WebSphere WebSphere Extended Deployment 버전 6.0.x

IBM

오브젝트 그리드 프로그래밍 모델 안내서

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 409 페이지의 『주의사항』의 일반 정보를 반드시 읽으십시오.

컴파일 날짜: 2006년 3월 27일

목차

의견을 보내는 방법

가장 정확한 양질의 정보를 제공하는 데 귀하의 피드백은 많은 도움이 됩니다.

- WebSphere Extended Deployment Information Center(`http://www.ibm.com/software/webservers/appserv/extend/library/`)의 항목에 대한 의견을 보내려면 다음을 수행하십시오.
  1. 웹 브라우저의 항목을 표시하고 항목의 끝으로 화면이동하십시오.
  2. 항목 맨 아래에 있는 **Feedback** 링크를 채운 후 제출하십시오.
- 이 항목 또는 다른 PDF 서적에 대한 주석을 보낼 경우 귀하의 의견을 전자 우편(**wasdoc@us.ibm.com**)으로 보낼 수 있습니다.

  문서 이름 및 번호를 표시하고 가능한 경우 의견에 대한 특정 페이지, 테이블 또는 그림 번호를 포함시키십시오.

귀하가 IBM에 정보를 제공하는 경우, IBM은 귀하의 권리를 침해하지 않는 범위 내에서 IBM이 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

제 1 장 샘플 응용프로그램을 실행하여 오브젝트 그리드 시작하기

이 주제를 참고하여 응용프로그램 세트에 대해 오브젝트를 사용 가능하게 하는 분산 컴퓨팅 프레임워크인 오브젝트 그리드를 시작하십시오.

WebSphere Extended Deployment Version 6.0 이상 및 WebSphere Application Server 버전 6.0.2 이상이 사용자 환경의 시스템에 이미 설치되어 있어야 합니다.

제한사항: WebSphere Extended Deployment 버전 6.0에서 오브젝트 그리드를 사용하는 경우, J2SE(Java 2 Platform, Standard Edition) 버전 1.4.2 이상의 환경 또는 WebSphere Application Server 버전 6.02 이상의 환경에서 오브젝트 그리드를 사용하려면 이에 대한 라이센스가 필요합니다. 세부사항은 영업 담당자에게 문의하십시오.

WebSphere Extended Deployment가 설치된 서버 시스템에 액세스하지 않고 오브젝트 그리드 응용프로그램을 개발하는 경우 로컬 시스템에서 해당 응용프로그램을 실행할 수 있습니다. 로컬 시스템에는 IBM SDK(Software Developer Kit) 또는 Eclipse가 설치되어 있어야 합니다.

로컬 시스템에서 오브젝트 그리드 응용프로그램을 개발하려면 사용자 설치로부터 로컬 시스템으로 다음 디렉토리를 복사하십시오.

- WebSphere Extended Deployment 버전 6.0.1을 사용 중인 경우 /lib/wsobjectgrid.jar 파일 및 /optionalLibraries/ObjectGrid/objectgridSamples.jar 파일을 작업 디렉토리로 복사하십시오.
- 혼합 서버 환경 설치를 통해 오브젝트 그리드를 설치한 경우 /ObjectGrid/lib/objectgrid.jar 및 /ObjectGrid/samples/objectgridSamples.jar 파일을 작업 디렉토리로 복사하십시오.

오브젝트 그리드로 설치된 Java 아카이브(JAR) 파일에 대한 자세한 정보는 오브젝트 그리드 패키징을 참조하십시오.

이 타스크를 참고하여 오브젝트 그리드 샘플 응용프로그램을 실행한 후 단계에 따라 진행하십시오. 이 타스크의 응용프로그램은 Java 명령행, Eclipse 또는 J2EE(Java 2 Platform, Enterprise Edition) 환경에서 실행할 수 있습니다.

- 명령행에서 오브젝트 그리드 샘플 응용프로그램을 실행하려면 명령행에서 오브젝트 그리드 샘플 응용프로그램 실행을 참조하십시오.
- Eclipse에서 오브젝트 그리드 샘플 응용프로그램을 실행하려면 Eclipse에서 오브젝트 그리드 샘플 응용프로그램 가져오기 및 사용을 참조하십시오.

- WebSphere Extended Deployment에서 오브젝트 그리드 샘플 응용프로그램을 실행하려면 WebSphere Extended Deployment에서 오브젝트 그리드 샘플 응용프로그램 로드 및 실행을 참조하십시오.

샘플 응용프로그램을 실행하고 개발 환경으로 샘플을 로드하여 오브젝트 그리드를 시작합니다.

명령행에서 오브젝트 그리드 샘플 응용프로그램 실행

이 주제를 참고하여 Java 명령행에서 오브젝트 그리드 사용 응용프로그램을 실행하고 오브젝트 그리드 구성을 테스트하십시오.

이 타스크를 시작하기 전에 독립형 오브젝트 그리드를 포함하는 혼합 서버 환경을 설치해야 합니다.

SDK(Software Development Kit)가 설치되어 있어야 합니다. 또한 오브젝트 그리드 샘플 응용프로그램에 대한 액세스 권한이 있어야 합니다. 자세한 정보는 오브젝트 그리드 시작하기를 참조하십시오.

이 타스크를 참고하여 오브젝트 그리드가 사용 가능한 응용프로그램을 빠르게 실행할 수 있습니다.

1. SDK(Software Development Kit) 버전을 확인하십시오. 오브젝트 그리드를 사용하려면 IBM SDK 1.4.2 이상이 필요합니다. 오브젝트 그리드 샘플 응용프로그램을 실행하기 전에 Java 환경을 테스트하려면 다음 단계를 수행하십시오.
   a. 명령행 프롬프트를 여십시오.
   b. 다음 명령을 입력하십시오.

      ```
      java -version
      ```

      명령을 올바르게 실행한 경우 다음 예와 유사한 텍스트가 표시됩니다.

      ```
      java version "1.4.2"
      Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)
      Classic VM (build 1.4.2, J2RE 1.4.2 IBM Windows 32 build cn142-20040820
      (JIT enabled: jitc))
      ```

      주: J2SE(Java 2 Platform, Standard Edition) 버전 1.3.x 소프트웨어 개발 킷(SDK)을 사용하여 이 샘플을 실행할 수도 있습니다. 자세한 정보는 오브젝트 그리드 패키징을 참조하십시오.

   오류가 표시된 경우 SDK가 설치되어 CLASSPATH에 있는지 확인하십시오.
2. 오브젝트 그리드 샘플 응용프로그램을 실행하십시오. 샘플 응용프로그램에서는 직원, 사무실 및 작업 위치와 관련된 단순한 사례를 설명합니다. 샘플 응용프로그램에서는 오브젝트 유형마다 맵을 포함하는 오브젝트 그리드 인스턴스를 작성합니다. 각 맵에는 오브젝트 그리드 캐싱 기능을 설명하도록 삽입 및 조정되는 항목이 있습니다.

a. 명령행을 열고 작업 디렉토리를 탐색하십시오. objectgrid.jar, asm.jar 및 cglib.jar 파일을 /ObjectGrid/lib 폴더에서 작업 디렉토리로 복사하십시오. /ObjectGrid/samples/objectgridSamples.jar를 작업 디렉토리로 복사하십시오.

b. 다음 명령을 실행하십시오.

```
cd working_directory
java -cp "objectgrid.jar;objectgridSamples.jar;asm.jar;cglib.jar"
com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample
```

다음 텍스트와 유사한 출력이 표시됩니다. 이 출력은 공개를 목적으로 요약되었습니다.

```
Initializing ObjectGridSample ...
resourcePath: META-INF/objectgrid-definition.xml
objectgridUrl:
 jar:file:/C:/temp/objg/objectgridSample.jar!/
META-INF/objectgrid-definition.xml
EmployeeOptimisticCallback returning version object for employee
 = Perry Cheng, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Hao Lee, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Ken Huang, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Jerry Anderson, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Kevin Bockhold, version = 0
-------------------------------------------
com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample status:
ivObjectGrid Name = clusterObjectGrid
ivObjectGrid      = com.ibm.ws.objectgrid.ObjectGridImpl@187b81e4
ivSession = com.ibm.ws.objectgrid.SessionImpl@6b0d81e4
ivEmpMap     = com.ibm.ws.objectgrid.ObjectMapImpl@6b1841e4
ivOfficeMap  = com.ibm.ws.objectgrid.ObjectMapImpl@6ba081e4
ivSiteMap    = com.ibm.ws.objectgrid.ObjectMapImpl@6bae01e4
ivCounterMap = com.ibm.ws.objectgrid.ObjectMapImpl@697b41e4
-------------------------------------------
interactiveMode = false
Action = populateMaps
CounterOptimisticCallback returning version object for
 counter name = Counter1, version = 0
CounterOptimisticCallback returning version object for
 counter name = Counter2, version = 0
CounterOptimisticCallback returning version object for
 counter name = Counter3, version = 0
ivCounterMap operations committed
ivOfficeMap operations committed
... ending with:
CounterOptimisticCallback returning version object for
 counter name = Counter1, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Ken Huang, version = 0
CounterOptimisticCallback returning version object for
 counter name = Counter2, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Perry Cheng, version = 0
CounterOptimisticCallback returning version object for counter name =
 Counter3, version = 0
EmployeeOptimisticCallback returning version object for employee =
```

```
 Jerry Anderson, version = 0
CounterOptimisticCallback returning version object for
  counter name = Counter4, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Hao Lee, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Kevin Bockhold, version = 1
DONE cleanup
```

3. 분산 오브젝트 그리드 샘플 응용프로그램을 실행하십시오.

   com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample 프로그램에서는 로컬 오브젝트 그리드 인스턴스를 데이터 캐시로 사용합니다. 모든 오브젝트는 로컬 JVM(Java virtual machine)에서 캐시됩니다. 오브젝트 그리드 클러스터에서 배치된 분산 오브젝트 그리드를 사용하려면 com.ibm.websphere.samples.objectgrid.distributed.DistributedObjectGridSample 프로그램을 사용하십시오. DistributedObjectGridSample 프로그램은 objectgridSamples.jar에 있습니다.

   a. 오브젝트 그리드 클러스터를 시작하십시오. 분산 오브젝트 그리드 샘플과 함께 사용하는 독립형 오브젝트 그리드 클러스터 시작에 관한 자세한 정보는 독립형 샘플 오브젝트 그리드 클러스터 시작을 참조하십시오.

   b. 오브젝트 그리드 서버를 시작한 다음 분산 오브젝트 그리드 샘플 응용프로그램을 다음 명령으로 실행할 수 있습니다.

      ```
      java -cp "objectgrid.jar;objectgridSamples.jar;asm.jar;cglib.jar"
      com.ibm.websphere.samples.objectgrid.distributed.DistributedObjectGridSample
      ```

   필수 오브젝트 그리드 클러스터를 시작한 후에는 DistributedObjectGridSample 프로그램은 ObjectGridSample 프로그램과 출력이 유사합니다.

Java 명령행에서 오브젝트 그리드 샘플 응용프로그램을 실행하여 오브젝트 그리드 기능을 테스트했습니다.

이 샘플의 소스는 objectgridSamples.jar 파일, 특히 com₩ibm₩websphere₩samples₩objectgrid₩basic₩ObjectGridSample.java 및 com₩ibm₩websphere₩samples₩objectgrid₩distributed₩DistributedObjectGridSample.java 파일에 있습니다.

독립형 샘플 오브젝트 그리드 클러스터 시작

분산 오브젝트 그리드 샘플을 실행하려면 필수 오브젝트 그리드를 호스트하는 오브젝트 그리드 클러스터를 시작해야 합니다.

혼합 서버 환경용 WebSphere Extended Deployment, 버전 6.0.x가 설치되었는지 확인하십시오.

이 타스크를 사용하여 `cluster-config-1.xml` 및 `cluster-objectgrid-definition.xml` 파일을 기초로 하는 오브젝트 그리드 서버를 시작하십시오. 이 타스크는 분산 오브젝트 그리드 샘플을 실행하는 데 필요합니다. 자세한 정보는 명령행에서 오브젝트 그리드 샘플 응용프로그램 실행 및 Eclipse에서 오브젝트 그리드 샘플 응용

프로그램 가져오기 및 사용을 참조하십시오. cluster-config-1.xml에만 오브젝트 그리드 서버 정의가 있습니다. 이 오브젝트 그리드 서버는 샘플 오브젝트 그리드 클러스터를 표시합니다.

1. *mse_install_root*/ObjectGrid/samples 디렉토리에서 objectgridSamples.jar 파일을 찾으십시오.
2. META-INF/cluster-config-1.xml 파일 및 META-INF/cluster-objectgrid-definition.xml 파일을 objectgridSamples.jar 파일에서 *mse_install_root*/ObjectGrid/samples 디렉토리로 추출하십시오.
3. JAVA_HOME 환경 변수가 설정되었는 지 그리고 Java 버전이 요구사항을 충족하는 지를 확인하십시오. 오브젝트 그리드 서버에서는 J2SE(Java2 Platform, Standard Edition) 버전 1.4.2 이상의 환경이 필요합니다. Java 환경을 확인하려면 다음 단계를 수행하십시오.
   a. JAVA_HOME 환경 변수를 확인하십시오. 명령 프롬프트에서 다음 명령을 발행하십시오.

      ```
      echo %JAVA_HOME%
      ```

      이 명령은 Java에 대한 경로를 표시합니다. JAVA_HOME 환경 변수를 설정해야 하는 경우, 다음 명령을 실행하십시오.

      ```
      set JAVA_HOME=JDK_INSTALL_ROOT
      ```

      *JDK_INSTALL_ROOT*를 Java 설치 디렉토리로 설정하십시오(예: c:₩java).
   b. Java 버전을 확인하십시오. 다음 명령을 실행하십시오.

      ```
      java -version
      ```

      버전이 J2SE(Java 2 Platform, Standard Edition) 버전 1.4.2 이상인지 확인하십시오.
4. 오브젝트 그리드 서버를 시작하십시오. 명령 프롬프트에서 다음 명령을 발행하십시오.

   ```
   cd mse_install_root/ObjectGrid/bin
   startOgServer.bat server1 -objectgridFile mse_install_root/ObjectGrid/
   samples/META-INF/cluster-objectgrid-definition.xml
   -clusterFile mse_install_root/ObjectGrid/samples/META-INF/
   cluster-config-1.xml
   -jvmArgs -cp mse_install_root/ObjectGrid/samples/objectgridSamples.jar
   ```

   중요사항: -jvmArgs 옵션을 사용하여 클래스 경로에 objectgridSamples.jar 파일을 지정해야 합니다. objectgridSamples.jar 파일에는 샘플 오브젝트 그리드 서버가 cluster-objectgrid-definition.xml 파일에 정의된 플러그인을 구현하는 데 필요한 클래스가 포함되어 있습니다. 이 JAR 파일은 맵에 저장된 오브젝트를 직렬화 또는 직렬화를 해제하는 데 사용

됩니다.

다음 텍스트와 유사한 출력이 표시됩니다. 이 출력은 공개를 목적으로 요약되었습니다.

```
************ Start Display Current Environment ************
[1/17/06 14:04:34:144 CST] 7daee176 Launcher
I CWOBJ2501I: Launching ObjectGrid server server1.
:
[1/17/06 14:04:37:719 CST] 7daee176 ServerRuntime
I CWOBJ1001I: ObjectGrid Server server1 is ready to process requests.
```

분산 오브젝트 그리드 샘플 응용프로그램을 실행하려면 명령행에서 오브젝트 그리드 샘플 응용프로그램 실행 또는 Eclipse에서 오브젝트 그리드 샘플 응용프로그램 가져오기 및 사용을 참조하십시오. 명령행에서 독립형 오브젝트 그리드 서버 시작 및 중지에 대한 자세한 정보는 89 페이지의 제 8 장 『명령행 지원』을 참조하십시오.

Eclipse에서 오브젝트 그리드 샘플 응용프로그램 가져오기 및 사용

이 타스크를 참고하여 Eclipse에서 오브젝트 그리드 샘플 응용프로그램을 가져와서 사용하십시오.

이 타스크를 시작하기 전에 독립형 오브젝트 그리드를 포함하는 혼합 서버 환경을 설치해야 합니다.

이 샘플 응용프로그램에서는 Eclipse 버전 3.1 이상을 사용하여 샘플을 가져온 다음 실행합니다. Eclipse.org에서 직접 다운로드하거나 Rational Application Developer를 설치하거나 또는 WebSphere Application Server에 포함된 Application Server Toolkit에서 Eclipse를 가져올 수 있습니다.

Eclipse를 사용하면 응용프로그램을 쉽게 디버그할 수 있습니다. 샘플 응용프로그램의 단계별 연습을 수행할 수 있습니다.

1. 프로젝트를 Eclipse로 가져오십시오.
   a. Eclipse 프로그램을 실행하십시오. Eclipse 설치 디렉토리에 있는 `eclipse.exe` 파일을 사용하십시오.
   b. Eclipse를 사용하여 새 프로젝트를 작성하십시오.
      1) **파일 > 새로 작성 > 프로젝트 > Java > Java 프로젝트**를 클릭하십시오. **다음**을 클릭하십시오.
      2) 프로젝트 이름을 입력하십시오. 예를 들어 `ObjectGridSamples`를 입력하십시오.
      3) **작업공간에서 새 프로젝트 작성**을 선택하십시오.
      4) 프로젝트 레이아웃 섹션에서 **기본값 구성**을 클릭하십시오.
      5) 소스 및 출력 폴더에서 **프로젝트**를 선택한 후 **확인**을 클릭하십시오.
      6) **다음**을 클릭하십시오.

7) **라이브러리** 탭을 클릭하십시오.

8) **외부 JAR 추가**를 클릭하십시오.

9) /ObjectGrid/lib 폴더를 탐색하여 **objectgrid.jar**, **asm.jar** 및 **cglib.jar** 파일을 선택하십시오. **JAR 선택** 마법사에서 **열기**를 클릭하십시오.

10) **완료**를 클릭하십시오.

2. objectgridSamples.jar 파일을 Java 프로젝트로 가져오십시오.
   a. Java 프로젝트를 마우스 오른쪽 단추로 클릭한 후 **가져오기**를 선택하십시오.
   b. **가져오기 소스 선택**에서 **Zip 파일**을 선택하십시오.
   c. **다음**을 클릭하십시오.
   d. **찾아보기**를 클릭하여 Zip 파일에서 가져오기 마법사를 여십시오.
   e. **objectgridSamples.jar** 파일을 여십시오. /ObjectGrid/samples 디렉토리를 탐색하십시오. **objectgridSamples.jar** 파일을 선택한 후 **열기**를 클릭하십시오.
   f. 루트 파일 트리의 선택란을 선택했는지 확인하십시오.
   g. **대상 폴더**에 이전 단계에서 작성한 Java 프로젝트(예: ObjectGridSamples 프로젝트)가 포함되어 있는지 확인하십시오.
   h. **완료**를 클릭하십시오.
3. Java 프로젝트의 특성을 선택하십시오.
   a. Java Perspective를 여십시오. **창 > Perspective 열기 > Java**를 클릭하십시오.
   b. 콘솔 보기로 이동하십시오. **창 > 보기 표시 > 콘솔**을 클릭하십시오.
   c. 패키지 탐색기 보기가 사용 가능하며 선택되었는지 확인하십시오. **창 > 보기 표시 > 패키지 탐색기**를 클릭하십시오.
   d. Java 프로젝트를 마우스 오른쪽 단추로 클릭한 후 **특성**을 선택하십시오.
   e. 왼쪽 패널에서 **Java 빌드 경로**를 클릭하십시오.
   f. 오른쪽 패널에서 **소스** 탭을 클릭하십시오.
   g. 프로젝트 루트가 빌드 경로 패널의 소스 폴더에 나열되었는지 확인하십시오.
   h. 오른쪽 패널에서 **라이브러리** 탭을 클릭하십시오.
   i. objectgrid.jar, asm.jar 및 cglib.jar 파일을 확인하고 JRE 시스템 라이브러리가 빌드 경로 패널의 JAR 및 클래스 폴더에 나열되었는지 확인하십시오.
   j. **확인**을 클릭하십시오.
4. 오브젝트 그리드 샘플을 실행하십시오.
   a. 패키지 탐색기 보기에서 Java 프로젝트를 펼치십시오.
   b. com.ibm.websphere.samples.objectgrid.basic 패키지를 펼치십시오.
   c. **ObjectGridSample.java** 파일을 마우스 오른쪽 단추로 클릭하십시오. **실행 > Java 응용프로그램**을 클릭하십시오.

d. Java 명령행에서 응용프로그램을 실행하면 유사한 출력이 콘솔에 표시됩니다. 출력에 대한 예는 명령행에서 오브젝트 그리드 샘플 응용프로그램 실행을 참조하십시오.

5. 분산 오브젝트 그리드 샘플을 실행하십시오. 분산 오브젝트 그리드 샘플을 실행하려면 오브젝트 그리드 클러스터를 구성해야 합니다. 이 샘플을 실행하는 경우 objectgridSamples.jar 파일에서 제공되는 미리 정의된 XML 구성 파일을 사용할 수 있습니다. 자세한 정보는 독립형 샘플 오브젝트 그리드 클러스터를 참조하십시오.

   오브젝트 그리드 서버가 시작되면, 다음 단계를 사용하여 분산 오브젝트 그리드 샘플 응용프로그램을 실행할 수 있습니다.

   a. 패키지 탐색기 보기에서 Java 프로젝트를 펼치십시오.
   b. com.ibm.websphere.samples.objectgrid.distributed 패키지를 펼치십시오.
   c. **DistributedObjectGridSample.java** 파일을 마우스 오른쪽 단추로 클릭하십시오. **실행 > Java 응용프로그램**을 클릭하십시오.
   d. 콘솔에서 오브젝트 그리드 샘플과 유사한 출력을 표시합니다.

프로젝트를 로드하고 디버거를 실행하는 단계도 SamplesGuide.htm 파일에 있습니다. SamplesGuide.htm 파일은 objectgridSamples.jar 파일의 doc 디렉토리에 있습니다.

관련 참조

오브젝트 그리드 패키징

WebSphere Extended Deployment 설치 또는 혼합 서버 환경을 설치하는 두 가지 방법으로 오브젝트 그리드 패키지에 액세스할 수 있습니다.

WebSphere Extended Deployment에서 오브젝트 그리드 샘플 응용프로그램 로드 및 실행

이 타스크를 참고하여 WebSphere Extended Deployment에서 J2EE(Java 2 Platform, Enterprise Edition) 오브젝트 그리드 샘플을 로드한 후 실행하십시오.

WebSphere Application Server 및 WebSphere Extended Deployment가 설치되어 있어야 합니다.

이 타스크를 사용하여 오브젝트 그리드 및 WebSphere Extended Deployment와의 통합을 이해하고 테스트할 수 있습니다. 자세한 정보는 313 페이지의 제 10 장 『WebSphere Application Server에서 오브젝트 그리드 통합』을 참조하십시오.

1. ObjectGridSample.ear 파일을 설치하십시오. 단일 Application Server 또는 클러스터에서 엔터프라이즈 아카이브(EAR) 파일을 설치할 수 있습니다. 관리 콘솔에서 ObjectGridSample.ear 파일을 설치하려면 다음 단계를 수행하십시오.

a. 관리 콘솔에서 **응용프로그램 > 새 응용프로그램 설치**를 클릭하십시오.

b. **응용프로그램 설치 준비** 페이지에서 오브젝트 그리드 샘플 응용프로그램의 위치를 지정하십시오. 예를 들어, `<install_root>/installableApps/ObjectGridSample.ear`을 찾아보십시오. **다음**을 클릭하십시오.

c. 두 번째 **응용프로그램 설치 준비** 페이지에서 기본 설정을 그대로 두고 **다음**을 클릭하십시오.

d. **설치 옵션 선택** 페이지에서 기본 설정을 그대로 두고 **다음**을 클릭하십시오.

e. **서버에 모듈 맵핑** 페이지에서 응용프로그램에 포함된 모듈을 설치할 전개 대상을 지정하십시오. 모든 모듈의 경우 **클러스터 및 서버** 목록에서 대상 서버 또는 클러스터를 선택하십시오. **모듈** 선택란을 선택하여 모든 응용프로그램 모듈을 선택하거나 개별 모듈을 선택하십시오.

f. 다음 페이지에서 기본값을 사용한 후 **완료**를 클릭하십시오.

g. 응용프로그램 설치를 완료한 후 **마스터 구성으로 저장**을 클릭하십시오.

h. **노드에서 변경사항 동기화** 옵션을 클릭하십시오. **엔터프라이즈 응용프로그램 > 저장** 페이지에서 **저장**을 클릭하십시오.

i. **확인**을 클릭하십시오.

2. 서버의 default_host에 해당하는 HTTP 포트를 확인하고 호스트 별명을 추가하십시오. 기본적으로설치 중 호스트 이름을 수정하지 않는 한 웹 모듈은 default_host 가상 호스트 이름으로 바인드됩니다. 클러스터에서 응용프로그램을 설치하는 경우 각 클러스터 구성원의 default_host에 해당하는 HTTP 포트의 호스트 별명을 하나 이상 구성해야 합니다. 또한 각 클러스터 구성원의 default_host에 해당하는 HTTP 포트를 확인하여 해당 호스트 별명을 관리 콘솔의 호스트 별명 목록에 추가해야 합니다. 서버의 default_host에 해당하는 HTTP 포트를 확인하려면 다음 단계를 수행하십시오.

   a. 관리 콘솔에서 **서버 > Application Server > *server_name***을 클릭하십시오.

   b. 통신 섹션에서 포트를 펼치십시오. **WC_defaulthost** 포트는 default_host 가상 호스트 포트입니다.

   호스트 별명을 추가하려면 다음 단계를 수행하십시오.

   a. 관리 콘솔에서 **환경 > 가상 호스트 > default_host > 호스트 별명 > 새로 작성**을 클릭하십시오.

   b. 호스트 이름의 기본값을 사용하고 포트를 지정하십시오.

   c. **확인**을 클릭하십시오.

3. 오브젝트 그리드 샘플 응용프로그램을 시작하십시오.

   - 서버에서 응용프로그램을 시작하려면 **서버 > Application Server**를 클릭하십시오. `ObjectGridSample.ear` 파일이 설치된 서버를 선택하십시오. **시작**을 클릭하십시오.

- 클러스터에서 응용프로그램을 시작하려면 **서버 > 클러스터**를 클릭하십시오. ObjectGridSample.ear 파일이 설치된 클러스터를 선택하십시오. **시작**을 클릭하십시오.

서버 또는 클러스터에서 응용프로그램을 시작하면 호스트 서버 또는 클러스터에서 독립적으로 응용프로그램을 중지 및 시작할 수 있습니다. 오브젝트 그리드 샘플 응용프로그램을 중지 또는 시작하려면 다음 단계를 수행하십시오.

a. 관리 콘솔에서 **응용프로그램 > 엔터프라이즈 응용프로그램**을 클릭하십시오.

b. 오브젝트 그리드 샘플 응용프로그램을 선택하십시오.

c. **시작** 또는 **중지**를 클릭하십시오.

4. 오브젝트 그리드 샘플에 액세스하십시오. 단일 서버 또는 클러스터에서 ObjectGridSample.ear 파일을 설치한 후 응용프로그램을 시작하면 다음 웹 주소에 있는 오브젝트 그리드 샘플에 액세스할 수 있습니다.

```
http://hostname:port/ObjectGridSample
```

예를 들어 호스트 이름이 localhost이고 포트 값이 9080인 경우 http://localhost:9080/ObjectGridSample을 웹 주소로 사용하십시오.

5. WebSphere Application Server 환경의 분산 오브젝트 그리드 기능을 테스트하십시오. ObjectGridSample.ear 파일에는 WebSphere Application Server 환경의 분산 오브젝트 그리드 사용에 대해 설명하는 DistributedObjectGridServlet servlet도 포함되어 있습니다. DistributedObjectGridServlet servlet을 호스트하는 Application Server는 필수 오브젝트 그리드 클러스터의 구성원인 오브젝트 그리드 서버도 호스트해야 합니다.

   - DistributedObjectGridServlet이 실행되도록 오브젝트 그리드 클러스터의 구성에 대한 자세한 정보는 WebSphere 환경에서 샘플 오브젝트 그리드 클러스터 시작을 참조하십시오.
   - Application Server에서 오브젝트 그리드 서버 시작에 대한 자세한 정보는 Application Server에서 오브젝트 그리드 서버 시작을 참조하십시오.

   ObjectGridSample.ear 파일이 설치된 Application Server에서 필수 오브젝트 그리드 서버를 호스트한 후, DistributedObjectGridServlet은 다른 Servlet과 동일한 방식으로 동작합니다. 다음 웹 주소에서 Servlet에 액세스할 수 있습니다: http://hostname:port/ObjectGridSample/DistributedObjectGridServlet. 예를 들어 호스트 이름이 localhost이고 포트 값이 9080인 경우, http://localhost:9080/ObjectGridSample/DistributedObjectGridServlet 웹 주소를 사용하십시오.

   다음 ObjectGrid*=all=enabled 추적 문자열을 사용하여 오브젝트 그리드 추적이 가능하도록 설정할 수 있습니다.

오브젝트 그리드 샘플 응용프로그램과 분산 오브젝트 그리드 샘플 응용프로그램을 WebSphere Extended Deployment 서버에 설치하고 구성했습니다.

서버 또는 클러스터에 응용프로그램을 설치했으면, 다음 웹 주소에서 응용프로그램을 시작한 다음 샘플 문서에 액세스할 수 있습니다.

`http://hostname:port/ObjectGridSample/docs/introduction.html`

예를 들어, 호스트 이름이 **localhost**이고 포트 값이 **9080**인 경우 `http://localhost:9080/ObjectGridSample/docs/introduction.html`을 웹 주소로 사용하십시오.

WebSphere 환경에서 샘플 오브젝트 그리드 클러스터 시작

이 타스크를 사용하여 단순한 오브젝트 그리드 클러스터를 시작하면 WebSphere Application Server 환경에서 분산 오브젝트 그리드 기능을 테스트할 수 있습니다.

WebSphere Extended Deployment가 설치되어야 합니다. ObjectGridSample.ear 파일을 Application Server에 설치해야 합니다. ObjectGridSample.ear 파일 설치에 대한 자세한 정보는 WebSphere Extended Deployment를 사용하여 오브젝트 그리드 샘플 응용프로그램 로드 및 실행을 참조하십시오.

이 타스크를 사용하여 cluster-config-1.xml 및 cluster-objectgrid-definition.xml 파일을 기반으로 오브젝트 그리드 서버를 호스트하는 Application Server를 설정하십시오.

cluster-config-1.xml 파일에만 오브젝트 그리드 서버 정의가 있습니다. 이 오브젝트 그리드 서버는 샘플 오브젝트 그리드 클러스터를 표시합니다. 독립형 Application Server나 하나의 클러스터 구성원이 있는 클러스터를 사용하여 샘플 오브젝트 그리드 서버를 호스트할 수 있습니다.

1. META-INF/cluster-config-1.xml 및 META-INF/cluster-objectgrid-definition.xml 파일을 /optionalLibraries/ObjectGrid/objectgridSamples.jar 파일에서 /optionalLibraries/ObjectGrid 디렉토리로 추출하십시오.
2. 필요한 일반 JVM 인수를 정의하십시오.
   a. 관리 콘솔에서 **서버 > Application Server > *server_name* > 프로세스 정의 > JVM(Java Virtual Machine)**을 클릭하십시오.
   b. 일반 JVM 인수 패널에서 다음 텍스트를 입력하십시오.

      ```
      -Dobjectgrid.server.name=server1
      -Dobjectgrid.xml.url=file:///<INSTALL_ROOT>₩optionalLibraries₩ObjectGrid₩
      META-INF₩cluster-objectgrid-definition.xml
      -Dobjectgrid.cluster.xml.url=file:///<INSTALL_ROOT>₩optionalLibraries₩
      ObjectGrid₩META-INF₩cluster-config-1.xml
      ```

      `INSTALL_ROOT`는 WebSphere Application Server 설치 루트 디렉토리입니다.
   c. **저장**을 클릭하십시오.
   d. **마스터 구성으로 저장**을 클릭하십시오.

e. **노드에서 변경사항 동기화** 옵션을 선택하십시오. **저장**을 클릭하십시오.

3. /optionalLibraries/ObjectGrid/objectgridSamples.jar 파일을 /classes 또는 lib/ext 디렉토리로 복사하십시오. objectgridSamples.jar에는 샘플 오브젝트 그리드 서버에서 cluster-objectgrid-definition.xml 파일에 정의된 플러그인을 구현하는 데 필요한 클래스가 들어 있습니다. 이 JAR 파일은 맵에 저장된 오브젝트 직렬화 또는 직렬화 해제에도 사용됩니다.
4. 서버를 재시작하여 변경을 적용하십시오.

Application Server에서 오브젝트 그리드 서버의 시작 및 중지에 대한 세부사항은 Application Server에서 오브젝트 그리드 서버 시작을 참조하십시오.

Application Server에서 오브젝트 그리드 서버 시작

Application Server 내에서 시작하도록 오브젝트 그리드 서버를 구성할 수 있습니다. WebSphere Application Server는 오브젝트 그리드 컴포넌트를 발견하고 자동으로 오브젝트 그리드 서버를 시작합니다.

WebSphere Extended Deployment 또는 WebSphere Business Integration Server 등의 애드온(add-on)이 설치될 경우를 포함하여 WebSphere Application Server 버전 6.0.2 이상에서 오브젝트 그리드 서버를 구성할 수 있습니다. WebSphere Application Server 버전 5.0.2와 같은 이전 버전 WebSphere Application Server에는 오브젝트 그리드를 클라이언트로 사용하는 응용프로그램이 있을 수 있지만, 오브젝트 그리드 서버 기능은 이전 Application Server 버전과 함께 배치될 수 없습니다.

복제가 가능한 클러스터 구성을 사용하는 경우, 고가용성 관리자가 필요합니다. 오브젝트 그리드 서버에서는 고가용성 관리자를 표준 Application Server와 다르게 사용합니다. 오브젝트 그리드 서버가 Application Server에 있는 경우, 오브젝트 그리드 서버에서는 고가용성 관리자 서비스를 구성, 초기화 또는 작성하지 않지만, Application Server에서 기존 고가용성 서비스를 사용합니다. 오브젝트 그리드 서버 사이의 복제인 경우, 오브젝트 그리드 서버는 동일한 코어 그룹의 구성원인 Application Server에서 실행되어야 합니다.

오브젝트 그리드 서버의 다른 모든 기능도 서버가 WebSphere Application Server에서 실행될 경우와 동일합니다. 오브젝트 그리드 클러스터 스펙에 세 서버가 포함되는 경우, 단일 코어 그룹에 있는 세 Application Server 모두는 이들 오브젝트 그리드 서버를 호스트할 수 있습니다. Application Server는 또한 클러스터가 동일한 코어 그룹에 속하는 동안에는 클러스터를 연결할 수도 있습니다. 가장 중요한 단계는 cluster.xml 파일에 있는 TCP/IP 호스트 이름 및 포트 정보를 서버와 상관시키는 것입니다.

이 타스크를 사용하여 WebSphere Application Server 환경의 Application Server 내에서 오브젝트 그리드 서버를 실행하십시오.

1. JVM(Java Virtual Machine)에 필수 사용자 정의 특성을 추가하십시오. 관리 콘솔에서 **서버 > Application Server > *server_name* > Java 및 프로세스 관리 > 프로세스 정의 > JVM(Java Virtual Machine) > 사용자 정의 특성**을 클릭하십시오. **새로 작성**을 클릭하십시오. 다음 사용자 정의 특성을 작성하십시오.

표 1. 오브젝트 그리드 서버에 대한 JVM 사용자 정의 특성

사용자 정의 특성 이름	설명	값 예
objectgrid.server.name	이 Application Server 내에서 사용할 오브젝트 그리드 서버 이름을 지정합니다. 제공된 이름은 오브젝트 그리드 클러스터 XML 파일에 정의된 서버 이름 중 하나여야 합니다.	server1
objectgrid.xml.url	오브젝트 그리드 XML 파일의 URL(Universal Resource Locator)을 지정합니다. 이 특성은 필수입니다.	file:///d:/was/etc/test/objectGridMatch.xml
objectgrid.cluster.xml.url	오브젝트 그리드 클러스터 XML 파일의 URL을 지정합니다. 이 특성은 필수입니다.	file:///d:/was/etc/test/csCluster0.xml

objectgrid.security.server.props	오브젝트 그리드 서버 보안 특성 파일의 URL을 지정합니다. 이 특성은 오브젝트 그리드 클러스터 XML 파일에서 보안을 사용 가능한 경우에만 필수입니다. 클러스터 XML 파일에서 보안이 가능한지를 판별하려면 다음 텍스트를 확인하십시오.
	`<cluster name="cluster1" securityEnabled="true"`
	`....`
	
	securityEnabled 속성이 false로 설정되었으면 이 특성을 정의할 필요가 없습니다.
	
	security.ogserver.props 파일을 템플리트로 사용하십시오. 이 파일에서 이 특성의 의미와 사용 방법에 대해서는 149 페이지의 『오브젝트 그리드 보안』을 참조하십시오.
		file:///d:/was/optionalLibraries/ObjectGrid/properties/security.ogserver.props

또한 관리 콘솔의 **Java Virtual Machine** 패널에 있는 **일반 JVM 인수** 필드에 이 JVM 특성을 정의할 수도 있습니다. 다음은 일반 JVM 인수 필드 값의 예제입니다.

```
-Dobjectgrid.server.name=server1
-Dobjectgrid.xml.url=file:///<INSTALL_ROOT>WoptionalLibrariesWObjectGridW
META-INFWcluster-objectgrid-definition.xml
-Dobjectgrid.cluster.xml.url=file:///<INSTALL_ROOT>WoptionalLibrariesW
ObjectGridWMETA-INFWcluster-config-1.xml
```

2. 변경사항을 저장하고 Application Server를 재시작하십시오. WebSphere Application Server는 오브젝트 그리드 컴포넌트를 발견하고 자동으로 오브젝트 그리드 서버를 시작합니다.

   Application Server에 있는 오브젝트 그리드에서는 채널 프레임워크를 사용하여 특히 클라이언트 액세스포트라는 오브젝트 그리드 클라이언트와 상호 작동합니다. 오브젝트 그리드 서버가 시작되면 WebSphere Application Server가 있는지 찾아보고 이미 Application Server에서 실행 중인 채널 프레임워크를 사용합니다. 오브젝트 그리드 서버는 Application Server에서 채널 프레임워크가 작성 또는 시작되지 않은 경우에만 자신의 채널 프레임워크를 작성 및 시작합니다.
3. 오브젝트 그리드 서버를 중지하십시오. 연관된 Application Server를 중지하여 오브젝트 그리드 서버를 중지합니다. 오브젝트 그리드 System Management 명령을 사용하여 오브젝트 그리드 서버를 중지할 수 없습니다.

WebSphere Application Server 환경에 있는 Application Server에서 오브젝트 그리드 서버를 실행 중입니다.

제 2 장 오브젝트 그리드

오브젝트 그리드는 J2SE(Java 2 Platform, Standard Edition) 및 J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램을 위한 확장 가능한 트랜잭션 오브젝트 캐싱 프레임워크입니다.

오브젝트 그리드 API를 사용하여 오브젝트 그리드 프레임워크에서 오브젝트를 검색, 저장, 삭제 및 갱신하기 위한 응용프로그램을 개발할 수 있습니다. 또한 캐시 갱신사항 모니터, 외부 데이터 소스로 데이터 검색 및 저장, 캐시에서 항목 제거 관리, 사용자 고유의 오브젝트 그리드 응용프로그램 환경에 대한 백그라운드 캐시 기능 처리를 수행하는 사용자 정의 플러그인을 구현할 수도 있습니다.

Map 기반 API

오브젝트 그리드는 java.util.Map 인터페이스를 기반으로 하는 API입니다. API는 트랜잭션 블록으로 조작 그룹화를 지원할 수 있도록 확장됩니다. 이 인터페이스는 java.util.Map 인터페이스의 수퍼세트이며 일괄처리 조작, 무효화, 키워드 연관 및 명시적 삽입 및 갱신 지원을 추가합니다. Java 맵 시멘틱은 다음 개선사항을 구현할 수 있도록 확장점을 사용하여 향상됩니다.

- 캐시 항목 지속 시간을 세밀하게 조정하기 위한 캐시 축출기
- 트랜잭션 관리를 주의하여 제어하고 선택적으로 J2EE 환경에서 WebSphere 트랜잭션 관리자와 동작하는 트랜잭션 콜백 인터페이스
- 응용프로그램 프로그래머가 오브젝트 그리드 맵의 get 및 put 조작을 사용할 때 자동으로 데이터를 검색하고 데이터베이스에서 데이터를 이동시키는 로더 구현
- 트랜잭션이 발생하고 전체적으로 오브젝트 그리드 프레임워크에 적용되거나 특정 맵 인스턴스에 적용될 때 확약된 모든 트랜잭션에 대한 정보를 제공할 수 있는 리스너 인터페이스
- 좀 더 효율적으로 키와 값을 복사하고 직렬화하는 오브젝트 변환기 인터페이스

오브젝트 그리드 환경

기존 오퍼링 중 하나를 설치하여 오브젝트 그리드 프레임워크를 사용할 수 있습니다.

- 오브젝트 그리드는 WebSphere Extended Deployment 버전 6.0.1과 통합되며 전체 설치의 일부분입니다.
- 독립형 오브젝트 그리드는 혼합 서버 환경(MSE) 설치의 일부분입니다.

위 오퍼링 모두에서 오브젝트 그리드는 클라이언트/서버 기능을 지원합니다. 서버 런타임은 분산 오브젝트 캐시의 전체 클러스터링, 복제 및 파티셔닝을 지원합니다. 클라이언트 런타임은 가까운 캐시 및 원격 클러스터로의 워크로드 관리 라우팅 로직 개념을 지원합니다. 클라이언트 런타임은 로컬 오브젝트 맵 작성도 지원합니다.

지원 레벨은 클라이언트 런타임, 서버 런타임, 통합 오브젝트 그리드 또는 독립형 오브젝트 그리드를 실행하는 경우에 따라 달라집니다.

WebSphere Extended Deployment 오퍼링과 통합된 오브젝트 그리드

서버 런타임: 서버 런타임이 통합됩니다. WebSphere Extended Deployment 버전 6.0.1의 경우, 통합된 런타임은 z/OS 플랫폼에서 지원되지 않습니다.

클라이언트 런타임: 클라이언트 런타임은 WebSphere Application Server 버전 5.0.2 이상을 포함하여 JDK 레벨 1.3.1 이상의 J2SE 및 J2EE에서 지원됩니다. 클라이언트 런타임은 z/OS 플랫폼에서 전체적으로 지원됩니다.

독립형 오브젝트 그리드 오퍼링

서버 런타임: 서버 런타임은 독립형 JVM(Java Virtual Machine)에서 단일 서버로서 또는 서버 클러스터로서 실행될 수 있습니다. 독립형 서버는 JDK 레벨 1.4.2 이상인 대부분의 J2SE 및 J2EE 플랫폼에서 지원됩니다. 독립형 서버는 WebSphere Application Server 버전 6.0.2 이상에서 지원됩니다. 독립형 서버 런타임은 WebSphere Extended Deployment 버전 6.0.1용 z/OS 플랫폼에서 지원되지 않습니다.

클라이언트 런타임: 클라이언트 런타임은 WebSphere Application Sever 버전 5.0.2 이상을 포함하여 JDK 레벨 1.3.1 이상의 J2SE 및 J2EE 플랫폼에서 지원됩니다.

세션 관리

오브젝트 그리드에 HTTP 세션 오브젝트를 저장하는 분산 HTTP 세션 관리 전체 구현이 제공됩니다.

간단한 설치

몇 가지의 간단한 단계로 오브젝트 그리드를 설치 및 구성할 수 있습니다. 이 단계에는 JAR(Java Archive) 파일을 클래스 경로에 복사하고 몇 개의 구성 지시문을 정의하는 작업이 포함됩니다.

트랜잭션 방식의 변경사항

모든 변경사항은 강력한 프로그램 인터페이스를 보장하기 위해 트랜잭션 컨텍스트로 수행됩니다. 응용프로그램 내에서 트랜잭션을 명시적으로 제어하거나 응용프로그램이 자동 확약 프로그래밍 모드를 사용할 수 있습니다. 이러한 트랜잭션 방식의 변경사항은 비동기화 및 동기화 모드의 오브젝트 그리드 클러스터에서 복제되어 확장 가능하고 내결함성이 있는 액세스를 제공할 수 있습니다.

단일 JVM(Java Virtual Machine)에서 실행되는 단일 그리드에서 하나 이상의 JVM(Java Virtual Machine) 오브젝트 그리드 클러스터가 포함된 그리드까지 오브젝트 그리드를 확장할 수 있습니다. 이 서버에서는 API를 대형 세트의 오브젝트 그리드 사용 가능한

클라이언트로 맵핑하여 데이터를 작성할 수 있습니다. 오브젝트 그리드 클라이언트에서는 기본 맵 Java API를 사용합니다. 그러나 오브젝트 그리드 클라이언트가 네트워크에서 정보를 보유하는 다른 오브젝트 그리드 서버에 연결될 수 있으므로 응용프로그램 개발자는 Java TCP/IP 및 원격 메소드 호출(RMI) API를 개발할 필요가 없습니다. 데이터 세트가 단일 JVM에 비해 너무 큰 경우 오브젝트 그리드를 사용하여 데이터를 파티셔닝할 수 있습니다.

오브젝트 그리드는 또한 고가용성 기능이 추가된 응용프로그램을 제공합니다. 오브젝트 공유는 기본 서버, 하나 이상의 복제 서버 및 하나 이상의 대기 서버가 있는 복제 모델을 기초로 합니다. 이 복제 서버 클러스터를 복제 그룹이라고도 합니다. 복제 그룹에 대한 액세스가 쓰기 조작인 경우, 요청은 기본 서버로 라우트됩니다. 액세스가 읽기 조작이거나 맵이 읽기 전용 맵인 경우, 요청은 기본 서버 또는 복제 서버로 라우트할 수 있습니다. 대기 서버는 서버가 실패하면 잠재 복제 서버로 정의됩니다. 기본 서버에서 실패하면 복제 서버가 기본 서버가 되어 사용 불가능을 최소화합니다. 이 동작은 사용자의 요구에 따라 구성 및 확장이 가능합니다.

좀 더 간단한 오브젝트 전파 방법을 사용하려면 더 낮은 품질의 피어 투 피어 모델을 사용할 수 있으며 이것은 Extended Deployment 버전 6.0에서 작성된 것입니다. 더 간단한 분산 트랜잭션 방식의 지원을 사용하면 피어에게 메시지 전송을 사용하여 변경을 알릴 수 있습니다. WebSphere Application Server 버전 6.0.2 이상을 실행 중인 경우 메시지 전송이 내장되어 있습니다. WebSphere Application Server 버전 6.0.2 이상을 실행하지 않는 경우, Java Messaging Service(JMS) 프로바이더와 같은 다른 메시지 전송을 지원해야 합니다.

주입 컨테이너 호환 가능 API

단순 XML 파일을 사용하거나 Java API를 사용하여 프로그램 방식으로 오브젝트 그리드를 구성하십시오. Java API는 주입 기반 프레임워크를 사용하여 응용프로그램을 구성 중인 환경에서도 작동하도록 설계되었습니다. IoC(Inversion of Control) 컨테이너로 오브젝트 그리드 오브젝트의 API 및 인터페이스를 호출한 후 키 오브젝트 그리드 오브젝트에 대한 참조를 응용프로그램에 주입할 수도 있습니다.

확장 가능 아키텍처

대부분의 오브젝트 그리드 프레임워크 요소는 플러그인을 개발하여 확장할 수 있습니다. 오브젝트 그리드를 조정하여 응용프로그램이 일관성과 성능 사이에서 취사 선택할 수 있게 할 수 있습니다. 플러그인 사용자 정의 코드는 다음 응용프로그램별 동작을 지원할 수도 있습니다.

- 초기화, 트랜잭션 시작, 트랜잭션 종료 및 제거에 대한 오브젝트 그리드 인스턴스 이벤트를 청취합니다.
- 트랜잭션 콜백을 호출하여 트랜잭션 특정 처리가 사용 가능하도록 합니다.
- 일반 오브젝트 그리드 트랜잭션으로 특정의 공통 트랜잭션 정책을 구현합니다.

- 외부 데이터 저장소 및 기타 정보저장소에 대해 투명하고 공통적인 시작점 및 종료점에 로더를 사용합니다.
- ObjectTransformer 인터페이스를 사용하여 특정 방법으로 직렬화할 수 없는 오브젝트를 처리합니다.

기본 오브젝트 그리드 캐시 API 인터페이스 사용에 영향을 주지 않고 이 작동을 각각 구현할 수 있습니다. 이와 같은 투명성으로, 캐시 하부 구조를 사용하는 응용프로그램은 응용프로그램에 영향을 주지 않으면서 데이터 스토어 및 트랜잭션 처리는 크게 변경시킬 수 있습니다.

오브젝트 그리드를 1차 API 또는 2단계 캐시로 사용

응용프로그램은 오브젝트 그리드 API를 lookaside 캐시 또는 동시 기록 캐시로 직접 사용할 수 있습니다. 동시 기록 모드에서 응용프로그램은 로더 오브젝트를 플러그인하여 오브젝트 그리드가 변경사항을 적용하고 응용프로그램으로 직접 투명하게 데이터를 페치할 수 있도록 합니다. 오브젝트 그리드는 어댑터를 작성하여 널리 사용되는 관계형 맵퍼에 2단계 캐시로 사용될 수도 있습니다. 응용프로그램이 오브젝트 관계형 맵퍼의 API를 1차 API로 사용하여 데이터에 액세스하기 때문에 이 모드에서는 응용프로그램에서 캐시를 볼 수 없습니다.

제 3 장 오브젝트 그리드 개요

오브젝트 그리드는 Java 맵 기반 데이터 액세스 모델 및 분산 캐싱 기술을 제공합니다. 오브젝트 그리드를 사용하여 사용 가능한 클러스터링 환경을 구성할 수 있습니다. 오브젝트 그리드 클라이언트는 대규모 통합 솔루션을 위해 여러 오브젝트 그리드 클러스터에 동시에 접속할 수 있습니다. 또한 오브젝트 그리드는 둘 이상의 JVM(Java Virtual Machine)에 있는 데이터로 표준화된 대량의 정보를 위해 고급 분산 데이터 파티셔닝 솔루션을 제공합니다. 기본적으로 오브젝트 그리드는 로컬 및 분산 캐싱을 허용하는 일련의 표준화된 Java API 및 네트워크 서비스입니다. 분산 및 확장 가능한 데이터 서비스의 방대한 배열에서 까지의 솔루션 확장이 고급 Java 맵 솔루션이 필요한 단일 JVM(Java Virtual Machine) 엔터프라이즈 전반에 대한 여러 오브젝트 그리드 클러스터에서 필요합니다.

단일 JVM(Java Virtual Machine)의 오브젝트 그리드

오브젝트 그리드의 가장 기본적인 사용법은 단일 JVM에 있습니다.

오브젝트 그리드를 사용하여 일련의 오브젝트 그리드 인스턴스를 작성할 수 있습니다. 각 오브젝트 그리드 인스턴스는 하나 이상의 Java 맵 호환 가능한 인스턴스를 포함할 수 있습니다. Java 맵 인스턴스는 Java 프로그래머에게 익숙한 GET 및 PUT 인스턴스와 함께 현재 Java 맵 인터페이스 및 기능에서 제공하지 않는 추가 기능을 제공합니다. 다음 다이어그램에서 오브젝트 그리드의 가장 기본적인 사용법에 대해 설명합니다.

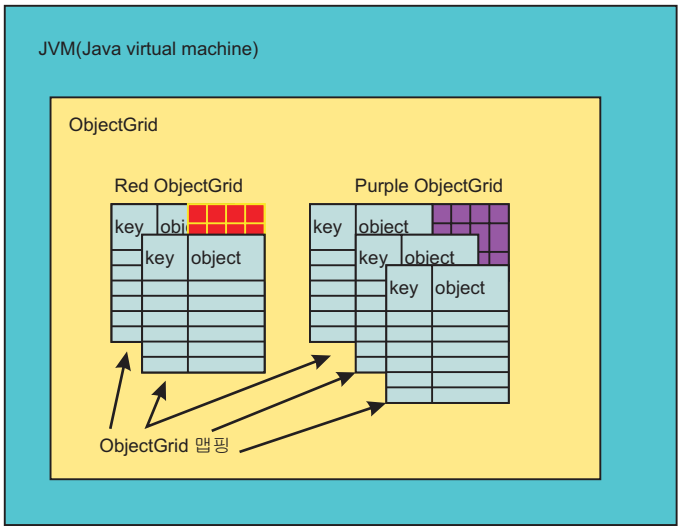


그림 1. 오브젝트 그리드 JVM 사용법

오브젝트 그리드 및 오브젝트 그리드 맵은 표준 Java 맵 인터페이스에 현재 제공되지 않는 다수의 기능을 포함합니다. 이 기능으로는 트랜잭션 액세스, 다양한 유형의 잠금

계획(없음, 변경이 예상되지 않음 및 변경이 예상됨), 플러그 앤 플레이 축출, GET 및 PUT API 사용의 부정적 영향으로 데이터베이스와 심리스 상호 작용, 그리고 기타 다수의 성능이 있습니다. 오브젝트 그리드에 대한 사용자 확장을 개발할 수도 있습니다. 예를 들어, 지정된 맵 인스턴스에 확약된 각 트랜잭션의 결과를 제공하는 맵 리스너를 개발할 수 있습니다. 예를 들어, 사용자는 변경사항을 지방 사무소 위치의 파일에 로그하여 트랜잭션 유실에 대비하거나 JMS(Java Message Service) 또는 일부 기타 하부 구조로 변경사항을 전달할 수 있습니다.

이전 다이어그램에서 JVM에는 두 개의 오브젝트 그리드 인스턴스가 있는데, 하나는 사용할 두 개의 Java 맵과 유사한 오브젝트를 가지고 있고 다른 하나는 세 개의 Map 오브젝트를 가지고 있습니다. Map 오브젝트는 2차원으로 키 및 오브젝트 쌍 지정을 표준 Java 맵처럼 조작할 수 있습니다. 단일 오브젝트 그리드 인스턴스는 다수의 특정 맵 인스턴스를 지원할 수 있습니다.

다음 구성은 Red 및 Purple 오브젝트 그리드 인스턴스의 기본 오브젝트 그리드 구성입니다.

```
<?xml version="1.0" encoding="UTF-8" ?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="Red">
<backingMap name="FirstRedMap" readOnly="false" />
<backingMap name="SecondRedMap readOnly="false" />
</objetGrid>
<objectGrid name="Purple">
<backingMap name="FirstPurpleMap readOnly="false" />
<backingMap name="SecondPurpleMap readOnly="false" />
<backingMap name="ThirdPurpleMap readOnly="false" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

분산 오브젝트 그리드

단일 JVM에포함된 오브젝트 그리드 JAR(Java Archive)을 사용하는 것 외에도 분산 환경에서 오브젝트 그리드를 사용할 수 있습니다. 이 환경에서 오브젝트 그리드 클러스터를 작성할 수 있습니다. 오브젝트 그리드 클러스터는 각각 단일 JVM을 보유하는 일련의 오브젝트 그리드 서버로 구성됩니다.

19 페이지의 『단일 JVM(Java Virtual Machine)의 오브젝트 그리드』 주제에서는 오브젝트 그리드의 Java 맵 개념 지원에 대해 설명합니다. 또한 이 개념은 단일 JVM 및 하나 이상의 원격 오브젝트 그리드 캐시 클러스터에 연결하는 Java 클라이언트에서 로컬로 지원됩니다. 오브젝트 그리드 서버는 단일 JVM 경우에 이미 위에서 설명한 기본 기능을 분배할 수 있습니다. 예를 들어, 여러 클라이언트가 없음, 변경이 예상되지 않음 또는 변경이 예상됨을 사용하여 동일한 오브젝트 그리드 인스턴스 맵을 공유할 수 있

습니다. 이외에도 오브젝트 그리드 클러스터 서버의 축출기가 서버측 Map 인스턴스 데이터의 축출을 관리할 수 있습니다. 모든 클라이언트는 공통 GET 및 PUT 시멘틱을 사용할 수 있고 각 클라이언트에 JDBC(Java Database Connectivity) 드라이버를 전개 및 관리하는 대신 오브젝트 그리드 클러스터 서버에 구성된 로더가 데이터베이스와의 모든 상호 작용을 수행합니다.

다음 다이어그램에서 JVM에는 두 개의 오브젝트 그리드 인스턴스가 있는데, 하나는 사용할 두 개의 Java 맵과 유사한 오브젝트를 가지고 있으며 다른 하나는 세 개의 Java 맵과 유사한 오브젝트를 가지고 있습니다. 각 맵은 키와 오브젝트를 허용하는 2차원 오브젝트입니다. 단일 오브젝트 그리드 인스턴스는 주로 응용프로그램의 요구사항에 따라 달라지는 다수의 맵을 지원할 수 있습니다. 이 경우에서 차이점은 맵이 오브젝트 그리드 클러스터 서버 내에 포함된다는 것입니다. 클라이언트는 보통 Java 응용프로그램 또는 J2EE(Java 2 Platform, Enterprise Edition) Application Server일 수 있습니다.

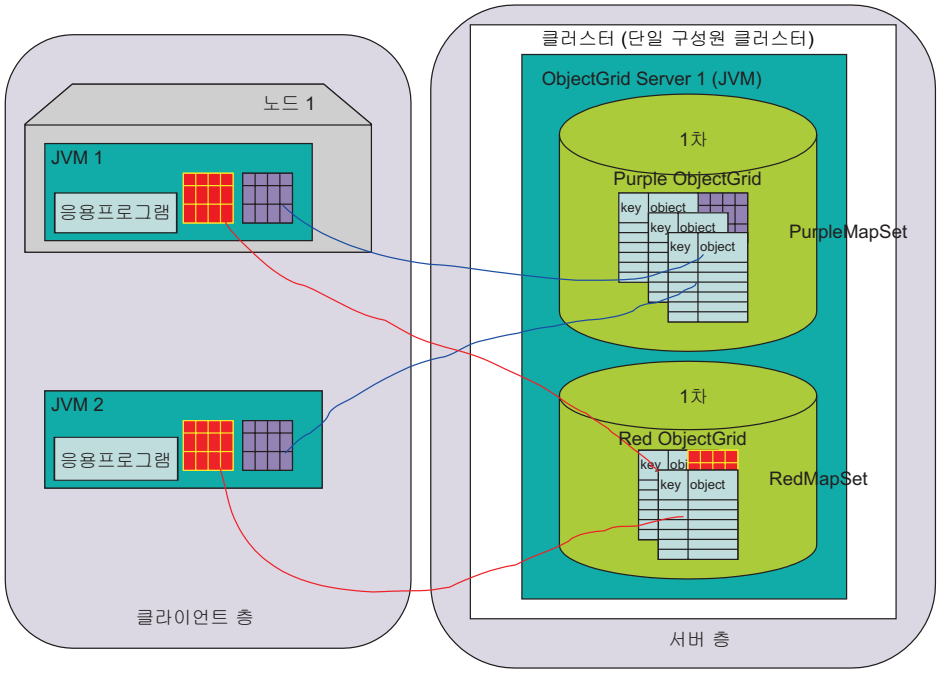


그림 2. 분산 오브젝트 그리드 단일 서버 토폴로지(두 개의 MapSet 보유)

오브젝트 그리드 클라이언트

오브젝트 그리드 클라이언트는 오브젝트 그리드 클러스터에 연결하고 오브젝트 그리드 클러스터의 광범위한 구성을 통해 부트스트랩한 다음 실제로 분산되어 있는 오브젝트 그리드 맵 조작을 수행하는 일련의 API로 구성되어 있습니다. 오브젝트 그리드 클라이언트는 분산 방식으로 오브젝트 그리드를 사용하는 고유 JVM 인스턴스 내의 Java 응용프로그램입니다. 또한 분산 오브젝트 그리드 클라이언트는 여전히 동일한 JVM(Java Virtual Machine)에서 분산되지 않은 기능을 사용할 수 있습니다. 오브젝트 그리드 클

라이언트 사용법은 여러 병렬 오브젝트 그리드 연결을 포함하고 각각 보안이 사용 가능하며 다른 사용자를 대신하는 전체 Application Server만큼 복잡할 수 있습니다.

분산 동작을 사용 가능하게 하려면 오브젝트 그리드 클러스터(오브젝트 그리드 솔루션의 서버측 서비스)를 작성해야 합니다. 필요한 추가 구성은 오브젝트 그리드 구성 파일 외에 오브젝트 그리드 클러스터 XML 파일입니다.

다음은 이전 다이어그램의 오브젝트 그리드 네트워크 전개를 구성하는 오브젝트 그리드 클러스터 XML입니다.

```
<?xml version="1.0" encoding="UTF-8" ?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!– single server ->
<serverDefinition name="server1" host="localhost" clientAccessPort="12053"
peerAccessPort="12500" />
</cluster>
<objectGridBinding ref="Red">
<mapSet name="RedMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstRedMap" />
<map ref="SecondRedMap" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsParitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

이 구성은 server1 서버를 포함하는 단일 클러스터인 ″cluster1″을 설명합니다. server1 서버는 두 개의 오브젝트 그리드인 ″Red″와 ″Purple″을 호스트합니다. 구성 파일은 파티션 및 복제를 위한 정보도 지정합니다. 오브젝트 그리드에서 오브젝트 그리드 클라이언트-서버를 지원하려면 프로그래머가 오브젝트 그리드 클러스터에 정의된 한 서버에 연결해야 합니다. 연결 처리 중에 오브젝트 그리드 및 오브젝트 그리드 클러스터 구성은 클라이언트에 동적으로 다운로드되어 사용법에 해당하는 클라이언트 준비와 클라이언트 측 구성 컨텐츠를 관리해야 하는 작업을 크게 단순화시킵니다. ″연결″ 조작을 수행하는 오브젝트 그리드 클라이언트 외에도 로컬 JVM과 오브젝트 그리드 클러스터에서 실제로 호스트되는 JVM으로 범위가 제한된 오브젝트 그리드를 사용하는 프로그래밍 API와 개념은 일반적으로 동일합니다.

오브젝트 그리드 클러스터 초기화

오브젝트 그리드와 함께 제공된 명령행 도구로 클러스터 내 오브젝트 그리드 서버를 시작할 수 있습니다. 오브젝트 그리드 응용프로그램은 오브젝트 그리드 클라이언트를 포함할 수 있고 다른 Java API 라이브러리가 응용프로그램 개발 프레임워크에 통합될 때 통합될 수 있습니다. 그러나, 두 경우 모두 오브젝트 그리드 사용법을 초기화해야 합니다.

로컬 JVM(Java Virtual Machine) 사용법 시나리오 또는 분산 오브젝트 그리드 클러스터에서 작업하려면 관리 가능한 접근 방식을 통해 부트스트랩에 유효한 구성을 확보해야 합니다. 오브젝트 그리드 클라이언트와 오브젝트 그리드 클러스터 서버는 동일한 구성을 사용해야 합니다. 프로그래머는 JVM의 단일 Java 응용프로그램 내에서 한정될 수 있는 매우 단순한 구성으로 시작할 수 있습니다.

그런 다음, 다중 클라이언트 동시 사용자 테스트를 시작할 준비가 되면 첫 번째 단일 서버 오브젝트 그리드 클러스터를 작성합니다. 초기 클라이언트-서버 기반 테스트가 완료된 후 관리 담당과 협력해서 복제 및 기타 고가용성 솔루션 요구사항을 실험할 수 있습니다. 각각의 개발 요구사항을 문제없이 진행하려면 더욱 풍부한 구성 파일이 필요합니다.

설명된 각 개발 단계에서 이러한 고급 기능을 각각 사용 가능하게 하는 구성 파일의 변경사항은 비교적 많지 않지만 솔루션 개발의 각 단계에서는 다른 버전의 구성 파일이 필요합니다. 변경사항을 상호 구축하는 것이 중요합니다. 솔루션을 개발하는 데이터 양이 단일 시스템에 과도하지 않거나 개발 목적을 위해 인위적으로 제한될 수 있는 경우 단일 시스템에서 복제된 솔루션을 단위 테스트할 수 있습니다.

XML을 사용하여 오브젝트 그리드 구성

하나 이상의 클라이언트와 하나 이상의 오브젝트 그리드 서버가 있는 분산 오브젝트 그리드 구성에는 XML 구성이 필요합니다. 기본 오브젝트 그리드 XML 구성 파일 외에 오브젝트 그리드 클러스터 XML 설명을 작성해야 합니다.

단일 오브젝트 그리드 XML 구성 설명과 오브젝트 그리드 클러스터 XML 구성 설명에서 단일 오브젝트 그리드 클러스터의 서버 및 클라이언트에게 예상된 기능을 수행하는 데 필요한 정보를 제공합니다. 사용자 환경에 여러 오브젝트 그리드 클러스터가 있을 수 있지만 클러스터 특정 오브젝트 그리드 클러스터 XML 문서는 특정 클러스터를 설명해야 합니다.

오브젝트 그리드가 시작하는 데 필요한 구성 정보는 표준 URL 접근 방식을 통해 확보할 수 있습니다. 예를 들어, 클라이언트 및 서버는 실제 파일 또는 HTTP URL을 통해 XML 파일을 얻을 수 있습니다.

다음 다이어그램에 설명된 분산 오브젝트 그리드 환경에서는 URL 또는 파일 시스템의 단순 파일(파일 URL은 복잡할 수 있기 때문)을 통해 오브젝트 그리드 서버의 구성을

검색하도록 오브젝트 그리드 서버의 초기 세트를 명령행으로 구성할 수 있습니다. 그러나, 더욱 효과적인 접근 방식은 이미 클러스터 내에서 작동하는 다른 서버로부터 동일한 오브젝트 그리드 클러스터 내에 있는 후속 서버를 부트스트랩하여 이 후속 서버를 시작하는 것입니다. 이 접근 방식은 관리자가 오브젝트 그리드 클라이언트 또는 서버를 호스트하는 각 시스템의 구성 파일을 추적할 필요가 없으므로 훨씬 더 관리하기가 용이합니다. 뿐만 아니라 부트스트랩하여 시작되는 서버에서 XML이 이미 성공적으로 처리되므로 XML 구성 오류가 확실하게 감소될 수 있습니다.

부트스트랩핑

오브젝트 그리드 서버 부트스트랩

다음 다이어그램은 동일한 오브젝트 그리드 구성을 호스트하지만 다수의 복제 클러스터 구성을 제공하는 일반 오브젝트 그리드 클러스터 환경의 부트스트랩을 설명합니다. 이 경우, 첫 번째 서버는 HTTP URL을 통해 부트스트랩하고 두 번째 및 세 번째 서버는 첫 번째에서 시작됩니다. 두 번째 및 세 번째 서버는 첫 번째 서버와 동일한 URL에서도 시작될 수 있습니다.

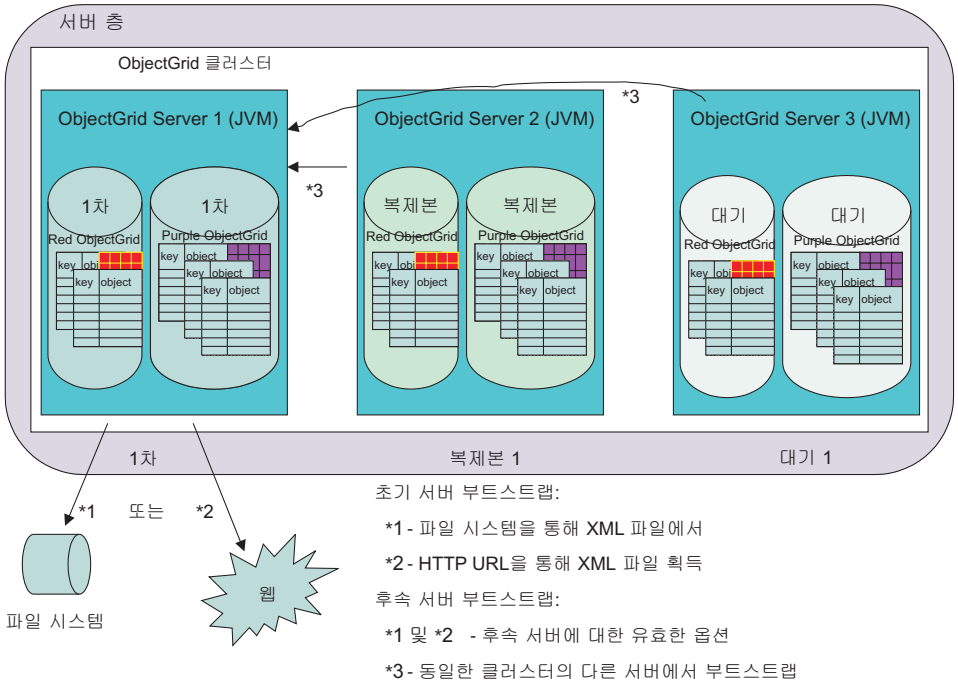


그림 3. XML 파일 구성을 통해 또는 기존 서버에서 초기 서버 부트스트랩

이전 다이어그램에 설명된 대로 cluster1 클러스터의 server1 서버가 부트스트랩할 초기 서버입니다. server1 서버는 파일 시스템의 XML 파일을 통해 또는 로컬 파일의 URL, 원격 HTTP 서버 또는 다른 유효한 URL 옵션을 통해 부트스트랩할 수 있습니다. server2 서버와 server3 서버는 이들의 중간 값을 통해 시작되거나 server1 서버를

구성 부트스트랩 호스트로 대상화하여 시작될 수 있습니다. 일반적으로 기타 서버에서 후속 서버를 부트스트랩하면 클러스터 구성원 전반에 걸쳐 구성의 일관성이 유지됩니다.

이 특정 시나리오에서, server1 서버가 실패하면 server2 및 server3은 여전이 작동 가능하고, server1은 server2 또는 server3에서 부트스트랩하거나 파일 또는 URL 접근 방식을 통해 다시 부트스트랩할 수 있습니다. 부트스트랩 및 특정 구성 옵션에 대한 자세한 내용은 107 페이지의 『오브젝트 그리드 클라이언트 연결 API』를 참조하십시오.

오브젝트 그리드 클라이언트 부트스트랩

오브젝트 그리드 클러스터 서버 구성원 서비스를 사용하려면 오브젝트 그리드 클라이언트가 클러스터 내 오브젝트 그리드 서버 중 하나에서 부트스트랩되어야 합니다. 각 클라이언트는 클러스터의 활성 구성원에 "연결"할 수 있습니다. 관리자는 특정 서버가 해당 서비스를 수행하도록 구성할 수 있습니다. 대규모 클라이언트 전개의 경우 구성된 오브젝트 그리드 클러스터 서버의 유일한 목적은 클라이언트 부트스트랩 지원을 제공하는 것입니다. 이러한 접근 방식은 클라이언트 수가 많고 자주 연결을 끊었다가 다시 연결하는 경우 유용합니다. 클라이언트가 "연결되면" 분산 참조를 클러스터 구성에 정의한 오브젝트 그리드로 가져올 수 있습니다. 자세한 정보는 99 페이지의 『ObjectGridManager 인터페이스』를 참조하십시오.

클라이언트는 오브젝트 그리드 클러스터에서 표준 구성을 확보하므로 관리자는 클라이언트 커뮤니티의 XML을 관리할 필요가 없습니다. 오브젝트 그리드 클라이언트는 오브젝트 그리드 클러스터 서버가 클라이언트에 특정한 설정을 대체할 때처럼 원격 URL을 사용할 수 있습니다.

분산 오브젝트 그리드 환경의 오브젝트 그리드 클라이언트

오브젝트 그리드 클라이언트는 동시에 둘 이상의 오브젝트 그리드 클러스터에 연결할 수 있습니다. JVM(Java Virtual Machine) 내 단일 Java 응용프로그램은 동일한 원격 클러스터에 여러 번 연결할 수 있습니다. 또한 이 응용프로그램은 동시에 다양한 원격 클러스터에 접속할 수 있습니다. 이러한 기능으로 클라이언트에서, 하나 이상의 오브젝트 그리드 클러스터를 통해 내보낸 정보의 다양한 자원에 액세스할 수 있으므로 이 기능이 중요합니다.

동일한 오브젝트 그리드 클라이언트가 동일한 오브젝트 그리드 서버에 접속할 수 있는 첫 번째 경우는 클라이언트가 Application Server인 보안 환경의 경우 중요하며 Application Server에서 원격 오브젝트 그리드 클러스터로 연결될 때마다 서로 다른 보안 신임을 사용합니다. 또다른 예는 단일 목적을 위해 여러 개의 오브젝트 그리드 클러스터의 데이터를 상관시켜야 하는 오브젝트 그리드 클라이언트입니다.

다음 다이어그램은 기업 웹 기반 클라이언트 사용자가 웹 응용프로그램을 통해 기업의 각기 다른 세 개 부서에서 보고서를 생성하는 시나리오를 설명합니다. Servlet 엔진은 Application Server 오브젝트 그리드 클라이언트 기능을 사용하여 각 기업 부서에서 관

리하는 세 개의 서로 다른 오브젝트 그리드 클러스터에 접속합니다. 다수의 기업에서 데이터를 수집할 수 있으므로 오브젝트 그리드의 핵심 목표는 정보를 더 쉽게 사용할 수 있게 하는 것입니다. 정보가 구체화되면 관심과 보안 신임을 보유한 기타 사용자가 새로운 방식으로 정보를 확보하고 사용할 수 있습니다. 데이터는 읽기 전용 모드로 제공되거나 해당되는 경우, 읽기/쓰기 갱신 시나리오에 제공될 수 있습니다.

이 시나리오에서는 보안 방식으로 데이터를 확보할 수 있습니다. 이 시나리오에서 오브젝트 그리드 캐싱은 각 기업 부서 내에서 공통 프로그램 방식으로 유연하게 데이터를 공유할 수 있도록 할 뿐만 아니라 다수의 Java 개발자가 흔히 사용했던 매우 안전하고 단순 명료한 프로그래밍 모델을 통해 얻어진 정보를 부서 데이터 전반에 걸쳐 액세스할 수 있도록 합니다.

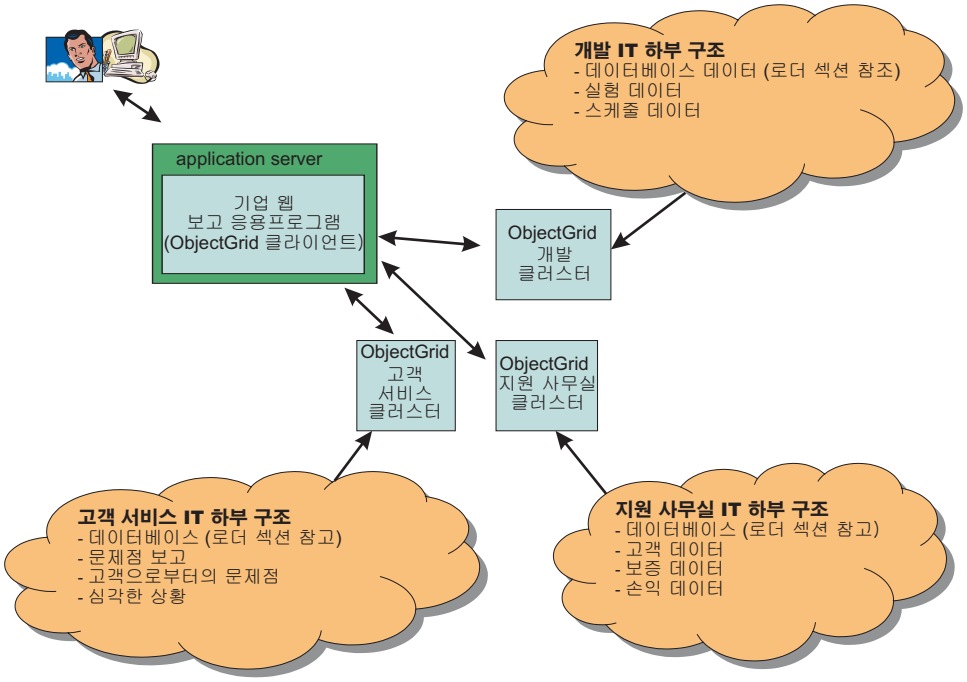


그림 4. 웹 기반 클라이언트 사용자는 기업의 각기 다른 세 개 부서에서 보고서를 생성합니다.

오브젝트 그리드 클러스터 개념

분산 오브젝트 그리드 항목에는 하나 이상의 오브젝트 그리드 클러스터와 상호 작용할 수 있는 클라이언트의 개념이 포함됩니다. 오브젝트 그리드 클러스터는 일 대 다 오브젝트 그리드 서버로 구성됩니다.

오브젝트 그리드 클라이언트

오브젝트 그리드는 두 가지 방법으로 생각할 수 있습니다. 그 중 하나는 클라이언트를 오브젝트 그리드 API를 사용하여 오브젝트 그리드 클러스터에 연결하고 해당 클러스터

에 대한 Java 맵 작업을 수행하는 JVM(Java Virtual Machine)으로 생각할 수 있습니다. 또 다른 하나는 클라이언트에 대해 더욱 공식적인 방법으로 생각하는 것인데, 동일한 JVM에 있는 다중 클라이언트의 개념을 고려하는 것입니다. 제공된 오브젝트 그리드 기능을 완전히 사용할 경우 동일한 JVM에 있는 다중 클라이언트를 사용할 수 있습니다.

프로그래머가 JVM에서 오브젝트 그리드 클라이언트 연결 조작을 실행할 때마다 클러스터 컨텍스트가 리턴됩니다. 실제로 이 컨텍스트는 하나의 클라이언트 인스턴스입니다. 따라서 비동기 스레드는 컨텍스트에 따라 캐싱의 다양한 요소를 처리합니다. 각 컨텍스트에 대해 ObjectGridManager를 사용하여 특정 원격 오브젝트 그리드 클러스터 인스턴스에서 호스트되는 오브젝트 그리드를 확보할 수 있습니다. 따라서 일반적으로 동일한 JVM에서 세 개의 원격 클러스터에 연결하면 동일한 JVM 내에 있는 세 개의 클라이언트 솔루션을 구현합니다.

이 시나리오의 중요한 고려사항은 다음과 같습니다. 단일 트랜잭션 세션에서는 동일한 클러스터 내에 있는 맵 세트를 확장할 수 없습니다. 사용자는 동일하거나 다른 오브젝트 그리드 클러스터에 접속된 다양한 클라이언트 상에서 단일 트랜잭션을 보유할 수 없습니다. 그러나, 정보 저장소를 통합하려는 사용자의 경우, 사용자는 트랜잭션을 사용하여 각 원격 오브젝트 그리드 클러스터에서 정보를 가져와서 통합 보고서를 인쇄하거나 정보를 결합하고 오브젝트 그리드 트랜잭션 데이터를 다른 오브젝트 그리드 클러스터로 보내거나 고객 특정 형식으로 개별 오브젝트 그리드 클러스터를 간단히 갱신할 수 있습니다. 이것은 일반적으로 별도의 트랜잭션 관리자가 제공하는 두 개의 단계 트랜잭션 지원과 반대되는 개념으로 주로 오브젝트 그리드가 단일 단계 트랜잭션 지원을 제공하기 때문입니다. 이 주제에 대한 자세한 정보는 40 페이지의 『오브젝트 그리드 트랜잭션 경계 설정』을 참조하십시오.

복제

동일한 오브젝트 그리드 클러스터 내의 오브젝트 그리드 서버 사이에서는 복제할 수 있습니다. 복제를 통해 사용자가 필요로 하는 특정 정보를 보유하는 1차 오브젝트 그리드 서버가 실패하거나 유지보수를 위해 시스템이 종료되었을 때 더욱 빠르게 장애를 복구할 수 있습니다. 다음 다이어그램은 서로 다른 두 개의 오브젝트 그리드 복제 그룹 구성원의 Red 오브젝트 그리드와 Purple 오브젝트 그리드입니다. 오브젝트 그리드에서 오브젝트 그리드의 서브세트인 각 MapSet는 한 개의 단위로 복제될 수 있습니다. PartitionSet는 다음 섹션에서 설명하겠지만 이 규칙에 해당되지 않습니다. 설명된 단일 서버 구성은 복제를 설명하기 위해 다음 다이어그램에서 수정되었습니다.

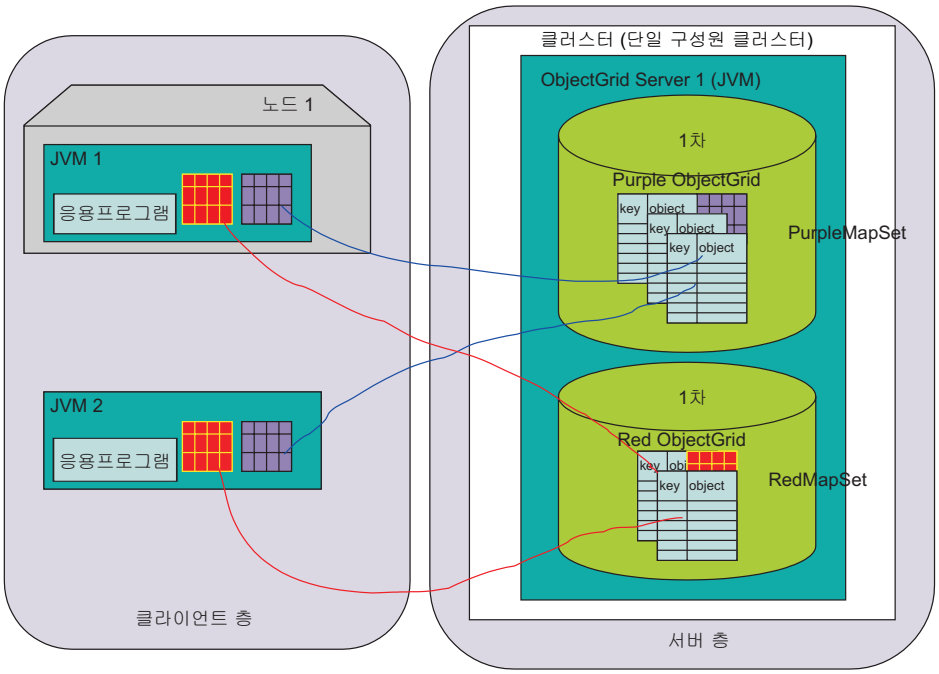


그림 5. 두 개의 MapSet를 포함한 분산 오브젝트 그리드 단일 서버 토폴로지

다이어그램은 Application Server를 클라이언트 응용프로그램과 독립형 Java 응용프로그램으로 설명합니다. 두 클라이언트 모두 단일 서버 오브젝트 그리드 클러스터에 있는 두 개의 오브젝트 그리드 인스턴스인 Red 및 Purple 인스턴스에 대한 액세스를 필요로 합니다. 이러한 각 인스턴스는 실제로 복제 그룹 구성원에 포함되어 있습니다. 복제 그룹 구성원은 핵심 개념이며 오브젝트 그리드 트랜잭션 경계 설정의 경계입니다. 트랜잭션은 단일 복제 그룹 구성원의 변경사항만 확약할 수 있습니다.

오브젝트 그리드 클러스터에서 Java 클라이언트는 오브젝트 그리드 트랜잭션(세션)을 시작하고 단일 복제 그룹 구성원 내 데이터를 갱신할 수 있습니다. 각 복제 그룹 구성원은 하나의 단위로 복제될 수 있으며 요구사항에 따라 둘 다 동기 또는 비동기식으로 복제되거나 모두 복제되지 않을 수 있습니다. 각 오브젝트 그리드 클라이언트 요청은 오브젝트 그리드 클러스터 서버 내 특정 복제 그룹 구성원으로 라우트됩니다. 요청을 수신 중인 복제 그룹 구성원 내 오브젝트 그리드는 요청을 처리하고 결과를 클라이언트로 리턴합니다. *동기* 복제의 경우 클라이언트로 리턴되기 전에 각 요청은 복제본 또는 다음 다이어그램에 있는 오브젝트 그리드 server2로 전송되어 복제본 복제 그룹 구성원이 올바르게 갱신을 적용했음을 확인한 다음 결과를 클라이언트로 리턴합니다. *비동기* 모드에서는 오브젝트 그리드 클라이언트가 변경사항을 적용할 수 있고 오브젝트 그리드 서버의 1차 복제 그룹 구성원이 결과를 클라이언트로 리턴하며 복제본이 변경사항이 수신되어 올바르게 적용되었음을 확인할 때까지 대기하지 않습니다. 비동기 모드에서는 트랜잭션이 1차 복제 그룹 구성원에서 성공적으로 확약된 후 갱신사항을 원격 서버의 복제본 복제 그룹 구성원으로 전송됩니다.

다음 다이어그램은 부트스트랩 예제의 다른 버전입니다. 이 경우 세 개의 서버는 각각 사용자가 상호 작용할 것으로 예상하는 두 개의 오브젝트 그리드 인스턴스 복제에서 고유한 역할을 수행합니다. 오브젝트 그리드 클러스터는 각각 두 개의 복제 그룹 구성원을 호스트하는 세 개의 서버로 구성되어 있습니다. server1 서버는 두 개의 1차를 호스트하고 server2 서버는 두 개의 복제본을 호스트하며 server3 서버는 두 개의 대기를 호스트합니다.

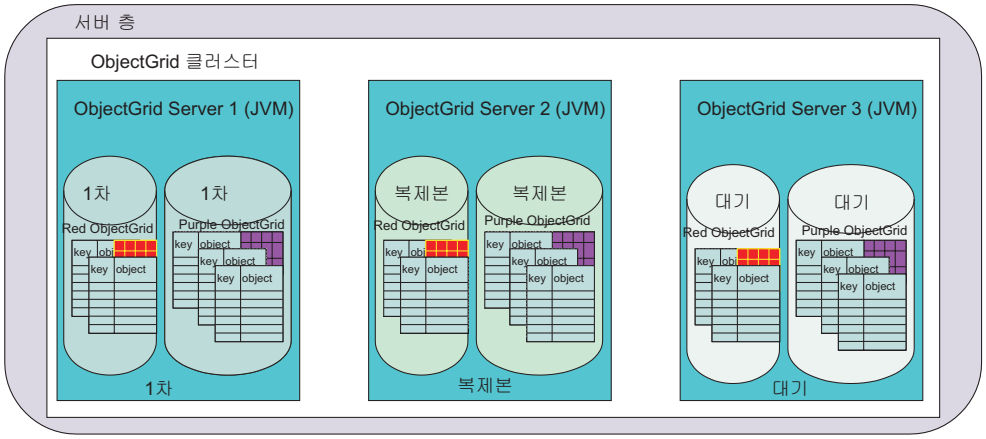


그림 6. 기본 샘플 구성 복제

31 페이지의 『고가용성 개요』에서 이 개념을 설명하지만 이해할 핵심 개념은 단일 서버에서 이전 다이어그램에 설명된 세 개의 서버 복제 솔루션으로 이동하는 데 필요한 구성의 차이입니다.

다중 서버 복제 구성 개요

다음 구성은 Red 및 Purple 오브젝트 그리드 인스턴스의 기본 오브젝트 그리드 구성입니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="Red">
<backingMap name="FirstRedMap" readOnly="false" />
<backingMap name="SecondRedMap" readOnly="false" />
</objectGrid>
<objectGrid name="Purple">
<backingMap name="FirstPurpleMap" readOnly="false" />
<backingMap name="SecondPurpleMap" readOnly="false" />
<backingMap name="ThirdPurpleMap" readOnly="false" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

이 구성을 분산 오브젝트 그리드 클러스터로 변환하려면 추가 구성 파일, 클러스터 XML 파일이 필요합니다. 단일 서버의 Red 및 Purple 오브젝트 인스턴스의 원래 구성을 변환하려면 다음 예에 표시된 추가사항만 필요합니다. 특히, 두 개의 서버 참조만 추가되었습니다. 복제 그룹은 다음 샘플에 설명된 대로 ColorMapsReplicationGroup 복제 그룹과 상호 참조된 초기 구성 파일에서 이미 존재했습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster ../objectGridCluster.xsd
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!- single server –>
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12504" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12506" />
<serverDefinition name="server3" host="localhost" clientAccessPort="12507"
peerAccessPort="12508" />
</cluster>
<objectgridBinding ref="Red">
<mapSet name="RedMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstRedMap" />
<map ref="SecondRedMap" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectgridBinding>
<partitionSet name="ColorMapsPartitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup" maxReplicas="1"
minReplicas="1">
<replicationGroupMember serverRef="server1" priority="1" />
<replicationGroupMember serverRef="server2" priority="2" /><!–New–>
<replicationGroupMember serverRef="server3" priority="3" /><!–New->
</replicationGroup>
</clusterConfig>
```

이전 예에서 아래에 설명된 두 MapSet는 모두 복제 그룹에 포함될 서버를 정의하는 ColorMapsReplicationGroup ReplicationGroup을 참조합니다. 구성 파일은 각 MapSet에 동일한 서버를 다른 순서로 포함하거나 다른 서버를 포함하여 고객 요구사항을 충족시키도록 하는 다른 ReplicationGroup을 포함하도록 확장되었을 수 있습니다. 오브젝트 그리드 클러스터 구성에서 스탠자를 재활용할 수 있습니다. 기본적으로 MapSet 복제 속성은 설정되지 않고 복제 그룹이 둘 이상의 서버를 포함하므로 복제는 사용 가능하고 모드는 비동기가 됩니다.

고가용성 개요

복제는 오브젝트 그리드 클러스터 내의 고가용성을 사용 가능하게 합니다.

복제 및 고가용성을 이해하려면 오브젝트 그리드 복제 그룹 구성원 유형을 이해해야 합니다. 오브젝트 그리드가 지원하는 복제 그룹 구성원 유형은 *1차*, *복제본* 및 *대기*입니다. 이러한 각 유형은 고가용성 구성에서 특정 역할을 보유합니다.

오브젝트 그리드 복제 그룹 구성원 유형

1차 복제 그룹 구성원

1차 복제 그룹 구성원은 사용 중인 클라이언트의 최신 데이터 보기를 보유합니다. 데이터가 갱신될 때 데이터는 복제본으로 전달됩니다. 1차는 오브젝트 그리드 로더 인터페이스를 통해 연결 데이터베이스와 통신하고 복제 구성에 따라 동기식 또는 비동기식으로 확약을 전달하거나 모두 전달하지 않는 인스턴스입니다.

복제본 복제 그룹 구성원

복제본 복제 그룹 구성원은 1차에서 전달된 데이터 버전을 보유합니다. 다양한 방식으로 변경사항을 전송하도록 1차를 구성할 수 있습니다. 복제 그룹에는 최소 두 개의 서버가 나열되어 1차와 복제본을 포함해야 하는데, 그렇지 않으면 복제는 사용 가능하지 않습니다.

대기 복제 그룹 구성원

복제본이 수행하는 것과 같이 1차에 변경사항이 작성될 때 대기 복제 그룹 구성원은 갱신사항을 수신하지 않습니다. 이것은 간단히 구성되고 1차 또는 복제본이 실패할 경우 갱신사항을 수신하도록 준비합니다. 1차가 실패할 경우 복제본이 새로운 1차가 되고 대기가 복제본으로 변환되어야 합니다.

고가용성 시나리오

일반적으로 복제는 오브젝트 그리드 클러스터 내의 고가용성을 사용 가능하게 합니다. 다음에서 두 개의 1차 장애 시나리오 및 복구를 설명합니다. 1차 복제 그룹 구성원이 복제되고 장애가 발생하면 복제본 중 하나가 선택되어 새로운 1차가 됩니다. 이 시나리오에는 하나의 복제본이 있습니다.

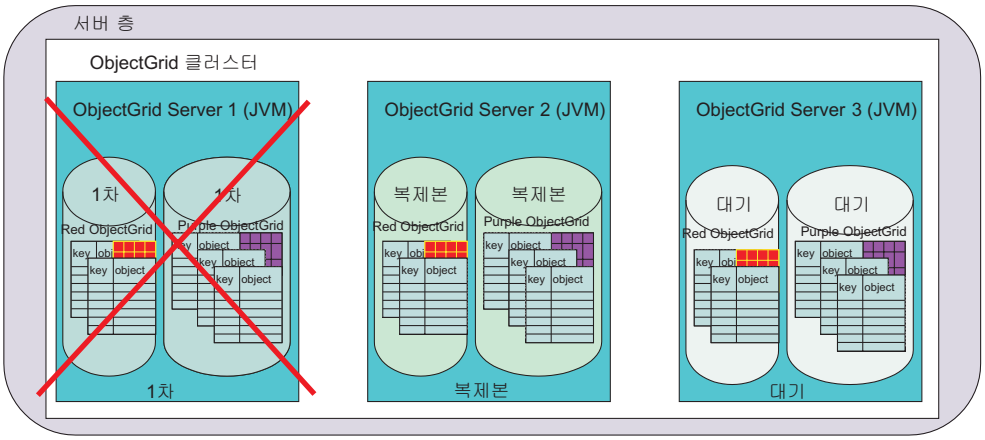


그림 7. 오브젝트 그리드 고가용성 시나리오

장애가 발견되면 1차는 사용 불가능하게 됩니다. 복제본이 1차가 됩니다. 대기가 있으면 다음 다이어그램의 복구 예와 유사하게 대기가 복제본이 됩니다.

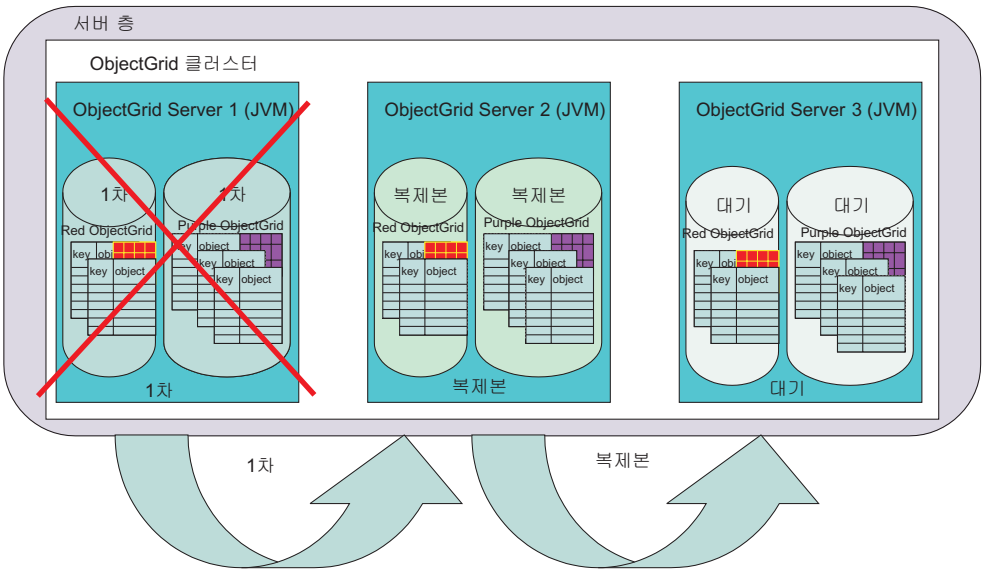


그림 8. 오브젝트 그리드 실패복구

오브젝트 그리드는 다음에 관련되는 서버에 연결하는 중에 이러한 조정을 인식하게 됩니다. 실패한 서버에 접속하는 클라이언트는 런타임 구성을 사용하고 클러스터의 다른 서버를 동적으로 시도할 수 있습니다. 클라이언트는 구성에 있는 다음 서버에 접속합니다. 서버가 켜졌는데 조작되지 않으면 클라이언트는 제한시간 동안 대기합니다. 클라이언트에서 복제된 복제 그룹의 구성원이 장애로부터 복구 중이라고 가정합니다. 일정 시간 후 클라이언트는 재시도하고 이제는 두 개의 구성원이 있는 복제 그룹이 조작되면 새 라우팅 테이블이 클라이언트에 제공됩니다. 라우팅 테이블은 현재 1차의 위치와 복제본 위치, 그리고 현재 대기 상태인 이 그룹의 복제 그룹 구성원을 설명합니다.

오브젝트 그리드 클러스터링 구성 세트

오브젝트 그리드 클러스터링을 구성할 때 오브젝트 그리드를 MapSet로 구분할 수 있습니다. 오브젝트 그리드는 많은 맵을 포함할 수 있으므로 이러한 구분은 중요합니다. MapSet는 PartitionSet를 통해 파티션될 수 있고 ReplicationGroup을 통해 복제될 수 있습니다. 이러한 각 구성 옵션은 오브젝트 그리드 서버 시작 중에 작성되는 복제 그룹 구성원 수에 영향을 줍니다. 각 세트 유형에 대한 간단한 요약은 각 유형의 역할을 설명하는 데 도움을 줍니다.

오브젝트 그리드 MapSet

각 오브젝트 그리드 맵은 서로 다른 사용법과 가용성 요구사항을 가질 수 있지만 일반 응용프로그램 사용법으로 상관됩니다. 예를 들어, 한 맵은 사전 로드 완료 후 변경사항이 없는 읽기 전용일 수 있고 다른 한 맵은 확장성 목적을 위해 읽기/쓰기가 가능하며 파티셔닝할 수 있습니다. 이 경우 각 맵은 고유 MapSet 내에 포함됩니다. 이전 예에서는 PurpleMapSet와 RedMapSet가 지정된 각 오브젝트 그리드에 해당하는 맵을 모두 보유했는데, 이것은 가장 단순한 옵션입니다.

MapSet는 오브젝트 그리드 서버에서 복제할 수 있는 단위이며 파티션되지 않은 복제 그룹 구성원과 상관됩니다. PartitionSet를 통해 MapSet에 연관된 각 복제 그룹 서버는 해당되는 복제 그룹 구성원을 호스트하여 요청된 구성을 지원합니다. 복제 그룹 구성원은 오브젝트 그리드 클러스터 내에 있는 고유한 엔드포인트이며 지정된 MapSet가 구성에서 명시하는 모든 맵을 호스트합니다.

예를 들어, 이전 다이어그램에서 server1 서버는 1차를 보유하고 server2 서버는 복제본을 보유하며 server3 서버는 복제 복구 시나리오에 따라 복제본 또는 1차가 될 수 있는 대기 단위를 보유합니다. MapSet는 세 개의 서버를 설명하는 PartitionSet에 상관됩니다. 따라서, 두 MapSet는 각각 맵 수가 다르더라도 PartitionSet 및 ReplicationGroup 스탠자가 동일하기 때문에 동일한 서버로 맵핑됩니다.

오브젝트 그리드 PartitionSet

일반적으로 Mapset 및 에뮬레이트된 복제 그룹 서버는 오브젝트 그리드 클러스터의 특정 MapSet를 지원하는 서버 수를 결정합니다. 복제 그룹 구성원은 각 호스트 복제 그룹 서버 내에서 작성됩니다. 그러나 파티션이 MapSet의 복제 그룹 구성원 수에 영향을 줄 수 있습니다. 파티션은 MapSet와 ReplicationGroup 간 PartitionSet 관계를 통해 구성 파일에서 관리됩니다.

하나의 JVM(Java Virtual Machine)이 단일 1차 복제 그룹 구성원에 전체 MapSet를 보유하지 않도록 PartitionSet가 MapSet를 여러 개 부분으로 나눕니다. 예를 들어, 1,000,000개의 키 데이터베이스라고 가정해 보십시오. 각 키가 나타내는 각 오브젝트가 클 경우 단일 32비트 JVM이 단일 1차, 복제본 또는 대기 복제 그룹 구성원에 메모리의 맵을 보유할 수 없는 경우도 있습니다. 그러나 대형 데이터 세트가 자주 필요합니

다. 데이터를 인위적으로 파티션해야 하는 경우를 방지하기 위해, 예를 들어, Purple 오브젝트 그리드 첫 번째 맵 인스턴스를 PurpleFirstMapMap1, PurpleFirstMap2, PurpleFirstMapN 맵으로 직접 파티션하지 않고 각각을 서로 다른 Mapset에 배치하여 오브젝트 그리드가 이 작업 대부분을 수행할 수 있습니다.

이 문서의 뒷부분에 PartitionKey의 개념이 정의되어 있습니다. 이것은 오브젝트 그리드가 삽입 중 특정 입력에 어떤 키 해시코드가 있는지 판별하기 위해 호출할 수 있는 API와 사실상 같아집니다. MapSet에 두 개의 파티션이 있으면 해당 MapSet에 대해 두 개의 복제 그룹 구성원이 작성됩니다. 종종 데이터가 방대하여 이러한 복제 그룹 구성원은 서로 다른 서버에 있습니다. 개발자의 경우 이러한 구성원은 초기 프로토타입 중에 동일한 서버에 있을 수 있습니다. 복제 그룹 구성원마다 해당되는 파티션된 복제 그룹 구성원 간에 동일한 값에 해시하는 키를 보유합니다. 간단한 예로 MapSet가 0, 1 및 2의 세 가지 방법으로 파티션 지정되었다고 가정하십시오. 세 개의 1차 복제 그룹 구성원이 설정되면 그 중 하나가 지정된 값 modulus 1차 복제 그룹 구성원 수에 해시하는 모든 키를 보유합니다. 예를 들어, 키의 해시 값이 7인 경우 7 modulus 3이 1이므로 파티션 지수가 1인 1차 복제 그룹 구성원이 인스턴스를 보유합니다.

PartitionSet 예

다음 다이어그램은 Purple 맵을 두 개의 파티션으로 분리하는 경우를 설명합니다. 맵의 각 키는 정수에 해시되고 해당 복제 그룹 구성원에 삽입할 때 설정된 특정 파티션에 지정됩니다. 각 파티션은 서로 다른 JVM에 존재할 수 있으므로 서로 다른 복제 그룹 구성원에 있으며 일반적으로 오브젝트 그리드를 통해 프로그래머는 PurpleFirstMap을 파티션되지 않은 단일 논리 맵 인스턴스로 처리합니다. 오브젝트 그리드 클라이언트 및 서버 지원에서 복제 그룹 구성원 간의 올바른 요청 라우팅을 관리합니다.

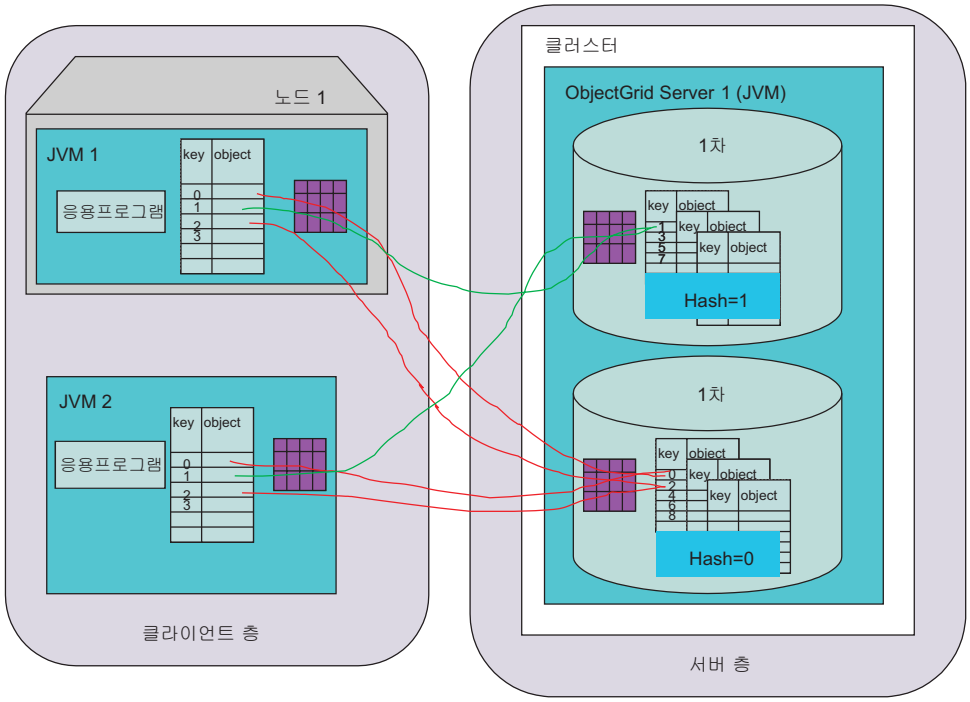


그림 9. 분산 오브젝트 그리드 토폴로지: 파티션을 포함한 단일 서버

이 구성을 사용 가능하게 하는 파티션 설정 구성 변경사항은 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!- single server –>
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsPartitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
<partition name="partition2" replicationGroupRef="ColorMapsReplicationGroup" />
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup" maxReplicas="1"
minReplicas="1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

위의 예제 구성에서 복제되고 파티션된 Purple 오브젝트 그리드의 설정 방법을 알 수 있습니다. 다음과 같은 경우에서 세 개의 서버가 존재하고 1차 복제 그룹 구성원 각각이 세 개의 서버 세트에 동일한 방법으로 맵핑됩니다. 해당 그룹 구성원들은 다른 ReplicationGroup 스탠자가 사용된 경우 다른 방법으로 쉽게 맵핑될 수 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!- single server –>
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12504" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12506" />
<serverDefinition name="server3" host="localhost" clientAccessPort="12507"
peerAccessPort="12508" />
</cluster>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsPartitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
<partition name="partition2" replicationGroupRef="ColorMapsReplicationGroup" />
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup" maxReplicas="1"
minReplicas="1">
<replicationGroupMember serverRef="server1" priority="1" />
<replicationGroupMember serverRef="server2" priority="2" />
<replicationGroupMember serverRef="server3" priority="3" />
</replicationGroup>
</clusterConfig>
```

앞의 예제를 다시 살펴보면 구성 파일을 일부 변경하는 것만으로 새로운 레벨의 성능을 끌어낼 수 있으므로 결과를 위해서는 응용프로그램을 수정해야 합니다. 다음 다이어그램은 이전 파티션 구성을 지원하기 위해 오브젝트 그리드 클러스터 보기 및 복제 그룹 구성원의 배치 방법을 설명합니다.

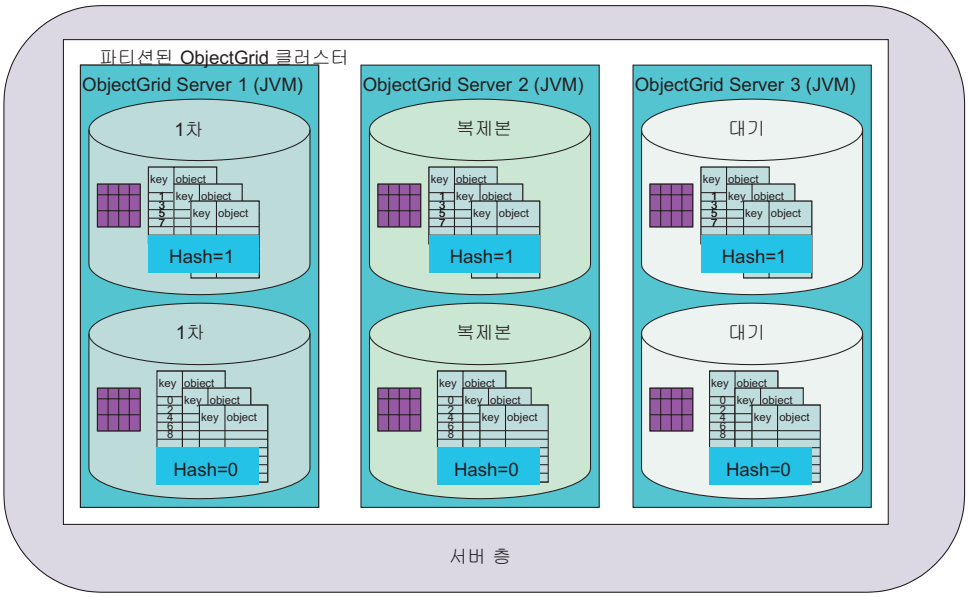


그림 10. 분산 오브젝트 그리드 토폴로지: 파티션을 포함한 다중 서버

다음은 PartitionSet 설명을 완료하기 위한 이전 예제의 다른 변형입니다. 이 경우 두 번째 ReplicationGroup이 작성되었습니다. 각 PartitionSet는 이제 개별 서버의 고유한 복제 그룹에 포함됩니다.

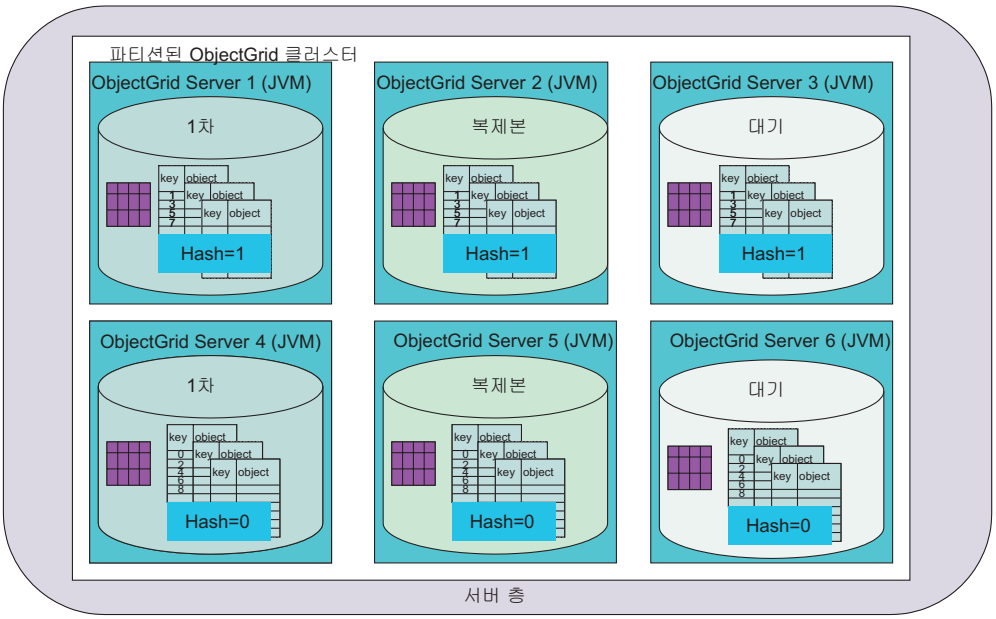


그림 11. 분산 오브젝트 그리드 토폴로지: 파티션을 포함한 다중 서버

이 토폴로지의 구성은 이전 예제와 매우 유사합니다. 변경사항에는 세 개 이상의 서버 인스턴스와 두 번째 PartitionSet가 참조하는 새 ReplicationGroup이 포함됩니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!- single server –>
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12504" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12506" />
<serverDefinition name="server3" host="localhost" clientAccessPort="12507"
peerAccessPort="12508" />
<serverDefinition name="server4" host="localhost" clientAccessPort="12513"
peerAccessPort="12514" /><!-*New*->
<serverDefinition name="server5" host="localhost" clientAccessPort="12514"
peerAccessPort="12516" /><!-*New*->
<serverDefinition name="server6" host="localhost" clientAccessPort="12517"
peerAccessPort="12518" /><!-*New*->
</cluster>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsPartitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
<partition name="partition2" replicationGroupRef="ColorMapsReplicationGroup" />
<!-NEW->
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup" maxReplicas="1"
minReplicas="1">
<replicationGroupMember serverRef="server1" priority="1" />
<replicationGroupMember serverRef="server2" priority="2" />
<replicationGroupMember serverRef="server3" priority="3" />
</replicationGroup>
<replicationGroup name="ColorMapsReplicationGroupNew" maxReplicas="1"
minReplicas="1"> <!-*NEW*->
<replicationGroupMember serverRef="server4" priority="1" />
<replicationGroupMember serverRef="server5" priority="2" />
<replicationGroupMember serverRef="server6" priority="3" />
</replicationGroup>
</clusterConfig>
```

이렇게 구성하면 복제 그룹 구성원 수는 동일하지만 두 번째 복제 그룹이 고유 XML 스탠자를 통해 특별히 구성되고 동시에 각 인스턴스를 이전 예제에서와 같이 배치하지 않고 다양한 서버 인스턴스에 제공합니다.

구성의 장점은 이러한 경우에, 각 파티션이 1.5기가바이트 이상의 데이터를 보유할 수 있고, 두 개의 복제 그룹 구성원이 고유의 JVM 인스턴스에 있으므로 총 3기가바이트 이상을 보유할 수 있다는 것입니다. 따라서 32비트 JVM 2기가바이트를 주소 지정 가능한 메모리 공간으로 최대한 활용할 수 있게 됩니다.

다중 오브젝트 그리드 클러스터에 접속하는 오브젝트 그리드 클라이언트

오브젝트 그리드는 JVM(Java Virtual Machine)에서 파티션된 지원을 확장할 뿐만 아니라 정보를 얻기 위해 표준 Java 맵 인터페이스가 영향을 미칠 수 있는 범위를 확장하도록 디자인되었습니다. 단일 클라이언트로 다수의 오브젝트 그리드 클러스터에 접속할 수 있습니다.

다음 시나리오에서 서버 층은 두 개의 오브젝트 그리드 클러스터를 포함합니다. Java 응용프로그램 클라이언트 중 하나와 Application Server 중 하나가 다중 클러스터에 접속해야 합니다. 이것은 강력한 기능으로 대규모 확장성을 허용합니다.

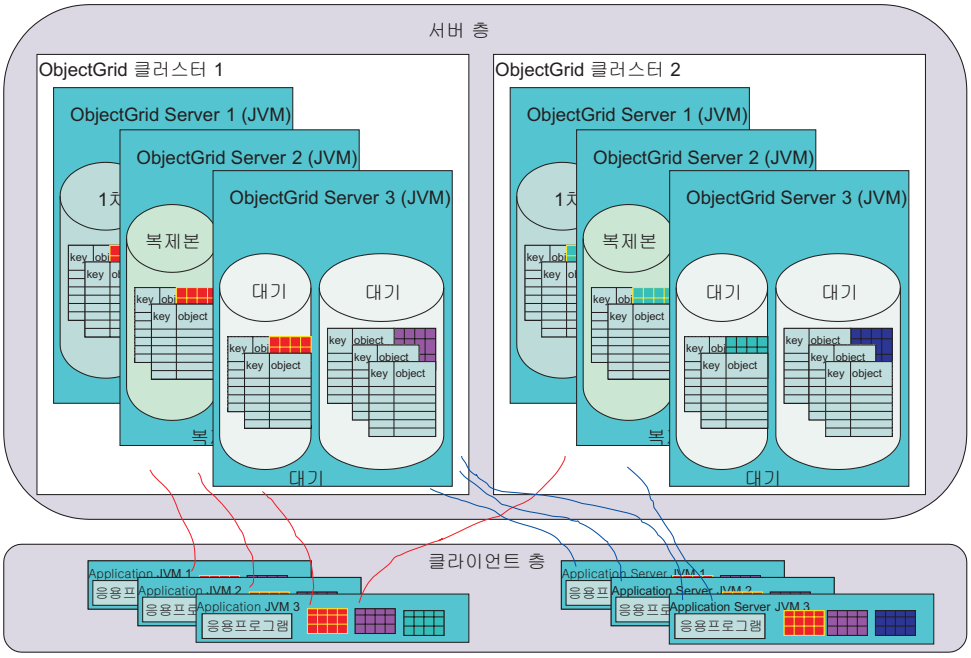


그림 12. 다중 오브젝트 그리드 클러스터와 상호 작용하는 오브젝트 그리드 클러스터

오브젝트 그리드 클러스터는 많은 MapSet 및 PartitionSet 구성을 각각 포함하는 다수의 오브젝트 그리드를 지원할 수 있습니다. 각 오브젝트 그리드 클러스터는 하나 이상(또는 더 많이)의 JVM으로 작성될 수 있습니다. 대형 엔터프라이즈의 경우 오브젝트 그리드 클라이어트가 하나만이 아니라 동시에 여러 오브젝트 그리드 클러스터에 접속할 수 있는 매우 중요한 기능입니다.

그러나 한 가지 주의할 것은 오브젝트 그리드 클라이언트는 단일 트랜잭션 내에서 다중 클러스터의 데이터를 참조할 수 없습니다. 오브젝트 그리드 클라이언트 응용프로그램은 하나 이상의 트랜잭션을 실행하여 JVM(Java Virtual Machine) 인스턴스의 데이터를 캐시하고 Java 오브젝트로 검색된 정보를 상관시켜야 합니다. 정보에 기반한 갱신 또한 각 클러스터의 단일 트랜잭션에 포함되어야 합니다. 이 문제에 대한 자세한 내용은 40 페이지의 『오브젝트 그리드 트랜잭션 경계 설정』을 참조하십시오.

오브젝트 그리드 클라이언트 주변 캐싱 지원

오브젝트 그리드 클라이언트는 실제로 캐싱 층입니다. 이전에 원격 서버에서 확보한 데이터가 무효가 아닌지 여부를 알고 있는 경우 로컬 캐싱 기능을 활용하도록 응용프로그램을 설계할 수 있습니다. 예를 들어, 데이터가 오브젝트 그리드 클러스터에서 갱신되었지만 이 클라이언트에서는 갱신되지 않은 경우(이것은 새로운 get(...) 요청을 가짐) 해당 클라이언트는 로컬 캐시가 오브젝트 그리드 클러스터와 일관되도록(모든 응용프로그램이 이것을 요구하지는 않음) 갱신해야 합니다.

가져오기 조작이 수행된 후 동일한 키 및 오브젝트 쌍의 작업을 가져오기 위한 후속 요청이 이루어 지면 오브젝트 그리드 클라이언트에서 데이터가 이미 검색되었음을 발견하고 네트워크 상에서 오브젝트 그리드 클러스터로 이동하여 데이터에 액세스하지 않고 ″JVM(Java Virtual Machine)에 있는″ 캐시 버전을 사용합니다. 네트워크를 통해 데이터가 한 번 검색된 후 로컬 입력이 수동으로 또는 표준 구성 축출기를 통해 축출될 때까지 데이터는 로컬 캐시에서 계속 제공될 수 있습니다.

예를 들어, 서버의 데이터가 6시간마다 한 번씩 새로 고쳐지는 경우 로컬 캐시 또는 주변 캐시 클라이언트 갱신이 발생하는 시기를 제어할 수 있습니다. 사용자는 주변 캐시 항목을 무효화시킨 다음 가져오기 요청을 발행합니다. 가져오기 요청은 서버에 접속하여 모두 올바르게 진행될 경우 정보를 확보합니다. 오브젝트가 이미지 파일이라고 가정해 보십시오. 갱신 창이 나타난 뒤 처음으로 이미지가 다운로드되고 후속 요청이 있을 때마다 서버가 이미지를 가져오도록 원격 프로시저 호출이 이루어지는 것은 아닙니다.

클라이언트에서 오브젝트 그리드 클러스터 데이터를 잠궈 요청된 잠금 계획을 강제 실행해야 하므로 주변 캐싱 지원이 변경이 예상됨 모드에서는 적용되지 않습니다. 오브젝트 그리드 클러스터의 정보 보기를 수정하지 않고 제거해야 하는 주변 캐시 항목 지우기 또는 무효화에 대한 자세한 내용은 beginNoWriteThrough() 메소드를 검토하십시오.

오브젝트 그리드 트랜잭션 경계 설정

프로그래머로서 명심해야 할 핵심 개념은 트랜잭션 경계 설정의 개념입니다. 오브젝트 그리드는 오브젝트 그리드 클러스터의 복제 그룹 구성원에서 트랜잭션 확약 처리에 대한 2단계 확약 프로토콜을 지원하지 않습니다. 단일 오브젝트 그리드 클러스터 기반 세션에서는 단일 1차 복제 그룹 구성원에 읽기/쓰기 갱신을 적용해야 합니다. 다중 복제 그룹 구성원이 관련되었다면 트랜잭션 확약 처리 중에 갱신사항을 작게 만들 수 없습니다. 이것은 단일 오브젝트 그리드의 모든 맵에 대해 확약을 허용하는 로컬 오브젝트 그리드 JAR 프로그래밍 모델과 약간 다릅니다.

특별히 눈여겨 볼 경우는 둘 이상의 파티션이 정의된 PartitionSet입니다. 이러한 시나리오에서 key 1은 server 1에 있고 key 2는 server 2에 있는 형태일 수 있습니다. 한 트랜잭션에 key 1 및 key 2에 대한 갱신사항이 있으면 오브젝트 그리드 세션은 두 개의 복제 그룹 구성원에 대해 확약할 수 없으므로 갱신이 실패합니다. 앞서 명시한 대로

둘 이상의 파티션을 지원하는 PartitionSet는 응용프로그램에서 신중하게 사용되어야 합니다. 트랜잭션 순서에서 둘 이상의 트랜잭션으로부터 데이터를 갱신하지 않도록 주의하십시오.

데이터베이스에 대한 오브젝트 그리드 관계

이 개요에서는 기타 Java 클라이언트가 아닌 다른 장소에서 초기화된 데이터를 확보하는 오브젝트 그리드 서버를 특별히 설명하지 않았습니다. 오브젝트 그리드가 시작될 때 MapSet의 맵마다 로더가 초기화되었습니다. 이 로더를 사용하여 Java 맵의 get 또는 put 형식의 사용자 요청을 검색하거나 해당되는 데이터베이스에 기록할 수 있습니다. 데이터베이스 작업은 Java 맵 사용자가 볼 수 없고 해당 응용프로그램에서는 특수 코딩이 필요하지 않습니다. 그러나, 일반 사용자 프로그래머가 로더 기능을 사용하려면 이 로더 기능을 프로그래머가 개발하고 오브젝트 그리드에서 사용하도록 구성되어야 합니다. 자세한 내용은 217 페이지의 『로더』를 참조하십시오.

이외에도 오브젝트 그리드 클러스터가 초기화되고 나면 데이터베이스에서 사전 로드하기 위해 사전 로드 지원이 존재할 뿐만 아니라 지정된 MapSet의 특정 복제 그룹 구성원 파티션에 대해 올바른 데이터를 읽을 수 있도록 하는 파티션된 사전 로드가 존재합니다. 사용자는 없음, 변경이 예상되지 않음 및 변경이 예상됨을 포함하여 다양한 잠금 계획으로 읽기/쓰기 정보에 액세스하고 갱신할 수 있습니다. 읽기 전용 데이터 액세스가 지원되며 여러 가지 최적화가 제공될 수 있을 때 가장 빠른 모델이 됩니다.

제 4 장 오브젝트 그리드 학습서: 응용프로그램 프로그래밍 모델

이 타스크를 참고하여 오브젝트 그리드 응용프로그램 프로그래밍 모델에 대해 학습하십시오.

오브젝트 그리드 응용프로그램을 실행하도록 환경을 준비하십시오. Java 아카이브(JAR) 파일 위치, Java 요구사항 및 환경을 올바르게 설정했는지 확인하기 위해 단순 파일을 실행하는 방법에 대해 학습하려면 1 페이지의 제 1 장 『샘플 응용프로그램을 실행하여 오브젝트 그리드 시작하기』를 참조하십시오.

이 타스크에서 사용할 프로그래밍 환경을 결정하십시오. Eclipse와 같은 IDE(Integrated Development Environment)를 사용할 수 있지만 명령행 Java 환경도 가능합니다. 오브젝트 그리드에 보다 익숙해지면 오브젝트 그리드를 엔터프라이즈 Bean 및 Servlet에 통합하십시오. 학습서의 예제에서 특정 Java 환경이 필요하지는 않으므로 사용자에게 익숙한 환경을 사용할 수 있습니다.

대부분의 기본 정의에 따르면 오브젝트 그리드는 오브젝트의 캐시 및 인메모리 저장소입니다. 오브젝트를 저장하고 해당 오브젝트에 액세스할 때 java.util.Map 맵을 사용하는 방법은 오브젝트 그리드를 사용하는 방법과 유사합니다. 동시에 오브젝트 그리드는 캐시 이상의 역할을 수행합니다. 이 타스크에서 다양한 기능 및 플러그인을 탐색하면 오브젝트 그리드가 확장성 및 유연성이 매우 뛰어나다는 점을 알 수 있습니다. 오브젝트 그리드를 단순 *Look-Aside* 캐시로 사용할 수도 있고, 자원 관리자가 지원하는 보다 정교한 캐시로 사용할 수도 있습니다.

이 학습서의 예제는 완전한 프로그램이 아닙니다. 가져오기, 예외 처리 및 일부 변수도 모든 예에서 완전히 선언되지는 않습니다. 샘플을 사용하여 자체 프로그램을 작성할 수 있습니다.

이 타스크를 참고하여 Java 프로그램에서 오브젝트 그리드를 사용하십시오.

1. 오브젝트 그리드 API 및 예외를 찾으십시오. 모든 공용 오브젝트 그리드 API 및 예외는 `com.ibm.websphere.objectgrid` 패키지에 들어 있습니다. 고급 시스템 또는 구성 주제는 `com.ibm.websphere.objectgrid.plugins` 패키지의 추가 API 및 예외를 참조하십시오. 플러그인 구현이 제공된 경우 `com.ibm.websphere.objectgrid.plugins.builtins` 패키지에서 해당 클래스를 찾으십시오. 오브젝트 그리드 보안 기능은 이름에 **security**가 포함된 패키지(예: `com.ibm.websphere.objectgrid.security`, `com.ibm.websphere.objectgrid.security.plugins` 및 `com.ibm.websphere.objectgrid.security.plugins.builtins`)에서 찾으십시오.

이 타스크는 com.ibm.websphere.objectgrid 패키지에 있는 API에 초점을 맞추고 있습니다. 오브젝트 그리드의 전체 JavaDoc은 <install_root>/web/xd/apidocs에서 찾을 수 있습니다.

```
com.ibm.websphere.objectgrid
com.ibm.websphere.objectgrid.plugins
com.ibm.websphere.objectgrid.plugins.builtins
com.ibm.websphere.objectgrid.security
com.ibm.websphere.objectgrid.security.plugins
com.ibm.websphere.objectgrid.security.plugins.builtins
```

2. 오브젝트 그리드 인스턴스를 확보하거나 작성하십시오. ObjectGridManagerFactory를 사용하여 ObjectGridManager 싱글톤 인스턴스를 확보하십시오. 그런 다음 명령문으로 오브젝트 그리드 인스턴스를 작성하십시오.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid objectGrid =
objectGridManager.createObjectGrid("someGrid");
```

ObjectGridManager 인터페이스에는 오브젝트 그리드 인스턴스를 작성, 검색 및 제거하는 여러 메소드가 있습니다. 상황에 맞는 변수를 선택하려면 99 페이지의 『ObjectGridManager 인터페이스』 주제를 참조하십시오. 또한 ObjectGridManager 인터페이스를 사용하여 추적 설정을 설정할 수 있습니다. WebSphere Extended Deployment 또는 WebSphere Application Server 내부에서 실행하는 경우 포함된 기능에서 추적을 관리하므로 이 메소드는 필요하지 않습니다. WebSphere Application Server 외부에서 실행하는 경우 이 메소드가 유용할 수 있습니다. 이 메소드에 대한 전체 정보는 107 페이지의 『오브젝트 그리드 추적』 주제를 참조하십시오.

3. 오브젝트 그리드를 초기화하십시오.
   a. create 메소드로 이름을 설정하지 않은 경우 오브젝트 그리드 이름을 설정하십시오.
   b. 시작 응용프로그램의 BackingMap에서 기본 구성을 사용하여 BackingMap을 정의하십시오.
   c. BackingMap을 정의한 후 오브젝트 그리드를 초기화하십시오. 오브젝트 그리드를 초기화하면 모든 구성이 완료되어 오브젝트 그리드를 사용할 수 있습니다.
   d. 오브젝트 그리드를 초기화한 후 Session 오브젝트를 확보하십시오. 자세한 정보는 114 페이지의 『오브젝트 그리드 인터페이스』 및 JavaDoc을 참조하십시오.

   이 단계의 지침으로 다음 예를 사용하십시오.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid objectGrid =
objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
```

4. 세션을 사용하여 트랜잭션 조작을 관리하십시오. 오브젝트 그리드 캐시에 대한 모든 액세스는 트랜잭션 방식입니다. 캐시의 오브젝트에 대한 다중 액세스, 삽입, 갱신 및 제거는 세션이라고 하는 단일 작업 단위에 들어 있습니다. 세션 종료 시 이 작업 단위에서 모든 변경사항을 확약하거나 작업 단위에서 모든 변경사항을 롤백한 후 지울 수 있습니다.

   또한 캐시에 대해 단일 원자 조작에 해당하는 자동 확약을 사용할 수 있습니다. 활성 세션 컨텍스트가 없는 경우, 캐시 컨텐츠에 대한 개별 액세스는 자동으로 확약된 자체 세션으로 묶여 있습니다.

   세션 인터페이스의 또다른 중요한 측면은 ObjectMap 인터페이스를 사용하여 BackingMap에 대한 트랜잭션 방식의 액세스를 확보하거나 BackingMap을 처리하는 것입니다. getMap 메소드를 사용하여 사전정의된 BackingMap에 대한 ObjectMap 핸들을 작성할 수 있습니다. 캐시에 대한 모든 조작(예: 삽입, 갱신, 삭제)은 ObjectMap 인스턴스에서 완료됩니다. 자세한 정보는 123 페이지의 『세션 인터페이스』 주제를 참조하십시오. 다음 예를 사용하여 세션을 확보한 후 관리하십시오.

   ```
   Session session = objectGrid.getSession();
   ObjectMap objectMap = session.getMap("someMap");
   session.begin();
   objectMap.insert("key1", "value1");
   objectMap.insert("key2", "value2");
   session.commit();
   objectMap.insert("key3", "value3"); // auto-commit
   ```

5. ObjectMap 인터페이스를 사용하여 캐시에 액세스한 후 캐시를 갱신하십시오. ObjectMap 인터페이스를 찾을 때 캐시에 액세스한 후 캐시를 갱신하는 여러 메소드에 주의하십시오. ObjectMap 인터페이스는 맵과 유사한 인터페이스로 모델화됩니다. 그러나 Eclipse와 같은 IDE를 사용하여 오브젝트 그리드 응용프로그램을 개발하는 경우, 이를 지원하기 위해 확인된 예외가 도입되었습니다. 확인된 예외 없이 java.util.Map 인터페이스를 사용할 경우 getJavaMap 메소드를 사용할 수 있습니다. 자세한 정보는 128 페이지의 『ObjectMap 및 JavaMap 인터페이스』를 참조하십시오.

   명시적인 insert 및 update 메소드로 모호한 put 조작을 피할 수 있습니다. put 메소드를 사용할 수도 있지만, 명시적인 insert 및 update 메소드를 사용하면 사용자의 의도가 보다 명확히 전달됩니다. insert 메소드로 선행하는 get 조작 없이 put 메소드를 정의하여 put 메소드 사용을 확실히 합니다. 선행하는 get 조작을 put 조작보다 먼저 시도하는 경우 캐시에 항목이 있는지 여부에 따라 put 조작을 insert 또는 update로 처리합니다.

   기본 ObjectMap 조작(get, put, insert, update, remove, touch, invalidate 및 containsKey)을 수행할 수 있습니다. 다양한 세부사항 및 응용은 47 페이지의 『시

스템 프로그래밍 모델 개요』 주제 또는 ObjectMap API 문서를 참조하십시오. 다음 예에서는 캐시를 수정하는 ObjectMap 사용을 보여줍니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid objectGrid =
objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
// Start a transaction/session...
session.begin();
objectMap.insert("key1", "value1");
objectMap.put("key2", "value2");
session.commit();
// Verify changes did commit
String value1 = (String)objectMap.get("key1");
String value2 = (String)objectMap.get("key2");
System.out.println("key1 = " + value1 + ", key2 = " + value2);
//Start a new transaction/session...
session.begin();
objectMap.update("key2", "newValue2");
objectMap.remove("key1");
    session.rollback();
// Verify changes didn't commit
String newValue1 = (String)objectMap.get("key1");
String newValue2 = (String)objectMap.get("key2");
System.out.println("key1 = " + newValue1 + ", key2 = " + newValue2);
```

6. 캐시된 오브젝트를 검색하려면 색인을 사용하십시오. 색인을 사용하면 응용프로그램이 특정 값 또는 값 범위로 오브젝트를 찾을 수 있습니다. 응용프로그램이 색인 기능을 사용하려면 BackingMap 맵이 색인 플러그인을 구성했어야 합니다. 응용프로그램은 ObjectMap 인터페이스의 getIndex() 메소드에서 색인 오브젝트를 확보하고 이것을 MapIndex 인터페이스, MapRangeIndex 인터페이스와 같은 올바른 색인 인터페이스 또는 사용자 정의 색인 인터페이스로 캐스트해야 합니다.

   현재 색인 지정 기능은 로컬 캐시에서만 지원됩니다. 색인 기능은 분산 캐시에서 지원하지 않습니다. 분산 캐시에 대해 색인 지정 조작을 수행하려는 경우 `UnsupportedOperationException` 예외가 발생합니다.

   다음 예는 색인 사용 방법을 설명합니다.

```
MapRangeIndex myIndex = (MapRangeIndex ) objectMap.getIndex("indexName");
Object searchCriteria = "targetAttributeValue";
Iterator iter = myIndex.findAll(searchCriteria);
    while (iter.hasNext()) {
Object key = iter.next();
System.out.println(objectMap.get(key));
}
```

이 섹션을 모두 읽고 코드 예제를 점검하게 되면 핵심 오브젝트 그리드 프로그래밍 모델에 보다 익숙해졌습니다.

자세한 정보는 99 페이지의 제 9 장『오브젝트 그리드 API(Application Programming Interface) 개요』를 참조하십시오.

원격 오브젝트 그리드 시작하기

여기에간단한 설명을 넣으십시오. 첫 번째 단락 및 요약에 사용됩니다.

일반적으로 43 페이지의 제 4 장『오브젝트 그리드 학습서: 응용프로그램 프로그래밍 모델』은 "로컬" 또는 오브젝트 그리드의 응용프로그램 사용법 내에서 다루었습니다. 응용프로그램은 오브젝트 그리드의 인스턴스를 작성했고 이 인스턴스를 사용했습니다. 응용프로그램 JVM(Java Virtual Machine)이 종료될 때 오브젝트 그리드 캐시도 종료되었습니다. 이름에서 알 수 있듯이 원격 오브젝트 그리드는 다른 JVM에 상주하는 오브젝트 그리드에 액세스할 수 있습니다. 다중 클라이언트는 동일한 API를 투명하게 사용하여 원격 오브젝트 그리드에 연결하고 이 오브젝트 그리드에 액세스할 수 있습니다.

1. 다음 섹션을 검토하여 시작하십시오.
   - 19 페이지의 제 3 장『오브젝트 그리드 개요』
   - 278 페이지의『오브젝트 그리드 구성』
   - 107 페이지의『오브젝트 그리드 클라이언트 연결 API』
   - 89 페이지의 제 8 장『명령행 지원』
2. 서버를 시작하려면 오브젝트 그리드 XML 파일 및 클러스터 XML 파일을 정의해야 합니다. 294 페이지의『분산 오브젝트 그리드 구성』을 참조하십시오. 이 주제는 `university.xml` 및 `universityCluster.xml` 파일을 참조합니다. 이 파일을 예로 사용하여 호스트 및 포트를 수정하여 서버를 실행하거나 시작할 수 있습니다. 오브젝트 그리드 서버 실행에 대한 자세한 내용은 89 페이지의 제 8 장『명령행 지원』을 참조하십시오.
3. 서버가 실행 중일 때 클라이언트는 이 실행 중인 서버에 연결할 수 있습니다. 클라이언트를 연결하고 오브젝트 그리드 조작을 수행하는 방법에 대한 자세한 정보는 107 페이지의『오브젝트 그리드 클라이언트 연결 API』를 참조하십시오.

시스템 프로그래밍 모델 개요

시스템 프로그래밍 모델은 오브젝트 그리드에 여러 가지 추가 기능 및 확장점을 제공합니다.

다음 다이어그램에서 시스템 프로그래밍 모델이 여러 가지 추가 기능 및 확장점을 제공하는 방법을 설명합니다.

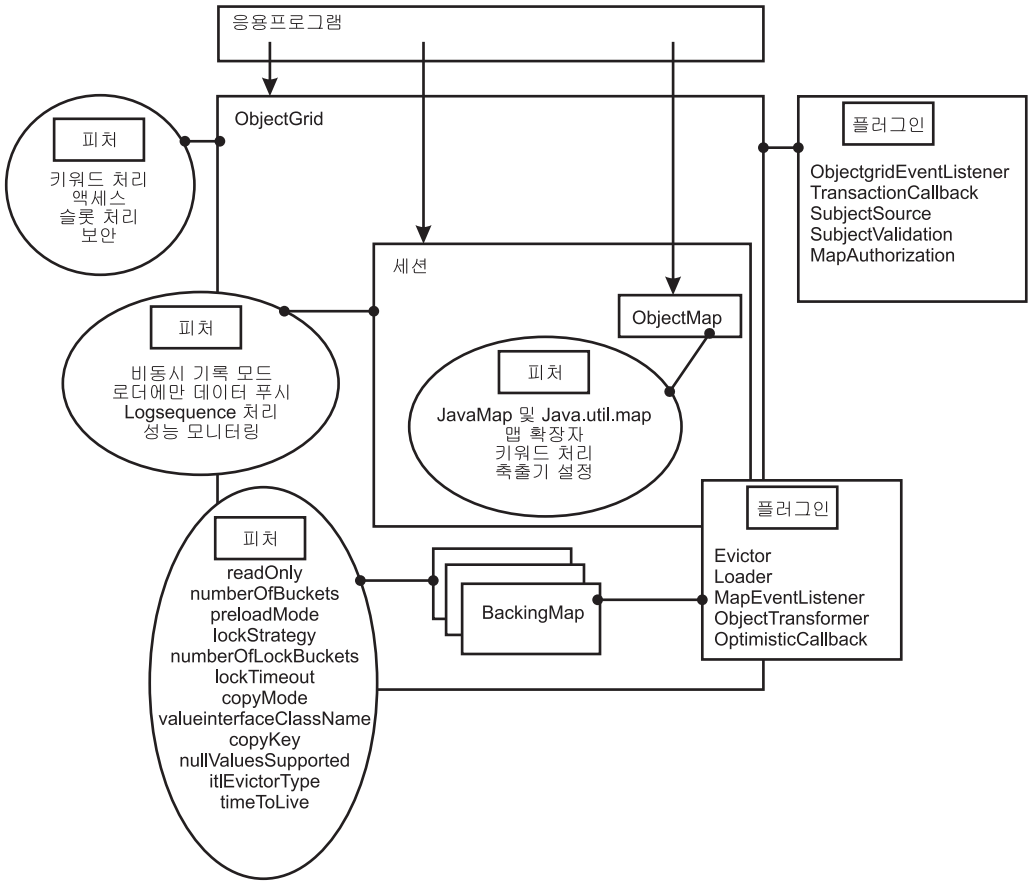


그림 13. 오브젝트 그리드 개요

오브젝트 그리드의 플러그인은 오브젝트 그리드 및 BackingMap을 포함하는 플러그 가능한 오브젝트 그리드 컴포넌트에 특정 유형의 기능을 제공하는 컴포넌트입니다. 기능은 ObjectGrid, Session, BackingMap, ObjectMap 등 오브젝트 그리드 컴포넌트의 특정 기능 또는 특성을 나타냅니다. 피처가 기능을 나타내는 경우, 특정 컴퓨팅 목적을 달성하기 위해 사용될 수 있습니다. 기능이 특성인 경우, 오브젝트 그리드 컴포넌트의 동작을 조정하는 데 사용될 수 있습니다.

다음 각섹션에서는 위 다이어그램에 설명된 일부 기능 및 확장을 설명합니다.

- 49 페이지의『시스템 프로그래밍 모델 개요: ObjectGrid 인터페이스 플러그 지점 및 기능』

  오브젝트 그리드 인터페이스에는 오브젝트 그리드와의 확장 가능한 상호 작용을 지원하는 여러 개의 플러그 지점 및 기능이 있습니다.

- 52 페이지의『시스템 프로그래밍 모델 개요: BackingMap 인터페이스 플러그 지점 및 기능』

  BackingMap 인터페이스에는 오브젝트 그리드와의 확장 가능한 상호 작용을 지원하는 여러 개의 선택적 플러그 지점과 기능이 있습니다.

- 60 페이지의『시스템 프로그래밍 모델 개요: Session 인터페이스 기능』

  세션 인터페이스에는 오브젝트 그리드와의 확장 가능한 상호 작용을 지원하는 여러 기능이 있습니다. 이 주제의 각 섹션에서는 기능을 설명하고 사용법 시나리오에 대한 간략한 코드 스니펫을 일부 제공합니다.

- 62 페이지의『시스템 프로그래밍 모델 개요: ObjectMap 인터페이스 기능』

  ObjectMap 인터페이스에는 오브젝트 그리드와의 확장 가능한 상호 작용을 지원하는 여러 기능이 있습니다. 이 주제의 각 섹션에서는 기능을 설명하고 사용법 시나리오에 대한 간략한 코드 스니펫을 일부 제공합니다.

개별 기능 및 플러그인에 대한 자세한 정보는 99 페이지의 제 9 장『오브젝트 그리드 API(Application Programming Interface) 개요』를 참조하십시오.

시스템 프로그래밍 모델 개요: ObjectGrid 인터페이스 플러그 지점 및 기능

오브젝트 그리드 인터페이스에는 오브젝트 그리드와의 확장 가능한 상호 작용을 지원하는 여러 개의 플러그 지점 및 기능이 있습니다.

다음의 각 섹션에서는 기능을 설명하고 사용법 시나리오의 간단한 일부 코드 스니펫을 제공합니다. 필요한 경우 대체 XML 구성을 표시하도록 XML 스니펫이 제공됩니다. 자세한 정보는 114 페이지의『오브젝트 그리드 인터페이스』및 278 페이지의『오브젝트 그리드 구성』주제를 참조하십시오.

키워드 처리

ObjectGrid 인터페이스에서는 키워드 기반의 유연한 무효화 메커니즘을 제공합니다. 키워드는 직렬화 가능한 오브젝트의 널이 아닌 인스턴스입니다. 어떤 방식으로든 키워드를 BackingMap 항목과 자유롭게 연관시킬 수 있습니다. 대부분의 키워드 처리는 ObjectMap 레벨에서 수행되지만 키워드의 계층 구조 트리를 구성하도록 하나의 키워드를 다른 키워드에 연관시키는 작업은 오브젝트 그리드 레벨에서 수행됩니다.

associateKeyword(java.io.Serializable parent, java.io.Serializable child) 메소드는 두 개의 키워드를 지향적 관계로 링크합니다. 상위가 무효화된 경우 하위도 무효화됩니다. 하위를 무효화해도 상위는 영향을 받지 않습니다. 예를 들어, 이 메소드를 사용하여 New York 맵 항목을 USA 맵 항목의 하위로 추가한 경우 USA가 무효화되면 모든 New York 항목도 무효화됩니다. 다음 코드 샘플을 참조하십시오.

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
// associate several cities with "USA" keyword
objectGrid.associateKeyword("USA", "New York");
objectGrid.associateKeyword("USA", "Rochester");
objectGrid.associateKeyword("USA", "Raleigh");
:
// insert several entries with various keywords
objectMap.insert("key1", "value1", "New York");
```

```
objectMap.insert("key2", "value2", "Mexico");
objectMap.insert("key3", "value3", "Raleigh");
objectMap.insert("key4", "value4", "USA");
objectMap.insert("key5", "value5", "Rochester");
objectMap.insert("key6", "value6", "France");
:
// invalidate all entries associated with "USA" keyword, leaving
// "key2" and "key6" entries
objectMap.invalidateUsingKeyword("USA", true);
:
```

자세한 정보는 132 페이지의 『키워드』를 참조하십시오.

BackingMap 액세스

오브젝트 그리드에서는 BackingMap 오브젝트에 대한 액세스를 제공합니다. defineMap 또는 getMap 메소드를 사용하여 BackingMap에 액세스할 수 있습니다. 자세한 정보는 119 페이지의 『BackingMap 인터페이스』를 참조하십시오. 다음 예에서는 두 개의 BackingMap 참조를 작성합니다.

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
BackingMap newBackingMap = objectGrid.defineMap("newMap");
:
```

슬롯 처리

트랜잭션 ID 오브젝트(TxID) 또는 데이터베이스 연결 오브젝트(Connection)와 같이 트랜잭션 도중 사용하는 오브젝트를 저장할 때 사용할 슬롯을 예약할 수 있습니다. 이 저장된 오브젝트는 reserveSlot 메소드에서 제공하는 특정 색인을 사용하여 참조됩니다. 슬롯 사용에 대한 자세한 정보는 217 페이지의 『로더』 및 234 페이지의 『TransactionCallback 플러그인』 주제를 참조하십시오. 다음 코드 스니펫에서는 슬롯 처리를 보여줍니다.

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
int index = objectGrid.reserveSlot
   (com.ibm.websphere.objectgrid.TxID.SLOT_NAME);
:
// Use the index later when storing or retrieving objects from
//the TxID object ...
TxID tx = session.getTxID();
tx.putSlot(index, someObject);
:
Object theTxObject = tx.getSlot(index);
:
```

보안 처리

보안 메커니즘을 사용하여 맵을 보호할 수 있습니다. 다음 메소드가 보안 기능 구성 및 사용 시 오브젝트 그리드에서 다음 메소드를 사용할 수 있습니다.

- getSession(Subject)
- SubjectSource
- SubjectValidation

- AuthorizationMechanism
- MapAuthorization
- PermissionCheckPeriod

사용 가능한 보안 메커니즘에 대한 자세한 정보는 149 페이지의 『오브젝트 그리드 보안』을 참조하십시오.

ObjectGridEventListener

ObjectGridEventListener 리스너에서는 트랜잭션을 시작하거나 확약할 때 응용프로그램이 알림을 수신하는 방법을 제공합니다. ObjectGridEventListener의 인스턴스를 오브젝트 그리드에서 설정할 수 있습니다. 자세한 정보는 200 페이지의 『리스너』 주제를 참조하십시오. 다음은 ObjectGridEventListener 인터페이스를 프로그램 방식으로 구현하는 방법에 대한 예입니다.

```
class MyObjectGridEventListener implements
com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.addEventListener(new MyObjectGridEventListener());
:
```

XML을 사용하여 동일한 구성을 수행할 수도 있습니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
<bean id="ObjectGridEventListner" className=
"com.somecompany.MyObjectGridEventListener" />
:
</objectGrid>
</objectGrids>
:
```

TransactionCallback 플러그인

세션에서 메소드를 호출하면 TransactionCallback 플러그인에 해당 이벤트가 전송됩니다. 오브젝트 그리드는 0개 또는 하나의 TransactionCallback 플러그인을 포함할 수 있습니다. TransactionCallback 플러그인을 사용하여 오브젝트 그리드에서 정의된 BackingMap에 해당 로더가 있어야 합니다. 자세한 정보는 234 페이지의 『TransactionCallback 플러그인』을 참조하십시오. 다음 코드 스니펫에서는 TransactionCallback 플러그인을 프로그램 방식으로 구현하는 방법을 보여줍니다.

```
class MyTransactionCallback implements
com.ibm.websphere.objectgrid.plugins.TransactionCallback { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.setTransactionCallback(new MyTransactionCallback());
:
```

XML을 사용하여 동일한 구성을 수행할 수 있습니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
<bean id="TransactionCallback" className=
"com.somecompany.MyTransactionCallback" />
</objectGrid>
</objectGrids>
:
```

시스템 프로그래밍 모델 개요: BackingMap 인터페이스 플러그 지점 및 기능

BackingMap 인터페이스에는 오브젝트 그리드와의 확장 가능한 상호 작용을 지원하는 여러 선택적 플러그 지점이 있습니다.

다음의 각 섹션에서는 기능을 설명하고 사용법 시나리오의 간단한 일부 코드 스니펫을 제공합니다. 필요한 경우 대체 XML 구성을 표시하도록 XML 스니펫이 제공됩니다. 추가 확장 정보는 119 페이지의 『BackingMap 인터페이스』 및 278 페이지의 『오브젝트 그리드 구성』 주제 또는 API 문서를 참조하십시오.

구성 속성

여러 구성 항목이 BackingMap과 연관되어 있습니다.

- **ReadOnly**(기본값: `false`): 이 속성을 `true`로 설정하면 BackingMap은 읽기 전용이 됩니다. `false`로 설정하면 BackingMap은 읽기 및 쓰기가 가능합니다. 값을 지정하지 않으면 기본적으로 읽고 쓰기가 가능합니다.
- **NullValuesSupported**(기본값: `true`): 널값 지원은 맵에 널값을 입력할 수 있음을 의미합니다. 이 속성을 true로 설정하면 ObjectMap에서 널값이 지원되지만 그렇지 않은 경우 널값은 지원되지 않습니다. 널값이 지원되는 경우 `get` 조작에서 널을 리턴하면 값이 널이거나 맵에 전달(passed-in) 키가 없음을 의미할 수 있습니다.
- **NumberOfBuckets**(기본값: `503`): 이 BackingMap에서 사용하는 버켓 수를 지정합니다. BackingMap 구현은 구현 시 해시 맵을 사용합니다. BackingMap에 항목이 많은 경우 버켓 수가 증가할 수록 충돌 위험은 낮아지므로 버켓이 많을수록 성능이 향상됩니다. 또한 버켓이 많을수록 동시성이 향상됩니다.
- **NumberOfLockBuckets**(기본값: 383): 이 BackingMap의 잠금 관리자가 사용하는 잠금 버켓 수를 지정합니다. lockStrategy 속성이 OPTIMISTIC 또는 PESSIMISTIC으로 설정된 경우 BackingMap에서 잠금 관리자가 작성됩니다. 잠금 관리자는 해시 맵을 사용하여 하나 이상의 트랜잭션에서 잠근 항목을 추적합니다. 해시 맵에 항목이 많은 경우 버켓 수가 증가할 수록 충돌 위험은 낮아지므로 잠금 버켓이 많을수록 성능이 향상됩니다. 또한 잠금 버켓이 많을수록 동시성이 향상됩니다. lockStrategy가 NONE인 경우 BackingMap에서 잠금 관리자를 사용하지 않습니다. 이 경우 numberOfLockBuckets 속성을 설정해도 적용되지 않습니다.

프로그램 방식의 구성 예제

다음 예는 BackingMap의 특성을 구성합니다.

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// override default of read/write
backingMap.setReadOnly(true);
// override default of allowing Null values
backingMap.setNullValuesSupported(false);
// override default (prime numbers work best)
backingMap.setNumberOfBuckets(251);
// override default (prime numbers work best)
backingMap.setNumberOfLockBuckets(251);
:
```

XML 구성 예제

다음 XML 구성 예는 위 프로그램 방식의 샘플과 동일한 특성을 구성합니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" readOnly="true" nullValuesSupported="false"
numberOfBuckets="251" numberOfLockBuckets="251" />
</objectGrid>
</objectGrids>
:
```

잠금 계획

잠금 계획이 OPTIMISTIC 또는 PESSIMISTIC으로 설정된 경우 BackingMap에서 잠금 관리자가 작성됩니다. 교착 상태를 방지하기 위해 잠금 관리자에서 잠금 부여를 대기하는 기본 제한시간 값을 설정합니다. 이 제한시간 한계를 초과하면 LockTimeoutException 예외가 발생합니다. 대부분의 응용프로그램에서 기본값(15초)으로 충분하지만 로드가 많은 시스템의 경우 실제 교착 상태가 없는 경우에도 제한시간을 초과할 수 있습니다. 이 경우 setLockTimeout 메소드를 사용하여 잠금 제한시간 값을 기본값에서 잘못된 제한시간 예외가 발생하지 않는 값으로 증가시킬 수 있습니다. 잠금 계획이 NONE인 경우 BackingMap에서 잠금 관리자를 사용하지 않습니다. 이 경우 lockTimeout 속성을 설정해도 적용되지 않습니다. 자세한 정보는 140 페이지의 『잠금』 주제를 참조하십시오.

프로그램 방식의 구성 예제

다음 예는 잠금 계획을 설정합니다.

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// override default value of OPTIMISTIC
backingMap.setLockStrategy(LockStrategy.PESSIMISTIC);
backingMap.setLockTimeout(30); // sets lock timeout to 30 seconds
:
```

XML 구성 예제

다음 예는 위 프로그램 방식의 예제와 동일한 잠금 계획을 설정합니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" lockStrategy="PESSIMISTIC" lockTimeout="30" />
</objectGrid>
</objectGrids>
:
```

키 및 값 복사

키 및 값을 복사하면 자원 및 성능 측면에서 비용이 높아질 수 있습니다. 사본을 작성하는 성능이 없으면 분명하지 않고 디버그하기 어려운 문제점이 발생할 수 있습니다. 오브젝트 그리드는 키를 복사할지 또는 값을 복사할지 여부 및 복사 시기를 구성하는 기능을 제공합니다. 일반적으로 키는 바꿀 수 없다고 간주되므로 키 오브젝트는 복사하지 않아도 됩니다. 키 오브젝트의 기본 모드는 복사하지 않는 것입니다. 값 오브젝트는 응용프로그램에서 수정될 가능성이 높습니다. 값 오브젝트에 대한 실제 참조 대비 값 오브젝트의 사본을 제공하는 시기는 구성 가능한 옵션입니다. CopyKey 및 CopyMode 설정에 대한 추가 정보는 357 페이지의 제 11 장『오브젝트 그리드 성능 우수 사례』주제 및 JavaDoc을 참조하십시오.

프로그램 방식의 구성 예제

다음은 복사 모드 및 복사 키를 설정하는 예제입니다.

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setCopyKey(true); // make a copy of each new key
backingMap.setCopyMode(NO_COPY); // Most efficient - trust the application
:
```

XML 구성 예제

다음 예제는 위 프로그램 방식의 구성과 동일한 구성을 생성합니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" copyKey="true" copyMode="NO_COPY" />
</objectGrid>
</objectGrids>
:
```

축출기

축출기는 정기적으로 맵에서 불필요한 항목을 정리할 때 사용됩니다. 제거되는 항목은 축출기에서 정의됩니다. 내장 축출기는 시간 기준이므로 축출 계획은 맵에서 항목이 활성(alive)인 시간에 근거합니다. 다른 축출 계획은 사용량, 크기 또는 이 요소의 조합을 기반으로 합니다.

- **내장 TTL(Time to Live) 축출기**: 내장 TTL(Time to Live) 축출기는 setTtlEvictorType 및 setTimeToLive 메소드를 사용하여 BackingMap에 설정되는 두, 세 개의 구성 항목을 제공합니다. 기본적으로 이 내장 TimeToLive 축출기는 활성화되지 않습니다. CREATION_TIME, LAST_ACCESS_TIME 또는 NONE(기본값)의 세 개의 값 중 하나를 사용하여 setTtlEvictorType 메소드를 호출하면 해당 축출기를 활성화할 수 있습니다. 그런 다음 선택한 TimeToLive 축출기 유형에 따라 각 맵 항목에서 지속 시간을 설정할 때 setTimeToLive 메소드 값을 사용합니다.
- **Evictor 플러그인:** 내장 TTL(Time to Live) 축출기 이외에도 응용프로그램은 고유한 Evictor 구현 플러그인을 제공할 수 있습니다. 정기적으로 맵 항목을 무효화하는 자체 알고리즘을 사용할 수 있습니다.

프로그램 방식의 구성

다음 클래스에서는 축출기를 작성합니다.

```
class MyEvictor implements com.ibm.websphere.objectgrid.plugins.Evictor { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// timer starts when entry is first created
backingMap.setTtlEvictorType(CREATION_TIME);
// Allow each map entry to live 30 seconds before invalidation
backingMap.setTimeToLive(30);
// Both builtin and custom Evictors will be active
backingMap.setEvictor(new MyEvictor()); :
```

XML 구성

다음 XML 코드에서는 위 프로그램 방식의 구성과 동일한 구성을 작성합니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" pluginCollection="default"
ttlEvictorType="CREATION_TIME" timeToLive="30" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Evictor" className="com.somecompany.MyEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

자세한 정보는 206 페이지의 『축출기』를 참조하십시오.

로더

오브젝트 그리드 로더는 오브젝트 그리드 맵이 동일한 시스템 또는 다른 시스템의 지속적 저장소에서 정상적으로 보관된 데이터의 메모리 캐시로 작동할 수 있게 하는 플러그 가능한 컴포넌트입니다. 일반적으로 데이터베이스 또는 파일 시스템을 지속적 저장소로 사용합니다. 로더에는 지속적 저장소에서 데이터를 읽고 쓰는 논리가 포함되어 있습니다.

로더는 오브젝트 그리드 BackingMap의 선택적 플러그인입니다. 하나의 로더만 지정된 BackingMap에 연관될 수 있으며 모든 BackingMap에는 고유한 로더 인스턴스가 들어 있습니다. BackingMap은 해당 로더에서 포함하지 않은 데이터를 요청합니다. 맵에서 변경된 사항은 로더로 푸시됩니다. 로더 플러그인에서는 BackingMap이 맵 및 해당 지속적 저장 사이에서 데이터를 이동시키는 방법을 제공합니다.

프로그램 방식의 구성

다음은 로더 구현의 예입니다.

```
class MyLoader implements com.ibm.websphere.objectgrid.plugins.Loader { .. }
:
Loader myLoader = new MyLoader();
myLoader.setDataBaseName("testdb");
myLoader.setIsolationLevel("ReadCommitted");
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setLoader(myLoader);
backingMap.setPreloadMode(true);
:
```

XML 구성

다음 XML 샘플은 위 프로그램 방식과 동일한 구성을 생성합니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" pluginCollectionRef="default" preloadMode="true" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Loader" classname="com.somecompany.MyLoader">
<property name="dataBaseName" type="java.lang.String" value="testdb" />
<property name="isolationLevel" type="java.lang.String" value="ReadCommitted" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

자세한 정보는 217 페이지의 『로더』 주제를 참조하십시오.

MapEventListener 인터페이스

응용프로그램으로 데이터 사전 로드 완료 또는 맵 항목의 축출과 같은 맵에 대한 이벤트를 수신하기 위해, MapEventListener 콜백 인터페이스가 응용프로그램에서 구현됩니다. 다음 코드 예에서는 BackingMap 인스턴스에서 MapEventListener 인스턴스를 설정하는 방법을 보여줍니다.

프로그램 방식의 구성

```
class MyMapEventListener implements
 com.ibm.websphere.objectgrid.plugins.MapEventListener { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.addMapEventListener(new MyMapEventListener() );
```

XML 구성

다음 예는 위 프로그램 방식과 동일한 구성을 생성합니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="MapEventListener" classname="com.somecompany.MyMapEventListener" />
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

자세한 정보는 200 페이지의 『리스너』 주제를 참조하십시오.

ObjectTransformer 인터페이스

ObjectTransformer를 사용하여 직렬화 가능으로 정의되지 않은 캐시 항목 키 및 값을 직렬화할 수 있습니다. 따라서 직렬화 가능 인터페이스를 직접 구현하거나 확장하지 않아도 고유한 직렬화 설계를 정의할 수 있습니다. 이 인터페이스에서는 키 및 값에서 복사 기능을 수행하는 메소드도 제공합니다. 다음은 ObjectTransformer 인터페이스를 구현하는 클래스입니다.

프로그램 방식의 구성

```
class MyObjectTransformer implements
 com.ibm.websphere.objectgrid.plugins.ObjectTransformer { ... }
:
ObjectTransformer myObjectTransformer = new MyObjectTransformer();
myObjectTransformer.setTransformType("full");
```

```
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setObjectTransformer(myObjectTransformer);
:
```

XML 구성

다음 XML 예는 위 프로그램 방식의 샘플과 동일한 구성을 생성합니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="ObjectTransformer" className="com.somecompany.MyObjectTransformer">
<property name="transformType" type="java.lang.String" value="full"
description="..." />
</bean>
</backingMapCollection>
</backingMapCollections>
:
```

자세한 정보는 229 페이지의 『ObjectTransformer 플러그인』 주제를 참조하십시오.

OptimisticCallback 인터페이스

OptimisticCallback 인터페이스를 사용하여 지정된 값 오브젝트와 연관된 버전 필드를 작성 및 처리할 수 있습니다. 값 오브젝트를 직접 사용하여 검색 이후 다른 캐시 클라이언트에서 값을 수정했는지 여부를 판별하는 방법은 대체로 매우 비효율적이며 오류가 발생하기 쉽습니다. 대안으로 값 오브젝트의 상태를 표시하는 다른 필드를 제공하는 방법이 있습니다. OptimisticCallback 인터페이스의 목적은 값 오브젝트를 표시하는 대체 버전화된 값 오브젝트를 제공하는 것입니다. 다음은 OptimisticCallback 인터페이스의 샘플 구성입니다.

프로그램 방식의 구성

```
class MyOptimisticCallback implements
 com.ibm.websphere.objectgrid.plugins.OptimisticCallback { ... }
:
OptimisticCallback myOptimisticCallback = new MyOptimisticCallback();
myOptimisticCallback.setVersionType("Integer");
backingMap.setOptimisticCallback(myOptimisticCallback);
:
```

XML 구성

다음 예는 위 프로그램 방식과 동일한 구성을 생성합니다.

```
:
<objectGrids>
<objectGrid name="someGrid">
```

```
<backingMap name="someMap" pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="OptimisticCallBack" classname="com.somecompany.MyOptimisticCallback">
<property name="versionType" type="java.lang.string" value="Integer"
description="..." />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

색인

MapIndexPlugin 또는 축약형 색인은 BackingMap이 저장된 오브젝트의 지정된 속성에 기반하여 색인을 빌드하기 위해 사용하는 옵션입니다. 응용프로그램은 색인을 사용하여 특정 값 또는 범위 값으로 오브젝트를 찾을 수 있습니다. 색인을 사용하려면 응용프로그램이 ObjectMap 인터페이스의 getIndex() 메소드에서 색인 오브젝트를 확보한 다음 이것을 MapIndex 또는 MapRangeIndex와 같은 올바른 색인 인터페이스 또는 사용자 정의 색인 인터페이스에 캐스트해야 합니다.

현재 색인 지정 기능은 분산 캐시가 아닌 로컬 캐시에서만 지원됩니다. 분산 캐시에 대해 색인 지정 조작을 수행하려면 UnsupportedOperationException이 발생합니다.

정적과 동적인 두 가지 유형의 색인이 있습니다. 정적 색인은 프로그램 방식 구성 및 XML 구성 모두를 통해 작성될 수 있습니다. 동적 색인은 프로그램 방식으로만 작성될 수 있습니다.

프로그램 방식의 구성

다음 코드 예는 BackingMap 인스턴스에 정적 색인을 추가하는 방법을 설명합니다.

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
//use the builtin com.ibm.websphere.objectgrid.plugins.index.HashIndex
//class as the index plugin class.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
personBackingMap.addMapIndexPlugin(mapIndexPlugin);
//Note: the previous Index configuration assumes that the stored object has
// an attribute named EmployeeCode and a method named getEmployeeCode()
//that returns the value of the EmployeeCode attribute.
:
```

다음 코드 예는 BackingMap 인스턴스에서 동적 색인을 작성하는 방법을 설명합니다.

```
class DynamicIndexCallbackImpl implements
com.ibm.websphere.objectgrid.plugins.index.DynamicIndexCallback { ... }
:
```

```
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
objectGrid.initiallize();
:
//insert, update, or remove data
//Dynamic index can be created after the containing ObjectGrid instance has
//been initialized
//If there is a need to create a dynamic index, create it without
//DynamicIndexCallback
personBackingMap.createDynamicIndex("CODE2", true, "employeeCode", null);
:
//Another option is to create dynamic index with DynamicIndexCallback
//Assuming there is a DynamicIndexCallbackImpl class implements
//DynamicIndexCallback interface
personBackingMap.createDynamicIndex("CODE3", true, "employeeCode",
new DynamicIndexCallbackImpl());
:
```

XML 구성

다음 예는 정적 색인의 위 프로그램 방식 예와 동일한 구성을 생성합니다.

```
:
<objectGrids>
<objectGrid name="indexSampleGrid">
<backingMap name="person" pluginCollectionRef="person" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="person">
<bean id="MapIndexPlugin
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="CODE"
description="index name" />
<property name="RangeIndex" type="boolean" value="true"
description="true for MapRangeIndex />
<property name="AttributeName" type="java.lang.String"
value="employeeCode" description="attriubte name" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

자세한 정보는 색인 지정 주제를 참조하십시오.

시스템 프로그래밍 모델 개요: Session 인터페이스 기능

Session 인터페이스에는 ObjectGrid와의 확장 가능한 상호 작용을 지원하는 여러 기능이 있습니다. 다음의 각 섹션에서는 기능을 설명하고 사용법 시나리오의 간단한 일부 코드 스니펫을 제공합니다.

Session 인터페이스에 대한 자세한 정보는 123 페이지의 『세션 인터페이스』를 참조하십시오.

비동시 기록 모드

응용프로그램은 때때로 로더가 아닌 기본 맵에만 변경사항을 적용하려고 합니다. Session 인터페이스의 beginNoWriteThrough 메소드는 이를 위해 디자인되었습니다. Session 인터페이스의 isWriteThroughEnabled 메소드는 백엔드 로더에 현재 세션을 쓰는지 확인하는 데 사용할 수 있습니다. 이 메소드는 현재 처리하는 세션 유형을 알고 있는 세션 오브젝트의 다른 사용자에게 유용할 수 있습니다. 다음 예는 비동시 기록 모드를 사용 가능하게 합니다.

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
boolean isWriteThroughEnabled = session.isWriteThroughEnabled();
// make updates to the map ...
session.commit();
:
```

로더에만 데이터 푸시

응용프로그램은 다음 예와 같이 flush 메소드를 호출하여 영구적으로 변경사항을 확약하지 않고 로더에 세션의 로컬 변경사항을 적용할 수 있습니다.

```
:
Session session = objectGrid.getSession();
session.begin();
// make some changes ...
session.flush(); // push these changes to the Loader, but don't commit yet
// make some more changes ...
session.commit();
:
```

processLogSequence 메소드

processLogSequence 메소드는 LogSequence를 처리할 때 사용됩니다. LogSequence의 각 LogElement를 점검하여 LogSequence MapName으로 식별된 BackingMap에서 적절한 조작(예: insert, update, invalidate 조작)을 수행합니다. 이 메소드를 호출하기 전에 먼저 오브젝트 그리드 세션이 활성화되어야 합니다. 그런 다음 호출자에서 세션을 완료하도록 적절한 확약 또는 롤백 호출을 발행해야 합니다. 자동 확약 처리는 이 메소드 호출 시 사용할 수 없습니다.

이 메소드의 기본 사용은 원격 JVM에서 수신한 LogSequence를 처리하는 것입니다. 예를 들어 분배 확약 지원을 사용하면 지정된 확약 세션과 연관된 LogSequence가 다른 JVM(Java Virtual Machine)의 다른 청취 오브젝트 그리드에 분배됩니다. 원격 JVM에서 LogSequence를 수신하면 리스너에서 beginNoWriteThrough 메소드를 사용하여 세션을 시작하고 이 processLogSequence 메소드를 호출한 후 세션에서 commit 메소드를 수행할 수 있습니다. 예를 들면 다음과 같습니다.

```
:
session.beginNoWriteThrough();
try {
 session.processLogSequence(inputSequence);
}
catch (Exception e) {
    session.rollback();
 throw e;
}
session.commit();
:
```

성능 모니터링

WebSphere Application Server에서 맵을 실행하는 동안, 선택적으로 성능 모니터링을 하도록 맵을 사용할 수 있습니다. Session의 setTransactionType 메소드를 사용하여 성능 모니터링 기능을 구성 및 사용할 수 있습니다. 자세한 정보는 318 페이지의 『WebSphere Application Server PMI(Performance Monitoring Infrastructure)에서 오브젝트 그리드 성능 모니터링』을 참조하십시오.

시스템 프로그래밍 모델 개요: ObjectMap 인터페이스 기능

ObjectMap 인터페이스에는 오브젝트 그리드와의 확장 가능한 상호 작용을 지원하는 여러 기능이 있습니다.

ObjectMap 인터페이스에 대한 자세한 정보는 128 페이지의 『ObjectMap 및 JavaMap 인터페이스』를 참조하십시오.

JavaMap 인터페이스 및 java.util.Map 인터페이스

응용프로그램에서 java.util.Map 인터페이스를 사용할 경우 ObjectMap에 getJavaMap 메소드가 있으므로 응용프로그램은 ObjectMap에서 지원하는 java.util.Map 인터페이스 구현을 확보할 수 있습니다. 그러면 java.util.Map 인터페이스를 확장하는 JavaMap 인터페이스에 리턴된 맵 인스턴스를 캐스트할 수 있습니다. JavaMap 인터페이스는 ObjectMap과 메소드 서명은 동일하지만 예외 처리는 서로 다릅니다. JavaMap 인터페이스는 java.util.Map 인터페이스를 확장하므로 모든 예외는 java.lang.RuntimeException 클래스의 인스턴스입니다. JavaMap 인터페이스가 java.util.Map 인터페이스를 확장하므로 오브젝트 캐싱에서 java.util.Map 인터페이스를 사용하는 기존 응용프로그램을 통해 오브젝트 그리드를 쉽고 빠르게 사용할 수 있습니다. 다음은 코드 스니펫입니다.

```
:
JavaMap javaMap = (JavaMap)objectMap.getJavaMap();
:
```

맵 확장

ObjectMap 인터페이스에서는 확인된 예외 성능 이외에도 추가 기능적 성능을 제공합니다. 예를 들어 getForUpdate 메소드를 사용하여 지정된 맵 항목을 갱신하도록 지정

할 수 있습니다. 이 메소드는 필요한 경우 처리 시 항목을 잠글 수 있음을 로더 플러그인 및 오브젝트 그리드 런타임에 표시합니다. 일괄처리는 getAll, putAll 및 removeAll 메소드의 또다른 추가 성능입니다. 이 메소드에 대한 자세한 정보는 API 문서를 참조하십시오.

키워드 처리

대부분의 맵 조작에는 키워드 매개변수 버전(예: insert, get, getForUpdate, put, remove 및 invalidate)이 있습니다. 사용하기 쉽도록 setDefaultKeyword 메소드도 제공됩니다. 이 메소드는 맵 조작의 키워드 버전을 사용하지 않고 항목을 키워드에 연관시킵니다. 키워드 예는 다음과 같습니다.

```
:
// setDefaultKeyword
session.begin();
objectMap.setDefaultKeyword("New York");
Person p = (Person) objectMap.get("Billy"); // "Billy" entry has "New York" keyword
p = (Person) objectMap.get("Bob", "Los Angeles"); // "Bob" entry
//has "Los Angeles" keyword
objectMap.setDefaultKeyword(null);
p = (Person) objectMap.get("Jimmy"); // "Jimmy" entry has no keyword
session.commit();
:
// keyword parameter version of insert operation
session.begin();
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person, "Rochester"); // "BillyBob" has
//"Rochester" keyword
session.commit();
:
```

자세한 정보는 132 페이지의 『키워드』를 참조하십시오.

복사 모드 메소드

setCopyMode 메소드를 사용하면 이 세션 또는 트랜잭션 전용으로 이 맵에서 맵의 복사 모드를 대체할 수 있습니다. 이 메소드를 사용하면 응용프로그램의 필요에 따라, 하나의 세션을 기반으로 최적 복사 모드를 사용할 수 있습니다. 활성 세션에서는 복사 모드를 변경할 수 없습니다. 복사 모드를 BackingMap에서 정의된 모드로 재설정하는 clearCopyMode 메소드가 있습니다. 활성 세션이 없는 경우에만 이 메소드를 호출할 수 있습니다. 복사 모드를 설정하는 예는 다음과 같습니다.

```
:
objectMap.setCopyMode(CopyMode.COPY_ON_READ, null);
session.begin();
// modify objectMap ...
session.commit();
objectMap.clearCopyMode(); // reset CopyMode to BackingMap setting
session.begin();
// modify objectMap ...
session.commit();
:
```

자세한 정보는 229 페이지의 『ObjectTransformer 플러그인』 및 357 페이지의 제 11 장 『오브젝트 그리드 성능 우수 사례』 주제를 참조하십시오.

축출기 설정

ObjectMap 레벨에서 내장 TimeToLive 축출기의 TimeToLive 제한시간 값을 대체할 수 있습니다. setTimeToLive 메소드는 지정된 캐시 항목이 활성(live)인 초 수를 설정합니다. 이 값을 수정하면 이전 TimeToLive 값이 리턴됩니다. 이 TimeToLive 값은 축출하기 전에 항목이 캐시에 남아 있는 최소 시간으로, 마지막 액세스 시간 이후 항목이 남아 있어야 하는 시간을 내장 TimeToLive 축출기에 표시합니다. 새 TimeToLive 값은 ObjectMap 인스턴스를 확보할 때 사용한 세션 오브젝트가 시작한 트랜잭션에서 액세스하는 ObjectMap 항목에만 적용됩니다. 새 TimeToLive 값은 세션에서 처리 중인 모든 트랜잭션 및 세션에서 실행할 향후 트랜잭션에 적용됩니다. 새 TimeToLive 값은 일부 다른 세션이 시작한 트랜잭션에서 액세스하는 ObjectMap 인스턴스의 항목에는 영향을 주지 않습니다. ObjectMap에서 이 메소드를 호출하면 이 ObjectMap 인스턴스에서 BackingMap의 setTimeToLive 메소드로 설정한 이전 값이 대체됩니다. 예를 들면 다음과 같습니다.

```
:
session.begin();
int oldTTL = objectMap.setTimeToLive(60); // set TTL to 60 seconds
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person); // "BillyBob" entry will have a TTL
//of 60 seconds
session.commit();
:
objectMap.setTimeToLive(oldTTL); // reset TTL to original value
Person person2 = new Person("Angelina", "Jolie", "somewhere");
objectMap.insert("Brad", person2); // "Brad" entry will use original TTL value
:
```

자세한 정보는 206 페이지의 『축출기』 주제를 참조하십시오.

제 5 장 오브젝트 그리드 샘플

이 주제에서는 WebSphere Extended Deployment 제품 설치 시 제공되는 오브젝트 그리드 샘플을 설명합니다.

개요

여러 오브젝트 그리드 샘플에서 J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램 및 파티션 기능(WPF)을 통합하는 방법을 표시합니다. 이 주제에서는 각각의 샘플, 각 샘플에서 보여주는 기능, 각 샘플의 위치 및 샘플을 실행하는 환경을 설명합니다.

이 주제에서는 WebSphere Extended Deployment 설치 시 제공되는 샘플을 설명합니다. 다른 개방형 소스 프레임워크와의 오브젝트 그리드 통합 및 JMS(Java Message Service) 통합을 사용하는 경우와 관련된 다른 샘플은 다음 URL을 참조하십시오 :http://www-1.ibm.com/support/docview.wss?uid=swg27006432

샘플

- **ObjectGridSamplesSA:** 이 샘플은 오브젝트 그리드 기능을 보여주는 objectgridSamples.jar 파일에 패키지된 J2SE(Java 2 Platform, Standard Edition) 예제 세트입니다. 이 J2SE 샘플은 J2SE 환경에서 실행 가능합니다. objectgridSamples.jar 파일에는 이 샘플을 실행하기 위한 지시사항이 포함된 SamplesGuide.htm 파일이 있습니다.
- **ObjectGridSample:** 이 샘플은 Servlet 및 세션 엔터프라이즈 Bean에서 오브젝트 그리드 기능을 사용하는 방법을 보여주는 J2EE 예제입니다. 이 샘플은 ObjectGridSample.ear EAR(Enterprise ARchive) 파일에 제공되어 있습니다. ObjectGridSample.ear 파일에는 이 샘플을 설정하고 실행하기 위한 지시사항을 포함하는 readme.txt 파일이 있습니다.
- **ObjectGridPartitionCluster:** 이 샘플은 WPF와 오브젝트 그리드를 함께 사용하는 방법, bjectGridEventListener를 사용하여 오브젝트 변경사항을 전달하는 방법 및 컨텍스트 기반 라우팅을 사용 가능하게 하여 오브젝트 그리드 무결성 및 일관성을 유지보수하는 방법을 보여주는 J2EE 샘플입니다. 이 샘플은 D_ObjectGridPartitionClusterSample.ear EAR 파일에서 제공됩니다. D_ObjectGridPartitionClusterSample.ear 파일에는 이 샘플을 설정하고 실행하기 위한 지시사항을 포함하는 readme.txt 파일이 있습니다.
- **ObjectGridJMSSamples:** 이 샘플은 ObjectGridJMSSamples.zip 파일에 패키지된 J2EE 샘플 세트로, JMS 기능을 사용하여 단일 JVM 또는 클러스터 환경에서 오브젝트 그리드 인스턴스의 변경사항을 다른 오브젝트 그리드 인스턴스로 전송하는 방법을 보여줍니다. 이 J2EE 샘플은 다음 웹 페이지에서만 사용 가능합니다: http://www-1.ibm.com/support/docview.wss?uid=swg27006432

샘플 기능

표 2. 샘플 기능

기능 영역	오브젝트 그리드 **SamplesSA** 샘플	오브젝트 그리드 **Sample** 샘플	**ObjectGridPartition Cluster** 샘플	오브젝트 그리드 **JMSSamples** 샘플
오브젝트 그리드 EventListener			x	x
트랜잭션 콜백	x	x	x	
로더	x	x	x	
MapEvent 리스너	x			
오브젝트 변환기	x	x	x	x
변경이 예상되지 않는 콜백	x	x	x	
BackingMap 복사 모드	x	x		
분배 무효화			x	x
분배 갱신			x	x
LogSequence 처리				x
파티션 기능(WPF)			x	
JMS(Java Message Service)				x
맵 색인	x			
오브젝트 그리드 클러스터	x	x		
오브젝트 그리드 클러스터 컨텍스트	x	x		
분산 오브젝트 그리드	x	x		
오브젝트 그리드 관리	x			
오브젝트 그리드 보안	x		x	

위치

WebSphere Extended Deployment를 설치하면 다음 .jar 파일이 다음 디렉토리에 들어 있습니다.

표 3. 샘플 위치

샘플	위치
ObjectGridSamplesSA	*install_root*₩optionalLibraries₩ObjectGrid₩objectgridSamples.jar
ObjectGridSample	*install_root*₩installableApps₩ObjectGridSample.ear
ObjectGridPartitionCluster	*install_root*₩installableApps₩D_ObjectGridPartitionClusterSample.ear

목록으로 제공된 샘플의 갱신된 버전 및 ObjectGridJMSSamples와 같은 추가 샘플은 http://www-1.ibm.com/support/docview.wss?uid=swg27006432에서 볼 수 있습니다. 또한 다음 웹 페이지http://www.ibm.com/developerworks에서관심이 있는 주제를 설명하는 IBM DeveloperWorks에 대한 항목을 찾을 수 있습니다. **오브젝트 그리드**를 검색하십시오.

샘플 환경

일부 샘플은 J2SE 환경에서 실행 가능하지만 일부 샘플은 J2EE 환경에서 실행해야 합니다. 일부는 단일 서버 인스턴스에서 실행 가능하지만 나머지는 클러스터에서 실행해야 합니다. 다음 표에서는 샘플의 실행 환경을 표시합니다.

제한사항: WebSphere Extended Deployment 버전 6.0 환경에서 오브젝트 그리드를 사용하는 경우, 추가 라이센스 부여 계약을 맺은 J2SE(Java 2 Platform, Standard Edition) 버전 1.4.2 이상의 환경 또는 WebSphere Application Server 버전 6.02 이상의 환경에서 오브젝트 그리드를 사용할 수도 있습니다. 세부사항은 영업 담당자에게 문의하십시오.

표 4. 샘플 실행 환경

		ObjectGrid SamplesSA	**ObjectGrid Sample**	**ObjectGrid Partition Cluster**	**ObjectGrid JMSSamples**
J2SE	Eclipse	x			
	명령행	x			
WebSphere Application Sever 버전 6.0.*x*	단일 서버		x		x
	클러스터		x		x
	Rational Application Developer 유닛 테스트 환경 (UTE)		x		
WebSphere Application Server 버전 5.0.2.*x* 및 버전 5.1.*x*	단일 서버		x		
	클러스터		x		
WebSphere Extended Deployment 버전 6.0.*x*	단일 서버		x		x
	클러스터		x	x	x

제 6 장 오브젝트 그리드 패키징

WebSphere Extended Deployment 설치 또는 혼합 서버 환경을 설치하는 두 가지 방법으로 오브젝트 그리드 패키지에 액세스할 수 있습니다.

WebSphere Extended Deployment 버전 6.0.1 오브젝트 그리드 패키지

WebSphere Extended Deployment 버전 6.0.1 이상을 설치할 경우, 다음 런타임 파일이 설치됩니다.

표 5. WebSphere Extended Deployment 오브젝트 그리드 런타임 파일

파일 이름	런타임 환경	설명
/lib/asm.jar /lib/cglib.jar	로컬, 클라이언트 및 서버	COW(Copy on Write) 복사 모드를 사용할 경우 이 JAR는 cglib 유틸리티 기능에 사용됩니다.
/lib/wsobjectgrid.jar	로컬, 클라이언트 및 서버	이 JAR(Java archive) 파일에는 WebSphere Extended Deployment 버전 6.0.1 이상의 환경에서 사용하기 위한 오브젝트 그리드 로컬, 클라이언트 및 서버 런타임이 들어 있습니다.

혼합 서버 환경용 WebSphere Extended Deployment 버전 6.0.1 오브젝트 그리드 패키지

혼합 서버 환경용 WebSphere Extended Deployment를 설치하면 다음의 런타임 파일이 설치됩니다.

표 6. 혼합 서버 환경용 WebSphere Extended Deployment 오브젝트 그리드 런타임 파일

파일 이름	런타임 환경	설명
/ObjectGrid/lib/asm.jar /ObjectGrid/lib/cglib.jar	로컬, 클라이언트 및 서버	COW(Copy on Write) 복사 모드를 사용할 경우 이 JAR 파일은 cglib 유틸리티 기능에 사용됩니다. COW(Copy on Write) 복사 모드를 사용하면서 cglib 프록시 기능을 사용하려는 경우 CLASSPATH에 이 Jar 파일을 포함시키십시오. 이 JAR 파일은 서버 런타임에 자동으로 포함됩니다. 이 파일을 클라이언트나 로컬 오브젝트 그리드 런타임에 추가하십시오.

표 6. 혼합 서버 환경용 WebSphere Extended Deployment 오브젝트 그리드 런타임 파일 (계속)

파일 이름	런타임 환경	설명
/ObjectGrid/lib/mx4j.jar /ObjectGrid/lib/mx4j-remote.jar /ObjectGrid/lib/mx4j-tools.jar	관리 게이트웨이 클라이언트 및 서버	이 JAR 파일은 관리 게이트웨이 클라이언트 프로그램과 함께 오브젝트 그리드 관리 게이트웨이 서버에서 사용하는 mx4j 유틸리티에 사용됩니다. 관리 게이트웨이 서버에 연결하는 경우 이 JAR 파일을 관리 게이트웨이 클라이언트 CLASSPATH에 추가하십시오.
/ObjectGrid/lib/objectgrid.jar	로컬, 클라이언트 및 서버	이 JAR 파일은 J2SE(Java 2 Platform, Standard Edition) 버전 1.4.2 이상의 독립형 서버 런타임에서 사용합니다. 또한 J2SE 버전 1.3 이상의 클라이언트 및 로컬 런타임에 이 Jar 파일을 사용할 수도 있습니다.
/ObjectGrid/lib/ogclient.jar	로컬 및 클라이언트	WebSphere 프로세스 외부에서 실행될 경우 이 JAR 파일에는 로컬 및 클라이언트 오브젝트 그리드 런타임만 포함됩니다. 풋프린트가 작아 objectgrid.jar에 이 JAR 파일을 사용하는 것이 더 바람직합니다. 이 jar을 J2SE 버전 1.3 이상과 함께 사용할 수 있습니다.
/ObjectGrid/lib/wsobjectgrid.jar	로컬, 클라이언트 및 서버	WebSphere Application Server 버전 6.0.2 이상에서 이 Jar 파일을 사용하십시오. 이 JAR 파일은 WebSphere Extended Deployment로 설치한 것과 동일한 Jar 파일입니다.
/ObjectGrid/lib/wsogclient.jar	로컬 및 클라이언트	WebSphere Application Server 버전 5.0.2 이상에 이 Jar 파일을 사용하십시오. 이 Jar 파일에는 로컬 및 클라이언트 오브젝트 그리드 런타임만이 들어 있습니다.

J2SE 버전 1.3에서 오브젝트 그리드 사용에 대한 고려사항

J2SE 버전 1.3.x 환경에서 ogclient.jar 또는 objectgrid.jar 파일을 사용하는 경우, 다음 요구사항을 J2SE 1.3.x 환경에 추가하여 오브젝트 그리드에서 동작하도록 해야 합니다.

- **JAAS(Java Authentication and Authorization Service) 구현**. J2SE 버전 1.3에는 JAAS 스펙의 일부인 javax.security.Subject 오브젝트가 들어 있지 않습니다.

오브젝트 그리드 및 세션 인터페이스에서는 이 오브젝트가 필요합니다. JAAS 구현을 jre/lib/ext Java 확장자 디렉토리에 두십시오.

- **JAXP(Java API for XML Processing) 구현**. XML 파일을 오브젝트 그리드 런타임으로 전달하는 경우, 오브젝트 그리드 런타임에서 XML 파일을 구문 분석하려면 JAXP 구현이 필요합니다. 오브젝트 그리드에서는 XML 스키마 정의 구문 유효성 검증을 사용하므로 스키마 유효성 검증을 지원하는 구현이 필요합니다. Apache Xerces 제품은 스키마 유효성 검증을 지원하는 구현의 한 예입니다.
- **JSSE(Java Secure Socket Extension) 구현**. 클라이언트 런타임을 사용하는 경우, JSSE 구현이 필요합니다. 보안을 사용하여 실행하는 경우 사용되는 JSSE 구현이 오브젝트 그리드 서버 Java Development Kit(JDK) 구현과 호환되는지 확인하십시오.

로컬 또는 클라이언트 오브젝트 그리드 런타임이 J2SE 버전 1.3을 사용하는 J2EE 버전 1.3 호환 환경에 포함된 경우, 필요한 모든 스펙 구현이 J2EE 버전 1.3의 일부로서 필요하므로 이러한 요구사항 모두를 충족합니다.

제 7 장 System Management 개요

WebSphere Extended Deployment 버전 6.0.1 릴리스에서 오브젝트 그리드는 사용자가 오브젝트 그리드 환경을 모니터하고 관리할 수 있도록 하는 System Management 하부 구조를 제공합니다. System Management 아키텍처는 3티어 접근 방식입니다. 사용자 클라이언트는 관리 게이트웨이 서버에 연결하여 오브젝트 그리드 클라이언트를 오브젝트 그리드 클러스터에 연결합니다.

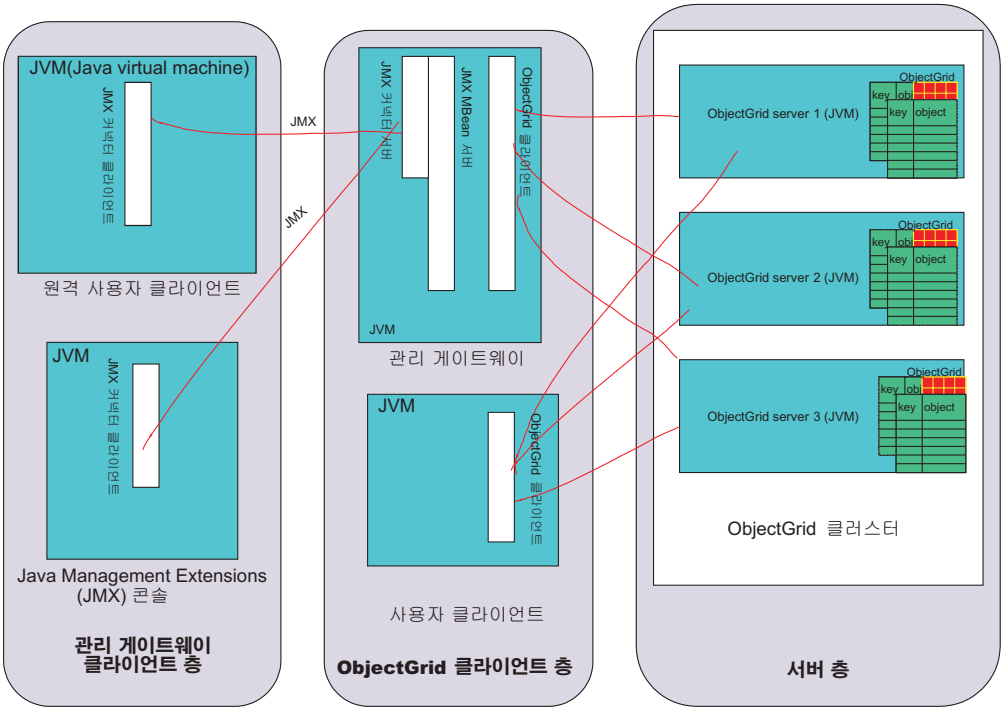


그림 14. System Management 다이어그램

*관리 게이트웨이 클라이언트 티어*는 JMX(Java Management Extensions)를 사용하여 관리 게이트웨이 서버에 연결하는 프로그램을 포함합니다. 써드파티 JMX 콘솔 뿐만 아니라 MX4J API를 사용하는 클라이언트 프로그램도 포함됩니다. *오브젝트 그리드 클라이언트 티어*는 관리 게이트웨이 서버로 구성됩니다. 관리 게이트웨이는 관리 게이트웨이 클라이언트 층의 경우 서버 역할을 수행하고 *서버 티어*에서 작동하고 있는 오브젝트 그리드 클러스터에 대해서는 클라이언트 역할을 수행합니다. 또한 오브젝트 그리드 클라이언트 프로그램은 사용자가 JMX를 포함하지 않으려는 경우 관리 게이트웨이 서버가 호출하는 동일한 API를 호출할 수 있습니다. 결과적으로 서버 층은 오브젝트 그리드 클러스터로 구성됩니다.

관리 게이트웨이는 일련의 MBean(Managed Bean)을 포함하고 JMX를 사용하여 오브젝트 그리드 환경을 관리 및 모니터하며 MX4J 오픈 소스 프로젝트를 통해 구현됩니다. MX4J는 오브젝트 그리드와 함께 제공됩니다.

JMX 환경을 관리하는 데 사용할 수 있는 다양한 JMX 콘솔을 이용할 수 있도록 오브젝트 그리드 JMX 및 MBean 관리 모델이 작성되었습니다. 선택한 JMX 콘솔을 사용하여 대시보드를 함께 배치할 수 있습니다. 콘솔은 ManagementGateway JVM(Java Virtual Machine)에서 실행 중인 MBean에 연결될 수 있고 대시보드는 이 MBean을 사용하여 어셈블될 수 있습니다. 콘솔은 숫자 및 문자열 값으로 된 그래픽 히스토리 또는 차트를 제공합니다.

System Management 명령을 실행하는 두 가지 옵션이 있습니다.

- ObjectGridAdministrator 인터페이스를 사용하는 곳에서 현재 클라이언트-서버 하부 구조를 통해 모든 명령을 호출합니다.
- ObjectGrid MBean을 ObjectGridAdministrator에 대한 랩퍼로 작동시키고 JMX를 사용하여 이러한 동일 명령을 호출합니다.

ManagementGateway 프로세스 시작

클러스터(또는 단일 서버)가 시작된 후 ManagementGateway 프로세스를 시작할 수 있습니다. ManagementGateway는 사용자 클라이언트 요청에 대해 서버 역할을 수행하고 연결된 클러스터에 대해 오브젝트 그리드 클라이언트 역할을 수행합니다.

옵션

다음은 ManagementGateway 프로세스에 전달될 수 있는 옵션의 목록입니다.

- **connectorPort**(필수) - JMX 커넥터의 포트 번호를 지정합니다.
- **clusterHost**(필수) - 오브젝트 그리드 클러스터에 있는 서버 중 하나의 호스트 이름을 지정합니다.
- **clusterPort**(필수) - 오브젝트 그리드 클러스터에 있는 서버 중 하나의 클라이언트 액세스 포트를 지정합니다.
- **clusterName**(필수) - 오브젝트 그리드 클러스터의 이름을 지정합니다.
- **traceEnabled** - ManagementGateway 프로세스에 추적이 사용 가능한지 여부를 지정합니다.
- **traceSpec** - ManagementGateway의 추적 스펙을 표시합니다.
- **traceFile** - 추적 출력이 인쇄될 파일을 지정합니다.
- **sslEnabled** - ManagementGateway에서 SSL이 사용 가능한지 여부를 지정합니다.
- **csConfig** - 보안 ManagementGateway에 ClientSecurityConfiguration 오브젝트를 지정합니다.

- **refreshInterval** - 관리 게이트웨이가 MBean 속성을 새로 고치는 간격(시간)을 지정합니다.

ManagementGateway 인터페이스

MBeans를 사용 가능하게 하려면 ManagementGateway 프로세스를 시작해야 합니다. ManagementGateway 인터페이스는 ManagementGateway를 시작할 때 전달될 수 있는 옵션을 표시합니다.

```
public interface ManagementGateway {
/**
* Start the JMX MBean connector server
*/
void startConnector();
/**
* Stop the JMX MBean connector server
*/
void stopConnector();
/**
* @param JMX connector port
*/
void setConnectorPort(int port);
/**
* @return JMX connector port
*/
int getConnectorPort();
/**
* @param a {@link com.ibm.websphere.objectgrid.security.config.
ClientSecurityConfiguration} object.
*/
void setCsConfig(ClientSecurityConfiguration csConfig);
/**
* @return a {@link com.ibm.websphere.objectgrid.security.config.
ClientSecurityConfiguration} object.
*/
ClientSecurityConfiguration getCsConfig();
/**
* @param port of server to which gateway client connects
*/
void setPort(String port);
/**
* @return port of server to which gateway client connects
*/
String getPort();
/**
* @param host of server to which gateway client connects
*/
void setHost(String host);
/**
* @return host of server to which gateway client connects
*/
String getHost();
/**
* @param boolean true if SSL enabled on gateway
*/
void setSSLEnabled(boolean sslEnabled);
```

```
/**
* @return boolean true if SSL enabled on gateway
*/
boolean getSSLEnabled();
/**
* @param cluster to which gateway client connects
*/
void setClusterName(String clusterName);
/**
* @return cluster to which gateway client connects
*/
String getClusterName();
/**
* @param true if trace is enabled on gateway
*/
void setTraceEnabled(boolean traceEnabled);
/**
* @return true if trace is enabled on gateway
*/
boolean getTraceEnabled();
/**
* @param trace specification on gateway
*/
void setTraceSpec(String traceSpec);
/**
* @return trace specification on gateway
*/
String getTraceSpec();
/**
* @param trace output file for gateway trace
*/
void setTraceFile(String traceFile);
/**
* @return trace output file for gateway trace
*/
String getTraceFile();
/**
* @param interval (in seconds) to refresh cluster MBean attributes
*/
void setRefreshInterval(int refreshInterval);
/**
* @return interval (in seconds) to refresh cluster MBean attributes
*/
int getRefreshInterval();
}
```

ManagementGateway 프로세스를 시작하는 옵션

프로그램 방식으로 ManagementGatewayFactory 사용

다음은 이 옵션을 사용하는 샘플 코드입니다.

```
ManagementGateway gw = ManagementGatewayFactory.getManagementGateway();
gw.setConnectorPort(1099);
gw.setClusterName("cluster1");
gw.setHost("localhost");
gw.setPort("12503");
gw.startConnector();
```

이 코드는 연결을 시도 중인 오브젝트 그리드 클러스터가 시작된 후 실행되는 사용자 프로그램에 있어야 합니다.

명령행에서 startManagementGateway 일괄처리 파일

예를 들면 다음과 같습니다.

```
startManagementGateway.bat -connectorPort 1099 -clusterName cluster1
-clusterHost localhost -clusterPort 12503
```

startManagementGateway 스크립트에 대한 자세한 정보는 95 페이지의 『관리 게이트웨이 서버 시작』을 참조하십시오.

ManagementGateway는 JMX를 호출하려는 클라이언트 프로세스에 대한 서버 역할을 수행하지만 사용자가 연결할 클러스터에 대해 오브젝트 그리드 클라이언트 역할도 수행합니다. ManagementGateway가 시작된 후 클러스터에 연결이 설정되고 JMX 커넥터 서비스가 사용 가능하게 됩니다. 그런 다음 MX4J 또는 J2SE(Java 2 Platform, Standard Edition) 버전 5 API를 통해 JMX 커넥터 서비스에 액세스할 수 있습니다.

예제

다음은 커넥터 포트 1을 사용하는 ManagementGateway를 통해 Server1 서버에서 MapStatsModule을 가져오는 방법을 보여주는 샘플 코드입니다.

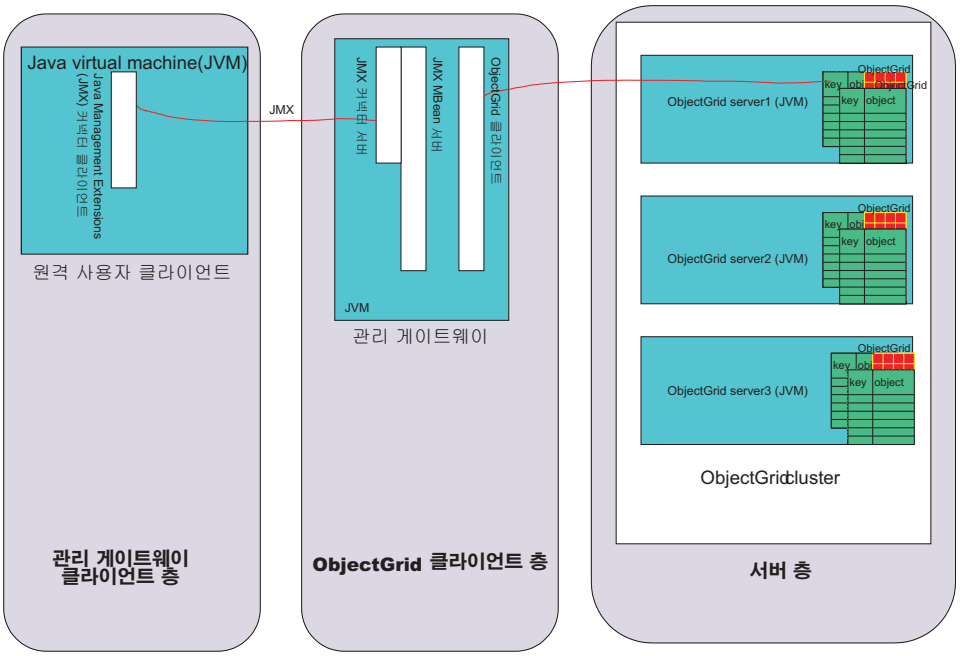


그림 15. server1 서버에서 맵 통계 가져오기

이전 다이어그램의 원격 사용자 클라이언트 섹션에서 실행되는 사용자 프로그램에 있는 다음 코드를 실행하십시오.

```
JMXServiceURL url = new
JMXServiceURL("service:jmx:rmi://host/jndi/rmi://localhost:1099/jmxconnector");
JMXConnector c = JMXConnectorFactory.connect(url);
MBeanServerConnection mbsc = c.getMBeanServerConnection();
Iterator it = mbsc
.queryMBeans(new ObjectName
("ManagementMap:type=ObjectGrid,OG=OG1,Map=map1,S=server1"), null)
.iterator();
ObjectInstance oi = (ObjectInstance) it.next();
ObjectName mapMBean = oi.getObjectName();
MapStatsModule stats = (MapStatsModule) mbsc.invoke(
mapMBean,
"retrieveStatsModule",
new Object[] { },
new String[] { });
```

ManagementGateway를 통해 server1 서버를 중지하려면 다음을 수행하십시오.

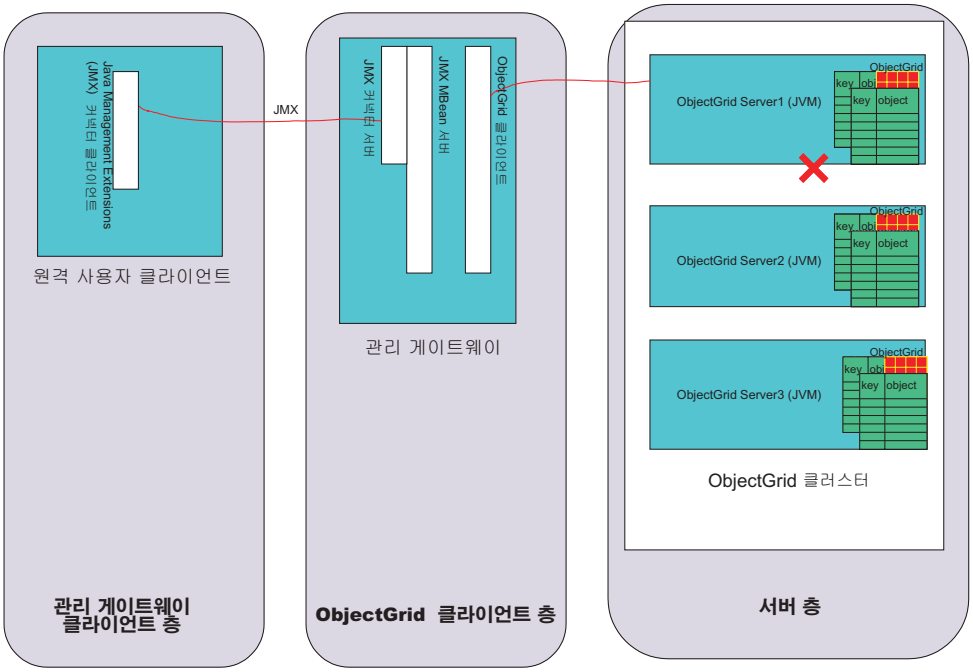


그림 16. server1 서버 중지

이전 다이어그램의 원격 사용자 클라이언트에서 실행되는 사용자 프로그램에서 다음 코드를 실행하십시오.

```
JMXServiceURL url = new JMXServiceURL(
"service:jmx:rmi://host/jndi/rmi://localhost:1099/jmxconnector");
JMXConnector c = JMXConnectorFactory.connect(url);
MBeanServerConnection mbsc = c.getMBeanServerConnection();
Iterator it = mbsc
.queryMBeans(new ObjectName("ManagementServer:type=ObjectGrid,S=Server1"), null)
.iterator();
ObjectInstance oi = (ObjectInstance) it.next();
ObjectName server1MBean = oi.getObjectName();
boolean stop = ((Boolean) mbsc.invoke(
```

```
server1MBean,
"stopServer",
new Object[] { },
new String[] { })).booleanValue();
```

이전 코드 샘플을 실행한 후 server1 서버가 중지됩니다. server1 서버는 중지된 후 ManagementGateway로 다시 시작될 수 없습니다. 명령행을 사용하여 서버를 다시 시작할 수 있습니다. 자세한 정보는 94 페이지의 『오브젝트 그리드 서버 중지』를 참조하십시오.

오브젝트 그리드 관리 Bean(MBean)

오브젝트 그리드 환경에는 다섯 가지 유형의 MBeans가 있습니다. 각 MBean은 맵, 오브젝트 그리드, 서버, 복제 그룹 또는 복제 그룹 구성원과 같은 특정 엔티티를 나타내며 속성 및 조작을 보유합니다.

오브젝트 그리드의 각 MBean은 속성 값을 나타내는 getxxx 메소드를 보유합니다. 이 getxxx 메소드는 사용자 프로그램에서 직접 호출될 수 없습니다. JMX(Java Management eXtensions) 스펙이 속성을 조작과 다르게 취급하기 때문입니다. 속성은 써드파티 JMX 콘솔을 통해 볼 수 있고 조작은 사용자 프로그램 또는 써드파티 JMX 콘솔을 통해 수행될 수 있습니다.

MapMbean Mbean

MapMBean을 사용하여 사용자는 클러스터에 정의된 각 맵의 통계를 모니터할 수 있습니다. 각 맵은 맵과 연관된 다음 통계를 보유합니다.

- 일괄처리 갱신 시간(min/max/mean/total)
- 계수
- 적중률

또한 여러 서버에서 맵을 파티션할 수 있으므로 특정 서버 또는 복제 그룹 구성원 범위의 맵 통계를 얻을 수 있습니다. 또한 전체 클러스터에 대한 통계를 맵핑할 수도 있습니다. 여러 가지 방법으로 MapMBean에 ObjectName을 지정할 수 있습니다.

- `"ManagementMap:type=ObjectGrid,OG=ObjectGridName,Map=MapName"`
- `"ManagementMap:type=ObjectGrid,OG=ObjectGridName,Map=MapName,`
  `S=ServerName"`
- `"ManagementMap:type=ObjectGrid,OG=ObjectGridName,Map=MapName,`
  `RG=ReplicationGroup,IDX=Index"`

OG1 오브젝트 그리드, Map1 맵과 함께 복제 그룹 RGI에 두 개의 서버인 server1과 server2가 있는 예제 구성을 보십시오. 또한 server1 서버가 1차이고 server2 서버가 복제본이라고 가정하십시오. 1차의 Map1 맵에 대한 통계를 가져오려면 다음 ObjectName 중 하나를 사용하십시오.

- `"ManagementMap:type=ObjectGrid,OG=OG1,Map=Map1,S=server1"`

- "ManagementMap:type=ObjectGrid,OG=OG1,Map=Map1,RG=RG1,IDX=0"

오브젝트 그리드 MBeans의 ObjectName에서 IDX=0일 때 이것은 복제 그룹의 1차를 나타내며 IDX=1에서 10까지는 복제 그룹의 복제본을 나타냅니다.

MapMBean 인터페이스 목록은 다음과 같습니다.

```
public interface MapMBean {
/**
* Operation to get MapStatsModule associated with the MBean.
*
* @return MapStatsModule
*/
MapStatsModule retrieveStatsModule();
/**
* Operation will only go to server to get StatsModule if the
* StatsModule is not cached in the ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* Map.
*
* @return name of map
*/
String getMapName();
/**
* ObjectGrid containing the map.
*
* @return name of the object grid
*/
String getObjectGridName();
/**
* Server name of the replication group member for the map.
*
* @return name of server of the replication group member
*/
String getServerName();
/**
* Name of replication group for the map.
*
* @return name of replication group
*/
String getReplicationGroup();
/**
* Index of replication group member for the map.
*
* @return index of replication group member
*/
int getIndex();
/**
* MapStatsModule attribute loaded up by the
* retrieveStatsModule call.
*
* @return String form of MapStatsModule
*/
String getMapStatsModule();
```

```
/**
* Map count attribute loaded up by the
* retrieveStatsModule call.
*
* @return number of entries in map
*/
long getMapCountStatistic();
/**
* Hit rate attribute loaded up by the
* retrieveStatsModule call.
*
* @return hit rate for map
*/
double getMapHitRateStatistic();
/**
* Mean batch update time attribute loaded up by the
* retrieveStatsModule call.
*
* @return mean batch update time for map
*/
double getMapBatchUpdateMeanTime();
/**
* Maximum batch update time attribute loaded up by the
* retrieveStatsModule call.
*
* @return maximum batch update time for map
*/
double getMapBatchUpdateMaxTime();
/**
* Minimum batch update time attribute loaded up by the
* retrieveStatsModule call.
*
* @return minimum batch update time for map
*/
double getMapBatchUpdateMinTime();
/**
* Total batch update time attribute loaded up by the
* retrieveStatsModule call.
*
* @return total batch update time for map
*/
double getMapBatchUpdateTotalTime();
}
```

ObjectGridMBean MBean

ObjectGridMBean MBean을 사용하여 사용자는 클러스터에 정의된 각 오브젝트 그리드의 모든 맵에 대한 통계를 모니터할 수 있습니다. 각 오브젝트 그리드는 오브젝트 그리드와 연관된 다음 통계를 보유합니다.

- 트랜잭션 시간(min/max/mean/total)
- 계수

또한 여러 서버에서 오브젝트 그리드를 파티션할 수 있으므로 특정 서버 또는 복제 그룹 구성원 범위의 오브젝트 그리드에 대한 통계를 얻을 수 있습니다. 또한 전체 클러스

터에 대한 오브젝트 그리드 통계를 가져올 수도 있습니다. 여러 가지 방법으로 ObjectGridMBean에 ObjectName을 지정할 수 있습니다.

- "ManagementObjectGrid:type=ObjectGrid,OG=ObjectGridName"
- "ManagementObjectGrid:type=ObjectGrid,OG=ObjectGridName, S=ServerName"
- "ManagementObjectGrid:type=ObjectGrid,OG=ObjectGridName, RG=ReplicationGroup,IDX=Index"

다음은 ObjectGridMbean 인터페이스 목록입니다.

```
public interface ObjectGridMBean {
/**
* Operation to get OGStatsModule associated with the MBean.
*
* @return OGStatsModule
*/
OGStatsModule retrieveStatsModule();
/**
* Operation will only go to server to get StatsModule if the
* StatsModule is not cached in the ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* ObjectGrid.
*
* @return name of the object grid
*/
String getObjectGridName();
/**
* Server name of the replication group member for the ObjectGrid.
*
* @return name of server of the replication group member
*/
String getServerName();
/**
* Name of replication group for the ObjectGrid.
*
* @return name of replication group
*/
String getReplicationGroup();
/**
* Index of replication group member for the ObjectGrid.
*
* @return index of replication group member
*/
int getIndex();
/**
* OGStatsModule attribute loaded up by the retrieveStatsModule call.
*
* @return String form of OGStatsModule
*/
String getOGStatsModule();
/**
* ObjectGrid count attribute loaded up by the retrieveStatsModule call.
*
* @return number of transactions
*/
long getOGCount();
```

```
/**
* Maximum transaction time attribute loaded up by the retrieveStatsModule call.
*
* @return maximum transaction time for the ObjectGrid
*/
long getOGMaxTranTime();
/**
* Minimum transaction time attribute loaded up by the retrieveStatsModule call.
*
* @return minimum transaction time for the ObjectGrid
*/
long getOGMinTranTime();
/**
* Mean transaction time attribute loaded up by the retrieveStatsModule call.
*
* @return mean transaction time for the ObjectGrid
*/
double getOGMeanTranTime();
/**
* Total transaction time attribute loaded up by the retrieveStatsModule call.
*
* @return total transaction time for the ObjectGrid
*/
long getOGTotalTranTime();
}
```

ServerMBean MBean

사용자는 ServerMBean Mbean을 사용하여 클러스터의 서버에서 조작을 수행할 수 있습니다. 여러 가지 방법으로 ServerMBean에 ObjectName을 지정할 수 있습니다.

- ″ManagementServer:type=ObjectGrid,S=ServerName″

ServerMBean 인터페이스 목록은 다음과 같습니다.

```
public interface ServerMBean {
/**
* Operation to load the replication status for the server.
*
*/
void retrieveReplicationStatus();
/**
* Return the name of the server.
*
* @return server name
*/
String getServerName();
/**
* Operation to get the status of the server.
*
* @return status of server (true if running, false if not)
*/
boolean retrieveServerStatus();
/**
* Operation to stop the server.
*
* @return true if server was stopped, false if not
*/
boolean stopServer();
```

```
/**
* Operation to force the server stop.
*
* @return true if server was stopped, false if not
*/
boolean forceStopServer();
/**
* Operation to stop the cluster the server is a part of.
*
* @param determines if servers are stopped with force
* @return true if cluster was stopped, false if not
*/
boolean stopCluster(Boolean force);
/**
* Operation to modify the trace spec for all servers in
* the cluster the server is a part of.
*
* @param trace specification
*/
void modifyClusterTraceSpec(String spec);
/**
* Operation to modify the trace spec for the server.
*
* @param trace specification
*/
void modifyServerTraceSpec(String spec);
}
```

ReplicationGroupMBean Mbean

ReplicationGroupMBean을 사용하여 1차인 서버와 최대 10개의 복제본을 포함하여 특정 복제 그룹과 연관된 모든 복제 그룹 구성원의 상태를 모니터할 수 있습니다. ReplicationGroupMBean에 ObjectName을 지정할 수 있습니다.

- ″ManagementReplicationGroup:type=ObjectGrid,RG=ReplicationGridName″

ReplicationGroupMBean 인터페이스 목록은 다음과 같습니다.

```
public interface ReplicationGroupMBean {
/**
* Operation to load up the status of the replication group attributes.
*
*/
String[] retrieveReplicationGroupStatus();
/**
* ReplicationGroupName attribute.
*
* @return name of the ReplicationGroup
*/
String getReplicationGroupName();
/**
* Primary attribute.
*
* @return name of the Primary
*/
String getPrimary();
/**
```

```
* Replica1 attribute.
*
* @return server name of Replica1
*/
String getReplica1();
/**
* Replica2 attribute.
*
* @return server name of Replica2
*/
String getReplica2();
/**
* Replica3 attribute.
*
* @return server name of Replica3
*/
String getReplica3();
/**
* Replica4 attribute.
*
* @return server name of Replica4
*/
String getReplica4();
/**
* Replica5 attribute.
*
* @return server name of Replica5
*/
String getReplica5();
/**
* Replica6 attribute.
*
* @return server name of Replica6
*/
String getReplica6();
/**
* Replica7 attribute.
*
* @return server name of Replica7
*/
String getReplica7();
/**
* Replica8 attribute.
*
* @return server name of Replica8
*/
String getReplica8();
/**
* Replica9 attribute.
*
* @return server name of Replica9
*/
String getReplica9();
/**
* Replica10 attribute.
*
* @return server name of Replica10
*/
```

```
String getReplica10();
/**
* All replicas for this replication group comma delimited
*
* @return server names of all replicas
*/
String getReplicas();
}
```

ReplicationGroupMemberMBean Mbean

ReplicationGroupMemberMBean을 사용하여 복제 그룹 구성원에 대한 다음 통계를 모니터할 수 있습니다.

- 복제 그룹 구성원의 상태. 1차 및 복제본 구성원을 모니터할 수 있습니다.
- 복제본 가중치 비율. 이 통계는 복제본인 복제 그룹 구성원에만 적용됩니다. 이 비율은 복제본의 맵이 1차의 맵과 근사값으로 동기화되는 정도입니다. 이 비율이 높을수록 복제본은 1차의 최신 정보에 더욱 근접하게 보유하게 됩니다.

다음과 같은 방법으로 ReplicationGroupMemberMBean에 ObjectName을 지정할 수 있습니다.

- `"ManagementReplicationGroupMember:type=ObjectGrid, RG=ReplicationGridName,S=ServerName"`
- `"ManagementReplicationGroupMember:type=ObjectGrid, RG=ReplicationGridName,IDX=Index"`

IDX=0을 지정하면 복제 그룹의 1차가 리턴되고 IDX=1에서 10까지는 복제본입니다. ReplicationGroupMBean 인터페이스 목록은 다음과 같습니다.

```
public interface ReplicationGroupMemberMBean {
/**
* Operation to load up the status of the replication group member attributes.
*
*/
void retrieveReplicationGroupMemberStatus();
/**
* Operation to load up the status of the replication group member
* attributes.
* Will use the cache as opposed to the retrieveReplicationGroupMemberStatus
* method which will go to the server to get status.
*
*/
void refreshReplicationGroupMemberStatus();
/**
* ReplicationGroupName attribute.
*
* @return name of the ReplicationGroup this member belongs to
*/
String getReplicationGroupName();
/**
* Status of the ReplicationGroupMember: primary/replica/standby.
*
* @return status of the ReplicationGroupMember
*/
String getStatus();
```

```
/**
* Statistic representing the percentage how close a replica is
* to being up to date with the primary maps.
*
* @return Replica statistic of the ReplicationGroupMember
*/
double getReplicaWeightRatio();
/**
* Name of server on which this ReplicationGroupMember resides.
*
* @return server name
*/
String getServerName();
/**
* Index of ReplicationGroupMember.
*
* @return index of replica
*/
int getIndex();
}
```

제 8 장 명령행 지원

명령행 스크립트를 사용하여 오브젝트 그리드 서버를 관리하십시오.

혼합 서버 환경 설치의 /ObjectGrid/bin 디렉토리에 일련의 스크립트 파일이 제공되어 있습니다. 이 스크립트는 오브젝트 그리드 서버를 시작하거나 중지하고 관리 게이트웨이 서버를 시작하며 특성 파일에서 암호를 인코드하는 데 사용될 수 있습니다. 스크립트 사용을 시도하기 전에 JAVA_HOME 환경 변수가 설정되었는지와 이 변수의 값이 오브젝트 그리드가 지원하는 Java 버전인지 확인하십시오. setupCmdLine.bat|sh 파일에서 JAVA_HOME을 갱신하여 환경 변수를 전체적으로 변경하지 않을 경우에는 해당 Java 버전을 지시하도록 하십시오.

명령행 스크립트에 대한 자세한 정보는 다음 주제를 참조하십시오.

- 『오브젝트 그리드 서버 시작』
- 94 페이지의 『오브젝트 그리드 서버 중지』
- 95 페이지의 『관리 게이트웨이 서버 시작』
- 97 페이지의 『암호 인코딩』

오브젝트 그리드 서버 시작

startOgServer 스크립트는 오브젝트 그리드 서버를 시작하기 위해 제공됩니다.

사용법

Windows 시스템에서 서버를 시작하려면 startOgServer.bat 파일을 사용하십시오. Linux 및 Unix 플랫폼에서 오브젝트 그리드 서버를 시작하려면 startOgServer.sh 파일을 사용하십시오.

XML 파일 사용

오브젝트 그리드 서버를 성공적으로 시작하려면 유효한 오브젝트 그리드 XML이 유효한 클러스터 XML 파일과 쌍을 이루어야 합니다. 일반 파일 이름 또는 URL(Uniform Resource Locator)을 사용하여 XML 파일을 startOgServer 스크립트에 전달할 수 있습니다. URL 옵션은 파일 프로토콜 외 다른 프로토콜(예: http, ftp 또는 jarfile)을 허용합니다.

XML 파일을 사용하여 서버를 시작하는 startOgServer 스크립트 인수는 다음과 같습니다.

```
startOgServer.bat <server> -objectgridFile <XML file> | -objectgridUrl
<XML file URL> -clusterFile <XML file> | -clusterUrl <XML file URL> [options]
```

예제

다음은 server1 오브젝트 그리드 서버를 시작하는 몇 가지 예입니다. 다음 예는 startOgServer.bat 파일을 사용합니다.

```
startOgServer.bat server1 -objectgridFile c:¥objectgrid¥xml¥university.xml
-clusterFile c:¥objectgrid¥xml¥universityCluster3Servers.xml
startOgServer.bat server1 -objectgridFile ..¥xml¥university.xml
-clusterUrl file:///c:/objectgrid/xml/universityCluster3Servers.xml
startOgServer.bat server1 -objectgridUrl file:///c:/objectgrid/xml/university.xml
-clusterFile ..¥xml¥universityCluster3Servers.xml
startOgServer.bat server1 -objectgridUrl file:///c:/objectgrid/xml/university.xml
-clusterUrl file:///c:/objectgrid/xml/universityCluster3Servers.xml
```

예에서는 universityCluster3Servers.xml 파일을 사용합니다. server1 서버가 시작할 서버로 지정되었으므로 universityCluster3Servers.xml 파일은 server1 이름을 가지는 serverDefinition 값을 보유해야 합니다.

universityCluster3Servers.xml 파일이 표시됩니다. server1 serverDefinition을 검사하고 호스트가 lion.ibm.com인지 살펴보십시오. server1 서버는 lion.ibm.com 호스트에서 시작되어야 합니다. 또한 이 파일은 server2 및 server3 서버를 정의합니다. 이 서버들은 tiger.ibm.com 및 bear.ibm.com 호스트에서 각각 시작되어야 합니다.

universityCluster3Servers.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
<serverDefinition name="server3" host="bear.ibm.com" clientAccessPort="12505"
peerAccessPort="12506" />
</cluster>
<objectgridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
```

```
<replicationGroupMember serverRef="server2" priority="2" />
<replicationGroupMember serverRef="server3" priority="3" />
</replicationGroup>
</clusterConfig>
```

부트스트랩핑

클러스터의 한 오브젝트 그리드 서버가 사용 가능하게 되면 클러스터의 다른 서버는 사용 가능한 서버에 부트스트랩할 수 있습니다. startOgServer 스크립트는 서버에 부트스트랩하기 위해 이미 사용 가능한 서버의 호스트 및 클라이언트 액세스 포트에 제공되어야 합니다.

클러스터의 첫 번째 서버는 XML로 실행되므로 이 서버는 이미 클러스터의 모든 서버에 대한 구성 정보를 보유합니다. 부트스트랩하게 되면 실행 서버가 사용 가능한 서버에 연결되고 구성을 다운로드할 수 있습니다.

다음은 사용 가능한 서버에 부트스트랩하여 서버를 시작할 수 있는 startOgServer 스크립트 인수입니다.

```
startOgServer.bat <server> -bootstrap <host:port,host:port> [options]
```

다음은 다른 서버로 부트스트랩핑하여 서버를 시작하는 몇 가지 예제입니다. 첫 번째 예의 경우 universityCluster3Servers.xml 파일의 server1 서버가 XML 파일을 사용하여 시작되었고 사용 가능하다고 가정하십시오. 이 예는 server1 서버에 부트스트랩하여 server2 서버를 시작하는 방법을 보여줍니다.

```
startOgServer.bat server2 -bootstrap lion.ibm.com:12501
```

다음 예의 경우 server2 서버가 성공적으로 시작되었고 server1 서버가 사용 불가능하게 되었다고 가정하십시오. 다른 서버에 부트스트랩할 때 쉼표로 구분된 호스트:포트 결합 목록을 사용할 수 있습니다. 사용 가능한 서버를 찾을 때까지 목록의 각 호스트 및 포트에 접속을 시도합니다. 다음 예에서 server3은 server1 서버의 호스트 및 포트에 접속을 시도합니다. 그러나, 이 시나리오에서는 server1 서버가 사용 불가능하므로 연결이 실패합니다. 목록에서 다음 항목을 가져오면 server3 서버가 server2 서버의 호스트 및 포트에 부트스트랩을 시도합니다. server2 서버가 사용 가능하므로 이 부트스트랩 시도는 성공합니다.

```
startOgServer.bat server3 -bootstrap lion.ibm.com:12501,tiger.ibm.com:12503
```

선택적 인수

startOgServer 스크립트에 전달될 수 있는 몇 가지 선택적 인수가 있습니다. 유효한 startOgServer 인수는 다음과 같습니다.

옵션:

- -traceSpec <추적 스펙>
- -traceFile <추적 파일>

- -serverSecurityFile <서버 보안 특성 파일>
- -timeout <초(seconds)>
- -script <스크립트 파일 이름>
- -jvmArgs <JVM 인수>

-traceSpec

-traceSpec 인수는 서버 시작 중 거의 즉시 적용되는 추적 스펙을 설정하는 데 사용될 수 있습니다. 표준 서버 시작 중에 클러스터 XML 파일 또는 부트스트랩된 구성에서 추적 스펙을 읽을 수 있을 때까지 추적 스펙은 설정되지 않습니다. 서버 시작 중에 문제점이 발생할 경우 더 일찍 추적 스펙을 설정하는 것이 도움이 될 수 있습니다.

다음은 -traceSpec 옵션을 설정하는 방법에 대한 예입니다.

```
startOgServer.bat server1 -objectgridFile c:₩objectgrid₩xml₩university.xml
-clusterFile c:₩objectgrid₩xml₩universityCluster3Servers.xml
-traceSpec ObjectGrid=all=enabled
```

-traceFile

-traceFile 인수는 서버 시작 중에 출력되는 추적의 위치를 지정하는 데 사용될 수 있습니다. 이 서버의 구성이 읽혀지면 클러스터 XML 파일에서 지정한 이 서버의 추적 설정이 적용됩니다.

다음은 -traceFile 옵션을 설정하는 방법에 대한 예입니다.

```
startOgServer.bat server2 -bootstrap lion.ibm.com:12501 -traceFile
c:₩objectgrid₩trace.log
```

-serverSecurityFile

-serverSecurityFile 인수는 보안 특성 파일을 서버에 전달하는 데 사용될 수 있습니다. 보안이 서버에서 사용 가능할 때 이 옵션은 필수입니다. 다음은 -serverSecurityFile 옵션을 설정하는 방법에 대한 예입니다.

```
startOgServer.bat server1 -objectgridUrl file:///c:/objectgrid/xml/
university.xml
-clusterFile ..₩xml₩universityCluster3Servers.xml
-serverSecurityFile c:₩objectgrid₩props₩serverSecurity.props
```

-timeout

-timeout 인수는 서버의 실행이 중단되기 전까지 경과될 수 있는 시간(초)을 지정하는 데 사용될 수 있습니다. 기본적으로 서버는 서버가 실행된 시간으로부터 90초 간 사용 가능하게 될 수 있습니다. 특정 시나리오의 경우 이 시간이 너무 짧으면 -timeout 인수를 사용하여 더 적절한 값으로 설정하십시오. 제한 시간 인수를 사용하는 방법에 대한 예는 다음과 같습니다.

```
startOgServer.bat server1 -objectgridFile ..₩xml₩university.xml
-clusterUrl file:///c:/objectgrid/xml/universityCluster3Servers.xml
-timeout 120
```

-script

-script 인수는 오브젝트 그리드 서버 프로세스를 실행하고 해당 프로세스의 출력을 현재 명령 프롬프트에 보관하는 스크립트를 작성하는 데 사용될 수 있습니다. 표준 환경에서 오브젝트 그리드 서버가 실행될 때 startOgServer 스크립트는 서버가 사용 가능해질 때까지 서버 프로세스의 출력을 명령 프롬프트에 표시합니다. 서버가 사용 가능하게 되면 startOgServer는 서버 프로세스의 출력 표시를 중지하고 종료됩니다. 일부의 경우 현재 명령 프롬프트에 출력하는 서버 프로세스를 실행하기를 원할 수 있습니다.

스크립트에 파일 이름을 지정할 때 파일의 경로를 제공하지 마십시오. 파일은 OBJECTGRID_HOME 경로의 bin 디렉토리에 있습니다. 파일의 이름을 제공하십시오. 작성된 스크립트 파일은 startOgServer 스크립트에 전달된 인수를 포함하므로 작성된 스크립트를 실행할 때 이 동일한 인수를 제공할 필요가 없습니다.

다음은 -script 옵션을 사용하는 방법에 대한 예입니다.

```
startOgServer.bat server1 -objectgridUrl
file:///c:/objectgrid/xml/university.xml
-clusterUrl file:///c:/objectgrid/xml/universityCluster3Servers.xml
-script universityClusterServer1.bat
```

이 예에서는 OBJECTGRID_HOME/bin 디렉토리에 universityClusterServer1.bat 스크립트를 작성합니다. 새로 작성된 스크립트를 실행하려면 명령 프롬프트에서 적절한 디렉토리로 이동하고 스크립트의 이름을 입력한 다음 **Enter** 키를 누르십시오.

-jvmArgs

-jvmArgs 인수는 실행 중인 오브젝트 그리드 서버 JVM(Java Virtual Machine)으로 인수를 전송하는 데 사용될 수 있습니다. 일반적으로 JVM으로 전달될 수 있는 인수는 -jvmArgs 인수를 사용하여 서버에 전달될 수 있습니다.

-jvmArgs 인수는 startOgServer 스크립트에 인수로 지정된 마지막 오브젝트 그리드의 선택적 인수여야 합니다. -jvmArgs 인수 뒤에 오는 것은 모두 서버 JVM에 JVM 인수로 전달됩니다. 다음은 -jvmArgs 인수를 설정하는 방법에 대한 예입니다.

```
startOgServer.bat server2 -bootstrap lion.ibm.com:12501
-jvmArgs -Xms768M -DmyProp=value1
```

-jvmArgs 인수가 -classpath 또는 -cp JVM 인수를 포함하는 경우 지정된 클래스 경로가 오브젝트 그리드 클래스 경로에 추가됩니다. 다음은 -jvmArgs 인수를 사용하여 오브젝트 그리드 서버를 실행하는 데 사용되는 클래스 경로에 Xerces JAR(Java archive) 파일을 포함하는 예입니다.

```
startOgServer.bat server2 -bootstrap lion.ibm.com:12501 -jvmArgs -cp
C:₩xerces2_7_1₩xml-apis.jar;c:₩xerces2_7_1₩xercesImpl.jar
```

오브젝트 그리드 서버 중지

오브젝트 그리드 서버를 중지하려면 stopOgServer 스크립트를 사용하십시오.

사용법

Windows 시스템에서 서버를 중지하려면 stopOgServer.bat 파일을 사용하십시오. Linux 및 Unix 플랫폼에서 오브젝트 그리드 서버를 중지하려면 stopOgServer.sh 파일을 사용하십시오. stopOgServer 스크립트는 클러스터에서 사용 가능한 서버에 연결하여 서버를 중지할 수 있는 클라이언트를 작성합니다. 이 스크립트의 동작은 사용 가능한 서버에 부트스트랩하여 다른 서버를 시작하는 것과 유사합니다. 다음은 서버를 중지하는 stopOgServer 스크립트 인수입니다.

```
stopOgServer.bat <server> -bootstrap <host:port,host:port> [options]
```

예제

다음은 다양한 오브젝트 그리드 서버를 중지하는 몇 가지 예입니다. 이 예에서는 stopOgServer.bat 파일을 사용합니다. 이 예제의 경우 89 페이지의 『오브젝트 그리드 서버 시작』의 universityCluster3Servers.xml 파일에서 정의하는 대로 server1, server2 및 server3의 세 개의 서버가 작동되고 실행 중이라고 가정하십시오.

첫 번째 예제에서는 서버의 호스트 및 클라이언트 액세스 포트에 부트스트랩하여 server1 서버를 중지합니다.

```
stopOgServer.bat server1 -bootstrap lion.ibm.com:12501
```

server1 서버가 성공적으로 중지되었다고 가정하십시오. 다음 예에서는 먼저 server1 서버에 부트스트랩을 시도하여 server2를 중지합니다. server1 서버가 이미 중지되었기 때문에 부트스트랩은 실패합니다. 목록에서 다음 호스트 및 포트는 server3 서버에 속합니다. server3 서버가 사용 가능하므로 server3 서버에 부트스트랩은 성공하고 server2는 중지됩니다.

```
stopOgServer.bat server2 -bootstrap lion.ibm.com:12501,bear.ibm.com:12505
```

선택적 인수

stopOgServer 스크립트에 전달될 수 있는 몇 가지 선택적 인수가 있습니다. 이 섹션에서는 이 선택적 인수 각각을 사용하는 방법을 보여줍니다. 다음은 유효한 stopOgServer 인수와 선택적 인수입니다.

```
stopOgServer.bat <server> -bootstrap <host:port,host:port> [options]
```

옵션:

- -traceSpec <추적 스펙>
- -traceFile <추적 파일 >
- -clientSecurityFile <클라이언트 보안 특성 파일>

-traceSpec

-traceSpec 인수는 오브젝트 그리드 서버 중지를 시도하는 클라이언트에 추적 스펙을 설정하는 데 사용될 수 있습니다. 다음은 -traceSpec 인수를 설정하는 방법에 대한 예입니다.

```
stopOgServer.bat server1 -bootstrap lion.ibm.com:12501 -traceSpec
ObjectGrid=all=enabled
```

-traceFile

-traceFile 인수는 서버 종료 중에 출력되는 클라이언트 추적의 위치를 지정하는 데 사용될 수 있습니다. 다음은 traceFile 인수를 설정하는 방법에 대한 예입니다.

```
stopOgServer.bat server2 -bootstrap lion.ibm.com:12501,bear.ibm.com:12505
-traceFile c:¥objectgrid¥trace.log
```

-clientSecurityFile

-clientSecurityFile 인수는 클라이언트에 보안 특성 파일을 전달하는 데 사용할 수 있습니다. 보안이 사용 가능한 서버로 연결을 시도할 때 이 인수는 필수입니다.

다음은 -clientSecurityFile 인수를 설정하는 방법에 대한 예입니다.

```
stopOgServer.bat server1 -bootstrap lion.ibm.com:12501 -clientSecurityFile
c:¥objectgrid¥props¥clientSecurity.props
```

관리 게이트웨이 서버 시작

JMX(Java Management eXtennsions)를 사용하여 오브젝트 그리드 클러스터를 모니터 및 관리하려면 명령행 스크립트를 통해 또는 프로그램 방식으로 관리 게이트웨이를 시작해야 합니다.

목적

명령행을 통해 관리 게이트웨이를 시작하려면 startManagementGateway 스크립트를 사용하십시오. Windows 시스템에서 ManagementGateway 서버를 시작하려면 `startManagementGateway.bat` 파일을 사용하십시오. Linux 및 Unix 플랫폼에서 ManagementGateway 서버를 시작하려면 `startManagementGateway.sh` 파일을 사용하십시오. 관리 게이트웨이 기능, 오브젝트 그리드 MBeans 및 JMX에 대한 자세한 정보는 73 페이지의 제 7 장 『System Management 개요』를 참조하십시오.

startManagementGateway 스크립트는 오브젝트 그리드 클러스터에 연결하여 서버를 중지하고 상태 및 통계를 수집하며 여러 추가 기능을 수행하는 오브젝트 그리드 클라이언트와 JMX 커넥터 서버를 작성합니다.

다음은 관리 게이트웨이 서버를 시작하기 위한 startManagementGateway 스크립트 인수입니다.

```
startManagementGateway.bat -connectorPort <port> -clusterHost <host>
-clusterPort <port> -clusterName <cluster> [options]
```

선택적 인수

몇 개의 선택적 인수를 startManagementGateway 스크립트에 전달할 수 있습니다. 유효한 startManagementGateway 인수 다음에 선택적 인수를 표시합니다.

```
startManagementGateway.bat -connectorPort <port> -clusterHost <host>
-clusterPort <port> -clusterName <cluster> [options]
```

옵션

- -traceEnabled <true/false 추적 사용 가능>
- -traceSpec <추적 스펙>
- -traceFile <추적 파일>
- -refreshInterval <MBean 속성 새로 고치기 간격>
- -sslEnabled <true/false 관리 게이트웨이에 대해 SSL 사용 가능>
- -clientSecurityFile <클라이언트 보안 파일 경로>

-traceEnabled

-traceEnabled 인수는 관리 게이트웨이 서버에 대해 추적이 켜져 있는지를 설정하는 데 사용할 수 있습니다. 기본값은 false이므로 오브젝트 그리드 추적을 확인하는 유일한 방법은 -traceEnabled를 ″true″로 설정하고 유효한 -traceSpec 및 -traceFile 값을 제공하여 추적을 사용 가능하게 하는 것입니다.

-traceSpec

-traceSpec 인수는 관리 게이트웨이 서버의 추적 스펙을 설정하는 데 사용할 수 있습니다.

-traceFile

-traceFile 인수는 관리 게이트웨이 추적 출력을 위한 위치를 지정하는 데 사용할 수 있습니다. 다음은 traceEnabled, traceSpec 및 traceFile 인수를 설정하는 방법에 대한 예입니다.

```
startManagementGateway.bat -connectorPort 1099 -clusterHost lion.ibm.com
-clusterPort 12501 -clusterName universityCluster -traceEnabled true
-traceSpec ObjectGrid=all=enabled -traceFile WWobjectgridWWtrace.log
```

-refreshInterval

-refreshInterval 인수는 관리 게이트웨이가 MBean 속성 값을 새로 고치는 사이 대기하는 시간(초)을 전달하는 데 사용할 수 있습니다. 기본값은 120초입니다. 다음은 refreshInterval 인수를 설정하는 방법에 대한 예입니다.

```
startManagementGateway.bat -connectorPort 1099 -clusterHost lion.ibm.com
-clusterPort 12501 -clusterName universityCluster -refreshInterval 60
```

-sslEnabled

-sslEnabled 인수는 관리 게이트웨이에 대해 SSL이 사용 가능한지 설정하는 데

사용할 수 있습니다. 이 인수의 값이 true이면 관리 게이트웨이 서버에 연결하는 사용자 클라이언트는 SSL 특성을 제공해야 합니다.

- -Djavax.net.ssl.trustStore
- -Djavax.net.ssl.trustStorePassword

-sslEnabled 인수가 제공되지 않은 경우 기본값은 ″false″입니다.

-clientSecurityFile

-clientSecurityFile 인수는 관리 게이트웨이 서버와 오브젝트 그리드 클러스터 간 보안 클라이언트 액세스의 클라이언트 보안 특성을 포함하는 파일 이름을 전달하는 데 사용될 수 있습니다. 보안이 사용 가능한 클러스터에 연결을 시도할 때 이 인수는 필수입니다. 오브젝트 그리드는 클라이언트 보안 특성 파일 템플리트인 security.ogclient.props를 제공합니다.

다음은 sslEnabled 및 clientSecurityFile 특성을 설정하는 방법의 예입니다.

```
startManagementGateway.bat -connectorPort 1099 -clusterHost lion.ibm.com
-clusterPort 12501 -clusterName universityCluster -sslEnabled true
-clientSecurityFile ..₩₩properties₩₩security.ogclient.props
```

암호 인코딩

암호 인코딩은 오브젝트 그리드 보안 특성 파일에서 일상적인 암호 관찰을 수행하지 못하게 합니다.

사용법

오브젝트 그리드에는 암호화되지 않은 몇몇 인코드된 암호가 있습니다. 오브젝트 그리드는 이 암호를 인코드하는 데 사용할 수 있는 FilePasswordEncoder 유틸리티를 제공합니다. Windows 시스템에서 암호를 인코드하려면 FilePasswordEncoder.bat 파일을 사용하십시오. Linux 및 Unix 플랫폼에서 암호를 인코드하려면 FilePasswordEncoder.sh 파일을 사용하십시오.

명령 구문은 다음과 같습니다.

```
FilePasswordEncoder.bat file_name password_properties_list [file_type]
```

옵션

FilePasswordEncoder 명령에 다음 옵션을 사용할 수 있습니다.

file_name

file_name은 인코드할 암호를 보유하는 파일 이름을 지정하는 데 사용됩니다. 예를 들어, security.ogserver.props가 있습니다.

password_prop_list

password_prop_list는 쉼표로 구분된 암호 특성 이름의 목록입니다(예: ″trustStorePassword,keyStorePassword″).

file_type

이 인수는 선택적입니다. file_type은 xml 또는 특성 값이 될 수 있으며, 이 값으로 제공된 파일이 특성 파일 또는 XML 파일인지 나타냅니다. 기본값은 property입니다. 현재 오브젝트 그리드는 XML 파일에 암호를 저장하지 않으므로, 이 옵션은 필수가 아닙니다. 다음 예는 올바른 구문을 설명합니다.

- `FilePasswordEncoder.bat security.ogclient.props ″trustStorePassword,keyStorePassword″`
- `FilePasswordEncoder.bat security.ogserver.props ″trustStorePassword,keyStorePassword, secureTokenKeyStorePassword, secureTokenKeyPairPassword, secureTokenSecretKeyPassword″`

이 FilePasswordEncoder 유틸리티는 WebSphere Extended Deployment에서 제공하지 않습니다. WebSphere Application Server에서 제공하는 PropFilePasswordEncoder 유틸리티를 사용하여 이 암호를 인코드할 수 있습니다. 자세한 정보는 PropFilePasswordEncoder 명령어 참조서를 참조하십시오.

제 9 장 오브젝트 그리드 API(Application Programming Interface) 개요

이 섹션에서는 XML 또는 프로그램 방식의 인터페이스를 사용하여 오브젝트 그리드를 구성하는 방법을 설명합니다. 오브젝트 그리드에서 제공하는 외부 인터페이스를 구현하는 정보도 들어 있습니다. 모든 경우에서 개요, API 인터페이스 및 예제를 설명합니다.

API 문서

오브젝트 그리드에 대한 JavaDoc은 API에 대한 정보를 얻는 가장 확실한 방법입니다. JavaDoc은 WebSphere Extended Deployment 설치 디렉토리, *install_root*₩web₩xd₩apidocs에 있습니다.

ObjectGridManager 인터페이스

ObjectGridManagerFactory 클래스 및 ObjectGridManager 인터페이스는 오브젝트 그리드 인스턴스를 작성, 액세스 및 캐시하기 위한 메커니즘을 제공합니다. ObjectGridManagerFactory 클래스는 ObjectGridManager 인터페이스, 싱글톤에 액세스하기 위한 정적 헬퍼 클래스입니다. ObjectGridManager 인터페이스에는 오브젝트 그리드 오브젝트의 인스턴스를 작성하는 데 편리한 여러 개의 메소드가 포함되어 있습니다. 또한 ObjectGridManager 인터페이스는 여러 명의 사용자가 액세스할 수 있는 오브젝트 그리드 인스턴스를 쉽게 작성하고 캐싱할 수 있도록 합니다.

createObjectGrid 메소드

ObjectGridManager 인터페이스에 있는 일곱 개의 createObjectGrid 메소드에 대해 학습하려면 이 주제를 사용하십시오.

createObjectGrid 메소드

ObjectGridManager 인터페이스에는 일곱 개의 createObjectGrid 메소드가 있습니다. 다음은 단순 시나리오입니다.

기본 구성의 단순 사례

다음은 많은 사용자들 간에 공유할 오브젝트 그리드를 작성하는 단순 사례입니다.

```
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
final ObjectGridManager oGridManager=
 ObjectGridManagerFactory.getObjectGridManager();
```

```
ObjectGrid employees = oGridManager.createObjectGrid("Employees",true);
employees.initialize();
employees.
/*sample continues..*/
```

위 Java 코드 스니펫은 Employees 오브젝트 그리드를 작성하고 캐시합니다. Employees 오브젝트 그리드는 기본 구성을 사용하여 초기화되며 사용할 준비가 되었습니다. createObjectGrid 메소드의 두 번째 매개변수는 true로 설정되어 ObjectGridManager가 작성하는 오브젝트 그리드 인스턴스를 캐시하도록 지시합니다. 이 매개변수가 false로 설정된 경우, 인스턴스는 캐시되지 않습니다. 모든 오브젝트 그리드마다 이름이 있으며, 해당 이름을 기준으로 다수의 클라이언트 또는 사용자 간에 인스턴스를 공유할 수 있습니다.

오브젝트 그리드 인스턴스가 피어 투 피어 공유에서 사용되면 캐싱을 true로 설정해야 합니다. 피어 투 피어 공유에 대한 자세한 정보는 369 페이지의 제 12 장 『피어 JVM(Java Virtual Machine) 사이에서 변경사항 분배』를 참조하십시오.

XML 구성

오브젝트 그리드는 쉽게 구성할 수 있습니다. 위 예에서는 구성을 사용하지 않고 단순 오브젝트 그리드를 작성하는 방법을 설명했습니다. 이 예를 사용하면 XML 구성 파일을 기반으로 사전 구성된 오브젝트 그리드 인스턴스를 작성할 수 있습니다. 오브젝트 그리드 인스턴스 프로그램 방식 또는 XML 기반의 구성 파일 사용을 구성할 수 있습니다. 두 가지 접근 방식을 조합하여 오브젝트 그리드를 구성할 수도 있습니다.

ObjectGridManager 인터페이스는 XML 구성을 기반으로 오브젝트 그리드 인스턴스 작성을 허용합니다. ObjectGridManager 인터페이스에는 URL을 인수로 갖는 여러 메소드가 있습니다. ObjectGridManager로 전달되는 모든 XML 파일을 스키마와 비교하여 유효성을 검증해야 합니다. 파일이 이전에 유효성 검증되었고 마지막 유효성 검증 이후로 파일이 변경되지 않은 경우에만 XML 유효성 검증을 사용 불가능하게 할 수 있습니다. 유효성 검증을 사용 불가능하게 하면 오버헤드를 약간 줄일 수 있지만 유효하지 않은 XML 파일을 사용할 가능성이 있습니다. IBM JDK(Java Developer Kit) 1.4.2는 XML 유효성 검증을 지원합니다. 이 지원이 없는 JDK를 사용하는 경우, XML 유효성을 검증하려면 Apache Xerces가 필요할 수도 있습니다.

다음 Java 코드는 XML 구성 파일을 전달하여 오브젝트 그리드를 작성하는 방법을 설명합니다.

```
import java.net.MalformedURLException;
import java.net.URL;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
boolean validateXML = true; // turn XML validation on
boolean cacheInstance = true; // Cache the instance
String objectGridName="Employees"; // Name of Object Grid URL
```

```
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees = oGridManager.createObjectGrid(objectGridName,
allObjectGrids,
validateXML,
                        cacheInstance);
```

XML 파일은 여러 오브젝트 그리드에 대한 구성 정보를 포함할 수 있습니다. 위의 코드 스니펫은 ″Employees″ 구성이 파일에 정의되어 있다고 가정하고 오브젝트 그리드 ″Employees″를 리턴합니다. XML 구문에 대해서는 278 페이지의 『오브젝트 그리드 구성』을 참조하십시오.

일곱 개의 createObjectGrid 메소드가 있습니다. 메소드는 다음 코드 블록에 설명되어 있습니다.

```
/**
* A simple factory method to return an instance of an
* Object Grid. A unique name is assigned.
* The instance of ObjectGrid is not cached.
* Users can then use {@link ObjectGrid#setName(String)} to change the
* ObjectGrid name.
*
* @return ObjectGrid an instance of ObjectGrid with a unique name assigned
* @throws ObjectGridException any error encountered during the ObjectGrid creation
* @ibm?api
*/
public ObjectGrid createObjectGrid() throws ObjectGridException;
/**
* A simple factory method to return an instance of an ObjectGrid with the
* specified name. The instances of ObjectGrid can be cached. If an ObjectGrid
* with the this name has already been cached, an ObjectGridException
* will be thrown. *
* @param objectGridName the name of the ObjectGrid to be created.
* @param cacheInstance true, if the ObjectGrid instance should be cached
* @return an ObjectGrid instance
* @this name has already been cached or
* any error during the ObjectGrid creation.
* @ibm?api
*/
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
throws ObjectGridException;
/**
* Create an ObjectGrid instance with the specified ObjectGrid name. The
* ObjectGrid instance created will be cached.
* @param objectGridName the Name of the ObjectGrid instance to be created.
* @return an ObjectGrid instance
* @throws ObjectGridException if an ObjectGrid with this name has already
* been cached, or any error encountered during the ObjectGrid creation
* @ibm?api
*/
public ObjectGrid createObjectGrid(String objectGridName)
throws ObjectGridException;
/**
* Create an ObjectGrid instance based on the specified ObjectGrid name and the
* XML file. The ObjectGrid instance defined in the XML file with the specified
* ObjectGrid name will be created and returned. If such an ObjectGrid
* cannot be found in the xml file, an exception will be thrown.
*
* This ObjecGrid instance can be cached.
*
* If the URL is null, it will be simply ignored. In this case, this method behaves
* the same as {@link #createObjectGrid(String, boolean)}.
```

```
*
* @param objectGridName the Name of the ObjectGrid instance to be returned. It
* must not be null.
* @param xmlFile a URL to a wellformed xml file based on the ObjectGrid schema.
* @param enableXmlValidation if true the XML is validated
* @param cacheInstance a boolean value indicating whether the ObjectGrid
* instance(s)
* defined in the XML will be cached or not. If true, the instance(s) will
* be cached. *
* @throws ObjectGridException if an ObjectGrid with the same name
* has been previously cached, no ObjectGrid name can be found in the xml file,
* or any other error during the ObjectGrid creation.
* @return an ObjectGrid instance
* @see ObjectGrid
* @ibm?api
*/
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance) throws
ObjectGridException;
/**
* Process an XML file and create a List of ObjectGrid objects based
* upon the file.
* These ObjecGrid instances can be cached.
* An ObjectGridException will be thrown when attempting to cache a
* newly created ObjectGrid
* that has the same name as an ObjectGrid that has already been cached.
*
* @param xmlFile the file that defines an ObjectGrid or multiple
* ObjectGrids
* @param enableXmlValidation setting to true will validate the XML
* file against the schema
* @param cacheInstances set to true to cache all ObjectGrid instances
* created based on the file
* @return an ObjectGrid instance
* @throws ObjectGridException if attempting to create and cache an
* ObjectGrid with the same name as
* an ObjectGrid that has already been cached, or any other error
* occurred during the
* ObjectGrid creation
* @ibm?api
*/
public List createObjectGrids(final URL xmlFile,
final boolean enableXmlValidation,
boolean cacheInstances)
throws ObjectGridException;
/*** Create all ObjectGrids that are found in the XML file. The XML file will be
* validated against the schema. Each ObjectGrid instance that is created will
* be cached. An ObjectGridException will be thrown when attempting to cache a
* newly created ObjectGrid that has the same name as an ObjectGrid that has
* already been cached.
* @param xmlFile The XML file to process. ObjectGrids will be created based
* on what is in the file.
* @return A List of ObjectGrid instances that have been created.
* @throws ObjectGridException if an ObjectGrid with the same name as any of
* those found in the XML has already been cached, or
* any other error encounterred during ObjectGrid creation.
* @ibm?api
*/
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;
/**
* Process the XML file and create a single ObjectGrid instance with the
* objectGridName specified only if an ObjectGrid with that name is found in
* the file. If there is no ObjectGrid with this name defined in the XML file,
* an ObjectGridException
* will be thrown. The ObjectGrid instance created will be cached.
* @param objectGridName name of the ObjectGrid to create. This ObjectGrid
* should be defined in the XML file.
* @param xmlFile the XML file to process
* @return A newly created ObjectGrid
```

```
* @throws ObjectGridException if an ObjectGrid with the same name has been
* previously cached, no ObjectGrid name can be found in the xml file,
* or any other error during the ObjectGrid creation.
* @ibm?api
*/
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
throws ObjectGridException;
```

getObjectGrid 메소드

캐시된 인스턴스를 검색하려면 getObjectGrid 메소드를 사용하십시오.

캐시된 인스턴스 검색

Employees 오브젝트 그리드 인스턴스는 ObjectGridManager 인터페이스로 캐시되었기 때문에 다른 사용자의 경우 다음 코드 스니펫을 사용하여 액세스할 수 있습니다.

```
ObjectGrid myEmployees = oGridManager.getObjectGrid("Employees");
```

다음은 캐시된 오브젝트 그리드 인스턴스를 리턴하는 두 개의 getObjectGrid 메소드입니다.

```
/**
* Get a List of the ObjectGrid instances that have been previously cached.
* Returns null if no ObjectGrid instances have been cached.
* @return a List of ObjectGrid instances that have been previously cached
* @ibm?api
*/
public List getObjectGrids();
/**
* Use this if a ObjectGrid already exists. It returns a cached
* ObjectGrid instance by name. This method returns null if no
* ObjectGrid with this objectGridName has been cached.
*
* @param objectGridName the cached objectgrid name.
* @return a cached ObjectGrid which currently exists.
*
* @since WAS XD 6.0
* @ibm?api
*
*/
public ObjectGrid getObjectGrid(String objectGridName);
```

removeObjectGrid 메소드

이 주제에서는 두 개의 removeObjectGrid 메소드 사용법을 설명합니다.

오브젝트 그리드 인스턴스 제거

캐시에서 오브젝트 그리드 인스턴스를 제거하려면 removeObjectGrid 메소드 중 하나를 사용하십시오. ObjectGridManager는 제거된 인스턴스의 참조를 보관하지 않습니다. 두 개의 제거 메소드가 있습니다. 하나의 메소드는 부울 매개변수를 사용합니다. 부울 매개변수가 **true**로 설정된 경우 오브젝트 그리드에서 destroy 메소드가 호출됩니다. 오

브젝트 그리드에서 destroy 메소드를 호출하면 오브젝트 그리드를 시스템 종료하고 여기에서 사용하는 자원을 해제합니다. 다음은 두 개의 removeObjectGrid 메소드 사용 방법을 설명합니다.

```
/**
* Remove an ObjectGrid from the cache of ObjectGrid instances
* @param objectGridName the name of the ObjectGrid instance to remove
* from the cache
* @throws ObjectGridException if an ObjectGrid with the objectGridName
* was not found in the cache
* @ibm-api
*/
public void removeObjectGrid(String objectGridName) throws ObjectGridException;
/**
* Remove an ObjectGrid from the cache of ObjectGrid instances and
* destroy its associated resources
* @param objectGridName the name of the ObjectGrid instance to remove
* from the cache
* @param destroy destroy the objectgrid instance and its associated
* resources
* @throws ObjectGridException if an ObjectGrid with the objectGridName
* was not found in the cache
* @ibm-api
*/
public void removeObjectGrid(String objectGridName, boolean destroy)
throws ObjectGridException;
```

getObjectGridAdministrator 메소드

클러스터의 ObjectGridAdministrator 인스턴스 리턴

```
public ObjectGridAdministrator getObjectGridAdministrator(ClientClusterContext ctx)
/**
* Return an ObjectGridAdministrator instance for this cluster. Each
* cluster will require the use of a different ObjectGridAdministrator.
*
* @param clientClusterContext. A unique cluster context, with which the client
* needs to interact.
* @param objectGridName the cached objectgrid name.
* @return an ObjectGrid
*
* @since WAS XD 6.0.1
* @ibm-api
*
*/
public ObjectGridAdministrator getObjectGridAdministrator(ClientClusterContext
ontext);
```

이 메소드에 대한 자세한 정보는 73 페이지의 제 7 장『System Management 개요』를 참조하십시오.

ObjectGridManager 인터페이스를 사용하여 오브젝트 그리드 인스턴스의 라이프 사이클 제어

이 주제에서는 ObjectGridManager 인터페이스를 통해 시작 Bean 및 Servlet을 사용하여 오브젝트 그리드 인스턴스의 라이프 사이클을 제어하는 방법을 설명합니다.

시작 Bean에서 오브젝트 그리드 인스턴스 라이프 사이클 관리

시작 Bean을 사용하여 오브젝트 그리드 인스턴스의 라이프 사이클을 제어할 수 있습니다. 시작 Bean은 응용프로그램이 시작될 때 로드됩니다. 시작 Bean을 사용하면 예상한 대로 응용프로그램이 시작되거나 중지될 때마다 코드를 실행할 수 있습니다. 시작 Bean을 작성하려면 홈 com.ibm.websphere.startupservice.AppStartUpHome 인터페이스를 사용하고 원격 com.ibm.websphere.startupservice.AppStartUp 인터페이스를 사용하십시오. Bean에 start 및 stop 메소드를 구현하십시오. start 메소드는 응용프로그램이 시작될 때마다 호출됩니다. stop 메소드는 응용프로그램이 시스템 종료될 때 호출됩니다. start 메소드를 사용하여 오브젝트 그리드 인스턴스를 작성할 수 있습니다. stop 메소드를 사용하여 오브젝트 그리드 인스턴스를 제거할 수 있습니다. 다음은 시작 Bean에서의 이 오브젝트 그리드 라이프 사이클 관리를 설명하는 코드 스니펫입니다.

```
public class MyStartupBean implements javax.ejb.SessionBean {
private ObjectGridManager objectGridManager;
/*
* The methods on the SessionBean interface have been
* left out of this example for the sake of brevity
*/
public boolean start(){
 // Starting the startup bean
 // This method is called when the application starts
 objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
try {
  // create 2 ObjectGrids and cache these instances
  ObjectGrid bookstoreGrid =
objectGridManager.createObjectGrid("bookstore", true);
bookstoreGrid.defineMap("book");
  ObjectGrid videostoreGrid =
objectGridManager.createObjectGrid("videostore", true);
  // within the JVM,
  // these ObjectGrids can now be retrieved from the
  //ObjectGridManager using the getObjectGrid(String) method
  } catch (ObjectGridException e) {
   e.printStackTrace();
  return false;
}
 return true;
}
public void stop(){
 // Stopping the startup bean
 // This method is called when the application is stopped
try {
  // remove the cached ObjectGrids and destroy them
objectGridManager.removeObjectGrid("bookstore", true);
objectGridManager.removeObjectGrid("videostore", true);
  } catch (ObjectGridException e) {
   e.printStackTrace();
}
}
}
```

start 메소드가 호출된 후에 새로 작성된 오브젝트 그리드 인스턴스를 ObjectGridManager에서 검색할 수 있습니다. 예를 들어, Servlet이 응용프로그램에 포함된 경우 Servlet은 다음 코드 스니펫을 사용하여 이런 오브젝트 그리드에 액세스할 수 있습니다.

```
ObjectGridManager objectGridManager =
  ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
ObjectGrid videostoreGrid = objectGridManager.getObjectGrid("videostore");
```

Servlet에서 오브젝트 그리드 라이프 사이클 관리

Servlet에서 오브젝트 그리드의 라이프 사이클을 관리하는 한 가지 방법은 init 메소드에서 오브젝트 그리드 인스턴스를 작성하고 destroy 메소드에서 오브젝트 그리드를 제거하는 것입니다. 오브젝트 그리드 인스턴스가 캐시된 경우, Servlet 코드에서 이를 검색하고 조작할 수 있습니다. 다음은 Servlet 내의 오브젝트 그리드 작성, 조작 및 제거를 설명하는 일부 샘플 코드입니다.

```
public class MyObjectGridServlet extends HttpServlet implements Servlet {
private ObjectGridManager objectGridManager;
 public MyObjectGridServlet() {
 super();
}
public void init(ServletConfig arg0) throws ServletException {
 super.init();
 objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
try {
  // create and cache an ObjectGrid named bookstore
  ObjectGrid bookstoreGrid =
objectGridManager.createObjectGrid("bookstore", true);
bookstoreGrid.defineMap("book");
  } catch (ObjectGridException e) {
   e.printStackTrace();
}
}
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
BackingMap bookMap = bookstoreGrid.getMap("book");
 // perform operations on the cached ObjectGrid
// ...
}
public void destroy() {
 super.destroy();
try {
  // remove and destroy the cached bookstore ObjectGrid
objectGridManager.removeObjectGrid("bookstore", true);
  } catch (ObjectGridException e) {
   e.printStackTrace();
}
}
}
```

오브젝트 그리드 추적

이 주제에서는 오브젝트 그리드에서 추적을 설정하는 방법을 설명합니다.

J2SE(Java 2 Platform, Standard Edition) 환경

IBM에 디버그 정보를 전송해야 하는 경우 추적 메커니즘을 사용하여 디버그 추적을 확보하십시오. 다음은 J2SE 환경에서 디버그 추적을 가져오는 방법에 대한 예제입니다.

```
oGridManager.setTraceFileName("debug.log");
oGridManager.setTraceSpecification("ObjectGrid=all=enabled");
```

위 예제는 오브젝트 그리드에 대한 내장 축출기 플러그인의 추적을 포함하지 않습니다. 오브젝트 그리드에서 제공하는 축출기 플러그인을 하나 이상 사용하는데 축출과 관련될 수 있는 문제점이 발생한 경우, 다음 예에서 표시한 대로 오브젝트 그리드 및 오브젝트 그리드 축출기에서 추적을 사용 가능하게 하십시오.

```
oGridManager.setTraceFileName("debug.log");
oGridManager.setTraceSpecification
("ObjectGridEvictors=all=enabled:ObjectGrid=all=enabled");
```

WebSphere Application Server 환경

WebSphere Application Server 환경에서 추적을 설정할 때 ObjectGridManager를 사용하지 않아도 됩니다. 관리 콘솔을 사용하여 추적 스펙을 설정할 수 있습니다.

오브젝트 그리드 클라이언트 연결 API

기본사항

클라이언트는 클러스터 내에서 활성 또는 실행 중인 서버 프로세스에 연결합니다. 클라이언트는 최소한 연결할 서버의 호스트 이름과 포트 번호를 필요로 합니다. 호스트 이름과 포트 정보는 초기에 서버를 시작하는 데 사용된 클러스터 정의 XML 파일에서 얻을 수 있습니다. XML 구성 세부사항에 대해서는 278 페이지의 『오브젝트 그리드 구성』을 참조하십시오. 다음은 이 섹션의 샘플로 사용된 클러스터 XML 정의의 스니펫입니다. 이 섹션에서 언급된 API를 사용하여 클라이언트는 클라이언트 요청을 수신하고 처리하도록 구성된 원격 오브젝트 그리드에 연결합니다. 클라이언트는 오브젝트 그리드 및 클러스터 XML 정의를 모두 "다운로드하여" 자체 부트스트랩합니다. 클라이언트 구성은 클라이언트가 연결하는 서버 구성에 기반합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster ../
objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1" securityEnabled="false" clientMaxRetries="15"
tcpConnectionTimeout="180"
singleSignOnEnabled="true" loginSessionExpirationTime="300"
statisticsEnabled="true" statisticsSpec="all=enabled">
<serverDefinition name="server1" host="s1.myco.com" clientAccessPort="12503"
```

```
peerAccessPort="12500" workingDirectory="/tmp/s1/"
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="s2.myco.com" clientAccessPort="12504"
peerAccessPort="12501" workingDirectory="/tmp/s2/"
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server3" host="10.5.1.22" clientAccessPort="12505"
peerAccessPort="12502" workingDirectory="/tmp/s3/"
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true"/>
</cluster>
<objectgrid-binding
.
.
.
```

이 클러스터 정의는 불완전하지만 이 예에서는 크게 문제되지 않습니다. cluster1 클러스터에는 server1, server2 및 server3 서버가 있습니다. clientAccessPort 속성은 서버가 청취하는 리스너 포트와 클라이언트가 처음 연결을 설정하는 포트를 지정합니다. 이전 XML 스니펫에서 server1, server2 및 server3 서버용 포트는 각각 12503, 12504 및 12504입니다.

연결 API

ObjectGridManager 인터페이스에는 다음 샘플에서 설명하는 connect 메소드가 있습니다. ObjectGridManager 인터페이스에서 다음 연결 API를 사용할 수 있습니다. 이 메소드에 대한 설명은 API 문서를 참조하십시오.

```
/**
* This allows a client to connect to a remote ObjectGrid
* The remote ObjectGrid is hosted as specified by the parameters:
* @param clusterName: The name of the cluster to which this client
* attaches itself
* @param host: The host on which to connect
* @param port: The clientAceess port that is listening
* @param ClientSecurityConfiguration: Security configuration,can be
* null if security is not configured
* @param overRideObjectGrid xml. This parameter can be null. If it is not null,
* the client side configuration of ObjectGrid plug-in is overridden.
* Not all plug-ins can be overridden. For details see the
* ObjectGrid documents
* @throws ConnectException
*
*/
public ClientClusterContext connect(String clusterName,
String host,
String port,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
*
* @param clusterName
* @param attributes Host and Port pair attributes that are tried in sequential
* order to connect. If an attempt to connect fails to one server, the next pair
* of host and port attributes is picked to retry the connect.
* @param ClientSecurityConfiguration: Security configuration. It can be null if
```

```
* security is not configured.
* @param overRideObjectGrid xml. This parameter can be null. If it is not null,
* the client side configuration of ObjectGrid plug-in is overridden.
* Not all plug-ins can be overridden. For details see the ObjectGrid documents.
* @return ClientClusterContext
* @throws ConnectException
*
*
*/
public ClientClusterContext connect(String clusterName,
HostPortConnectionAttributes[] attributes,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
* This method can be used only if a client is colocated with
* an ObjectGrid server, especially in a Java 2 Platform,
* Enterprise Edition (J2EE) environment with IBM WebSphere
* Application Server, which supports the
* embedded ObjectGrid server.
* This method connects the client to the Server which is running
* in the same Java virtual machine (JVM).
* @param securityProps. It can be null if not running in secure mode.
* @param overRideObjectGrid xml. This parameter can be null. If it is
* not null, the client side configuration of ObjectGrid plug-in is overridden.
* Not all plug-ins can be overridden. For details see the
* ObjectGrid documents
* @return ClientClusterContext
* @throws ConnectException
*
*/
public ClientClusterContext connect(ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException;
/**
* This allows a client to connect to a Remote ObjectGrid
* @param clusterConfigFile A URL to the clusterConfig File. This is the
* same file that is used to start servers.
* This is used to retrieve host port information. It cannot be null. If it is
* null a IllegalArgumentException exception results.
* @param serverName A String, the name of the specific server to connect to.
*If the server name is not in the configuration, IllegalArgumentException
* results.
* This parameter can be null, in which case an attempt is made to connect
* to one of servers specified in the cluster
* XML file. If an attempt fails to connect to one, another server is picked,
* It is done, until such time the list is exhausted.
* @param securityProps
* @param overRideObjectGrid xml. This parameter can be null. If it is not
* null, the client side configuration of ObjectGrid plug-in is overridden.
* Not all plug-ins can be overridden. For details, see the ObjectGrid
* documents
* @return ClientClusterContext
* @throws ConnectException
*
* @ibm-api
*/
public ClientClusterContext connect(URL clusterConfigFile,
String serverName,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
```

호스트 및 포트 매개변수를 사용하는 예제

다음 코드는 107 페이지의 『기본사항』에 설명된 클러스터 XML을 사용합니다. 이 클라이언트는 포트 12503에서 s1.myco.com 호스트에 연결됩니다.

```
import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ConnectException;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
public class C1 {
/**
* @param args
*/
public static void main(String[] args) {
final ObjectGridManager oGridManager=ObjectGridManagerFactory.
getObjectGridManager(); //step 1
ClientClusterContext ctx = null;
try {
ctx=oGridManager.connect("cluster1","s1.myco.com","12503",null,null);
//step 2
ObjectGrid employees = oGridManager.getObjectGrid(ctx,"employees");
// step 3
// Do objectGrid operations
// get
// update
// commit...etc..
} catch (ConnectException e) {
//connect failed
   e.printStackTrace();
//terminate
}finally {
if(ctx !=null) {
oGridManager.disconnect(ctx); // step 4
}
}
}
}
```

1. ObjectGridManagerFactory에서 ObjectGridManager 싱글톤 오브젝트를 가져옵니다.
2. 연결 API를 호출합니다.
3. 오브젝트 그리드 employees가 원격 오브젝트 그리드에 있다고 가정하면 ClientClusterContext 매개변수를 전달하여 getObjectGrid 메소드를 호출하십시오.
4. disconnect 메소드를 호출합니다. 마지막 단계로 작업이 완료되면, 모든 클라이언트에서 연결 끊기를 호출해야 합니다. 이것은 매우 중요한 단계입니다.

다중 호스트를 제공하여 ConnectException 예외의 경우 자동으로 연결 재시도

이 예에서는 HostPortConnectionAttributes 속성을 사용하여 클라이언트가 연결할 수 있는 호스트 포트 속성 배열을 제공합니다. API는 이 호스트 및 포트 쌍 속성을 순차적으로 사용하여 연결합니다. 한 서버에 대한 연결 시도가 실패하면 호스트 및 포트 속성의 다음 쌍이 선택되어 연결을 재시도합니다.

```
import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ConnectException;
import com.ibm.websphere.objectgrid.HostPortConnectionAttributes;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
public class C2 {
/**
* @param args
*/
public static void main(String[] args) {
final ObjectGridManager oGridManager=ObjectGridManagerFactory.
getObjectGridManager();
ClientClusterContext ctx = null;
HostPortConnectionAttributes[] hca = new HostPortConnectionAttributes[3];
hca[0]=new HostPortConnectionAttributes("s1.myco.com","12503");
hca[1]=new HostPortConnectionAttributes("s2.myco.com","12504");
hca[2]=new HostPortConnectionAttributes("10.5.1.22","12505");
try {
ctx=oGridManager.connect("cluster1",hca,null,null);
ObjectGrid employees = oGridManager.getObjectGrid(ctx,"employees");
// Do objectGrid operations such as
// get
// update
// commit.... etc...
} catch (ConnectException e) {
   e.printStackTrace();
}finally {
if(ctx !=null) {
oGridManager.disconnect(ctx);
}
}
}
}
```

동일한 프로세스의 클라이언트 및 서버

클라이언트가 서버와 동일한 JVM에 있으면 다음 connect 메소드를 사용할 수 있습니다.

```
ctx=oGridManager.connect(null,null);
```

클러스터 XML 지정

클라이언트가 클러스터 XML 파일에 액세스할 수 있으면 호스트 이름 또는 포트 번호를 지정할 필요가 없습니다. 이 API가 서버 이름 및 포트 번호를 검색하고 사용하여 연결합니다. 서버 이름은 선택적이며 API가 클러스터 XML 파일에 정의된 서버 중 하나에 연결을 시도하는 경우 널이 될 수 있습니다.

```
ctx=oGridManager.connect(urlToClusterxml,"server1",null,null);
// connect to server1
// or
ctx=oGridManager.connect(urlToClusterxml,null,null,null);
//connect to any server in the cluster
```

연결 API를 통한 클라이언트 보안

모든 예에서 ClientSecurityConfiguration 매개변수는 널이었습니다. 널 값을 전달한다는 것은 보안이 사용 불가능하다는 의미입니다. 보안이 사용 가능하면 ClientSecurityConfiguration 오브젝트를 인수로 전달합니다. 자세한 정보는 149 페이지의 『오브젝트 그리드 보안』을 참조하십시오.

오브젝트 그리드 XML 구성 대체

클라이언트는 서버에서 오브젝트 그리드 정의를 ″다운로드″하여 자체 구성됩니다. 오브젝트 그리드에 정의된 모든 플러그인이 클라이언트에 사용될 수 있도록 합니다. 원래 로컬 오브젝트 그리드가 서버측 오브젝트 그리드와 통신하는 클라이언트측에 존재합니다. 연결 API의 대체 XML 파일을 제공하여 클라이언트 사용만을 위한 플러그인 구성을 ″대체″할 수 있습니다. 이러한 플러그인은 다음과 같습니다.

오브젝트 그리드 플러그인:

- TransactionCallback 플러그인
- ObjectGridEventListener 플러그인

BackingMap 플러그인:

- Evictor 플러그인
- MapEventListener 플러그인

대체 XML에 정의된 기타 플러그인은 무시됩니다.

예

클라이언트가 특정 BackingMap의 축출기 구성을 대체해야 한다고 가정하십시오. 즉, 클라이언트측의 축출기는 서버측에 구성된 축출기와 달라야 합니다.

서버측 축출기가 다음과 같이 사용된다고 가정하십시오. 내장 LFUEvictor 축출기를 사용합니다.

```
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
<property name="maxSize" type="int" value="100" description="..." />
</bean>
```

클라이언트측 요구사항은 다릅니다. 대신 사용자가 정의한 myco.og.MyEvictor 축출기가 사용되어야 합니다. 대체 XML은 아래에 표시된 스니펫을 포함할 수 있습니다. LFUEvictor를 사용하도록 구성된 모든 backingMaps는 사용자 정의를 사용합니다.

```
<bean id="Evictor" className="myco.og.MyEvictor">
<property name="name" type="java.lang.String" value="MyEvictor"
description="..." />
</bean>
```

XML 완료

다음 XML 코드는 두 개의 XML 파일을 표시하는데, 하나는 서버측에 사용되고 두 번째는 클라이언트에 사용됩니다. 이러한 구성을 사용하여 클라이언트는 dow BackingMap의 축출기 구성을 대체할 수 있습니다.

서버측 오브젝트 그리드 XML 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="market">
<backingMap name="dow" ttlEvictorType="NONE" readOnly="false"
pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.
builtins.LRUEvictor">
<property name="maxSize" type="int" value="2"
description="set max size for LRU Evictor" />
<property name="numberOfLRUQueues" type="int" value="1"
description="set number of LRU queues" />
<property name="sleepTime" type="int" value="2" description="evictor
thread sleep time" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

연결 중 대체할 클라이언트측 오브젝트 그리드 XML 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="market">
<backingMap name="dow" pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Evictor" className="myco.og.MyEvictor">
<property name="name" type="java.lang.String" value="MyEvictor"
description="" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

응용프로그램 설계 고려사항

클라이언트 connect() API는 소모적인 조작입니다. 작업에 따라 클라이언트는 단일 서버에 대한 하나 이상의 실제 연결을 설정합니다. 클라이언트 연결 수는 클러스터 구성 XML 정의의 클러스터 요소 내에 정의된 tcpMinConnections 및 tcpMaxConnections 속성으로 지정된 값 사이에서 달라집니다. 이것은 서버를 시작하는 데 사용된 동일한 클러스터 XML 정의입니다. 연결 관리자는 이러한 실제 연결을 풀링하고 오브젝트 그리드는 필요에 따라 이 연결을 다시 사용합니다. tcpMinConnections 및 tcpMaxConnections 속성은 단일 서버에 대한 클라이언트 연결 수만 지정합니다. 클라이언트가 둘 이상의 서버에 연결될 경우 최대 클라이언트 연결 수는 클라이언트가 연결할 서버의 tcpMaxConnections 속성 횟수보다 작거나 같습니다. 예를 들어, 클라이언트가 세 개의 서버에 연결되고 tcpMaxConnections가 5로 지정되면 클라이언트는 tcpMinConnection의 설정이 1이라고 가정할 때 최대 연결 수를 (5*3)=15로 가지고 최소 연결 수를 3으로 가집니다. 연결은 모든 클라이언트에서 공유됩니다.

threadsPerClientConnect 속성은 작업자 스레드 수를 지정합니다. 이러한 작업자 스레드는 실제 연결을 통해 작업을 디스패치합니다. 이 스레드들은 구성 데이터, 클라이언트 요청, 서버 응답 및 시스템 관리 요청을 처리합니다. 기본값은 5입니다. 이 속성은 첫 번째 iFix에서 사용 가능합니다. iFix를 사용할 수 없으면 -Dthreads JVM(Java Virtual Machine) 시스템 특성을 사용하여 작업자 스레드 수를 지정하십시오. 응용프로그램에 따라 이 수를 늘리는 것이 성능에 도움을 줄 수 있습니다. 일반적으로 이 수는 클라이언트가 사용하는 실제 연결 수보다 많아야 합니다.

사용자

응용프로그램 설계에서 허용되는 경우 클라이언트는 다중 스레드를 작성하여 *ClientClusterContext* 메소드를 재사용함으로써 한 연결 호출 내에서 작업을 완료할 수 있습니다.

오브젝트 그리드 인터페이스

오브젝트 그리드를 수정하는 데 필요한 메소드를 참조하려면 이 주제를 사용하십시오.

소개

오브젝트 그리드는 JavaMap 인터페이스를 기반으로 하는 확장 가능한 트랜잭션 오브젝트 캐싱 프레임워크입니다. 오브젝트 그리드 API 조작은 트랜잭션 작업 단위로 그룹화되며, 사용자 정의 설계된 플러그인 지원을 통해 확장성을 허용합니다. 오브젝트 그리드는 다수의 BackingMap을 포함하는 이름 지정된 논리 컨테이너입니다. BackingMap에 대한 자세한 정보는 119 페이지의 『BackingMap 인터페이스』를 참조하십시오.

작성 및 초기화

오브젝트 그리드 인터페이스를 작성하는 데 필요한 단계는 ObjectGridManager 인터페이스 주제를 참조하십시오. 오브젝트 그리드를 작성하기 위한 두 개의 별개 메소드(프

로그램 방식 또는 XML 구성 파일 사용)가 있습니다. 자세한 정보는 99 페이지의 『ObjectGridManager 인터페이스』를 참조하십시오.

Get 또는 Set 및 팩토리 메소드

경고: 오브젝트 그리드 인스턴스를 초기화하기 전에 *set* 메소드를 호출해야 합니다. initialize 메소드가 호출된 후 set 메소드를 호출하면 `java.lang.IllegalStateException`이 발생합니다. 또한 오브젝트 그리드 인터페이스의 각 getSession 메소드는 암시적으로 initialize 메소드를 호출합니다. 따라서 getSession 메소드를 호출하기 전에 set 메소드를 호출해야 합니다. EventListener 오브젝트 설정, 추가 및 제거에는 이 규칙이 적용되지 않습니다. 이러한 오브젝트는 "초기화" 처리가 완료된 후 처리될 수 있습니다.

오브젝트 그리드 인터페이스에는 다음 메소드가 있습니다.

표 7. 오브젝트 그리드 인터페이스 메소드

메소드	설명
BackingMap defineMap(String name);	*defineMap*: 고유하게 이름 지정된 BackingMap을 정의하기 위한 팩토리 메소드입니다. BackingMap에 대한 자세한 정보는 119 페이지의 『BackingMap 인터페이스』를 참조하십시오.
BackingMap getMap(String name);	*getMap*: *defineMap*을 호출하여 이전에 정의된 BackingMap을 리턴합니다. XML 구성을 통해 BackingMap이 구성되지 않았으면 이 메소드를 사용하여 BackingMap을 구성할 수 있습니다.
BackingMap createMap(String name);	*createMap*: BackingMap을 작성하지만 이 오브젝트 그리드가 사용하도록 캐시하지 않습니다. 이 오브젝트 그리드와 함께 사용할 BackingMaps를 캐시하는 오브젝트 그리드 인터페이스의 setMaps(List) 메소드와 함께 이 메소드를 사용하십시오. 스프링 프레임워크로 오브젝트 그리드를 구성할 때 이 메소드를 사용하십시오.
void setMaps(List mapList);	*setMaps*: 이전에 이 오브젝트 그리드에 정의된 BackingMaps를 지우고 제공된 BackingMaps 목록으로 바꿉니다.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	*getSession*: 작업 단위에 대해 시작, 확약, 롤백 기능을 제공하는 로더를 리턴합니다. Session 오브젝트에 대한 자세한 정보는 123 페이지의 『세션 인터페이스』를 참조하십시오.
Session getSession(CredentialGenerator cg);	*getSession(CredentialGenerator cg)*: CredentialGenerator 오브젝트와 함께 세션을 가져옵니다. 이 메소드는 클라이언트 서버 환경에서 오브젝트 그리드 클라이언트만 호출할 수 있습니다.
Session getSession(Subject subject);	*getSession(Subject subject)*: 세션을 가져오기 위해 오브젝트 그리드에 구성된 오브젝트가 아닌 특정 Subject 오브젝트의 사용을 허용합니다.

표 7. 오브젝트 그리드 인터페이스 메소드 (계속)

메소드	설명
void initialize() throws ObjectGridException;	*initialize*: 오브젝트 그리드가 초기화되고 일반적으로 사용할 수 있습니다. 오브젝트 그리드가 초기화된 상태가 아닌 경우 이 메소드는 `getSession` 메소드가 호출될 때 내부적으로 호출됩니다.
void destroy();	*destroy*: 이 메소드를 호출한 후에는 프레임워크가 해체되어 사용할 수 없습니다.
void setTxTimeout(int timeout);	*setTxTimeout*: 이 메소드를 사용하여 오브젝트 그리드 인스턴스가 작성한 세션에서 시작된 트랜잭션을 완료하는데 허용된 시간(초)을 설정합니다. 트랜잭션이 지정된 시간 내에 완료되지 않으면 트랜잭션을 시작한 세션이 "제한시간 초과"됨으로 표시됩니다. 세션이 제한시간 초과됨으로 표시되면 제한시간 초과된 세션에서 호출하는 다음 ObjectMap 메소드에서 `com.ibm.websphere.objectgrid.TransactionTimeoutException` 예외를 발생시킵니다. 세션은 *롤백만*으로 표시되어 응용프로그램이 `TransactionTimeoutException` 예외를 발견한 후 rollback 메소드 대신 commit 메소드를 호출하는 경우에도 트랜잭션은 롤백됩니다. 제한시간 값 `0`은 트랜잭션이 완료하는 데 허용되는 시간에 제한이 없음을 나타냅니다. 트랜잭션은 제한시간 초과 값 `0`이 사용되면 제한시간을 초과하지 않습니다. 이 메소드가 호출되지 않으면 이 인터페이스의 getSession 메소드에서 리턴하는 세션은 기본적으로 트랜잭션 제한시간 값을 `0`으로 설정한 것입니다. 응용프로그램은 com.ibm.websphere.objectgrid.Session 인터페이스의 setTransactionTimeout 메소드를 사용하여 세션 기준에 따른 트랜잭션 제한시간 초과 설정을 대체할 수 있습니다.
int getTxTimeout();	*getTxTimeout*: 트랜잭션 제한시간 값을 초 단위로 리턴합니다. 이 메소드는 setTxTimeout 메소드의 제한시간 매개변수로 전달된 동일한 값을 리턴합니다. setTxTimeout 메소드가 호출되지 않았으면 메소드는 `0`을 리턴하여 트랜잭션이 완료하는 데 시간 제한이 없음을 표시합니다.
//Keywords.	
void associateKeyword(Serializable parent, Serializable child);	*associateKeyword*: 오브젝트 그리드는 키워드를 기반으로 유연한 무효화 메커니즘을 제공합니다. 키워드에 대한 자세한 정보는 132 페이지의 『키워드』를 참조하십시오. 이 메소드는 방향 관계로 두 개의 키워드를 함께 연결합니다. 상위가 무효화되면 하위도 무효화됩니다. 하위를 무효화해도 상위는 영향을 받지 않습니다.

표 7. 오브젝트 그리드 인터페이스 메소드 (계속)

메소드	설명
//Security	
void setSecurityEnabled()	*setSecurityEnabled*: 보안을 사용 가능하게 합니다. 기본적으로, 보안은 사용 불가능합니다.
void setPermissionCheckPeriod(long period);	*setPermissionCheckPeriod*: 이 메소드는 클라이언트 액세스를 허용하는 데 사용한 권한의 검사 빈도를 표시하는 단일 매개변수를 사용합니다. 매개변수가 0인 경우, 모든 메소드는 권한 부여 메커니즘(JAAS 권한 부여 또는 사용자 정의 권한 부여)에 현재 대상(subject)에 권한이 있는지 여부를 검사하도록 요청합니다. 이 계획으로 인해 권한 부여 구현에 따라 성능 문제점이 발생할 수 있습니다. 그러나 필요한 경우 이 권한 부여 유형을 사용할 수 있습니다. 또는 매개변수가 0미만인 경우, 권한을 새로 고치기 위해 권한 부여 메커니즘으로 리턴하기 전에 권한 세트를 캐시할 밀리초 수를 표시합니다. 이 매개변수는 성능을 향상시키지만 이 시점에 백엔드 권한이 변경된 경우에는 백엔드 보안 프로바이더가 수정되었더라도 오브젝트 그리드가 액세스를 허용하거나 방지할 수 있습니다.
void setAuthorizationMechanism(int authMechanism);	*setAuthorizationMechanism*: 권한 부여 메커니즘을 설정합니다. 기본값은 `SecurityConstants.JAAS_AUTHORIZATION`입니다.
setMapAuthorization(MapAuthorization ma);	*setMapAuthorization*: 이 오브젝트 그리드 인스턴스의 MapAuthorization 플러그인을 설정합니다. 이 플러그인을 사용하여 Subject 오브젝트에 포함된 프린시펄에 대한 ObjectMap 또는 JavaMap 액세스 권한을 부여할 수 있습니다. 이 플러그인의 일반 구현은 Subject 오브젝트에서 프린시펄을 검색한 후 프린시펄에 지정된 권한이 부여되었는지 여부를 확인하는 것입니다.
setSubjectSource(SubjectSource ss);	*setSubjectSource*: SubjectSource 플러그인을 설정합니다. 이 플러그인을 사용하여 오브젝트 그리드 클라이언트를 나타내는 Subject 오브젝트를 가져올 수 있습니다. 이 대상(subject)은 오브젝트 그리드 권한 부여에 사용됩니다. ObjectGrid.getSession 메소드가 사용되어 세션을 가져오고 보안이 사용 가능할 때 SubjectSource.getSubject 메소드는 오브젝트 그리드 런타임에서 호출합니다. 이 플러그인은 이미 인증된 클라이언트에 유용합니다. 이것은 인증된 Subject 오브젝트를 검색한 다음 오브젝트 그리드 인스턴스로 전달할 수 있습니다. 다른 인증은 필요하지 않습니다.

표 7. 오브젝트 그리드 인터페이스 메소드 (계속)

메소드	설명
setSubjectValidation(SubjectValidation sv);	*setSubjectValidation*: 이 오브젝트 그리드 인스턴스의 SubjectValidation 플러그인을 설정합니다. 이 플러그인을 사용하여 오브젝트 그리드로 전달된 javax.security.auth.Subject 대상(subject)이 변경되지 않은 유효한 대상(subject)인지 유효성을 검증할 수 있습니다. Subject 오브젝트가 변경되었는지 여부를 작성자만 알고 있기 때문에 이 플러그인을 구현하려면 Subject 오브젝트 작성자의 지원이 필요합니다. 그러나 대상(subject) 작성자가 Subject가 변경되었는지 여부를 모를 수도 있습니다. 이 경우, 이 플러그인을 사용하면 안 됩니다.

오브젝트 그리드 인터페이스: 플러그인

BackingMap 인터페이스에는 보다 확장 가능한 상호 작용을 위해 여러 개의 선택적 플러그 지점이 있습니다.

```
void addEventListener(ObjectGridEventListener cb);
void setEventListeners(List cbList);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
int reserveSlot(String);
// Security related plug-ins
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
void setMapAuthorization(MapAuthorization mapAuthorization);
```

- *ObjectGridEventListener*: ObjectGridEventListener 인터페이스는 오브젝트 그리드에서 중요 이벤트가 발생할 때 알림을 수신하는 데 사용됩니다. 이런 이벤트에는 오브젝트 그리드 초기화, 트랜잭션 시작, 트랜잭션 종료 및 오브젝트 그리드 제거가 포함됩니다. 이런 이벤트를 청취하려면 ObjectGridEventListener 인터페이스를 구현하는 클래스를 작성하고 이를 오브젝트 그리드에 추가하십시오. 이 리스너는 각 Session과 연관됩니다. 자세한 정보는 200 페이지의 『리스너』 및 123 페이지의 『세션 인터페이스』를 참조하십시오.
- *TransactionCallback*: TransactionCallback 리스너 인터페이스는 시작, 확약 및 롤백 신호와 같은 트랜잭션 이벤트를 이 인터페이스로 전송할 수 있도록 합니다. 일반적으로 TransactionCallback 리스너 인터페이스는 로더와 함께 사용됩니다. 자세한 정보는 234 페이지의 『TransactionCallback 플러그인』 및 217 페이지의 『로더』를 참조하십시오. 이런 이벤트를 사용하여 외부 자원과 또는 다중 로더 내에서 트랜잭션을 조정할 수 있습니다.
- *reserveSlot*: 이 오브젝트 그리드의 플러그인이 TxID와 같은 슬롯을 가지는 오브젝트 인스턴스에서 사용하도록 슬롯을 예약할 수 있습니다.

- *SubjectValidation*. 보안 사용이 가능한 경우, 이 플러그인을 사용하여 오브젝트 그리드로 전달되는 javax.security.auth.Subject 클래스의 유효성을 검증할 수 있습니다.
- *MapAuthorization*. 보안 사용이 가능한 경우, 이 플러그인을 사용하여 Subject 오브젝트가 나타내는 프린시펄에 대한 ObjectMap 액세스 권한을 부여할 수 있습니다.
- *SubjectSource*. 보안 사용이 가능한 경우, 이 플러그인을 사용하여 오브젝트 그리드 클라이언트를 나타내는 Subject 오브젝트를 가져올 수 있습니다. 그런 다음 이 대상(subject)은 오브젝트 그리드 권한 부여에 사용됩니다.

BackingMap 인터페이스

각 오브젝트 그리드 인스턴스는 BackingMap 오브젝트 콜렉션을 포함하고 있습니다.

각 BackingMap은 오브젝트 그리드 인터페이스의 defineMap 메소드 또는 createMap 메소드를 사용하여 이름 지정되고 오브젝트 그리드 인스턴스에 추가됩니다. 이 메소드는 개별 Map의 동작을 정의하는 데 사용되는 BackingMap 인스턴스를 리턴합니다. 자세한 정보는 114 페이지의 『오브젝트 그리드 인터페이스』를 참조하십시오.

세션 인터페이스는 트랜잭션을 시작하고 응용프로그램과 BackingMap 오브젝트 간의 트랜잭션 방식의 상호 작용을 수행하는 데 필요한 ObjectMap 또는 JavaMap을 얻는 데 사용됩니다. 그러나 트랜잭션 변경사항은 트랜잭션이 확약된 후에 BackingMap 오브젝트에 적용됩니다. BackingMap을 개별 Map에 대해 확약된 데이터의 인메모리 캐시로 간주할 수 있습니다. 세션 인터페이스에 대한 자세한 정보는 세션 인터페이스를 참조하십시오.

com.ibm.websphere.objectgrid.BackingMap 인터페이스는 BackingMap 속성 설정 메소드를 제공합니다. 일부 set 메소드는 사용자 정의 설계된 여러 플러그인을 통해 BackingMap 확장성을 허용합니다. 다음은 속성을 설정하고 사용자 정의 설계된 플러그인 지원을 제공하기 위한 set 메소드 목록입니다.

```
// For setting BackingMap attributes.
public void setReadOnly(boolean readOnlyEnabled);
public void setNullValuesSupported(boolean nullValuesSupported);
public void setLockStrategy( LockStrategy lockStrategy );
public void setCopyMode(CopyMode mode, Class valueInterface);
public void setCopyKey(boolean b);
public void setNumberOfBuckets(int numBuckets);
public void setNumberOfLockBuckets(int numBuckets);
public void setLockTimeout(int seconds);
public void setTimeToLive(int seconds);
public void setTtlEvictorType(TTLType type);
// For setting an optional custom plug-in provided by application.
public abstract void setObjectTransformer(ObjectTransformer t);
public abstract void setOptimisticCallback(OptimisticCallback checker);
public abstract void setLoader(Loader loader);
public abstract void setPreloadMode(boolean async);
public abstract void setEvictor(Evictor e);
public void setMapEventListeners( List /*MapEventListener*/ eventListenerList );
public void addMapEventListener(MapEventListener eventListener );
public void removeMapEventListener(MapEventListener eventListener );
```

```
public void addMapIndexPlugin(MapIndexPlugin index);
public void setMapIndexPlugins(List /* MapIndexPlugin */ indexList );
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb);
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback cb);
public void removeDynamicIndex(String name);
```

나열된 각 set 메소드에 대해 대응하는 get 메소드가 존재합니다.

BackingMap 속성

각 BackingMap에는 BackingMap 동작을 수정하거나 제어하기 위해 설정할 수 있는 다음 속성이 있습니다.

- *ReadOnly* 속성. 이 속성은 Map이 읽기 전용 Map인지 읽기 및 쓰기 Map인지 여부를 표시합니다. Map에 대해 이 속성이 설정되지 않은 경우, Map은 기본적으로 읽기 및 쓰기 Map으로 설정됩니다. BackingMap이 읽기 전용으로 설정된 경우, 오브젝트 그리드는 적절할 때 읽기 전용으로 성능을 최적화합니다.
- *NullValuesSupported* 속성. 이 속성은 Map에 널값을 넣을 수 있는지 여부를 표시합니다. 이 속성이 설정되지 않은 경우, Map은 널값을 지원하지 않습니다. Map이 널값을 지원하지 않는 경우, 널을 리턴하는 get 조작은 값이 널이거나 맵이 get 조작으로 지정한 키를 포함하지 않고 있음을 의미할 수 있습니다.
- *LockStrategy* 속성. 이 속성은 이 BackingMap이 잠금 관리자를 사용하는지 여부를 결정합니다. 잠금 관리자를 사용하는 경우, LockStrategy 속성은 맵 항목 잠금에 잠금 중 변경이 예상되지 않는 접근이 사용되는지 잠금 중 변경이 예상되는 접근이 사용되는지 여부를 표시하는 데 사용됩니다. 이 속성이 설정되지 않은 경우, 잠금 중 변경이 예상되지 않는 계획(optimistic LockStrategy)이 사용됩니다. 지원되는 잠금 계획에 관한 세부사항은 140 페이지의 『잠금』 주제를 참조하십시오.
- *CopyMode* 속성. 이 속성은 트랜잭션의 확약 주기 동안 맵에서 값을 읽거나 BackingMap에 값을 넣을 때 BackingMap이 값 오브젝트의 사본을 작성하는지 여부를 결정합니다. 응용프로그램이 성능과 데이터 무결성 사이에서 취사 선택할 수 있도록 다양한 복사 모드가 지원됩니다. 이 속성이 설정되지 않은 경우, `COPY_ON_READ_AND_COMMIT` 복사 모드가 사용됩니다. 이 복사 모드는 최상의 성능을 제공할 수 없지만 데이터 무결성 문제점을 최대한 보호합니다. 복사 모드에 대한 자세한 정보는 copyMode 메소드 우수 사례를 참조하십시오.
- *CopyKey* 속성. 이 속성은 맵에 항목을 처음 작성할 때 BackingMap이 키 오브젝트의 복사본을 만들지 판별합니다. 키는 일반적으로 변경할 수 없는 오브젝트이므로 기본 조치는 키 오브젝트의 복사본을 만들지 않는 것입니다.
- *NumberOfBuckets* 속성. 이 속성은 BackingMap이 사용할 해시 버켓 수를 표시합니다. BackingMap 구현은 구현 시 해시 맵을 사용합니다. BackingMap에 다수의 항목이 존재하는 경우, 버켓이 많을수록 성능이 향상됩니다. 버켓 수가 늘어날수록 동일한 버켓을 가진 키 수는 줄어듭니다. 또한 버켓이 많을수록 동시성이 향상됩니다.

이 속성은 성능을 세밀하게 조정하는 데 유용합니다. 응용프로그램에서 NumberOfBuckets 속성을 설정하지 않는 경우 기본값 503이 사용됩니다.

- *NumberOfLockBuckets* 속성. 이 속성은 이 BackingMap에 잠금 관리자가 사용하는 잠금 버켓 수를 표시합니다. LockStrategy가 OPTIMISTIC 또는 PESSIMISTIC으로 설정된 경우, BackingMap에 대해 잠금 관리자가 작성됩니다. 잠금 관리자는 해시 맵을 사용하여 하나 이상의 트랜잭션에서 잠근 항목을 추적합니다. 해시 맵에 많은 항목이 존재하는 경우, 버켓 수가 늘어날수록 동일한 버켓에서 충돌하는 키 수가 줄어들기 때문에 잠금 버켓이 많을수록 성능이 향상됩니다. 또한 잠금 버켓이 많을수록 동시성이 향상됩니다. LockStrategy 속성이 NONE으로 설정된 경우, 이 BackingMap은 잠금 관리자를 사용하지 않습니다. 이 경우, numberOfLockBuckets 설정은 영향을 주지 않습니다. 이 속성이 설정되지 않은 경우, 기본값 383이 사용됩니다.
- *LockTimeout* 속성. 이 속성은 BackingMap이 잠금 관리자를 사용 중인 경우에 사용됩니다. BackingMap은 LockStrategy 속성이 OPTIMISTIC 또는 PESSIMISTIC으로 설정된 경우 잠금 관리자를 사용합니다. 속성 값은 초 단위이며, 잠금 관리자가 잠금이 부여되기를 기다리는 시간을 결정합니다. 이 속성이 설정되지 않은 경우, LockTimeout 값으로 15초가 사용됩니다. 발생할 수 있는 잠금 대기 제한시간 예외에 대한 자세한 내용은 잠금 중 변경이 예상됨(Pessimistic)을 참조하십시오.
- *TtlEvictorType* 속성. 모든 BackingMap에는 시간 기준 알고리즘을 사용하여 축출할 맵 항목을 결정하는 TTL(Time to Live) 축출기가 내장되어 있습니다. 기본적으로 내장 TTL(Time to Live) 축출기는 활성화되어 있지 않습니다. 세 개 값 (CREATION_TIME, LAST_ACCESS_TIME 또는 NONE) 중 하나를 사용하여 setTtlEvictorType 메소드를 호출하면 TTL(Time to Live) 축출기를 활성화할 수 있습니다. CREATION_TIME 값은 축출기가 BackingMap에서 맵 항목이 작성된 시간에 TimeToLive 속성을 추가하여 축출기가 BackingMap에서 맵 항목을 축출해야 하는 시기를 판별함을 표시합니다. LAST_ACCESS_TIME 값은 축출기가 응용프로그램이 실행 중인 일부 트랜잭션에서 마지막으로 맵 항목에 액세스한 시간에 TimeToLive 속성을 추가하여 축출기로 맵 항목을 축출할 시기를 판별함을 표시합니다. TimeToLive 속성으로 지정한 기간 동안 트랜잭션이 맵 항목에 액세스하지 않은 경우에만 맵 항목이 축출됩니다. NONE 값은 축출기가 비활성 상태를 유지해야 하고 맵 항목을 축출하지 않아야 함을 표시합니다. 이 속성이 설정되지 않은 경우, NONE이 기본값으로 사용되고 TTL(Time to Live) 축출기가 활성화되지 않습니다. 내장 TTL(Time to Live) 축출기에 대한 자세한 내용은 축출기를 참조하십시오.
- *TimeToLive* 속성. 이 속성은 TtlEvictorType 속성에 대해 설명한 대로 각 항목에 대해 내장 TTL(Time to Live) 축출기가 작성 또는 마지막 액세스 시간에 추가해야 하는 초 수를 지정하는 데 사용됩니다. 이 속성이 설정되지 않은 경우, 특수 값 0이 사용되어 TTL(Time to Live)이 무한대임을 표시합니다. 이 속성이 무한대로 설정된 경우, 축출기는 맵 항목을 축출하지 않습니다.

다음 예는 someGrid 오브젝트 그리드 인스턴스에 someMap BackingMap을 정의하고 BackingMap 인터페이스의 set 메소드를 사용하여 BackingMap의 다양한 속성을 설정하는 방법을 설명합니다.

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("someGrid");
BackingMap bm = objectGrid.getMap("someMap");
bm.setReadOnly( true ); // override default of read/write
bm.setNullValuesSupported(false); // override default of allowing Null values
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // override default of OPTIMISTIC
bm.setLockTimeout( 60 ); // override default of 15 seconds.
bm.setNumberOfBuckets(251); // override default (prime numbers work best)
bm.setNumberOfLockBuckets(251); // override default (prime numbers work best)
...
```

BackingMap 플러그인

BackingMap 인터페이스에는 오브젝트 그리드와의 확장 가능한 상호 작용을 지원하는 여러 개의 선택적 플러그 지점이 있습니다.

- **ObjectTransformer 플러그인**: 일부 맵 조작의 경우, BackingMap이 BackingMap의 항목 값 또는 키를 직렬화, 직렬화 해제 또는 복사해야 할 수도 있습니다. BackingMap은 ObjectTransformer 인터페이스의 기본 구현을 제공하여 이런 조치를 수행할 수 있습니다. 응용프로그램은 BackingMap이 BackingMap의 항목 값 또는 키를 직렬화, 직렬화 해제 또는 복사하는 데 사용하는 사용자 정의 설계된 ObjectTransformer 플러그인을 제공하여 성능을 향상시킬 수 있습니다. 자세한 정보는 229 페이지의 『ObjectTransformer 플러그인』을 참조하십시오.
- **Evictor 플러그인**: 내장 TTL(Time to Live) 축출기는 기준 시간 알고리즘을 사용하여 BackingMap에서 항목을 축출해야 하는 시기를 결정합니다. 일부 응용프로그램은 다른 알고리즘을 사용하여 BackingMap에서 항목을 축출해야 하는 시기를 결정해야 할 수도 있습니다. Evictor 플러그인은 사용할 BackingMap에 사용자 정의 설계된 축출기를 사용할 수 있게 합니다. 내장 TTL(Time to Live) 축출기 이외에 Evictor 플러그인이 있습니다. 이 플러그인은 TTL(Time to Live) 축출기를 대체하지 않습니다. 오브젝트 그리드는 ″가장 오래 전에 사용″ 또는 ″가장 적게 사용″과 같은 잘 알려진 알고리즘을 구현하는 사용자 정의 Evictor 플러그인을 제공합니다. 응용프로그램은 제공된 Evictor 플러그인 중 하나를 플러그인하거나 고유의 Evictor 플러그인을 제공할 수 있습니다. 자세한 정보는 206 페이지의 『축출기』를 참조하십시오.
- **MapEventListener 플러그인**: 응용프로그램은 맵 항목 축출 또는 BackingMap 완료 사전 로드와 같은 BackingMap 이벤트에 대해 알려고 할 수 있습니다. BackingMap은 MapEventListener 플러그인에서 메소드를 호출하여 응용프로그램에 BackingMap 이벤트를 알립니다. 응용프로그램은 setMapEventListener 메소드를 사용하여 다양한 BackingMap 이벤트에 대한 알림을 수신하여 하나 이상의 사용자 정

의 설계된 MapEventListener 플러그인을 BackingMap에 제공할 수 있습니다. 응용프로그램은 addMapEventListener 메소드 또는 removeMapEventListener 메소드를 사용하여 나열된 MapEventListener 오브젝트를 수정할 수 있습니다. 자세한 정보는 203 페이지의 『MapEventListener 인터페이스』를 참조하십시오.

- **Loader 플러그인**: BackingMap은 Map의 인메모리 캐시입니다. Loader 플러그인은 BackingMap이 메모리 간에 데이터를 이동하는 데 사용하는 옵션이며, BackingMap의 지속적 저장에 사용됩니다. 예를 들어, JDBC(Java Database Connectivity) Loader를 사용하여 관계형 데이터베이스의 하나 이상의 관계형 테이블 및 BackingMap 간에 데이터를 이동할 수 있습니다. 관계형 데이터베이스를 BackingMap의 지속적 저장으로 사용하지 않아도 됩니다. Loader를 사용하여 BackingMap과 파일 간에, BackingMap과 Hibernate 맵 간에, BackingMap과 J2EE(Java 2 Platform, Enterprise Edition) 엔티티 Bean 간에, BackingMap과 다른 Application Server 간에 데이터를 이동할 수도 있습니다. 응용프로그램은 사용자 정의 설계된 Loader 플러그인을 제공하여 BackingMap과 사용되는 모든 기술의 지속적 저장 간에 데이터를 이동해야 합니다. Loader가 제공되지 않는 경우, BackingMap은 단순 인메모리 캐시가 됩니다. 이 플러그인에 대한 자세한 정보는 217 페이지의 『로더』를 참조하십시오.
- **OptimisticCallback 플러그인**: BackingMap의 LockStrategy 속성이 OPTIMISTIC로 설정된 경우, BackingMap 또는 Loader 플러그인은 맵 값에 대한 비교 조작을 수행해야 합니다. BackingMap 및 Loader는 OptimisticCallback 플러그인을 사용하여 변경이 예상되지 않는 버전화 비교 조작을 수행합니다. 자세한 정보는 242 페이지의 『OptimisticCallback 인터페이스』를 참조하십시오.
- **MapIndexPlugin 플러그인**: MapIndexPlugin 플러그인 또는 축약형 색인은 BackingMap이 저장된 오브젝트의 지정된 속성에 따라 색인을 빌드하기 위해 사용하는 옵션입니다. 응용프로그램은 색인을 사용하여 특정 값 또는 범위 값으로 오브젝트를 찾을 수 있습니다. 정적 및 동적과 같은 두 가지 유형의 색인이 있습니다. 자세한 정보는 256 페이지의 『색인 지정』을 참조하십시오.

세션 인터페이스

이 섹션에서는 응용프로그램이 세션 인터페이스를 사용하여 트랜잭션을 시작 및 종료하는 방법을 설명합니다. 세션 인터페이스에서는 응용프로그램 기반 ObjectMap 및 JavaMap 인터페이스의 액세스도 제공합니다.

소개

각 ObjectMap 또는 JavaMap 인스턴스는 특정 Session 오브젝트에 직접 연결되어 있습니다. 오브젝트 그리드에 액세스할 각 스레드는 먼저 오브젝트 그리드 오브젝트에서 Session을 확보해야 합니다. Session 인스턴스는 스레드 사이에서 동시에 공유될 수 없습니다. 오브젝트 그리드는 스레드 로컬 기억장치를 사용하지 않지만 플랫폼 제한사항 때문에 스레드 사이에서 Session을 전달할 기회가 제한될 수 있습니다.

메소드

다음은 세션 인터페이스에서 사용 가능한 메소드입니다. 다음 메소드에 대한 자세한 정보는 API 문서를 참조하십시오.

```
public interface Session {
 ObjectMap getMap(String cacheName)
 throws UndefinedMapException;
 void begin()
 throws TransactionAlreadyActiveException, TransactionException;
 void beginNoWriteThrough()
 throws TransactionAlreadyActiveException, TransactionException;
 public void commit()
 throws NoActiveTransactionException, TransactionException;
 public void rollback()
 throws NoActiveTransactionException, TransactionException;
 public void flush()
 throws TransactionException; ObjectGrid getObjectGrid();
 TxID getTxID()
 throws NoActiveTransactionException;
 boolean isWriteThroughEnabled();
 void setTransactionType(String tranType);
 public void processLogSequence(LogSequence logSequence)
 throws NoActiveTransactionException, UndefinedMapException, ObjectGridException;

public ObjectGrid getObjectGrid();
public void setTransactionTimeout(int timeout);
public int getTransactionTimeout();
public boolean transactionTimedOut();
public boolean isCommitting();
public boolean isFlushing();
public void markRollbackOnly(Throwable t) throws NoActiveTransactionException;
public boolean isMarkedRollbackOnly();
}
```

Get 메소드

응용프로그램은 ObjectGrid.getSession 메소드를 사용하여 오브젝트 그리드 오브젝트에서 Session 인스턴스를 확보합니다. 다음 코드 스니펫은 Session 인스턴스를 확보하는 방법을 설명합니다.

```
ObjectGrid objectGrid = ...;
Session sess = objectGrid.getSession();
```

Session을 확보하면 스레드에서 사용할 목적으로 세션에 대한 참조를 보관합니다. getSession 메소드를 여러 번 호출하면 항상 새 Session 오브젝트를 리턴합니다.

트랜잭션 및 세션 메소드

Session을 사용하여 트랜잭션을 시작, 확약 또는 롤백할 수 있습니다. ObjectMap 및 JavaMap을 사용하는 BackingMap에서의 조작은 Session 트랜잭션에서 가장 효과적으로 수행됩니다. 트랜잭션을 시작하면 해당 트랜잭션 범위의 임의의 BackingMap 변경사항이 트랜잭션을 확약할 때까지 특수 트랜잭션 캐시에 저장됩니다. 트랜잭션을 확약하면 보류 중인 변경사항이 BackingMap 및 Loader에 적용되고 해당 오브젝트 그리드의 다른 클라이언트에 표시됩니다.

또한 오브젝트 그리드는 트랜잭션을 자동으로 확약하는 기능(자동 확약이라고도 함)을 지원합니다. 임의의 ObjectMap 조작이 활성 트랜잭션 컨텍스트 외부에서 수행되는 경우 조작보다 먼저 암시적 트랜잭션을 시작하고 응용프로그램에 대한 제어를 리턴하기 전에 먼저 트랜잭션을 자동으로 확약합니다.

```
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
session.begin();
objectMap.insert("key1", "value1");
objectMap.insert("key2", "value2");
session.commit();
objectMap.insert("key3", "value3"); // auto?commit
```

Session.flush 메소드

Session.flush 메소드는 Loader를 BackingMap과 연관시킬 경우에만 적용됩니다. flush 메소드는 트랜잭션 캐시의 현재 변경사항 세트를 사용하여 Loader를 호출합니다. Loader는 변경사항을 백엔드에 적용합니다. 이 변경사항은 flush를 호출하는 경우 확약되지 않습니다. flush 호출 후 Session 트랜잭션이 확약된 경우, flush 호출 이후 수행된 갱신만 Loader에 적용됩니다. flush 호출 후 Session 트랜잭션이 롤백된 경우에는 트랜잭션에서 보류 중인 모든 다른 변경사항과 함께 비워진 변경사항을 버립니다. Flush 메소드는 로더에서 일괄처리 조작의 기회를 제한하므로 필요한 경우에만 Flush 메소드를 사용하십시오. 다음은 Session.flush 메소드 사용법에 대한 예입니다.

```
Session session = objectGrid.getSession();
session.begin();
// make some changes
...
session.flush(); // push these changes to the Loader, but don't commit yet
// make some more changes
...
session.commit();
```

비동시 기록 메소드

일부 오브젝트 그리드 맵은 맵의 데이터를 저장할 지속적 기억장치를 제공하는 로더에서 지원됩니다. 종종 오브젝트 그리드 맵에서만 데이터를 확약하고 로더에 데이터를 푸시(push)하지 말아야 할 필요가 있습니다. 세션 인터페이스는 이 목적으로 beginNoWriteThough 메소드를 제공합니다. beginNoWriteThrough 메소드는 begin 메소드와 같이 트랜잭션을 시작합니다. beginNoWriteThrough 메소드를 사용하는 경우 트랜잭션을 확약하면 오브젝트 그리드 인메모리 맵에서만 데이터가 확약되고 로더에서 제공하는 지속적 기억장치에서는 확약되지 않습니다. 이 메소드는 맵에서 데이터 사전 로드를 수행하는 경우에 매우 유용합니다.

분산 오브젝트 그리드 인스턴스를 사용할 때 beginNoWriteThrough 메소드는 서버에서 격리된 캐시는 수정하지 않고 주변 캐시에만 변경사항을 작성할 경우 유용합니다. 데이터가 주변 캐시에서 무효로 알려진 경우 beginNoWriteThrough 메소드를 사용하면 항목을 서버에서 무효화하지 않고 주변 캐시에서 항목을 무효화할 수 있습니다.

세션 인터페이스에서는 현재 활성 상태인 트랜잭션 유형을 판별하는 isWriteThroughEnabled 메소드도 제공합니다.

```
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// make some changes ...
session.commit(); // these changes will not get pushed to the Loader
```

TxID 오브젝트 메소드 얻기

TxID 오브젝트는 활성 트랜잭션을 식별하는 불투명 오브젝트입니다. 다음의 경우 TxID 오브젝트를 사용하십시오.

- 특정 트랜잭션을 찾을 때 비교하기 위해
- TransactionCallback 및 Loader 오브젝트 사이에서 공유 데이터를 저장하기 위해

오브젝트 슬롯 기능에 대한 자세한 정보는 234 페이지의 『TransactionCallback 플러그인』 및 217 페이지의 『로더』를 참조하십시오.

성능 모니터 메소드에서 트랜잭션 유형 설정

WebSphere Application Server 응용프로그램 서버에서 오브젝트 그리드를 사용하는 경우 성능 모니터링에서 트랜잭션 유형을 재설정해야 할 수도 있습니다. setTransactionType 메소드를 사용하여 트랜잭션 유형을 설정할 수 있습니다. setTransactionType 메소드에 대한 자세한 정보는 318 페이지의 『WebSphere Application Server PMI(Performance Monitoring Infrastructure)에서 오브젝트 그리드 성능 모니터링』을 참조하십시오.

전체 LogSequence 메소드 처리

오브젝트 그리드는 JVM(Java Virtual Machine) 사이에서 맵을 분배하는 수단으로 다른 오브젝트 그리드 리스터에 맵 변경사항 세트를 전달할 수 있습니다. 리스너에서 수신한 LogSequence를 쉽게 처리하도록 세션 인터페이스에서 processLogSequence 메소드를 제공합니다. 이 메소드는 LogSequence의 각 LogElement를 점검하여 LogSequence MapName으로 식별된 BackingMap에서 적절한 조작(예: insert, update, invalidate 등)을 수행합니다. processLogSequence 메소드를 호출하기 전에 먼저 오브젝트 그리드 세션이 활성화되어야 합니다. 또한 응용프로그램에서 Session을 완료하도록 적절한 commit 또는 rollback 호출을 발행해야 합니다. 자동 확약 처리는 이 메소드 호출 시 사용할 수 없습니다.

원격 JVM에서 수신한 ObjectGridEventListener에서의 표준 처리는 변경사항의 무한 전달을 방지하는 beginNoWriteThrough 메소드를 사용하여 Session을 시작합니다. 그 다음 이 processLogSequence 메소드를 호출한 후 트랜잭션을 확약 또는 롤백합니다.

```
// Use the Session object that was passed in during
//ObjectGridEventListener.initialization...
session.beginNoWriteThrough();
```

```
// process the received LogSequence
try {
 session.processLogSequence(receivedLogSequence);
}catch (Exception e) {
 session.rollback(); throw e;
}
// commit the changes
session.commit();
```

markRollbackOnly 메소드

이 메소드는 현재 트랜잭션을 "롤백만"으로 표시하는 데 사용됩니다. 트랜잭션을 "롤백만"으로 표시하면 응용프로그램에서 commit 메소드를 호출한 경우에도 트랜잭션은 롤백됩니다. 일반적으로 이 메소드는 오브젝트 그리드 자체에서 사용하거나 트랜잭션의 확약이 허용된 경우 데이터 손상이 발생할 수 있음을 알고 있을 때 응용프로그램에서 사용합니다.

이 메소드가 호출된 후 이 메소드에 전달된 throwable 오브젝트는 이전에 "롤백만"으로 표시된 세션에서 commit 메소드가 호출된 경우 이 메소드에 의해 생성되는 com.ibm.websphere.objectgrid.TransactionException 예외에 체인됩니다. 이미 "롤백만"으로 표시된 트랜잭션에 대한 이 메소드의 후속 호출은 무시됩니다. 즉, 널이 아닌 throwable 참조를 전달하는 첫 번째 호출만 사용됩니다. 표시된 트랜잭션이 완료되면 "롤백만" 표시가 제거되어 세션에 의해 시작되는 다음 트랜잭션이 확약될 수 있습니다.

isMarkedRollbackOnly 메소드

세션이 현재 "롤백만"으로 표시된 경우 리턴합니다. markRollbackOnly 메소드가 이 세션에서 이전에 호출되었고 해당 세션에서 시작한 트랜잭션이 여전히 활성인 경우에만 이 메소드는 부울 true를 리턴합니다.

setTransactionTimeout 메소드

이 세션에서 시작하는 다음 트랜잭션의 트랜잭션 제한시간을 지정된 수(초)로 설정하십시오. 이 메소드는 이전에 이 세션에서 시작한 트랜잭션의 트랜잭션 제한시간에는 영향을 주지 않습니다. 이 메소드가 호출된 후 시작되는 트랜잭션에만 영향을 줍니다. 이 메소드가 호출되지 않으면 com.ibm.websphere.objectgrid.ObjectGrid 메소드의 setTxTimeout 메소드로 전달된 제한시간 값이 사용됩니다.

getTransactionTimeout 메소드

이 메소드는 트랜잭션 제한시간 값을 초 단위로 리턴합니다. 이 메소드는 setTransactionTimeout 메소드에 제한시간 값으로 전달된 마지막 값을 리턴합니다. setTransactionTimeout 메소드가 호출되지 않으면 com.ibm.websphere.objectgrid.ObjectGrid 메소드의 setTxTimeout 메소드로 전달된 제한시간 값이 사용됩니다.

transactionTimedOut

이 세션에서 시작된 현재 트랜잭션이 제한시간을 초과한 경우 이 메소드는 부울 true를 리턴합니다.

isFlushing 메소드

호출 중인 세션 인터페이스의 flush 메소드 결과로 모든 트랜잭션 변경사항이 로더 플러그인으로 비워지는 경우에만 이 메소드는 부울 true를 리턴합니다. 로더 플러그인은 batchUpdate 메소드가 호출된 이유를 알아야 할 때 이 메소드가 유용함을 알 수 있습니다.

isCommitting 메소드

호출 중인 세션 인터페이스의 commit 메소드 결과로 모든 트랜잭션 변경사항이 확약되는 경우에만 이 메소드는 부울 true를 리턴합니다. 로더 플러그인은 batchUpdate 메소드가 호출된 이유를 알아야 할 때 이 메소드가 유용함을 알 수 있습니다.

ObjectMap 및 JavaMap 인터페이스

이 주제에서는 ObjectMap 및 JavaMap 인터페이스를 사용하여 응용프로그램이 오브젝트 그리드와 상호 작용하는 방법을 설명합니다. 위 두 인터페이스는 응용프로그램과 BackingMap 사이의 트랜잭션 상호 작용에 사용됩니다.

ObjectMap 인터페이스

현재 스레드에 대응하는 Session 오브젝트에서 ObjectMap 인스턴스를 얻습니다. ObjectMap 인터페이스는 응용프로그램이 BackingMap에서 항목에 대한 변경사항을 작성하는 데 사용하는 기본 수단입니다.

ObjectMap 인스턴스 확보

응용프로그램은 Session.getMap(String) 메소드를 사용하여 Session 오브젝트에서 ObjectMap 인스턴스를 가져옵니다. 다음 코드 스니펫은 ObjectMap 인스턴스를 얻는 방법을 설명합니다.

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
```

각 ObjectMap 인스턴스는 특정 Session 오브젝트에 대응합니다. 특정 Session 오브젝트에서 동일한 BackingMap 이름을 사용하여 getMap 메소드를 여러 번 호출하면 항상 동일한 BackingMap 인스턴스가 리턴됩니다.

자동 확약 트랜잭션

앞서 설명한 대로 ObjectMap 및 JavaMap을 사용하는 BackingMap에 대한 조작은 Session 트랜잭션 내에서 가장 효율적으로 수행됩니다. 오브젝트 그리드는 ObjectMap 및 JavaMap 인터페이스의 메소드가 Session 트랜잭션 외부에서 호출될 때 자동 확약 지원을 제공합니다. 메소드는 내부 트랜잭션을 시작하고 요청된 조작을 수행하며, 내부 트랜잭션을 확약합니다.

메소드 시멘틱

다음은 ObjectMap 및 JavaMap 인터페이스의 각 메소드 이면의 시멘틱에 대한 설명입니다. setDefaultKeyword 메소드, invalidateUsingKeyword 메소드 및 Serializable 인수를 갖는 메소드는 132 페이지의 『키워드』 주제에 설명되어 있습니다. setTimeToLive 메소드는 206 페이지의 『축출기』 주제에 설명되어 있습니다. 이런 메소드에 대한 자세한 정보는 자세한 정보는 API 문서를 참조하십시오.

containsKey 메소드

키가 BackingMap 또는 로더에 값을 갖는지 여부를 판별합니다. 응용프로그램이 널값을 지원하는 경우, 이 메소드를 사용하여 get 조작에서 리턴된 널 참조가 널값을 참조하는지 또는 BackingMap 및 로더가 키를 포함하지 않는지 여부를 판별할 수 있습니다.

flush 메소드

이 메소드의 시멘틱은 세션 인터페이스의 flush 메소드와 유사합니다. 두드러진 차이점은 Session flush는 현재 세션에서 수정된 모든 맵에 대해 현재 보류 중인 변경사항을 적용한다는 점입니다. 이 메소드를 사용하면 이 ObjectMap의 변경사항만 로더로 비워집니다.

get 메소드

BackingMap에서 항목을 페치합니다. BackingMap에서 항목을 찾을 수 없지만 로더가 BackingMap과 연관된 경우, 로더에서 항목을 페치하려고 시도합니다. 일괄처리 페치 처리를 허용할 수 있도록 getAll 메소드가 제공됩니다.

getForUpdate 메소드

getforUpdate 메소드는 get 메소드와 유사하지만 getForUpdate 메소드를 사용하면 항목 갱신 의도가 있음을 BackingMap 및 Loader에 알립니다. Loader는 이 힌트를 사용하여 데이터베이스 백엔드에 `SELECT for UPDATE` 조회를 발행할 수 있습니다. 잠금 중 변경이 예상되는 계획(Pessimistic LockingStrategy)이 BackingMap에 대해 정의된 경우, 잠금 관리자가 항목을 잠급니다. 일괄처리 페치 처리를 허용할 수 있도록 getAllForUpdate 메소드가 제공됩니다.

insert 메소드

BackingMap 및 Loader에 항목을 삽입합니다. 이 메소드를 사용하면 이전에 존재하지 않는 항목을 삽입하려 함을 BackingMap 및 Loader에 알립니다. 기존 항목에서 이 메소드를 호출하는 경우, 메소드가 호출될 때 또는 현재 트랜잭션이 확약될 때 예외가 발생합니다.

invalidate 메소드

invalidate 메소드의 시멘틱은 메소드로 전달되는 **isGlobal** 매개변수 값에 따라 다릅니다. 일괄처리 무효화 처리를 허용할 수 있도록 invalidateAll 메소드가 제공됩니다.

false 값이 invalidate 메소드의 **isGlobal** 매개변수로 전달될 때 로컬 무효화가 지정됩니다. 로컬 무효화는 트랜잭션 캐시에 있는 항목 변경사항을 버립니다. 응용프로그램이 get 메소드를 발행하는 경우, BackingMap에 있는 마지막으로 확약된 값에서 항목이 페치됩니다. BackingMap에 항목이 없는 경우, Loader에 있는 마지막으로 비우거나 확약된 값에서 항목이 페치됩니다. 트랜잭션이 확약될 때 로컬로 무효화됨으로 표시된 항목은 BackingMap에 영향을 주지 않습니다. Loader로 비워진 변경사항은 항목이 무효화되더라도 여전히 확약됩니다.

*true*가 invalidate 메소드의 **isGlobal** 매개변수로 전달될 때 글로벌 무효화가 지정됩니다. 글로벌 무효화는 트랜잭션 캐시에 있는 보류 중인 항목 변경사항을 버리고 항목에 대해 수행되는 후속 조작에서 BackingMap 값을 생략합니다. 트랜잭션이 확약될 때 글로벌로 무효화됨으로 표시된 항목은 BackingMap에서 축출됩니다.

다음 무효화 유스 케이스의 예제를 고려하십시오. 자동 증분 열이 있는 데이터베이스 테이블이 BackingMap을 지원합니다. 증분 열은 레코드에 고유 번호를 지정하는 데 유용합니다. 응용프로그램은 항목을 삽입합니다. 삽입 후에 응용프로그램은 삽입된 행의 순서 번호를 알아야 합니다. 오브젝트 사본이 이전 것임을 알고 있으므로 글로벌 무효화를 사용하여 로더에서 값을 가져옵니다. 다음 코드는 이러한 유스 케이스를 설명합니다.

```
Session sess = objectGrid.getSession();
ObjectMap map = sess.getMap("mymap");
sess.begin();
map.insert("Billy", new Person("Joe", "Bloggs", "Manhattan"));
sess.flush();
map.invalidate("Billy", true);
Person p = map.get("Billy");
System.out.println("Version column is: " + p.getVersion());
map.commit();
```

이 코드 샘플은 *Billy*에 대한 항목을 추가합니다. Person의 버전 속성은 데이터베이스의 자동 증분 열을 사용하여 설정됩니다. 응용프로그램은 먼저 insert 명령을 수행합니다. 그런 다음, insert가 로더 및 데이터베이스로 전송되도록 flush를 발행합니다. 데이터베이스는 버전 열을 순서의 다음 번호로 설정하여 트랜잭션의 Person 오브젝트가 갱신되도록 합니다. 오브젝트를 갱신하기 위해 응용프로그램은 글로벌 무효화를 수행합니다. 발행되는 다음 get 메소드는 트랜잭션의 값을 무시하고 로더에서 항목을 가져옵니다. 갱신된 버전 값으로 항목이 데이터베이스에서 페치됩니다.

put 메소드

put 메소드의 시멘틱은 이전 get 메소드가 키에 대해 트랜잭션에서 호출되었는지 여부에 따라 다릅니다. 응용프로그램이 BackingMap 또는 로더의 기존 항목을 리턴하는 get 조작을 발행하는 경우, put 메소드 호출은 갱신으로 해석되며 트랜잭션의 이전 값을 리턴합니다. 이전 get 메소드 호출 또는 항목을 찾지 못한 이전 get 메소드 호출이 없는 put 메소드 호출은 삽입으로 해석됩니다. insert 및 update 메소드의 시멘틱은 put 조작이 확약될 때 적용됩니다. 일괄처리 삽입 및 갱신 처리가 가능하도록 putAll 메소드가 제공됩니다.

remove 메소드

항목이 플러그인된 경우 BackingMap 및 로더에서 항목을 제거합니다. 제거된 오브젝트의 값은 이 메소드로 리턴됩니다. 오브젝트가 없으면 이 메소드는 널 값을 리턴합니다. 리턴값 없이 일괄처리 삭제 처리가 가능하도록 removeAll 메소드가 제공됩니다.

setCopyMode 메소드

이 ObjectMap의 CopyMode를 지정합니다. 이 메소드를 사용하면 응용프로그램이 BackingMap에 지정된 CopyMode를 대체할 수 있습니다. 지정된 CopyMode는 clearCopyMode 메소드가 호출될 때까지 유효합니다. 두 메소드 모두 트랜잭션 범위 밖에서 호출됩니다. 트랜잭션 도중에 CopyMode를 변경할 수는 없습니다.

touch 메소드

항목에 대한 마지막 액세스 시간을 갱신합니다. 이 메소드는 BackingMap에서 값을 검색하지 않습니다. 자체 트랜잭션으로 이 메소드를 사용하십시오. 제공된 키가 무효화 또는 제거로 인해 BackingMap에 존재하지 않는 경우, 확약 처리 중 예외가 발생합니다.

update 메소드

BackingMap 및 로더에서 항목을 명시적으로 갱신합니다. 이 메소드를 사용하면 기존 항목을 갱신하려 함을 BackingMap 및 로더에 표시합니다. 메소드가 호출될 때 또는 확약 처리 중에 존재하지 않는 항목에 대해 이 메소드를 호출하는 경우 예외가 발생합니다.

getIndex 메소드

BackingMap에 빌드되어 있는 이름 지정된 색인을 확보하려고 시도합니다. 색인은 스레드 간에 공유될 수 없으며 세션과 동일한 규칙으로 작동합니다. 리턴된 색인 오브젝트는 MapIndex 인터페이스, MapRangeIndex 인터페이스 또는 사용자 정의 색인 인터페이스와 같은 올바른 응용프로그램 색인 인터페이스로 캐스트되어야 합니다.

JavaMap 인터페이스

ObjectMap 오브젝트에서 JavaMap 인스턴스를 얻습니다. JavaMap 인터페이스는 ObjectMap과 메소드 서명은 동일하지만 예외 처리는 서로 다릅니다. JavaMap은 java.util.Map 인터페이스를 확장하므로 모든 예외는 java.lang.RuntimeException 클래스의 인스턴스입니다. JavaMap이 java.util.Map 인터페이스를 확장하므로 오브젝트 캐싱에서 java.util.Map 인터페이스를 사용하는 기존 응용프로그램을 통해 오브젝트 그리드를 쉽고 빠르게 사용할 수 있습니다.

JavaMap 인스턴스 얻기

응용프로그램은 ObjectMap.getJavaMap 메소드를 사용하여 ObjectMap 오브젝트에서 JavaMap 인스턴스를 가져옵니다. 다음 코드 스니펫은 JavaMap 인스턴스를 얻는 방법을 설명합니다.

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
java.util.Map map = objectMap.getJavaMap();
JavaMap javaMap = (JavaMap) javaMap;
```

JavaMap을 가져온 ObjectMap이 이를 지원합니다. 특정 ObjectMap을 사용하여 getJavaMap을 여러 번 호출하면 항상 동일한 JavaMap 인스턴스가 리턴됩니다.

지원되는 메소드

JavaMap 인터페이스는 java.util.Map 인터페이스의 메소드 서브세트만 지원합니다. java.util.Map 인터페이스는 다음 메소드를 지원합니다.

- `containsKey(java.lang.Object)`
- `get(java.lang.Object)`
- `put(java.lang.Object, java.lang.Object)`
- `putAll(java.util.Map)`
- `remove(java.lang.Object)`

java.util.Map 인터페이스에서 상속된 모든 기타 메소드로 인해 `java.lang.UnsupportedOperationException` 예외가 발생합니다.

키워드

오브젝트 그리드는 키워드를 기반으로 유연한 무효화 메커니즘을 제공합니다. *키워드*는 직렬화 가능 오브젝트의 널이 아닌 인스턴스입니다. 선택한 방법으로 키워드를 BackingMap 항목과 연관시킬 수 있습니다.

키워드를 항목과 연관

항목 세트를 0개 이상의 키워드와 연관시킬 수 있습니다. get, update, put, insert, touch 메소드 등 항목을 조작하는 ObjectMap 및 JavaMap의 메소드는 모두 단일 키워드를 메소드가 변경하는 모든 항목과 연관시킬 수 있게 하는 버전이 있습니다. 새 키워드 연관은 트랜잭션이 확약될 때까지는 현재 트랜잭션에서만 볼 수 있습니다. 확약 후에 새 연관은 BackingMap에 적용되며, 다른 트랜잭션에서 볼 수 있습니다. 확약 처리 중에 롤백을 일으키는 오류가 발생하거나 사용자가 활성 트랜잭션으로 롤백하는 경우, 새 키워드 연관은 롤백됩니다. 다음 코드는 새 항목을 키워드와 연관시키는 방법을 설명합니다.

```
Session sess = objectGrid.getSession();
ObjectMap map = sess.getMap("MapA");
sess.begin();
map.insert("Billy", new Person("Joe", "Bloggs", "Manhattan"), "New York");
sess.commit();
```

위 코드 예는 새 항목을 BackingMap에 삽입하고 "New York" 키워드와 연관시킵니다. 또한 항목을 삽입하는 응용프로그램은 항목이 검색될 때 키워드를 연관시켜야 합니다. 응용프로그램은 항목을 가져올 때마다 키워드를 항목과 연관시켜야 합니다. 다음 코드 샘플을 고려하십시오.

```
sess.begin();
Person p = (Person)map.get("Billy", "New York");
sess.commit();
```

위 코드 예는 검색된 항목이 "New York" 키워드와 연관되도록 합니다. 응용프로그램은 다중 키워드를 하나의 항목과 연관시킬 수 있지만 메소드 호출당 하나의 키워드만 연관시킬 수 있습니다. 더 많은 키워드를 연관시키려면 다음 샘플과 같이 다른 메소드 호출을 발행하십시오.

```
sess.begin();
Person p = (Person)map.get("Billy", "New York");
map.touch("Billy", "Another keyword");
map.get("Billy", "Yet another keyword");
sess.commit();
```

기본 키워드

ObjectMap 및 JavaMap 인터페이스의 setDefaultKeyword 메소드는 get, insert, put, update 또는 touch 메소드의 키워드 버전을 사용하지 않고 항목을 특정 키워드와 연관시키는 방법을 제공합니다. 메소드의 키워드 버전이 사용되는 경우, 기본 키워드는 무시되고 제공된 키워드 오브젝트가 사용됩니다.

```
sess.begin();
map.setDefaultKeyword("New York");
Person p = (Person)map.get("Billy");
```

```
p = (Person)map.get("Bob", "Los Angeles");
map.setDefaultKeyword(null);
p = (Person)map.get("Jimmy");
sess.commit();
```

위 예에서 Billy는 기본 키워드인 ″New York″과 연관됩니다. Bob 항목을 검색하기 위한 get 호출로 명시적 키워드가 전달되었기 때문에 Bob은 기본 키워드와 연관되지 않습니다. 기본 키워드가 재설정되었고 get 메소드 호출로 명시적 키워드 인수가 전달되지 않았기 때문에 키워드는 ″Jimmy″와 연관되지 않습니다.

키워드를 사용하여 항목 무효화

ObjectMap 및 JavaMap 인터페이스의 invalidateUsingKeyword 메소드를 사용하면 대응하는 BackingMap의 키워드와 연관된 모든 항목을 무효화합니다. 이 접근 방식을 사용하면 관련된 항목을 단일 조작으로 효율적으로 무효화할 수 있습니다.

```
sess.begin();
map.insert("Billy", new Person("Joe", "Bloggs", "Manhattan"), "New York");
map.invalidateUsingKeyword("New York", false);
map.insert("Bob", new Person("Paul", "Henry", "Albany"), "New York");
sess.commit();
```

위 예에서는 ″Billy″에 대한 항목이 무효화되고 BackingMap에 삽입되지 않습니다. ″Bob″에 대한 항목은 invalidateUsingKeyword 메소드 호출 이후에 삽입되었기 때문에 무효화되지 않습니다. invalidateUsingKeyword 메소드는 메소드가 호출될 때의 키워드 연관을 기반으로 항목을 무효화합니다.

키워드 그룹화

상위-하위 관계로 키워드를 함께 그룹화할 수도 있습니다. 상위 키워드는 여러 개의 하위를 가질 수 있으며, 하위 키워드는 여러 개의 상위를 가질 수 있습니다. 예를 들어, 응용프로그램이 ″Dublin″, ″Paris″, ″New York″ 및 ″Los Angeles″ 키워드를 사용하는 경우 다음 키워드 그룹화를 추가할 수 있습니다.

- ″USA″ 그룹 ″New York″ 및 ″Los Angeles″
- ″Europe″ 그룹 ″Dublin″ 및 ″Paris″
- ″World″ 그룹 ″USA″ 및 ″Europe″

″USA″ 키워드를 무효화하면 ″New York″ 및 ″Los Angeles″ 키워드와 연관된 모든 항목이 무효화됩니다. ″World″ 키워드를 무효화하면 ″USA″ 및 ″Europe″ 그룹화와 연관된 모든 항목이 무효화됩니다. 키워드 연관은 오브젝트 그리드 인터페이스의 associateKeyword 메소드를 사용하여 정의됩니다. invalidateUsingKeyword 메소드 호출 이후에 상위 키워드에 하위 키워드를 추가하면 하위 키워드와 연관된 항목이 무효화되지 않습니다. 다음 코드 예는 설명된 키워드 연관 세트를 정의합니다.

```
ObjectGrid objectGrid = ...;
objectGrid.associateKeyword("USA", "New York");
objectGrid.associateKeyword("USA", "Los Angeles");
```

```
objectGrid.associateKeyword("Europe", "Dublin");
objectGrid.associateKeyword("Europe", "Paris");
objectGrid.associateKeyword("World", "USA");
objectGrid.associateKeyword("World", "Europe");
```

LogElement 및 LogSequence 오브젝트

응용프로그램이 트랜잭션 중에 맵 변경사항을 작성하는 경우, LogSequence 오브젝트가 해당 변경사항을 추적합니다. 응용프로그램이 맵에 있는 항목을 변경하는 경우, 변경 세부사항을 제공할 수 있도록 대응하는 LogElement가 존재합니다. 응용프로그램이 트랜잭션에 대한 확약 또는 비우기를 요구할 때마다 특정 맵에 대한 LogSequence 오브젝트가 로더에 제공됩니다. 로더는 LogSequence 내의 LogElement에 대해 반복되며 각 LogElement를 백엔드에 적용합니다.

오브젝트 그리드에 등록된 ObjectGridEventListener도 LogSequence 오브젝트를 사용합니다. 이 리스너에는 확약된 트랜잭션의 각 맵에 대한 LogSequence 오브젝트가 제공됩니다. 응용프로그램은 일반 데이터베이스의 트리거처럼 이 리스너를 사용하여 특정 항목이 변경되기를 기다릴 수 있습니다.

이 주제에서는 오브젝트 그리드 프레임워크가 제공하는 네 개의 로그 관련 인터페이스 또는 클래스를 설명합니다.

- `com.ibm.websphere.objectgrid.plugins.LogElement`
- `com.ibm.websphere.objectgrid.plugins.LogSequence`
- `com.ibm.websphere.objectgrid.plugins.LogSequenceFilter`
- `com.ibm.websphere.objectgrid.plugins.LogSequenceTransformer`

LogElement 인터페이스

LogElement는 트랜잭션 중에 항목에 대한 조작을 나타냅니다. LogElement 오브젝트는 다음 속성을 보유합니다. 가장 일반적으로 사용되는 속성은 *유형* 및 *현재 값* 속성입니다.

유형 **속성**

유형 로그 요소는 이 로그 요소가 나타내는 조작 유형을 표시합니다. *유형*을 `LogElement` 인터페이스에 정의된 상수 중 하나(`INSERT`, `UPDATE`, `DELETE`, `EVICT`, `FETCH` 또는 `TOUCH`)로 설정할 수 있습니다.

실행 취소 유형 **속성**

트랜잭션이 맵 항목에 작성한 변경사항 이전으로 "실행 취소"하도록 수행해야 하는 조작을 리턴합니다.

현재 값 **속성**

*현재 값*은 `INSERT`, `UPDATE` 또는 `FETCH` 조작의 새 값을 나타냅니다. 조작이

TOUCH, DELETE 또는 EVICT인 경우, 현재 값은 널입니다. ValueInterface를 사용 중인 경우 이 값을 ValueProxyInfo로 캐스트할 수 있습니다.

CacheEntry **속성**

LogElement에서 CacheEntry 오브젝트에 대한 참조를 가져오고 CacheEntry 오브젝트에 정의된 메소드를 사용하여 필요한 정보를 검색할 수 있습니다.

보류 상태 **속성**

*보류 상태*가 true인 경우, 이 로그 요소가 나타내는 변경사항은 로더에 아직 적용되지 않았습니다. false인 경우, 변경사항이 로더에 적용되었습니다(보통 flush 조작을 통해).

버전화된 값 **속성**

버전화된 값은 버전화에 사용할 수 있는 값입니다.

새 키워드 **속성**

새 키워드 콜렉션은 이 항목과 연관된 새 키워드를 포함합니다.

마지막 액세스 시간 **속성**

항목에 대한 마지막 액세스 시간을 나타냅니다.

이미지 전 / 이미지 후 **속성**

Getter 메소드는 변경사항을 맵에 적용하기 전 또는 적용한 후 값 오브젝트의 이미지를 가져오는 데 사용할 수 있습니다.

LogSequence 인터페이스

대부분의 트랜잭션에서 맵에 있는 두 개 이상의 항목에 대한 조작이 발생하므로 다중 LogElement 오브젝트가 작성됩니다. 다중 LogElement 오브젝트의 복합체 역할을 하는 오브젝트가 있을 수도 있습니다. LogSequence 인터페이스는 LogElement 오브젝트 목록을 포함하여 이 목적을 달성합니다. LogSequence 인터페이스에는 다음 메소드가 있습니다.

size 메소드

지정된 순서로 LogElement 오브젝트 수를 리턴합니다.

getAllChanges 메소드

지정된 로그 순서로 모든 변경사항의 반복기를 리턴합니다.

getPendingChanges 메소드

보류 중인 모든 변경사항의 반복기를 리턴합니다. 대개 로더가 지속적 저장에 대한 보류 중인 변경사항만 적용하기 위해 사용합니다.

getChangesByKeys 메소드

입력 매개변수를 기준으로 대상 키를 가진 LogElement 오브젝트의 반복기를 리턴합니다.

getChangesByTypes 메소드

지정된 LogElement 유형을 가진 LogElement 오브젝트의 반복기를 리턴합니다.

getMapName 메소드

변경사항이 적용되는 BackingMap의 이름을 리턴합니다. 호출자는 이 이름을 `Session.getMap(string)` 메소드에 대한 입력으로 사용합니다.

isDirty 메소드

이 LogSequence에 더티(*dirty*) Map인 LogElement가 있는지 여부를 리턴합니다. 즉, LogSequence가 Fetch 또는 Get 이외의 다른 유형의 LogElement 오브젝트를 포함하는 경우 LogSequence는 ″더티(dirty)″로 간주됩니다.

isRollback 메소드

이 LogSequence를 생성하여 트랜잭션을 롤백한 경우 리턴합니다.

getObjectGridName 메소드

이 변경사항이 적용되는 맵을 포함하는 오브젝트 그리드의 이름을 리턴합니다.

LogElement 및 LogSequence는 오브젝트 그리드에서 및 조작이 한 컴포넌트 또는 서버에서 다른 컴포넌트 또는 서버로 전달될 때 사용자가 작성하는 오브젝트 그리드 플러그인에 널리 사용됩니다. 예를 들어, 분산 오브젝트 그리드 트랜잭션 전달 기능이 LogSequence 오브젝트를 사용하여 다른 서버로 변경사항을 전달하거나 로더가 지속적 저장에 이를 적용할 수 있습니다. LogSequence는 주로 다음 인터페이스에서 사용됩니다.

- `com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener`
- `com.ibm.websphere.objectgrid.plugins.Loader`
- `com.ibm.websphere.objectgrid.plugins.Evictor`
- `com.ibm.websphere.objectgrid.Session`

이런 인터페이스에 대한 세부사항은 API 문서를 참조하십시오.

로더 예제

이 섹션에서는 로더에서 LogSequence 및 LogElement 오브젝트를 사용하는 방법을 설명합니다. *Loader*는 지속적 저장에서 데이터를 로드하고 지속적 저장에 데이터를 지속시키는 데 사용됩니다. Loader 인터페이스의 `batchUpdate` 메소드는 LogSequence를 사용합니다.

```
void batchUpdate(TxID txid, LogSequence sequence)
throws LoaderException, OptimisticCollisionException;
```

`batchUpdate` 메소드는 오브젝트 그리드가 Loader에 현재 변경사항을 모두 적용해야 할 때마다 호출됩니다. Loader에는 맵에 대한 LogElement 오브젝트 목록이 제공되며, LogSequence 오브젝트에 캡슐화됩니다. `batchUpdate` 메소드 구현은 변경사항에 대

해 반복되어야 하며 변경사항을 백엔드에 적용해야 합니다. 다음 코드 스니펫은 Loader가 LogSequence 오브젝트를 사용하는 방법을 표시합니다. 스니펫은 변경사항 세트에 대해 반복되며 세 개의 일괄처리 JDBC(Java Database Connectivity) 문(하나는 insert를, 다른 하나는 update를, 나머지 하나는 delete를 갖는 명령문)을 빌드합니다.

```
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
 // Get a SQL connection to use.
 Connection conn = getConnection(tx);
try
{
  // Process the list of changes and build a set of prepared
  // statements for executing a batch update, insert, or delete
  // SQL operations. The statements are cached in stmtCache.
  Iterator iter = sequence.getPendingChanges();
 while ( iter.hasNext() )
{
   LogElement logElement = (LogElement)iter.next();
   Object key = logElement.getCacheEntry().getKey();
   Object value = logElement.getCurrentValue();
switch ( logElement.getType().getCode() )
{
   case LogElement.CODE_INSERT:
   buildBatchSQLInsert( key, value, conn );
break;
   case LogElement.CODE_UPDATE:
   buildBatchSQLUpdate( key, value, conn );
break;
   case LogElement.CODE_DELETE:
   buildBatchSQLDelete( key, conn );
break;
}
}
 // Run the batch statements that were built by above loop.
 Collection statements = getPreparedStatementCollection( tx, conn );
 iter = statements.iterator();
 while ( iter.hasNext() )
{
  PreparedStatement pstmt = (PreparedStatement) iter.next();
  pstmt.executeBatch();
}
}
catch (SQLException e)
{
 LoaderException ex = new LoaderException(e);
 throw ex;
}
}
```

위 샘플은 LogSequence 인수를 처리하는 상위 레벨 로직을 설명하며, SQL `insert`, `update` 또는 `delete` 문 빌드 방법에 대한 세부사항은 설명되어 있지 않습니다. 이 예는 `getPendingChanges` 메소드가 LogSequence 인수에서 호출되어 로더가 처리해야 하는 LogElement 오브젝트의 반복기를 가져오고, LogElement가 SQL `insert`, `update` 또는 `delete` 조작에 대한 것인지 여부를 판별하는 데 `LogElement.getType()`.

getCode() 메소드가 사용됨을 표시합니다.

축출기 샘플

이 예는 축출기에서 LogSequence 및 LogElement를 사용하는 방법을 탐색합니다. 축출기는 특정 기준에 따라 BackingMap에서 맵 항목을 축출하는 데 사용됩니다. Evictor 인터페이스의 apply 메소드는 LogSequence를 사용합니다.

```
/**
* This is called during cache commit to allow the evictor to track object usage
* in a backing map. This will also report any entries that have been successfully
* evicted.
*
* @param sequence LogSequence of changes to the map
*/
void apply(LogSequence sequence);
```

apply 메소드가 LogSequence를 사용하는 방법에 관한 정보는 206 페이지의 『축출기』 주제의 코드 샘플을 참조하십시오.

LogSequenceFilter 및 LogSequenceTransformer 인터페이스

특정 기준을 가진 LogElement 오브젝트만 승인하고 기타 오브젝트는 거부하도록 LogElement 오브젝트를 필터해야 하는 경우가 있습니다. 예를 들어, 일정 기준에 따라 특정 LogElement를 직렬화하려고 할 수 있습니다. LogSequenceFilter는 다음 메소드를 사용하여 이 문제점을 해결합니다.

```
public boolean accept (LogElement logElement);
```

이 메소드는 주어진 LogElement를 조작 시 사용해야 하는 경우에는 true를 리턴하고 주어진 LogElement를 사용하지 않아야 하는 경우에는 false를 리턴합니다.

LogSequenceTransformer는 앞서 설명한 LogSequenceFilter 함수가 이용하는 클래스입니다. LogSequenceFilter를 사용하여 일부 LogElement 오브젝트를 필터한 후 승인된 LogElement 오브젝트를 직렬화합니다. 이 클래스에는 두 개의 메소드가 있습니다. 첫 번째 메소드는 다음과 같습니다.

```
public static void serialize(Collection logSequences, ObjectOutputStream stream,
LogSequenceFilter filter, DistributionMode mode)
throws IOException
```

호출자는 이 메소드를 사용하여 직렬화 프로세스에 포함시킬 LogElement를 판별하기 위한 필터를 제공할 수 있습니다. 호출자는 **DistributionMode** 매개변수를 사용하여 직렬화 프로세스를 제어할 수 있습니다. 예를 들어, 분배 모드가 무효화 전용인 경우에는 값을 직렬화할 필요가 없습니다. 이 클래스의 두 번째 메소드는 다음과 같습니다.

```
public static Collection inflate(ObjectInputStream stream, ObjectGrid objectGrid)
throws IOException, ClassNotFoundException.
```

이 메소드는 serialize 메소드로 작성된 로그 순서 직렬화 양식을 제공된 오브젝트 입력 스트림에서 읽습니다.

잠금

이 주제에서는 오브젝트 그리드 BackingMap이 지원하는 잠금 계획을 설명합니다.

다음 잠금 계획 중 하나를 사용하도록 각 BackingMap을 구성할 수 있습니다.

- 잠금 중 변경이 예상됨(Pessimistic)
- 잠금 중 변경이 예상되지 않음(Optimistic)
- 없음(None)

다음은 map1, map2 및 map3 BackingMap에서 잠금 계획을 설정할 수 있는 방법의 예인데, 각 맵은 서로 다른 잠금 계획을 사용하고 있습니다.

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm = og.defineMap("map2");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

java.lang.IllegalStateException 예외가 발생하지 않도록 하려면 오브젝트 그리드 인스턴스에서 initialize 또는 getSession 메소드를 호출하기 전에 setLockStrategy 메소드를 호출해야 합니다.

PESSIMISTIC 또는 OPTIMISTIC 잠금 계획을 사용하는 경우, BackingMap에 대해 잠금 관리자가 작성됩니다. 잠금 관리자는 해시 맵을 사용하여 하나 이상의 트랜잭션에서 잠근 항목을 추적합니다. 해시 맵에 다수의 맵 항목이 존재하는 경우, 잠금 버켓이 많을수록 성능이 향상됩니다. 버켓 수가 늘어날수록 Java 동기화 충돌 위험이 줄어듭니다. 또한 잠금 버켓이 많을수록 동시성이 향상됩니다. 다음 예는 응용프로그램이 주어진 BackingMap에 사용할 잠금 버켓 수를 설정하는 방법을 표시합니다.

```
bm.setNumberOfLockBuckets( 503 );
```

다시, java.lang.IllegalStateException 예외가 발생하지 않도록 하려면 오브젝트 그리드 인스턴스에서 initialize 또는 getSession 메소드를 호출하기 전에 setNumberOfLockBuckets 메소드를 호출해야 합니다. setNumberOfLockBuckets 메소드 매개변수는 사용할 잠금 버켓 수를 지정하는 Java 기본 정수입니다. 소수를 사용하면 잠금 버켓을 통해 맵 항목을 균등하게 분배할 수 있습니다. 최상의 성능을 위해 잠금 버켓 수를 BackingMap 예상 항목 수의 약 10퍼센트로 설정하여 시작하는 것이 좋습니다.

잠금 중 변경이 예상됨(Pessimistic)

다른 잠금 계획이 가능하지 않는 경우 읽기 및 쓰기 맵에 잠금 중 변경이 예상되는 계획을 사용하십시오.

PESSIMISTIC 잠금 계획을 사용하여 오브젝트 그리드 Map을 구성하는 경우 트랜잭션이 BackingMap에서 처음으로 항목을 확보할 때 맵 항목의 변경이 예상되는 트랜잭션 잠금을 얻습니다. 잠금 중 변경이 예상됨은 응용프로그램에서 트랜잭션을 완료할 때까지 보관됩니다. 일반적으로 잠금 중 변경이 예상되는 계획은 다음 상황에서 사용됩니다.

- BackingMap을 로더의 존재 여부와 상관없이 구성하고 버전화 정보를 사용할 수 없는 경우
- BackingMap이 동시성 제어를 위해 오브젝트 그리드의 도움이 필요한 응용프로그램에서 직접 사용되는 경우
- 버전화 정보가 사용 가능하지만 갱신 트랜잭션이 백킹 항목에서 자주 충돌하여 갱신 중 변경이 예상되지 않는 장애가 발생하는 경우

잠금 중 변경이 예상되는 계획은 성능과 확장성에 가장 큰 영향을 주므로 이 계획은 다른 잠금 계획이 실행 가능하지 않은 경우 읽기 및 쓰기 맵에서만 사용되어야 합니다. 예를 들어 갱신 중 변경이 예상되지 않는 장애가 자주 발생하거나 응용프로그램에서 변경이 예상되지 않는 장애 복구를 처리하기 어렵습니다.

ObjectMap 메소드 및 잠금 모드

응용프로그램이 ObjectMap 인터페이스의 메소드를 사용하는 경우, 오브젝트 그리드는 자동으로 액세스할 맵 항목의 잠금 중 변경이 예상됨을 시도합니다. 오브젝트 그리드는 응용프로그램이 ObjectMap 인터페이스에서 호출하는 메소드에 따라 다음 잠금 모드를 사용합니다.

- `get` 및 `getAll` 메소드는 맵 항목의 키에서 *S 잠금* 또는 공유 잠금 모드를 확보합니다. S 잠금은 트랜잭션을 완료할 때까지 유지됩니다. S 잠금 모드인 경우 동일한 키에서 S 또는 업그레이드 가능 잠금(U 잠금) 모드를 확보하려는 트랜잭션 사이에서는 동시성을 허용하지만 동일한 키에서 독점 잠금(X 잠금) 모드를 확보하려는 다른 트랜잭션은 차단합니다.
- `getForUpdate` 및 `getAllForUpdate` 메소드는 맵 항목의 키에서 *U 잠금* 또는 업그레이드 가능 잠금 모드를 확보합니다. U 잠금은 트랜잭션을 완료할 때까지 유지됩니다. U 잠금 모드인 경우 동일한 키에서 S 잠금 모드를 확보하는 트랜잭션 사이에서는 동시성을 허용하지만 동일한 키에서 U 잠금 또는 X 잠금 모드를 확보하려는 다른 트랜잭션은 차단합니다.
- `put`, `putAll`, `remove`, `removeAll`, `insert`, `update` 및 `touch`는 맵 항목의 키에서 *X 잠금* 또는 독점 잠금 모드를 확보합니다. X 잠금은 트랜잭션을 완료할 때까지 유지됩니다. X 잠금 모드인 경우 하나의 트랜잭션에서만 지정된 키 값의 맵 항목을

삽입, 갱신 또는 제거합니다. X 잠금은 동일한 키에서 S, U 또는 X 잠금 모드를 확보하려는 모든 다른 트랜잭션을 차단합니다.

- `global invalidate` 및 `global invalidateAll` 메소드는 무효화된 각 맵 항목에서 X 잠금을 확보합니다. X 잠금은 트랜잭션을 완료할 때까지 유지됩니다. local invalidate 메소드 호출로 어떤 BackingMap 항목도 무효화되지 않으므로 `local invalidate` 및 `local invalidateAll` 메소드에서 잠금을 확보하지 못합니다.

이전 정의에서 동일한 맵 항목에 액세스하는 경우 S 잠금 모드는 더 많은 트랜잭션에서 동시에 실행할 수 있으므로 U 잠금 모드보다 확실히 더 약합니다. U 잠금 모드는 U 또는 X 잠금 모드를 요청하는 다른 트랜잭션을 차단하므로 S 잠금 모드보다 조금 더 강합니다. S 잠금 모드는 X 잠금 모드를 요청하는 다른 트랜잭션만 차단합니다. 이 작은 차이가 일부 교착 상태를 방지하는 데 중요합니다. X 잠금 모드는 동일한 맵 항목에서 S, U 또는 X 잠금 모드를 확보하려는 모든 다른 트랜잭션을 차단하므로 가장 강력한 잠금 모드입니다. X 잠금 모드의 기본적인 목적은 둘 이상의 트랜잭션에서 동일한 맵 항목을 갱신할 경우 맵 항목을 삽입, 갱신 또는 제거하고 갱신이 유실되지 않도록 하는 것입니다.

다음 표는 설명한 잠금 모드를 요약하고 서로 호환 가능한 잠금 모드를 판별할 때 사용하는 잠금 모드 호환성 매트릭스입니다. 이 매트릭스에서, 매트릭스의 행은 이미 부여된 잠금 모드를 표시합니다. 열은 다른 트랜잭션에서 요청된 잠금 모드를 표시합니다. 열에 **예(Y)**가 표시된 경우 이미 부여된 잠금 모드와 호환 가능함을 의미하므로 다른 트랜잭션에서 요청한 잠금 모드가 부여된 것입니다. **아니오(N)**는 잠금 모드가 호환 가능하지 않으므로 다른 트랜잭션이 첫 번째 트랜잭션에서 소유한 잠금을 해제하기 위해 대기해야 함을 의미합니다.

표 8. 잠금 모드 호환성 및 강도

잠금	호환 가능한 잠금			강도
	S(공유)	U(업그레이드 가능)	X(독점)	
S(공유)	예	예	아니오	가장 약함
U(업그레이드 가능)	예	아니오	아니오	표준
X(독점)	아니오	아니오	아니오	가장 강함

잠금 대기 제한시간

각 오브젝트 그리드 BackingMap에는 기본 잠금 대기 제한시간 값이 있습니다. 제한시간 값을 사용하면 응용프로그램 오류로 발생하는 교착 상태 조건 때문에 응용프로그램에서 잠금 모드가 부여되기를 한없이 기대되지 않아도 됩니다. 응용프로그램은 BackingMap 인터페이스를 사용하여 기본 잠금 대기 제한시간 값을 대체할 수 있습니다. 다음 예에서는 map1 BackingMap의 잠금 대기 제한시간 값을 60초로 설정하는 방법을 표시합니다.

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setLockTimeout( 60 );
```

java.lang.IllegalStateException 예외를 방지하려면 오브젝트 그리드 인스턴스에서 initialize 또는 getSession 메소드를 호출하기 전에 먼저 setLockStrategy 메소드와 setLockTimeout 메소드를 모두 호출하십시오. setLockTimeout 메소드 매개변수는 오브젝트 그리드에서 잠금 모드가 부여되기를 기다리는 초 수를 지정하는 Java 기본 요소 정수입니다. 트랜잭션이 BackingMap에 구성된 잠금 대기 제한시간 값을 초과하여 대기하면, com.ibm.websphere.objectgrid.LockTimeoutException 예외가 발생합니다.

LockTimeoutException이 발생하면, 응용프로그램이 예상보다 느리게 실행되어 제한시간을 초과한 것인지 또는 교착 상태 조건 때문에 제한시간을 초과한 것인지 판별해야 합니다. 실제 교착 상태 조건이 발생한 경우 잠금 대기 제한시간 값이 증가하므로 예외가 발생합니다. 제한시간이 증가하면 예외가 더 오랜 시간 후에 발생합니다. 그러나 잠금 대기 제한시간 값이 증가해도 예외가 발생하지 않는 경우 응용프로그램이 예상보다 느리게 실행된 것이므로 문제점이 발생합니다. 이 경우 응용프로그램은 성능이 느린 이유를 판별해야 합니다. 자세한 정보는 379 페이지의 제 14 장『문제점 해결』및 357 페이지의 제 11 장『오브젝트 그리드 성능 우수 사례』를 참조하십시오.

교착 상태

잠금 모드 요청 시 다음 순서를 고려하십시오.

```
X lock is granted to transaction 1 for key1.
X lock is granted to transaction 2 for key2.
X lock requested by transaction 1 for key2.
 (Transaction 1 blocks waiting for lock owned by transaction 2.)
X lock requested by transaction 2 for key1.
 (Transaction 2 blocks waiting for lock owned by transaction 1.)
```

위 순서는 둘 이상의 잠금을 확보하려는 두 개의 트랜잭션 교착 상태에 대한 표준 예입니다. 이때 각 트랜잭션은 서로 다른 순서로 잠금을 확보합니다. 이 교착 상태를 방지하려면 각 트랜잭션에서 동일한 순서로 다중 잠금을 얻어야 합니다. 응용프로그램에서 OPTIMISTIC 잠금 계획을 사용하고 ObjectMap 인터페이스의 flush 메소드를 사용하지 않는 경우 확약 주기 동안에만 트랜잭션이 잠금 모드를 요청합니다. 확약 주기 동안 오브젝트 그리드는 잠가야 하는 맵 항목의 키를 판별하고 키 순서대로 잠금 모드를 요청합니다. 오브젝트 그리드는 이 방법을 사용하여 대부분의 대형 표준 교착 상태를 방지합니다. 그러나 오브젝트 그리드는 모든 가능한 교착 상태 시나리오를 방지하는

것은 불가능합니다. 응용프로그램에서 고려해야 하는 두, 세 개의 시나리오가 있습니다. 다음은 응용프로그램에서 관심을 가지고 예방 조치를 취해야 하는 시나리오입니다.

하나의 시나리오는 오브젝트 그리드가 잠금 대기 제한시간을 대기하지 않고 교착 상태 발생을 발견할 수 있는 경우에 해당합니다. 이 시나리오가 발생한 경우, com.ibm.websphere.objectgrid.LockDeadlockException 예외가 발생합니다. 다음 코드 스니펫을 고려하십시오.

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Billy");
// Billy had a birthday, so we make him 1 year older.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
```

이 상황에서 Billy의 부인이 Billy의 실제 나이보다 더 증가시키려고 하여, Billy와 부인 모두 이 트랜잭션을 동시에 실행하게 됩니다. 이 경우 두 트랜잭션은 모두 person.get(“Billy”) 메소드 호출의 결과로 PERSON 맵의 **Billy** 항목에서 S 잠금 모드를 소유합니다. person.put (“Billy”, p) 메소드를 호출하면 두 트랜잭션 모두 S 잠금 모드를 X 잠금 모드로 업그레이드하려고 합니다. 두 트랜잭션 모두 다른 트랜잭션에서 소유한 S 잠금 모드를 해제하도록 대기하는 것을 차단합니다. 그 결과 순환 대기 조건이 두 트랜잭션 사이에서 나타나므로 교착 상태가 발생합니다. 둘 이상의 트랜잭션이 동일한 맵 항목에서 더 약한 잠금 모드를 더 강한 모드로 승격하려는 경우 순환 대기 조건이 발생합니다. 이 시나리오에서 오브제트 그리드는 LockTimeoutException 예외가 아닌 LockDeadlockException 예외를 처리합니다. 자세한 정보는 384 페이지의 『LockDeadlockException』을 참조하십시오.

응용프로그램은 PESSIMISTIC 잠금 계획 대신 OPTIMISTIC 잠금 계획을 사용하여 위 예에서 LockDeadlockException 예외를 방지할 수 있습니다. OPTIMISTIC 잠금 계획 사용은 맵을 대부분 읽은 상태이고 맵에 대한 갱신이 드문 경우 바람직한 솔루션입니다. 변경이 예상되지 않는 계획에 대한 자세한 정보는 147 페이지의 『잠금 중 변경이 예상되지 않음(Optimistic)』을 참조하십시오. PESSIMISTIC 잠금 계획을 사용해야 하는 경우, 위 예에서 get 메소드 대신 getForUpdate 메소드를 사용할 수 있습니다. 그러면 getForUpdate 메소드를 호출하는 첫 번째 트랜잭션이 S 잠금 모드가 아닌 U 잠금 모드를 확보합니다. 이 잠금 모드를 확보하면 하나의 트랜잭션에서만 U 잠금 모드가 부여되므로 두 번째 트랜잭션에서 getForUpdate 메소드를 호출하는 경우 해당 트랜잭션이 차단됩니다. 두 번째 트랜잭션이 차단되므로 Billy 맵 항목에서 어떤 잠금 모드도 소유하지 못합니다. 첫 번째 트랜잭션에서 put 메소드를 호출하여 U 잠금 모드를 X 잠금 모드로 업그레이드하려는 경우 첫 번째 트랜잭션은 차단되지 않습니다. 이 기능은 U 잠금 모드를 ″업그레이드 가능″ 잠금 모드라고 부르는 이유를 보여줍니다. 첫 번째 트랜잭션를 완료하면 두 번째 트랜잭션의 차단이 해제되고 두 번째 트랜잭션에 U

잠금 모드를 부여합니다. 응용프로그램은 PESSIMISTIC 잠금 계획을 사용하는 경우 get 메소드 대신 getForUpdate 메소드를 사용하여 잠금 승격 교착 상태 시나리오를 방지할 수 있습니다.

중요사항: 이 솔루션으로는 읽기 전용 트랜잭션이 맵 항목을 읽는 것을 방지하지 못합니다. 읽기 전용 트랜잭션은 get 메소드를 호출하지만 put, insert, update 또는 remove 메소드는 호출하지 못합니다. 일반 get 메소드를 사용하는 경우만큼 동시성이 높게 나타납니다. 동일한 맵 항목에서 둘 이상의 트랜잭션이 getForUpdate 메소드를 호출하는 경우에만 동시성이 감소합니다.

각 트랜잭션에서 동일한 순서로 U 잠금을 확보하도록 하나의 트랜잭션이 둘 이상의 맵 항목에서 getForUpdate 메소드를 호출하는 시기에 주의해야 합니다. 예를 들어, 첫 번째 트랜잭션이 키 1에서 getForUpdate 메소드를, 키 2에서 getForUpdate 메소드를 호출한다고 가정하십시오. 다른 동시 트랜잭션이 동일한 키에서 getForUpdate 메소드를 호출하지만 호출 순서가 정반대입니다. 이 순서 때문에 서로 다른 트랜잭션에서 서로 다른 순서로 다중 잠금을 얻으므로 표준 교착 상태가 발생합니다. 응용프로그램에서는 계속 다중 맵 항목에 액세스하는 모든 트랜잭션을 교착 상태가 발생하지 않는 키 순서대로 수행해야 합니다. 확약 시간이 아닌 getForUpdate 메소드 호출 시간에 U 잠금을 확보하므로 오브젝트 그리드는 확약 주기 중 정렬한 순서대로 잠금 요청을 정렬할 수 없습니다. 이 경우 응용프로그램에서 잠금 순서를 제어해야 합니다.

확약 전에 ObjectMap 인터페이스에서 flush 메소드를 사용하면 추가 잠금 정렬 고려사항을 소개할 수 있습니다. 일반적으로 flush 메소드는 Loader 플러그인을 통해 맵에서 변경된 사항을 백엔드 외부에서 강제 실행할 때 사용됩니다. 이 상황에서 백엔드는 고유한 잠금 관리자를 사용하여 동시성을 제어하므로 오브젝트 그리드 잠금 관리자가 아닌 백엔드에서 잠금 대기 조건 및 교착 상태가 발생할 수 있습니다. 다음 트랜잭션을 고려하십시오.

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
sess.begin();
 activeTran = true;
Person p = (IPerson)person.get("Billy");
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
 person.flush();
...
p = (IPerson)person.get("Tom");
p.setAge( p.getAge() + 1 );
sess.commit();
 activeTran = false;
}
```

```
finally
{
 if ( activeTran ) sess.rollback();
}
```

일부 다른 트랜잭션에서 사용자 Tom도 갱신하고 flush 메소드를 호출하며 사용자 Billy를 갱신한다고 가정하십시오. 이 상황이 발생하면 다음과 같은 두 개의 트랜잭션 사이에서 데이터베이스 교착 상태 조건이 발생합니다.

```
X lock is granted to transaction 1 for "Billy" when flush is executed.
X lock is granted to transaction 2 for "Tom" when flush is executed..
X lock requested by transaction 1 for "Tom" during commit processing.
 (Transaction 1 blocks waiting for lock owned by transaction 2.)
X lock requested by transaction 2 for "Billy" during commit processing.
 (Transaction 2 blocks waiting for lock owned by transaction 1.)
```

이 예에서는 flush 메소드를 사용하여 오브젝트 그리드가 아닌 데이터베이스에서 교착 상태가 발생할 수 있는 경우를 보여줍니다. 이 교착 상태 예는 사용하는 잠금 계획과 무관하게 발생할 수 있습니다. flush 메소드를 사용하는 경우 및 BackingMap으로 Loader를 플러그인하는 경우 응용프로그램에서 이 종류의 교착 상태가 발생하지 않도록 주의해야 합니다. 위 예에서는 오브젝트 그리드에 잠금 대기 제한시간 메커니즘이 있는 또다른 이유도 표시합니다. 데이터베이스 잠금을 대기 중인 트랜잭션은 오브젝트 그리드 맵 항목 잠금을 소유하는 동안 대기 중일 수 있습니다. 결과적으로 데이터베이스 레벨에서 문제점이 발생하면 오브젝트 그리드 잠금 모드에서의 대기 시간이 길어져 LockTimeoutException 예외가 발생합니다.

예외 처리

이 주제의 예는 예외 처리를 포함하지 않습니다. LockTimeoutException 예외 또는 LockDeadlockException이 발생한 경우 잠금 보유 시간이 길어지지 않으려면, 예상치 않은 상황이 발생한 경우 응용프로그램에서 예상치 않은 예외를 발견하고 rollback 메소드를 호출해야 합니다. 다음 예에서 보여준 대로 이전 코드 스니펫을 변경하십시오.

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
sess.begin();
 activeTran = true;
Person p = (IPerson)person.get("Billy");
// Billy had a birthday, so we make him 1 year older.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
 activeTran = false;
}
```

```
finally
{
 if ( activeTran ) sess.rollback();
}
```

코드 스니펫의 finally 블록을 통해 예상치 않은 예외가 발생한 경우 트랜잭션이 롤백됩니다. LockDeadlockException 예외 뿐만 아니라 발생할 수 있는 다른 예상치 않은 예외도 처리합니다. finally 블록은 commit 메소드 호출 중 예외가 발생한 경우를 처리합니다. 이 예가 예상치 않은 예외를 처리하는 유일한 방법은 아니며, 응용프로그램에서 발생 가능한 일부 예상치 않은 예외를 발견하여 해당 응용프로그램 예외 중 하나를 표시하는 경우도 있을 수 있습니다. 필요한 경우 catch 블록을 추가할 수 있지만 응용프로그램에서 트랜잭션을 완료하지 않으면 코드 스니펫을 종료하지 않도록 해야 합니다.

잠금 중 변경이 예상되지 않음(Optimistic)

잠금 중 변경이 예상되지 않는 계획은 두 개의 트랜잭션이 동시에 실행 중일 때 동일한 맵 항목을 갱신하려고 시도하지 않음을 확신합니다. 따라서 두 개 이상의 트랜잭션이 동시에 맵 항목 갱신을 시도하지 않으므로 트랜잭션이 지속되는 동안 잠금 모드를 유지할 필요가 없습니다.

잠금 중 변경이 예상되지 않는 계획은 일반적으로 다음 경우에 사용됩니다.

- BackingMap이 로더를 사용하여 또는 로더를 사용하지 않고 구성되어 있으며 버전화 정보를 사용할 수 있습니다.
- 대부분의 경우 BackingMap을 읽습니다. 즉, 트랜잭션에서 자주 맵 항목을 읽고 가끔씩 맵 항목을 삽입, 갱신 또는 제거합니다.
- BackingMap을 삽입, 갱신 또는 제거하는 경우 보다 드물게 트랜잭션은 동일한 맵 항목에서 거의 충돌하지 않습니다.

잠금 중 변경이 예상되는 계획과 마찬가지로 ObjectMap 인터페이스의 메소드는 오브젝트 그리드가 액세스하는 맵 항목에 대해 자동으로 잠금 모드를 획득하려고 시도하는 방법을 결정합니다. 그러나 변경이 예상되는 계획과 변경이 예상되지 않는 계획 간에는 아주 중요한 몇 가지 차이점이 있습니다.

- 잠금 중 변경이 예상되는 계획과 마찬가지로 메소드가 호출될 때 get 및 getAll 메소드로 *S 잠금* 모드를 획득합니다. 그러나 잠금 중 변경이 예상되지 않음을 사용하면 트랜잭션이 완료될 때까지 S 잠금 모드가 유지되지 않습니다. 대신, 메소드가 응용프로그램으로 리턴되기 전에 S 잠금 모드가 해제됩니다. 잠금 모드를 획득하는 목적은 오브젝트 그리드가 기타 트랜잭션에서 확약된 데이터만 현재 트랜잭션에서 볼 수 있도록 하기 위해서입니다. 오브젝트 그리드가 데이터가 확약되었음을 확인한 후에 S 잠금 모드가 해제됩니다. 확약 시 변경이 예상되지 않는 버전화 검사가 수행되어 현재 트랜잭션이 S 잠금 모드를 해제한 후에 기타 트랜잭션이 맵 항목을 변경하지 않았는지 확인합니다. 항목이 갱신, 무효화 또는 삭제되기 전에 맵에서 페치되지

않으면 오브젝트 그리드 런타임은 내재적으로 맵에서 항목을 페치합니다. 이러한 내재적 가져오기 조작이 수행되어 항목 수정이 요청되었을 당시의 현재 값을 가져옵니다.

- 잠금 중 변경이 예상되는 계획과 달리, 잠금 중 변경이 예상되지 않는 계획이 사용되는 경우에는 `getForUpdate` 및 `getAllForUpdate` 메소드가 `get` 및 `getAll` 메소드와 똑같이 처리됩니다. 즉, 메소드 시작 시 S 잠금 모드를 획득하고 응용프로그램으로 리턴되기 전에 S 잠금 모드가 해제됩니다.
- 기타 모든 ObjectMap 메소드는 잠금 중 변경이 예상되는 계획의 경우와 똑같이 처리됩니다. 즉, commit 메소드가 호출될 때 삽입, 갱신, 제거, 접촉 또는 무효화된 맵 항목에 대한 X 잠금 모드를 획득하며 X 잠금 모드는 트랜잭션이 확약 처리 완료될 때까지 유지됩니다.

변경이 예상되지 않으므로 이 잠금 계획을 변경이 예상되지 않음이라고 합니다. 잠금 중 변경이 예상되지 않는 계획은 두 개의 트랜잭션이 동시에 실행 중에 동일한 맵 항목을 갱신하려고 시도할 수 없다는 것입니다. 이런 믿음 때문에 두 개 이상의 트랜잭션이 동시에 맵 항목 갱신을 시도하지 않으므로 트랜잭션이 지속되는 동안 잠금 모드를 유지할 필요가 없습니다. 그러나 잠금 모드가 유지되지 않기 때문에 현재 트랜잭션이 S 잠금 모드를 해제한 후에 또다른 동시 트랜잭션이 맵 항목을 갱신할 가능성이 있습니다. 이 가능성을 처리하기 위해 오브젝트 그리드는 확약 시 X 잠금을 획득하고 변경이 예상되지 않는 버전화 검사를 수행하여 현재 트랜잭션이 BackingMap에서 맵 항목을 읽은 후에 기타 트랜잭션이 맵 항목을 변경하지 않았는지 확인합니다. 또다른 트랜잭션이 맵 항목을 변경하는 경우, 버전 검사는 실패하고 `OptimisticCollisionException` 예외가 발생합니다. 이 예외는 현재 트랜잭션이 강제로 롤백되도록 하며, 응용프로그램은 전체 트랜잭션을 재시도해야 합니다. 대부분의 경우 맵이 읽히며 동일한 맵 항목에 대한 갱신이 발생하지 않을 가능성이 높은 경우에는 잠금 중 변경이 예상되지 않는 계획이 아주 유용합니다.

잠금 중 BackingMap이 없는 계획

NONE 잠금 계획을 사용하도록 BackingMap이 구성된 경우, 맵 항목에 대한 트랜잭션 잠금을 확보하지 못합니다. 이 계획이 유용한 시나리오는 응용프로그램이 J2EE(Java 2 Platform, Enterprise Edition) EJB(Enterprise JavaBeans) 컨테이너와 같은 지속 관리자이거나 Hibernate를 사용하여 지속 데이터를 얻는 경우입니다. 이 시나리오에서 BackingMap은 로더를 사용하지 않고 구성되며, 지속 관리자에 의해 데이터 캐시로 사용됩니다. 이 시나리오에서 지속 관리자는 동일한 오브젝트 그리드 Map 항목에 액세스 중인 트랜잭션 사이의 동시성 제어를 제공합니다. 오브젝트 그리드는 동시성 제어를 목적으로 트랜잭션 잠금을 얻지 않아도 됩니다. 이는 지속 관리자가 오브젝트 그리드 맵을 확약된 변경사항으로 갱신하기 전에 트랜잭션 잠금을 해제하지 않는다고 가정합니다. 그렇지 않은 경우, PESSIMISTIC 또는 OPTIMISTIC 잠금 계획을 사용해야 합니다. 예를 들어, EJB 컨테이너의 지속 관리자가 EJB 컨테이너 관리 트랜잭션에서 확약

된 데이터로 오브젝트 그리드 맵을 갱신 중이라고 가정하십시오. 지속 관리자 트랜잭션 잠금이 해제되기 전에 오브젝트 그리드 맵 갱신이 발생하는 경우, NONE 잠금 계획을 사용할 수 있습니다. 지속 관리자 트랜잭션 잠금이 해제되기 전에 오브젝트 그리드 맵 갱신이 발생하는 경우, OPTIMISTIC 또는 PESSIMISTIC 잠금 계획이 필요합니다.

NONE 잠금 계획을 사용할 수 있는 또다른 시나리오는 응용프로그램이 BackingMap을 직접 사용하고 맵에 대해 로더가 구성된 경우입니다. 이 시나리오에서 로더는 JDBC(Java Database Connectivity) 또는 Hibernate를 사용하여 관계형 데이터베이스의 데이터에 액세스하는 RDBMS(Relational Database Management System)가 제공하는 동시성 제어 지원을 사용합니다. 로더 구현은 변경이 예상되지 않는 또는 변경이 예상되는 접근을 사용할 수 있습니다.

잠금 중 변경이 예상되지 않음 또는 버전화 접근을 사용하는 로더는 동시성 및 성능을 극대화하는 데 도움이 됩니다. 잠금 중 변경이 예상되지 않는 접근 방식 구현에 대한 자세한 정보는 227 페이지의 『OptimisticCallback』 섹션의 223 페이지의 『로더 고려사항』 주제를 참조하십시오.

기본 백엔드에 대한 잠금 중 변경이 예상됨 지원을 사용하는 로더는 Loader 인터페이스의 get 메소드에 전달된 **forUpdate** 매개변수를 사용하려고 할 수 있습니다. 응용프로그램이 ObjectMap 인터페이스의 `getForUpdate` 메소드를 사용하여 데이터를 가져온 경우, 이 매개변수는 *true*로 설정됩니다. 로더는 이 매개변수를 사용하여 읽고 있는 행에서 업그레이드 가능한 잠금을 요청할 것인지 여부를 결정할 수 있습니다. 예를 들어, SQL select 문에 `for update` 절이 포함된 경우 DB2는 업그레이드 가능한 잠금을 얻습니다. 이 접근 방식은 잠금 중 변경이 예상됨 주제에 설명된 것과 동일한 교착상태 방지에 대해 설명합니다.

오브젝트 그리드 보안

구성 또는 프로그래밍을 통해 맵 데이터 액세스 및 관리 타스크를 보안하려면 ObjectGrid 보안 메커니즘을 사용하십시오.

오브젝트 그리드에서는 맵 데이터 및 관리 타스크에 액세스를 보안하는 보안 메커니즘을 제공합니다. 오브젝트 그리드 보안은 JAAS(Java Authentication and Authorization Service) 메커니즘으로 빌드됩니다. JAAS는 Java 2 보안의 핵심 부분입니다.

이 섹션에서는 오브젝트 그리드 보안 메커니즘과 오브젝트 그리드 보안 API를 사용하는 방법에 대해 설명합니다.

- 150 페이지의 『오브젝트 그리드 보안 개요』에서는 오브젝트 그리드 보안을 개괄적으로 설명합니다.
- 155 페이지의 『클라이언트 서버 보안』에서는 분산 오브젝트 그리드 프로그래밍 모델의 클라이언트 서버 보안을 설명합니다.

- 176 페이지의 『로컬 오브젝트 그리드 보안』에서는 로컬 오브젝트 그리드 보안을 설명합니다.
- 183 페이지의 『권한 부여』에서는 분산 및 로컬 오브젝트 그리드 프로그래밍 모델 모두에 적용되는 권한 메커니즘 및 관련 플러그인을 설명합니다.
- 193 페이지의 『오브젝트 그리드 클러스터 보안』에서는 오브젝트 그리드 클러스터 보안 메커니즘과 관련된 플러그인에 대해 설명합니다.
- 196 페이지의 『게이트웨이 보안』에서는 게이트웨이 보안을 설명합니다.
- 199 페이지의 『WebSphere Application Server에서 보안 통합』에서는 WebSphere Application Server와 통합을 강조표시합니다.

이 섹션에 나타난 대부분의 샘플 코드는 오브젝트 그리드에서 제공하는 샘플에 있는 것입니다. 65 페이지의 제 5 장 『오브젝트 그리드 샘플』에서 보안 샘플 개요를 찾을 수 있습니다.

오브젝트 그리드 보안 개요

오브젝트 그리드는 분산 캐싱 시스템입니다. 캐시 데이터에 대한 액세스를 보안할 수 있습니다. 일반적으로 보안은 다음 세 가지 주요 개념에 기반합니다.

- *신뢰할 수 있는 인증*: 요청자의 ID를 안심하고 확인합니다.
- *권한*: 권한을 보유한 요청자에게 액세스 권한을 부여합니다.
- *보안 전송*: 네트워크 상에서 데이터를 안전하게 전송합니다.

오브젝트 그리드에서는 다음과 같은 측면의 보안을 제공합니다.

- 151 페이지의 『클라이언트 서버 보안』에서는 SSL(Secure Sockets Layer)을 사용하여 인증 및 클라이언트 서버 통신 보안을 설명합니다.
- 151 페이지의 『권한 부여』 메커니즘은 권한이 부여된 클라이언트만 오브젝트 그리드 맵 데이터 및 관리 타스크에 액세스할 수 있도록 합니다.
- 151 페이지의 『오브젝트 그리드 클러스터 보안』에서는 권한이 부여된 서버만 오브젝트 그리드 클러스터를 결합할 수 있다고 확인합니다.
- 152 페이지의 『게이트웨이 보안』에서는 게이트웨이 클라이언트 인증을 설명합니다.
- 152 페이지의 『로컬 오브젝트 그리드 보안』은 응용프로그램이 직접 오브젝트 그리드 인스턴스를 인스턴스화할 때 보안 메커니즘을 제공합니다.

오브젝트 그리드 보안은 개방형 아키텍처에 빌드되고 사용자 정의를 위해 여러 개의 플러그인 지점을 제공합니다. 플러그인 메커니즘은 중요한 역할을 수행합니다. 또한 오브젝트 그리드는 이러한 플러그인에게 일부 내장 구현을 제공합니다. 일부 구현은 독창적(out-of-box) 프로덕션에 사용할 수 있는 것이고 다른 일부 구현은 테스트 또는 샘플 목적을 위한 것입니다. 플러그인 및 내장 구현의 요약에 대해서는 152 페이지의 『보안 플러그인』을 참조하십시오.

클라이언트 서버 보안

오브젝트 그리드는 분산 클라이언트 서버 프레임워크를 지원합니다. 클라이언트 서버 보안 하부 구조는 오브젝트 그리드 서버에 대한 액세스를 적절하게 보안합니다.

오브젝트 그리드 클라이언트는 필요한 신임을 사용하여 오브젝트 그리드 서버에 인증할 수 있습니다. 이 신임을 서버 인증 메커니즘이 이해하도록 클라이언트와 서버 간에 약속되어야 합니다. SSL(Secure Sockets Layer)이 사용될 때 클라이언트는 SSL 인증을 사용하여 오브젝트 그리드 서버에 인증할 수도 있습니다.

클라이언트 서버 통신을 보안하기 위해 오브젝트 그리드는 SSL을 지원합니다. SSL 프로토콜은 오브젝트 그리드 클라이언트와 서버 간 보안 연결을 위해 확실성, 무결성 및 기밀성을 갖춘 전송 레이어 보안을 제공합니다. SSL이 제공하는 일부 보안 기능에는 데이터 플로우 중 보안 정보 노출을 방지하는 데이터 암호화, 데이터 플로우 중 인증되지 않은 데이터 수정을 방지하기 위한 데이터 서명, 그리고 담당자 또는 해당 시스템에 토크를 보장하는 클라이언트 및 서버 인증이 있습니다. SSL은 엔터프라이즈 환경 보안에서 효과적일 수 있습니다.

자세한 정보는 155 페이지의 『클라이언트 서버 보안』을 참조하십시오.

권한 부여

오브젝트 그리드 권한은 주제 및 권한에 기반합니다. 오브젝트 그리드에는 데이터 액세스를 위한 권한 및 관리 타스크를 위한 권한과 같은 두 가지 권한 카테고리가 존재합니다. Java 인증 및 권한 서비스(JAAS)를 사용하여 액세스 권한을 부여하거나 사용자 메커니즘을 플러그인하여 권한을 처리할 수 있습니다.

자세한 정보는 183 페이지의 『권한 부여』를 참조하십시오.

오브젝트 그리드 클러스터 보안

보안 환경에서 서버는 다른 서버의 확실성을 확인할 수 있어야 합니다. 오브젝트 그리드는 이러한 목적을 위해 공유 비밀 키 문자열 메커니즘을 사용합니다. 이 비밀 키 메커니즘은 공유 암호와 유사합니다. 모든 오브젝트 그리드 서버는 공유 비밀에 동의합니다. 서버와 클러스터가 결합할 때 비밀 문자열을 제시해야 합니다. 결합하는 서버의 비밀 문자열이 마스터 서버의 비밀 문자열과 일치할 경우 결합하는 서버는 클러스터와 결합할 수 있고, 그렇지 않으면 결합 요청은 거부됩니다.

텍스트 비밀 지우기 전송은 보안되지 않습니다. 오브젝트 그리드 보안 하부 구조는 서버가 이 비밀을 전송하기 전에 보안할 수 있도록 SecureTokenManager 플러그인을 제공합니다. "보안" 작업을 구현하는 방법은 진행 중입니다. 오브젝트 그리드는 "보안" 작업이 구현되어 비밀을 암호화하고 서명하는 독창적(out-of-box) 구현을 제공합니다.

자세한 정보는 193 페이지의 『오브젝트 그리드 클러스터 보안』을 참조하십시오.

게이트웨이 보안

오브젝트 그리드 게이트웨이는 클라이언트 관리 요청을 오브젝트 그리드 서버로 위임하는 연락 창구 역할을 수행합니다. 관리 게이트웨이는 일련의 mbean을 포함합니다. 게이트웨이 클라이언트는 이러한 mbean을 호출하여 오브젝트 그리드 서버를 관리 또는 모니터합니다.

관리 게이트웨이 및 서버 통신은 게이트웨이를 오브젝트 그리드 클라이언트로 취급하는 오브젝트 그리드 클라이언트 서버 통신 메커니즘을 사용합니다. 게이트웨이 클라이언트 및 게이트웨이(MBean 서버) 통신은 SSL을 통해 보안될 수 있습니다. 이 기능은 개방형 소스 프로젝트 mx4j에서 구현하는 JMX 커넥터 레이어에 의해 제공됩니다. 게이트웨이를 올바르게 작동시키려면 오브젝트 그리드에 mx4j가 필요합니다.

인증을 위해 게이트웨이는 게이트웨이 클라이언트가 제시한 신임을 오브젝트 그리드 서버로 전달합니다. 인증과 권한 모두 오브젝트 그리드 서버에서 강제 실행됩니다.

자세한 정보는 196 페이지의 『게이트웨이 보안』을 참조하십시오.

로컬 오브젝트 그리드 보안

WebSphere Extended Deployment Server 릴리스 6.0에서 로컬 오브젝트 그리드 프로그래밍 모델이 소개되었습니다. 이 모델에서는 응용프로그램이 직접 오브젝트 그리드 인스턴스를 인스턴스화하여 사용합니다. 사용자의 응용프로그램 및 오브젝트 그리드는 동일한 JVM(Java Virtual Machine)에 있습니다. 클라이언트 또는 서버 개념은 이 모델에 존재하지 않습니다.

로컬 오브젝트 그리드 프로그래밍 모델에서는 인증이 지원되지 않습니다. 응용프로그램은 고유 인증을 관리한 다음 인증된 Subject 오브젝트를 오브젝트 그리드로 전달해야 합니다.

클라이언트 서버 모델에 대해 사용된 것과 동일한 인증 메커니즘이 로컬 오브젝트 그리드 프로그래밍 모델에 사용됩니다.

자세한 정보는 176 페이지의 『로컬 오브젝트 그리드 보안』을 참조하십시오.

보안 플러그인

오브젝트 그리드 보안 프레임워크는 보안 플러그인으로 보완할 수 있습니다. 이러한 플러그인을 구현하여 보안 프레임워크를 확장하거나 사용자 정의할 수 있습니다.

하나의 예는 com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator 인터페이스로 표시되는 CredentialGenerator 플러그인입니다. 오브젝트 그리드 클라이언트가 오브젝트 그리드 서버에 연결할 때 이 플러그인의 getCredential() 메소드가 호출되어 Credential 오브젝트를 생성합니다. 여기서 이 Credential 오브젝트는 서버로 전송됩니다. 그러면 서버가 이 Credential 오브젝트를 사용하여 thecom.ibm.websphere.

objectgrid.security.plugins.CredentialGenerator.Authenticator 인터페이스로 나타나는 Authenticator 플러그인을 사용하여 인증합니다.

플러그인은 오브젝트 그리드 보안 프레임워크에서 중요한 역할을 수행합니다. CredentialGenerator를 구현하여 특정 신임을 생성할 수 있습니다(예: 사용자 ID 및 암호 쌍, Kerberos 티켓 또는 보안 토큰). Authenticator 플러그인을 구현하여 클라이언트를 인증할 수 있습니다. 원하는 경우 Authenticator 플러그인을 구현하여 사용자 암호 또는 보안 토큰을 모두 지원할 수 있습니다.

다음 테이블에

보안을 위해 사용할 수 있는 모든 플러그인이 나열되어 있습니다.

표 9. 보안 플러그인

카테고리	플러그인 클래스 이름	인스턴스
인증	com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator	클라이언트
	com.ibm.websphere.objectgrid.security.plugins.Credential	클라이언트
	com.ibm.websphere.objectgrid.security.plugins.Authenticator	서버
권한	com.ibm.websphere.objectgrid.security.plugins.MapAuthorization	오브젝트 그리드
	com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization	클러스터
오브젝트 그리드 클러스터 보안	com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager	서버
기타	com.ibm.websphere.objectgrid.security.plugins.SubjectSource	로컬 오브젝트 그리드
	com.ibm.websphere.objectgrid.security.plugins.SubjectValidation	로컬 오브젝트 그리드

다음 다이어그램은 적용된 플러그인과 인스턴스를 표시합니다. 예를 들어, MapAuthorization 플러그인은 오브젝트 그리드 인스턴스에 적용되지만 AdminAuthorization은 서버 인스턴스에 적용됩니다.

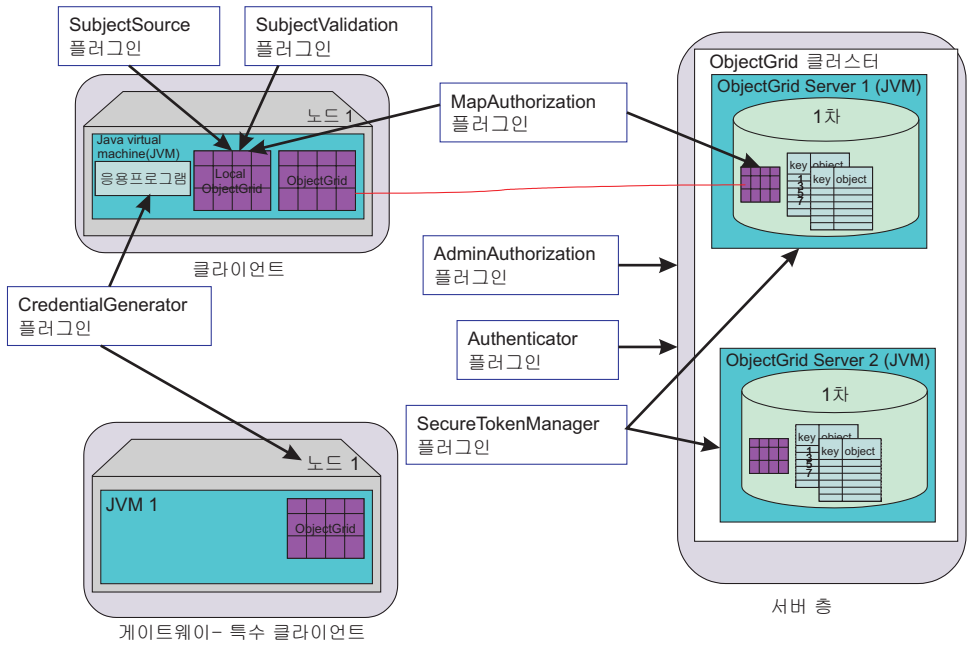


그림 17. 보안 플러그인

다음 표에서는 내장 구현을 표시합니다. 목적 열은 목적을 표시합니다. 목적은 독창적(out-of-box) 프로덕션에 사용하거나 테스트를 위한 것일 수 있습니다.

표 10. 보안 내장 구현

플러그인	내장 클래스 이름	목적
신임 생성기 신임	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator	프로덕션
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator	production
	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential	프로덕션
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential	프로덕션
	com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential	프로덕션
Authenticator	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator	프로덕션
	com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator	테스팅
	com.ibm.websphere.objectgrid.security.plugins.builtins.CertificateMappingAuthenticator	테스팅
	com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator	테스팅
맵 권한	com.ibm.websphere.objectgrid.security.plugins.builtins.JAASMapAuthorizationImpl	프로덕션
	com.ibm.websphere.objectgrid.security.plugins.builtins.TAMMapAuthorizationImpl	테스팅

표 10. 보안 내장 구현 (계속)

플러그인	내장 클래스 이름	목적
SubjectSource	com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl	프로덕션
주제 유효성 검증	com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl	프로덕션

클라이언트 서버 보안

이 주제에서는 인증 메커니즘과 클라이언트 서버 통신을 보안하는 방법에 대해 설명합니다.

클라이언트 서버 보안에는 다음과 같은 중요한 요소가 있습니다.

- 『클라이언트 서버 보안 사용 가능』 방법
- 156 페이지의 『신임 및 신임 생성자』에서 클라이언트를 나타내는 신임을 가져오는 방법
- 174 페이지의 『보안 통신』을 사용하여 SSL 구성에 사용된 매개변수를 구성하는 방법
- 163 페이지의 『인증기』를 사용하여 서버측에서 클라이언트를 인증하는 방법

클라이언트 서버 보안 사용 가능

클라이언트 보안 사용 가능

클라이언트 보안을 사용 가능하게 하려면 security.ogclient.props 파일의 securityEnabled 특성을 **true**로 설정하십시오. 오브젝트 그리드는 클라이언트 보안 특성 템플리트 파일인 security.ogclient.props 파일을 WebSphere 설치의 경우 [WAS_HOME]/optionalLibraries/ObjectGrid/properties 디렉토리에 제공하거나 혼합 서버 설치의 경우 /ObjectGrid/properties 디렉토리에 제공합니다. 이 템플리트 파일을 적절한 값으로 수정할 수 있습니다.

securityEnabled 특성에 대한 설명은 다음과 같습니다.

securityEnabled (true, false+)

이 특성은 보안이 사용 가능한지를 표시합니다. 클라이언트가 서버에 연결할 때 클라이언트 및 서버측의 securityEnabled 값은 모두 true이거나 모두 false여야 합니다. 예를 들어, 연결된 서버 보안이 사용 가능하면 클라이언트는 이 특성을 true로 설정하여 서버에 연결해야 합니다.

com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration 인터페이스는 security.ogclient.props 파일을 나타냅니다. com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory 공용 API를 사용하여 이 인터페이스의 인

스턴스를 기본값으로 작성하거나 오브젝트 그리드 클라이언트 보안 특성 파일을 전달하여 인스턴스를 작성할 수 있습니다. security.ogclient.props 파일에는 다른 특성이 있습니다.

서버 보안 사용 가능

서버측의 보안을 사용 가능하게 하기 위해 클러스터 XML의 securityEnabled 특성을 true로 설정할 수 있습니다. 예를 들면 다음과 같습니다.

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

신임 및 신임 생성자

서버를 연결할 때 클라이언트는 고유의 신임을 제시해야 합니다. 클라이언트 신임은 com.ibm.websphere.objectgrid.security.plugins.Credential 인터페이스로 표시됩니다. 신임은 사용자 암호 쌍, Kerberos 티켓 등을 포함할 수 있습니다.

신임 인터페이스는 다음과 같습니다.

```
package com.ibm.websphere.objectgrid.security.plugins;
import java.io.Serializable;
/**
* This interface represents a credential used by an ObjectGrid client. It
* represents one client identity. This credential is sent to the
* ObjectGrid server for authentication. It must be serializable.
*
* A credential has to implement the equals(Object)and
* hashCode() methods. Two Credential objects are considered equal
* if and only if they represent the same identity and security information. For
* example, if the credential contains a user ID and password. Two credentials
* are equal if and only if both their user IDs and passwords are equal.
*
* ObjectGrid provides three built-in implementations for this interface:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* ClientCertificateCredential:
* A credential containing an SSL certificate chain.
* com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential:
* A credential containing a user ID and password pair.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential:
* A credential containing WebSphere Application Server specific authentication
* and authorization tokens.
*
* Refer to the respective API documentation for more details.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see CredentialGenerator
*/
public interface Credential extends Serializable {
/**
* Checks two Credential objects for equality.
*
* Two Credential objects are considered equal if and only if
* they represent the same identity and security information.
*
* @param o the object we are testing for equality with this object.
*
* @return true if both Credential objects are equivalent.
*/
```

```
boolean equals(Object o);
/**
* Returns the hashcode of the Credential object
*
* @return the hash code of the Credential object
*/
int hashCode();
}
```

이 인터페이스는 equals(Object) 및 hashCode() 메소드를 명시적으로 정의합니다. 이 메소드는 동작을 보장하는 데 중요합니다. 인증된 Subject 오브젝트는 서버측의 Credential 오브젝트에 기반하여 캐시됩니다.

오브젝트 그리드는 신임 인터페이스에 대한 세 개의 기본 구현을 제공합니다.

1. com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential 구현. 이 신임은 사용자 ID와 암호 쌍을 포함합니다.
2. com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential 구현. 이 신임은 클라이언트 인증서 체인을 포함합니다. 이 신임을 오브젝트 그리드 클라이언트 증명 인증에 사용할 수 있습니다. 클라이언트측에는 이 신임을 작성할 수 없습니다. 이 신임은 서버에서 SSL 핸드쉐이크의 일부분으로 생성되어야 합니다.
3. com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential 구현. 이 신임은 WebSphere Application Server 특정 인증 및 권한 토큰을 포함합니다. 이 토큰을 사용하여 동일한 보안 도메인의 Application Server에서 보안 속성을 전달할 수 있습니다.

자세한 내용은 API 문서를 참조하십시오.

또한 오브젝트 그리드는 신임을 생성하는 플러그인을 제공합니다. 이 플러그인은 com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator 인터페이스로 표시됩니다. 다음은 CredentialGenerator 인터페이스입니다.

```
/**
* This plug-in is used to get a Credential representing this client. It is a
* factory for the Credential object.
* One example implementation is to return a Credential object
* containing a user ID and password pair. The implementation of the Credential
* generated by an implementation of this class must to be understood by the
* server's Authenticator plug-in.
*
* An implementation class of this interface must have a default constructor.
* When launching the client in a secure environment, set the
* implementation class name (credentialGeneratorClass) in the client security
* configuration property file. The client runtime constructs an object of
* this implementation class and calls getCredential() to get the Credential
* to connect to an ObjectGrid cluster.
*
*Users can also specify the additional properties for this factory using the
* credentialGeneratorProps property in the client security configuration
* property file.
* These properties are be passed to this
* factory by using the setProperties(String) method. This way, you can
* customize your factory.
*
```

```
* You can also set CredentialGenerator programmatically by calling
* ClientSecurityCinfiguration.setCredentialGenerator
* (CredentialGenerator) method.
*
*
* For example, you can have the following settings in the client security
* configuration property file:
* credentialGeneratorClass=com.myco.CredGenFactory
*
* credentialGeneratorProps=user1 password1
*
*
* , a String "user1 password1" is passed to the setProperties(String)
* method, with the "user1" indicating the user name, and "password1"
* indicating the password.
*
* ObjectGrid provides two built-in implementations for this interface:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* UserPasswordCredentialGenerator:
* A credential generator generating a UserPasswordCredential
* containing a user ID and password pair.
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* WSTokenCredentialGenerator:
* A credential generator generating a WSTokenCredential containing WebSphere
* Application Server
* specific authentication and authorization tokens.
*
*
* The relationship between CredentialGenerator and Credential can be
* one to one relationship or one to many relationship. For example, the
* UserPasswordCredentialGenerator
* has a one to one relationship with UserPasswordCredential, but the
* WSTokenCredentialGenerator
* has a one to many relationship with WSTokenCredential because it could generate
* different WSTokenCredential based on what Subject is associated with the current
* thread.
*
* Refer to the respective API documentation for more details.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Authenticator
* @see ClientSecurityConfiguration#setCredentialGenerator(CredentialGenerator)
* @see Credential
* @see CredentialGeneratorFactory#getCredentialGenerator()
*/
public interface CredentialGenerator {
/**
* Gets a Credential which represents the client.
*
* @return the Credential representing the client
*
* @throws CannotGenerateCredentialException if a failure occurs when
* generating the Credential for the client.
*
* @see Credential
*/
Credential getCredential() throws CannotGenerateCredentialException;
/**
* Set the user defined properties to the factory
*
* This method is used to add addtional CredentialGenerator properties
* to the object. These properties can be set using the credentialGeneratorProps
* property in the client security configuration property file.
* This way, you can customize your factory.
*
```

```
* @param properties user defined properties
*/
void setProperties(String properties);
}
```

오브젝트 그리드는 두 개의 기본 내장 구현을 제공합니다.

1. com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator 생성자는 사용자 ID 및 암호를 가져옵니다. getCredential() 메소드가 호출되면 사용자 ID 및 암호가 포함된 UserPasswordCredential 오브젝트를 리턴합니다.
2. com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator는 WebSphere Application Server에서 실행될 때 신임(보안 토큰) 생성자를 나타냅니다. getCredential() 메소드가 호출될 때 현재 스레드와 연관된 Subject가 검색됩니다. 그런 다음 이 Subject 오브젝트의 보안 정보가 WSTokenCredential 오브젝트로 변환됩니다. 상수 WSTokenCredentialGenerator.RUN_AS_SUBJECT 또는 WSTokenCredentialGenerator.CALLER_SUBJECT를 사용하여 스레드에서 caller 주제 또는 runAs 주제를 검색할지 여부를 지정할 수 있습니다.

자세한 내용은 API 문서를 참조하십시오.

연결

오브젝트 그리드 클라이언트가 서버에 보안 상태로 연결하려는 경우 ObjectGridManager 인터페이스의 connect 메소드를 사용할 수 있습니다. 예로 다음 connect 메소드를 고려하십시오.

```
/**
* This allows client to connect to a Remote ObjectGrid
* The RemoteObject Grid is hosted as specified by the paramaters
* @param clusterName: The name of the cluster to which this client
* will attach iteself
* @param host: The host on which to connect to
* @param port: The clientAceess port which is listening.
* @param ClientSecurityConfiguration: Security configuration. It can be null
* if security is not configured
* @param overRideObjectGrid xml. This parameter can be null. If it is not
* null, the client side configuration of objectgrid plug-in is overridden.
* Not all plug-ins can be overridden. For details see the ObjectGrid documents
* @throws ConnectException
* @ibm-api
*/
public ClientClusterContext connect(String clusterName, String host, String port,
ClientSecurityConfiguration securityProps, URL overRideObjectGrid) throws
ConnectException ;
```

이 메소드는 ClientSecurityConfiguration 유형의 매개변수를 가져옵니다. 이 인터페이스는 클라이언트 보안 구성을 나타냅니다. com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory 공용 API를 사용하여 이 인터페이스의 인스턴스를 기본값으로 작성하거나 오브젝트 그리드 클라이언트 보안 특성 파일을 전달하

여 인스턴스를 작성할 수 있습니다. security.ogclient.props 파일에는 인증과 관련된 다음 특성이 있습니다. + 표시가 있는 값이 기본값입니다.

- securityEnabled (true, false+): 이 특성은 보안이 사용 가능한지 여부를 표시합니다. 클라이언트가 서버에 연결할 때 클라이언트 및 서버측의 securityEnabled 값은 모두 true이거나 모두 false여야 합니다. 예를 들어, 연결된 서버 보안이 사용 가능하면 클라이언트는 이 특성을 true로 설정하여 서버에 연결해야 합니다.
- credentialAuthentication(Never, Supported+, Required): 이 특성은 클라이언트가 신임 인증을 지원하는지 표시합니다.
  - 특성 값이 **Never**이면 이 클라이언트에서는 신임 인증을 지원하지 않습니다.
  - 특성 값이 **Supported**이면 클라이언트 인증은 신임 인증을 지원하거나 필요로 하는 서버와 통신할 때 수행됩니다. 클라이언트 신임 인증은 신임 또는 단일 사인온(SSO) 토큰을 전송합니다.
  - 특성 값이 **Required**이면 클라이언트는 인증을 위해 서버에 신임을 전송해야 합니다.
- authenticationRetryCount(정수 값, 0+). 이 특성은 신임이 만기되는 시점에서 로그인 재시도 횟수를 결정합니다. 값이 0이면 재시도가 이루어지지 않은 것입니다. 인증 재시도는 신임이 만기되는 경우에만 적용됩니다. 신임이 유효하지 않으면 재시도는 이루어지지 않습니다. 해당 응용프로그램이 조작 재시도를 담당합니다.
- clientCertificateAuthentication(Never+, Supported, Required): 이 특성은 클라이언트가 클라이언트 증명 인증을 지원하는지 표시합니다.
  - 특성 값이 **Never**이면 클라이언트 증명 인증이 클라이언트측에서 지원되지 않습니다.
  - 특성 값이 **Supported**이면 전송 레이어 클라이언트 인증이 수행될 수 있고 클라이언트는 인증 단계에서 서버에 디지털 인증서를 전송합니다.
  - 특성 값이 **Required**이면 클라이언트는 전송 레이어 클라이언트 인증을 지원하는 서버에서만 인증합니다.
- transportType(TCP/IP, SSSL-Supported+, SSL-Required): 클라이언트가 서버에 연결하려는 전송 프로토콜을 표시합니다. 또한 클라이언트가 서버에 연결하는 프로토콜은 서버측의 transportType 설정에 따라 달라집니다. 자세한 내용은 174 페이지의 『보안 통신』을 참조하십시오.
  - 값이 **TCP/IP**이면 클라이언트는 TCP/IP를 사용하여 서버에 연결해야 합니다.
  - 값이 **SSL-Supported**이면 클라이언트는 TCP/IP 또는 SSL을 사용하여 서버에 연결할 수 있습니다. 클라이언트는 서버에 연결하는 데 먼저 SSL을 사용합니다. SSL 연결이 실패하면 클라이언트는 TCP/IP를 사용합니다.
  - 값이 **SSL-Required**이면 클라이언트는 SSL을 사용하여 서버에 연결해야 합니다.
- SSOEnabled: 클라이언트가 서버에 SSO 토큰 전달을 지원하는지 지정합니다. 클라이언트가 모든 서버에 인증하면 이 특성을 false로 설정하십시오. 클라이언트가 한

서버에만 인증하면 이 특성을 true로 설정하십시오. 클라이언트에서 SSOEnabled를 true로 설정했으면 클러스터 XML 구성의 single-sign-on-enabled 특성도 true로 설정되어 있는지 확인하십시오.

또한 ClientSecurityConfiguration 인터페이스의 Setter를 사용하여 이 특성을 설정할 수 있습니다.

ClientSecurityConfiguration 유형 오브젝트를 작성한 후 다음 메소드를 사용하여 오브젝트에 credentialGenerator를 설정하십시오.

```
/**
* Set the {@link CredentialGenerator} object for this client.
* @param generator the CredentialGenerator object associated with this client
*/
void setCredentialGenerator(CredentialGenerator generator);
```

오브젝트 그리드 클라이언트 보안 특성 파일에서도 CredentialGenerator를 설정할 수 있습니다. 다음은 특성입니다.

- **credentialGeneratorClass**: CredentialGenerator의 클래스 구현 이름. 기본 생성자가 있어야 합니다.
- **credentialGeneratorProps**: CredentialGenerator 클래스의 특성. 값이 널이 아니면 setProperties(String) 메소드를 사용하여 이 값을 생성된 CredentialGenerator 오브젝트로 설정합니다.

다음은 ClientSecurityConfiguration을 인스턴스화한 다음 이것을 사용하여 서버에 연결하는 샘플입니다.

```
/**
* Get a secure ClientClusterContext
* @return a secure ClientClusterContext object
*/
protected ClientClusterContext connect() throws ConnectException {
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration("/properties/security.ogclient.props");
UserPasswordCredentialGenerator gen= new
UserPasswordCredentialGenerator("manager", "manager1");
csConfig.setCredentialGenerator(gen);
return objectGridManager.connect(csConfig, null);
}
```

연결이 호출될 때 오브젝트 그리드 클라이언트는 CredentialGenerator.getCredential() 메소드를 호출하여 클라이언트 신임을 가져옵니다. 이 신임은 연결 요청과 함께 인증을 위해 서버로 전송됩니다.

세션마다 다른 CredentialGenerator 사용

일부 경우 오브젝트 그리드 클라이언트는 한 개의 클라이언트 ID만을 나타내는 반면 다른 경우에는 여러 개의 ID를 나타낼 수 있습니다. 다음은 후자 경우의 한 시나리오입니다. 오브젝트 그리드 클라이언트가 작성되고 웹 서버에서 공유됩니다. 이 웹 서버의

모든 Servlet은 이 하나의 오브젝트 그리드 클라이언트를 사용합니다. 모든 Servlet은 상이한 웹 클라이언트를 나타내므로 요청을 오브젝트 그리드 서버로 전송할 때 다양한 신임을 사용합니다.

오브젝트 그리드는 세션 레벨에서 신임을 변경할 수 있습니다. 즉, 모든 세션이 상이한 CredentialGenerator를 사용할 수 있습니다. 따라서 Servlet이 다른 CredentialGenerator로 세션을 가져올 수 있도록 이전 시나리오를 완료할 수 있습니다. 다음은 ObjectGridManager 인터페이스의 메소드입니다.

```
/**
* Get a session with a CredentialGenerator. This method can only be called
* by the ObjectGrid client in a client server environment.
*
* If ObjectGrid is used in a core model, that is, within the same JVM with
* no client or server existing, getSession(Subject) should be used to secure
* the ObjectGrid.
*
* @since WAS XD 6.0.1
*/
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;
```

예를 들면 다음과 같습니다.

```
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
CredentialGenerator credGenManager = new UserPasswordCredentialGenerator
("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator
("employee", "xxxxxx");
ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");
// Get a session with CredentialGenerator;
Session session = og.getSession(credGenManager );
// Get the employee map
ObjectMap om = session.getMap("employee");
// start a transaction.
session.begin();
Object rec1 = map.get("xxxxxx");
session.commit();
// Get another session with a different CredentialGenerator;
session = og.getSession(credGenEmployee );
// Get the employee map
om = session.getMap("employee");
// start a transaction.
session.begin();
Object rec2 = map.get("xxxxx");
session.commit();
```

ObjectGird.getSession() 메소드를 사용하여 Session 오브젝트를 가져오는 경우 세션은 ClientConfigurationSecurity 오브젝트에 설정된 CredentialGenerator를 사용합니다. 따라서 ObjectGird.getSession(CredentialGenerator) 메소드에 전달된 CredentialGenerator가 ClientConfigurationSecurity 오브젝트에 설정된 CredentialGenerator를 대체하는 것으로 간주할 수 있습니다.

Session 오브젝트를 다시 사용할 수 있는 경우 성능 이득을 얻습니다. 그러나, ObjectGrid.getSession(CredentialGenerator) 메소드는 큰 비용이 소요되지 않으므로 주

오버헤드는 오브젝트 가비지 콜렉션 시간 증가입니다. Session 오브젝트로 수행한 후 참조를 릴리스했는지 확인하십시오. 요약하면 Session 오브젝트가 ID를 공유할 수 있으면 Session 오브젝트 재사용을 시도하고, Session 오브젝트가 ID를 공유할 수 없으면 ObjectGrid.getSession(CredentialGenerator) 메소드를 사용하십시오.

인증기

오브젝트 그리드 클라이언트가 CredentialGenerator 오브젝트를 사용하여 Credential 오브젝트를 검색한 후 Credential 오브젝트는 클라이언트 요청과 함께 오브젝트 그리드 서버로 전송됩니다. 오브젝트 그리드 서버는 요청을 처리하기 전에 Credential 오브젝트를 인증합니다. Credential 오브젝트가 성공적으로 인증되면 Subject 오브젝트가 리턴되어 이 Credential 오브젝트를 나타냅니다. 그런 다음, Subject 오브젝트는 요청 권한 부여에 사용됩니다.

이 Subject 오브젝트도 캐시됩니다. 이 오브젝트는 지속 시간이 세션 제한시간 값에 도달하면 만기됩니다. 클러스터 XML 파일에서 loginSessionExpirationTime 특성을 사용하여 로그인 세션 제한시간 값을 설정할 수 있습니다. 예를 들어, loginSessionExpirationTime="300"을 설정하면 Subject 오브젝트가 300초 후 만기됩니다.

오브젝트 그리드 서버는 Authenticator 플러그인을 사용하여 Credential 오브젝트를 인증합니다. 다음은 Authenticator 인터페이스입니다.

```
/**
* This plug-in can be used to authenticate an ObjectGrid client to an ObjectGrid
* server based on the credential provided by the client. A Subject
* object is returned as a result of authentication.
*
* This plug-in is used in an ObjectGrid server. It can be configured in the
* ObjectGrid cluster XML file.
*
* The Credential passed in the authenticate(Credential)
* method can contain any credential information users desire. For example,
* it could be a Credential object containing a user password pair.
*
* ObjectGrid provides several built-in implementations for this interface:
* * com.ibm.websphere.objectgrid.security.plugins.builtins.
* CertificateMappingAuthenticator:
* An authenticator that simply maps a SSL certficate to a Subject.
* com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator:
* An authenticator that authenticates a user ID and password to a key file.
* com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator:
* An authenticator that authenticates a user ID and password to a LDAP server.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator:
* An authenticator that authenticates a WebSphere Application Server securty token.
*
* Refer to the respective API documentation for more details.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Credential
*/
public interface Authenticator {
/**
* Authenticates a user represented by the credential object.
```

```
*
* @param credential the user Credential
*
* @return a Subject object representing the user
*
* @throws InvalidCredentialException if the credential is invalid
* @throws ExpiredCredentialException if the credential is expired
*
* @see Credential
*/
Subject authenticate(Credential credential)
throws InvalidCredentialException, ExpiredCredentialException;
}
```

이것은 구현에서 Credential 오브젝트를 가져온 다음 이 오브젝트를 사용자 레지스트리(예: LDAP(Lightweight Directory Access Protocol) 서버)에 인증하는 경우입니다. 오브젝트 그리드는 독창적(out-of-box) 사용자 레지스트리 구성을 제공하지 않습니다. 사용자 레지스트리에 연결하고 인증하는 것이 이 플러그인에 구현되어야 합니다.

예를 들어, 한 인증기 구현이 신임에서 사용자 ID 및 암호를 추출하고 이것을 사용하여 LDAP 서버에 연결하며 유효성을 검증하고 Subject 오브젝트를 인증의 결과로 작성합니다. 구현은 JAAS 로그인 모듈을 활용할 수 있습니다. Subject 오브젝트가 인증의 결과로 리턴됩니다.

이 메소드는 두 가지 예외인 `InvalidCredentialException`과 `ExpiredCredentialException`을 처리합니다. `InvalidCredentialException` 예외는 신임이 유효하지 않음을 표시합니다. `ExpiredCredentialException` 예외는 신임이 만기되었음을 표시합니다. 이 두 예외 중 하나가 authenticate 메소드의 결과로 발생한 것이면 예외는 다시 클라이언트로 전송됩니다. 그러나, 클라이언트 런타임은 이 두 가지 예외를 다르게 처리합니다.

- 예외가 InvalidCredentialException인 경우 클라이언트 런타임은 이 예외를 표시합니다. 응용프로그램이 이 예외를 처리하게 됩니다. 예를 들어, CredentialGenerator를 정정한 다음 조작을 재시도할 수 있습니다.
- 예외가 ExpiredCredentialException이고 재시도 횟수가 0이 아닌 경우 클라이언트 런타임은 CredentialGenerator.getCredential() 메소드를 다시 호출하고 새 Credential 오브젝트를 서버로 전송합니다. 새 신임 인증이 성공하면 서버는 요청을 처리합니다. 새 신임 인증이 실패하면 예외가 다시 클라이언트로 전송됩니다. 인증 재시도 수가 허용된 값에 도달하고 클라이언트가 여전히 ExpiredCredentialException을 가져오면 ExpiredCredentialException이 발생됩니다. 해당 응용프로그램이 이 예외를 처리해야 합니다.

Authenticator 인터페이스는 탁월한 유연성을 제공합니다. 어떠한 방법으로든 Authenticator 인터페이스를 구현할 수 있습니다. 예를 들어, 이 인터페이스를 구현하여 신임 인증과 클라이언트 증명 인증을 모두 수행하고 두 인증을 모두 지원할 수 있습니다. 또는 인터페이스를 구현하여 두 개의 서로 다른 사용자 레지스트리를 지원할 수 있습니다.

오브젝트 그리드는 신임 인증과 클라이언트 증명 인증이라는 두 가지 종류의 인증을 지원합니다. 사용하는 메커니즘은 클라이언트 및 서버측 보안 특성 설정에 따라 달라집니다. 이 특성은 다음과 같습니다.

- `security.ogclient.props` 파일의 credentialAuthentication
- `security.ogserver.props` 파일의 credentialAuthentication
- `security.ogclient.props` 파일의 clientCertificateAuthentication
- `security.ogserver.props` 파일의 clientCertificateAuthentication

또한 프로그래밍 API를 사용하여 이 특성을 설정할 수 있습니다.

다음 두 테이블은 다양한 설정 하에서 사용되는 인증 메커니즘을 표시합니다.

표 11. 클라이언트 및 서버 설정 하에 신임 인증

클라이언트 **credentialAuthentication**	서버 **credentialAuthentication**	결과
아니오	불필요	사용 불가능
	지원됨	사용 불가능
	필수	오류 사례
지원됨	불필요	사용 불가능
	지원됨	사용 가능
	필수	사용 가능
필수	불필요	오류 사례
	지원됨	사용 가능
	필수	사용 가능

신임 인증이 없으면(결과가 사용 불가능임) 클라이언트 증명 인증이 발생할 수 있습니다.

다음 테이블은 다양한 설정 하에서 클라이언트 증명 인증이 사용되는지 여부를 표시합니다. 클라이언트 증명 인증은 SSL이 통신 프로토콜로 사용되고 신임 인증이 사용되지 않는 경우에만 가능합니다.

표 12. 클라이언트 및 서버 설정 하에 클라이언트 증명 인증.

클라이언트 **clientCertificate** 인증	서버 **clientCertificate** 인증	결과
아니오	불필요	사용 불가능
	지원됨	사용 불가능
	필수	오류 사례
지원됨	불필요	사용 불가능
	지원됨	사용 가능*
	필수	사용 가능*

표 12. 클라이언트 및 서버 설정 하에 클라이언트 증명 인증. (계속)

클라이언트 **clientCertificate** 인증	서버 **clientCertificate** 인증	결과
필수	불필요	오류 사례
	지원됨	사용 가능*
	필수	사용 가능*

* ClientCertificateAuthentication은 SSL이 프로토콜로 사용되고 CredentialAuthentication이 사용되지 않을 때에만 발생합니다.

드문 경우로 신임 인증과 클라이언트 증명 인증이 모두 사용되지만 클라이언트에서 전송된 신임이 널일 때 클라이언트 증명 인증이 사용됩니다.

인증기는 클러스터 XML 파일에서 구성될 수 있습니다. 예를 들면 다음과 같습니다.

```
<cluster name="cluster1" securityEnabled="true" singleSignOnEnabled="true"
loginSessionExpirationTime="300" statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12500" workingDirectory="" traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator
className ="com.ibm.websphere.objectgrid.security.plugins.builtins.
WSTokenAuthenticator">
</authenticator>
</cluster>
```

오브젝트 그리드는 키 파일 사용자 레지스트에 대한 사용자 ID와 암호 인증, LDAP 서버에 대한 사용자 ID 및 암호 인증, SSL 클라이언트 인증 단순 맵핑 인증, 그리고 WebSphere Application Server 보안 메커니즘과 같은 네 가지 기본 인증 내장 구현을 제공합니다. WebSphere Application Server 보안 메커니즘에 대한 인증기 구현을 제외하면 내장 구현은 테스트 목적만을 위한 것입니다. 이러한 두 가지 내장의 기본 목적은 사용자에게 코드 작성 없이 단순 테스트 수행을 허용하는 것입니다. WebSphere Application Server 인증기 구현은 오브젝트 그리드 서버 및 클라이언트 모두 동일한 보안 도메인에 있을 때 플러그인할 수 있는 독창적(out-of-box) 구현입니다.

WebSphere Application Server 사용자 레지스트리를 사용하려는 오브젝트 그리드 서버의 경우 WebSphere Application Server API를 사용하여 Application Server에 구성된 사용자 레지스트리를 가져온 다음 인증기 구현에서 사용할 수 있습니다. 그러나, 이 구현에 대해서는 이 프로그래밍 안내서에서 설명하지 않습니다.

키 파일 레지스트리 인증기 구현

키 스토어 파일이라는 파일에 사용자 ID 및 암호를 저장할 수 있습니다. 키 도구를 사용하여 키 스토어 파일 및 항목을 작성할 수 있습니다. 예를 들어, 다음 명령은 별명이 user1인 항목을 작성합니다.

```
keytool -genkey -v -keystore ./keys.jks -storepass password -alias user1
-keypass password -dname CN=user1,O=MyCompany,L=MyCity,ST=MyState
```

테스트 목적을 위해 오브젝트 그리드는 이 플러그인의 com.ibm.websphere.objectgrid.security.plugins.builtins. KeyStoreLoginAuthenticator 기본 구현을 제공하여 사용자 이름 및 암호 인증을 처리합니다. 이 구현에서 사용자는 로그인 이름 KeyStoreLogin을 사용하여 키 스토어 파일에 로그인합니다.

다음은 KeyStoreLoginAuthenticator 클래스의 authenticate(Credential) 메소드 구현을 보여주는 코드 스니펫입니다.

```
public Subject authenticate(Credential credential)throws
InvalidCredentialException,
ExpiredCredentialException {
UserPasswordCredential cred = (UserPasswordCredential) credential;
LoginContext lc = null;
lc = new LoginContext("KeyStoreLogin",
new UserPasswordCallbackHandlerImpl(cred.getUserName(),
cred.getPassword().toCharArray()));
lc.login();
Subject subject = lc.getSubject();
```

이 스니펫은 클라이언트가 UserPasswordCredential 유형 오브젝트만 전달할 수 있는 클라이언트와 연계되므로 Credential 인터페이스의 구현인 UserPasswordCredential로 신임을 먼저 캐스트합니다. 그런 다음 KeyStoreLogin 로그인 모듈을 호출하여 로그인합니다.

오브젝트 그리드는 이 목적을 위해 com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule 로그인 모듈을 제공합니다. 각 사용자의 사용자 이름 및 암호 쌍을 포함하는 키 스토어 파일을 제공해야 합니다. 키 스토어 파일은 로그인 모듈에 대한 옵션으로 구성됩니다.

다음은 로그인 모델이 키 파일에 로그인하는 방법을 보여주는 코드 스니펫입니다.

```
/**
* Authenticates a user based on the keystore file.
*
* @see javax.security.auth.spi.LoginModule#login()
*/
public boolean login() throws LoginException {
if (debug) {
System.out.println("[KeyStoreLoginModule] login: entry");
}
String name = null;
char pwd[] = null;
if (keyStore == null || subject == null || handler == null) {
throw new LoginException("Module initialization failed");
}
NameCallback nameCallback = new NameCallback("Username:");
PasswordCallback pwdCallback = new PasswordCallback("Password:", false);
try {
handler.handle(new Callback[] { nameCallback, pwdCallback });
```

```
}
catch (Exception e) {
throw new LoginException("Callback failed: " + e);
}
name = nameCallback.getName();
char[] tempPwd = pwdCallback.getPassword();
if (tempPwd == null) {
// treat a NULL password as an empty password
tempPwd = new char[0];
}
pwd = new char[tempPwd.length];
System.arraycopy(tempPwd, 0, pwd, 0, tempPwd.length);
pwdCallback.clearPassword();
if (debug) {
System.out.println("[KeyStoreLoginModule] login: "
+ "user entered user name: " + name);
}
if (ObjectGridManagerImpl.isTraceEnabled && TC.isDebugEnabled())
Tr.debug(TC, "login", "userName="+name);
// Validate the user name and password
try {
validate(name, pwd);
}
catch (SecurityException se) {
principals.clear();
publicCreds.clear();
privateCreds.clear();
LoginException le = new LoginException(
"Exception encountered during login");
le.initCause(se);
throw le;
}
if (debug) {
System.out.println("[KeyStoreLoginModule] login: exit");
}
 return true;
}
/**
* Validate the user name and password based on the keystore.
*
* @param userName user name
* @param password password
* @throws SecurityException if any exceptions encountered
*/
protected void validate(String userName, char password[])
throws SecurityException {
PrivateKey privateKey = null;
// Get the private key from the keystore
try {
privateKey = (PrivateKey) keyStore.getKey(userName, password);
}
catch (NoSuchAlgorithmException nsae) {
SecurityException se = new SecurityException();
se.initCause(nsae);
throw se;
}
catch (KeyStoreException kse) {
SecurityException se = new SecurityException();
se.initCause(kse);
throw se;
}
catch (UnrecoverableKeyException uke) {
```

```
SecurityException se = new SecurityException();
se.initCause(uke);
throw se;
}
if (privateKey == null) {
throw new SecurityException("Invalid name: " + userName);
}
// Check the certificats
Certificate certs[] = null;
try {
certs = keyStore.getCertificateChain(userName);
}
catch (KeyStoreException kse) {
SecurityException se = new SecurityException();
se.initCause(kse);
throw se;
}
if (certs != null && certs.length > 0) {
// If the first certificate is an X509Certificate
if (certs[0] instanceof X509Certificate) {
try {
// Get the first certificate which represents the user
X509Certificate certX509 = (X509Certificate) certs[0];
// Create a principal
X500Principal principal = new X500Principal(certX509
.getIssuerDN()
.getName());
principals.add(principal);
if (debug) {
System.out.println(" Principal added: " + principal);
}
}
catch (CertificateException ce) {
SecurityException se = new SecurityException();
se.initCause(ce);
throw se;
}
}
}
}
```

JAAS 인증 구성 파일에 로그인 이름 ″KeyStoreLogin″을 작성해야 합니다. JAAS 인증 구성 파일에 익숙하지 않은 경우 자세한 정보를 보려면 JAAS 인증 학습서를 참조하십시오.

```
KeyStoreLogin {
com.ibm.websphere.objectgrid.jaas.KeystoreLoginModule required
keyStoreFile="${user.dir}${/}security${/}.keystore";
};
```

이 구현은 테스트 목적만을 위한 것입니다.

LDAP 인증기 구현

오브젝트 그리드는 이 플러그인에 com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator 기본 구현을 제공하여 LDAP 서버에 대한 사용자 이름 및 암호 인증을 처리합니다. 이 구현은 LDAPLogin 로그인 모듈을 사용하여 사용자를 LDAP 서버에 로그인시킵니다.

다음 스니펫은 authenticate 메소드가 구현되는 방법을 나타냅니다.

```
/**
* @see com.ibm.ws.objectgrid.security.plugins.Authenticator#
* authenticate(LDAPLogin)
*/
public Subject authenticate(Credential credential) throws
InvalidCredentialException, ExpiredCredentialException {
UserPasswordCredential cred = (UserPasswordCredential) credential;
LoginContext lc = null;
try {
lc = new LoginContext("LDAPLogin",
new UserPasswordCallbackHandlerImpl(cred.getUserName(),
cred.getPassword().toCharArray()));
lc.login();
Subject subject = lc.getSubject();
return subject;
}
catch (LoginException le) {
throw new InvalidCredentialException(le);
}
catch (IllegalArgumentException ile) {
throw new InvalidCredentialException(ile);
}
}
```

오브젝트 그리드는 이 목적을 위해 로그인 모듈 com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule을 제공합니다. JAAS 로그인 구성 파일에 다음 두 개의 옵션을 제공해야 합니다.

- **providerURL**: LDAP 서버 프로바이더 URL
- **factoryClass**: LDAP 컨텍스트 팩토리 구현 클래스

LDAPLoginModule은 com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthentcationHelper.authenticate 메소드를 호출합니다. 다음 코드 스니펫은 LDAPAuthentcationHelper의 authenticate 메소드가 구현되는 방법을 보여줍니다.

```
/**
* Authenticate the user to the LDAP directory.
* @param user the user ID, e.g., uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
* @param pwd the password
*
* @throws NamingException
*/
public String[] authenticate(String user, String pwd)
throws NamingException {
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, factoryClass);
env.put(Context.PROVIDER_URL, providerURL);
env.put(Context.SECURITY_PRINCIPAL, user);
env.put(Context.SECURITY_CREDENTIALS, pwd);
env.put(Context.SECURITY_AUTHENTICATION, "simple");
InitialContext initialContext = new InitialContext(env);
// Look up for the user
DirContext dirCtx = (DirContext) initialContext.lookup(user);
String uid = null;
```

```
int iComma = user.indexOf(",");
int iEqual = user.indexOf("=");
if (iComma > 0 && iComma > 0) {
uid = user.substring(iEqual + 1, iComma);
}
else {
uid = user;
}
Attributes attributes = dirCtx.getAttributes("");
// Check the UID
String thisUID = (String) (attributes.get(UID).get());
String thisDept = (String) (attributes.get(HR_DEPT).get());
if (thisUID.equals(uid)) {
return new String[] { thisUID, thisDept };
}
else {
return null;
}
}
```

인증이 성공하면 ID와 암호는 유효한 것으로 간주됩니다. 그런 다음 로그인 모듈은 이 authenticate 메소드에서 UID 정보 및 부서 정보를 가져옵니다. 로그인 모듈은 SimpleUserPrincipal 및 SimpleDeptPrincipal과 같은 두 개의 프린시펄을 작성합니다. 그룹 권한(이 경우, 부서가 그룹임) 및 개별 권한에 대해 인증된 주제를 사용할 수 있습니다.

다음은 LDAP 서버에 로그인하는 데 사용되는 로그인 모듈 구성 예입니다.

```
LDAPLogin { com.ibm.websphere.objectgrid.security.plugins.builtins.
LDAPLoginModule required
providerURL="ldap://directory.acme.com:389/"
factoryClass="com.sun.jndi.ldap.LdapCtxFactory";
};
```

이전 구성에서 LDAP 서버는 `ldap://directory.acme.com:389/` 서버를 지시합니다. 이 설정을 LDAP 서버로 변경하십시오. 이 로그인 모듈은 제공된 사용자 ID 및 암호를 사용하여 LDAP 서버에 연결합니다. 이 구현은 테스트 목적만을 위한 것입니다.

WebSphere Application Server 인증기 구현

또한 오브젝트 그리드는 com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator 내장 구현을 제공하여 WebSphere Application Server 보안 하부 구조를 사용합니다. 이 내장 구현은 다음 조건이 있을 때 사용될 수 있습니다.

- WebSphere Application Server 글로벌 보안이 설정되어 있습니다.
- 오브젝트 그리드 클라이언트 및 오브젝트 그리드 서버 모두 WebSphere Application Sever JVM(Java Virtual Machines)에서 실행됩니다.
- 이 Application Server는 동일한 보안 도메인에 있습니다.
- 오브젝트 그리드 클라이언트가 이미 WebSphere Application Server에서 인증되었습니다.

오브젝트 그리드 클라이언트는 com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator 클래스를 사용하여 신임을 생성할 수 있으며 오브젝트 그리드 서버는 이 인증기 구현 클래스를 사용하여 신임을 인증합니다. 토큰이 성공적으로 인증된 경우 Subject 오브젝트가 리턴됩니다.

이 시나리오는 오브젝트 그리드 클라이언트가 이미 인증되었다는 사실을 활용합니다. 오브젝트 그리드 서버가 있는 Application Server는 오브젝트 그리드 클라이언트를 포함하는 Application Server와 동일한 보안 도메인에 있으므로 보안 토큰을 오브젝트 그리드 클라이언트에서 오브젝트 그리드 서버로 전달할 수 있어 동일한 사용자 레지스트리를 다시 인증할 필요가 없습니다.

단순 인증 맵핑 인증기 구현

또한 오브젝트 그리드는 내장 구현 com.ibm.websphere.objectgrid.security.plugins.builtins. CertificateMappingAuthenticator를 제공하여 인증서를 Subject 오브젝트에 맵핑합니다. 구현은 체인의 첫 번째 인증서 식별 이름(DN)을 추출하고 해당 이름을 가지는 프린시펄을 작성합니다. 이 구현은 테스트 목적만을 위한 것입니다.

Tivoli Access Manager 인증기 구현

Tivoli Access Manager는 보안 서버로 광범위하게 존재합니다. 또한 Tivoli Access Manager 제공 로그인 모듈을 사용하여 인증기를 구현할 수 있습니다.

Tivoli Access Manager를 사용하여 사용자를 인증하려면 Tivoli 제공 LoginModule, com.tivoli.mts.PDLoginModule에서 호출 응용프로그램이 다음을 제공해야 합니다.

- 축약 이름 또는 X.500 이름(DN)으로 지정된 프린시펄 이름
- 암호

LoginModule은 프린시펄을 인증하고 Tivoli Access Manager 신임을 리턴합니다. LoginModule은 호출 응용프로그램이 다음 정보를 제공할 것으로 예상합니다.

- javax.security.auth.callback.NameCallback을 통한 사용자 이름
- javax.security.auth.callback.PasswordCallback을 통한 암호.

Tivoli Access Manager 신임이 성공적으로 검색될 때 JAAS LoginModule은 주제 및 PDPrincipal을 작성합니다. 이것은 단지 PDLoginModule에서 사소하므로 TAM 인증을 위한 내장 구현이 제공되지 않습니다. 자세한 정보는IBM Tivoli Access Manager 권한 Java 클래스 개발자 참조서를 참조하십시오.

SSO(Single Sign-On)

오브젝트 그리드 클라이언트가 성공적으로 서버에 인증하면 오브젝트 그리드 서버는 Subject 오브젝트를 작성합니다. 클라이언트와 서버 모두 SSO(Single Sign-On)를 지

원하면 이 Subject 오브젝트가 SSO 토큰으로 변환됩니다. 이 토큰은 다시 소켓과 연관시킬 클라이언트측으로 전달됩니다. 이 SSO 토큰은 인증을 위해 새 서버로 전달될 수 있으므로 다른 서버에서 다시 인증할 필요가 없습니다.

SSO 토큰은 오브젝트 그리드 보안 토큰 관리자 메커니즘을 사용하여 구현됩니다. 보안 토큰 관리자에 대한 자세한 정보는 193 페이지의 『오브젝트 그리드 클러스터 보안』을 참조하십시오. 기본적으로 보안 토큰 메커니즘은 암호 키(비밀 키)를 사용하여 서버 간에 전달하는 사용자 데이터를 암호화 및 암호 해독하고 공용-개인 키를 사용하여 데이터에 서명합니다.

또한 SSO 토큰은 만기 시간을 포함합니다. 보호 도메인에 참여하는 모든 제품은 시간, 날짜 및 시간대를 동기화해야 합니다. 그렇지 않으면, SSO 토큰이 조기에 만기됨으로 나타나 인증 또는 유효성 검증이 실패합니다. (세계시가 사용될 경우에는 필요하지 않습니다.)

오브젝트 그리드 클라이언트가 다른 서버에 연결할 때 이 SSO 토큰을 새 서버로 전달할 수 있습니다. 이 서버는 SSO 토큰의 유효성을 검증하여 이 토큰이 서명이 제거되고 암호 해독되어 조작되지 않았음을 확인합니다. 또한 시간소인을 검사하여 만기되지 않았음을 확인합니다. 토큰이 유효하면 클라이언트가 이 서버에 인증할 필요가 없습니다.

SSO 토큰이 만기되면 서버는 클라이언트를 다시 인증해야 합니다. 서버는 클라이언트가 다시 신임을 제공하도록 요청합니다.

클라이언트의 단일 사인온 사용

클라이언트 단일 사인온은 다음 두 가지 방법으로 사용 가능하게 할 수 있습니다.

- **구성**. security.ogclient.props 파일에서 SSOEnabled 특성을 사용하여 클라이언트측의 단일 사인온을 사용 가능하게 합니다.
- **프로그래밍**. ClientSecurityConfiguration을 사용하여 다음 메소드로 단일 사인온을 사용 가능하게 합니다.

```
/**
* Set whether single sign on is enabled.
* @param enabled whether single sign on is enabled for this
* client or not.
*/
void setSingleSignOnEnabled(boolean enabled);
```

서버의 단일 사인온 사용

서버측의 단일 사인온을 사용 가능하게 하려면 클러스터 XML 파일에서 singleSignOnEnabled 속성을 true로 설정하십시오. 예를 들면 다음과 같습니다.

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

보안이 사용 가능한 경우에만 단일 사인온이 사용 가능합니다.

보안 통신

오브젝트 그리드는 보안 통신을 위해 TCP/IP 및 SSL을 모두 지원합니다. SSL은 클라이언트와 서버 간에 보안 통신을 제공합니다. 사용되는 통신 메커니즘은 다음 특성의 설정에 따라 달라집니다.

- security.ogclient.props 파일의 transportType 특성
- security.ogserver.props 파일의 transportType 특성

표 13. 클라이언트 전송 및 서버 전송 하에서 사용할 전송 프로토콜

클라이언트 **transportType**	서버 **transportType**	결과 프로토콜
TCP/IP	TCP/IP	TCP/IP
	SSL 지원됨	TCP/IP
	SSL 필수	오류
SSL 지원됨	TCP/IP	TCP/IP
	SSL 지원됨	SSL(SSL 실패의 경우 TCP/IP)
	SSL 필수	SSL
SSL 필수	TCP/IP	오류
	SSL 지원됨	SSL
	SSL 필수	SSL

SSL이 사용될 때 클라이언트측과 서버측 모두에 SSL 구성을 제공해야 합니다.

오브젝트 그리드 클라이언트의 SSL 매개변수 구성

클라이언트측 SSL 매개변수는 다음 방법으로 구성될 수 있습니다.

- com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory 팩토리 클래스를 사용하여 com.ibm.websphere.objectgrid.security.config.SSLConfiguration 오브젝트를 작성합니다. 자세한 내용은 API 문서를 참조하십시오.
- `security.ogclient.props` 파일의 매개변수를 구성한 다음 ClientSecurityConfigurationFactory.getClientSecurityConfiguration (String) 메소드를 사용하여 오브젝트 인스턴스를 채웁니다.

다음 특성은 `security.ogclient.props` 파일의 SSL 구성에 대한 것입니다.

- **provider**: SSL JSSE 프로바이더를 지정합니다. IBMJSSE+, IBMJSSE2, SunJSSE 등이 가능한 값입니다. 사용하는 JDK(Java Development Kit)에 기반하여 이 값을 설정하십시오.
- **protocol**: SSL 프로토콜을 지정합니다. SSL+, SSLV2, SSLV3, TLS, TLSv1 등이 가능한 값입니다. 사용하는 JSSE(Java Secure Socket Extension) 프로바이더에 기반하여 이 프로토콜 값을 설정하십시오.

- **alias**: 문자열은 키 스토어의 별명을 나타냅니다. 기본값은 없습니다. 이 특성은 키 스토어에 다중 키 쌍 인증서가 있고 사용자가 인증서 중 하나를 선택하려는 경우 사용됩니다.
- **keyStoreType**: SSL 키 스토어 유형을 지정합니다. JKS+, JCEK, PKCS12 등이 가능한 값입니다. 사용하는 JSSE(Java Secure Socket Extension) 프로바이더에 기반하여 이 값을 설정하십시오.
- **keyStore**: 클라이언트 공용 인증서 및 개인 키가 있는 키 스토어 경로 파일 이름을 지정합니다. 예를 들어, `[OBJECTGRID_HOME]/properties/DummyClientKeyFile.jks`가 있습니다. 이 릴리스에서 하드웨어 지원은 해당되지 않습니다.
- **keyStorePassword**: 키 스토어 경로를 보호할 암호를 지정합니다. 암호는 오브젝트 그리드별 "xor" 알고리즘을 사용하여 간단히 인코드됩니다. PropFilePasswordEncoder 도구를 사용하여 이 특성 파일을 인코드하십시오. 인코드된 암호의 예를 들면, `{x0r}CDo9HgwWW`와 같습니다.
- **trustStoreType**: 신뢰 저장 유형을 지정합니다. JKS+, JCEK, PKCS12 등이 가능한 값입니다. 사용하는 JSSE 프로바이더에 기반하여 이 값을 설정할 수 있습니다.
- **trustStore**: 서버 공용 인증서를 가지는 신뢰 저장 경로 파일 이름을 지정합니다. 예를 들어, `[OBJECTGRID_HOME]/properties/DummyClientTrustFile.jks`가 있습니다.
- **trustStorePassword**: 신뢰 저장 경로를 보호할 암호를 지정합니다. 암호는 오브젝트 그리드별 xor 알고리즘을 사용하여 간단히 인코드됩니다. PropFilePasswordEncoder 도구를 사용하여 이 특성 파일을 인코드하십시오. 인코드된 암호의 예를 들면, `{x0r}CDo9HgwW`와 같습니다.
- **certReqSubjectDN**: 이것은 서버의 인증서 주제 식별 이름(DN)에 필요한 문자열입니다. 클라이언트는 서버 인증서 DN이 이 문자열을 포함하는 경우에만 서버에 연결할 수 있습니다. 값이 널이면 클라이언트는 서버 인증서의 특정 주제 DN을 필요로 하지 않습니다. 예를 들어, 인증서 주제 DN이 "CN=Server1, OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country"이면, "CN=server1", "O=Your Organization", "OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country"가 일치하지만 "CN=server2" 및 "OU=Your Organizational Unit, L=smething, O=Your Organization, S=Your State,C=Your Country"는 일치하지 않습니다. 와일드 카드 일치는 지원되지 않습니다.

오브젝트 서버의 SSL 매개변수 구성

클라이언트측 SSL 매개변수는 security.ogserver.props 파일에서 구성될 수 있습니다. 이 특성 파일은 오브젝트 그리드 서버를 실행할 때 매개변수로 전달될 수 있습니다.

이전 SSL 특성을 제외하면 서버측 SSL 구성은 추가 특성으로 다음을 보유합니다.

- **clientAuthentication**(true+, false). 이 특성이 true로 설정되면 SSL 클라이언트를 인증해야 합니다. 이것은 클라이언트 증명 인증과 다릅니다. 클라이언트 증명 인증은 인증서 체인에 기반하여 클라이언트를 사용자 레지스트리에 인증한다는 의미이지만 이 특성은 서버가 올바른 클라이언트에 연결되도록 합니다.

로컬 오브젝트 그리드 보안

이 주제에서는 로컬 오브젝트 그리드 프로그래밍 모델의 보안에 대해 설명합니다. 로컬 오브젝트 그리드 프로그래밍 모델에서 기본 보안 기능은 권한입니다. 로컬 오브젝트 그리드 프로그래밍 모델은 권한을 지원하지 않습니다. 오브젝트 그리드의 외부에서 인증해야 합니다. 그러나, 오브젝트 그리드는 Subject 오브젝트를 가져오고 유효화하는 플러그인을 제공합니다.

다음 두 가지 방법으로 오브젝트 그리드 보안을 사용 가능하게 할 수 있습니다.

- 구성. ObjectGrid XML 파일을 사용하여 오브젝트 그리드를 정의하고 해당 오브젝트 그리드에서 보안을 사용 가능하게 할 수 있습니다. 다음은 ObjectGridSample 엔터프라이즈 응용프로그램 샘플에서 사용하는 secure-objectgrid-definition.xml 파일입니다. 이 XML 파일에서는 securityEnabled 속성이 true로 설정되어 보안이 사용 가능합니다.

```
<objectGrids>
<objectGrid name="secureClusterObjectGrid" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JASS">
<bean id="TransactionCallback"
classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
...
</objectGrids>
```

- 프로그래밍. API를 사용하여 오브젝트 그리드를 작성할 경우, 오브젝트 그리드 인터페이스에서 다음 메소드를 호출하여 보안을 사용 가능하게 하십시오.

```
/**
* Enable the ObjectGrid security
*/
void setSecurityEnabled();
```

로컬 오브젝트 그리드 프로그래밍 모델에는 인증이 없습니다. 이 메소드로 보안을 설정할 때 권한을 구성합니다. 이 조건은 클라이언트 서버 모델과 일관됩니다. 클라이언트 서버 모델에서 오브젝트 그리드의 보안을 사용 가능하게 하면 해당 오브젝트 그리드 인스턴스의 권한만 사용 가능하게 됩니다.

인증

로컬 오브젝트 그리드 프로그래밍 모델에서 오브젝트 그리드는 인증 메커니즘을 제공하지 않습니다. 오브젝트 그리드는 환경(Application Server 또는 응용프로그램)에 따라 인증이 다릅니다. 오브젝트 그리드를 WebSphere Application Server 또는 WebSphere Extended Deployment에서 사용하는 경우, 응용프로그램은 WebSphere Application Server 보안 인증 메커니즘을 사용할 수 있습니다. 오브젝트 그리드를 J2SE(Java 2 Platform, Standard Edition) 환경에서 실행 중인 경우, 응용프로그램은 JAAS(Java Authentication and Authorization Service) 인증 또는 기타 인증 메커니즘을 사용하여 인증을 관리해야 합니다. JAAS 인증 사용에 대한 자세한 정보는 JAAS 참조 안내서를 참조하십시오.

응용프로그램과 오브젝트 그리드 인스턴스 간의 계약은 javax.security.auth.Subject 오브젝트입니다. Application Server 또는 응용프로그램이 클라이언트를 인증한 후에 응용프로그램은 인증된 authenticated javax.security.auth.Subject 오브젝트를 검색하고 이 Subject 오브젝트를 사용하여 ObjectGrid.getSession(Subject) 메소드를 호출하여 오브젝트 그리드 인스턴스에서 세션을 가져올 수 있습니다. 이 Subject 오브젝트는 맵 데이터에 대한 액세스 권한을 부여하는 데 사용됩니다. 이 계약을 대상(subject) 전달 메커니즘이라고 합니다. 다음은 ObjectGrid.getSession(Subject) API입니다.

```
/**
* This API allows the cache to use a specific subject rather than the one
* configured on the ObjectGrid to get a session.
* @param subject
* @return An instance of Session
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws InvalidSubjectException the subject passed in is invalid based
* on the SubjectValidation mechanism.
*/
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

오브젝트 그리드 인터페이스의 getSession 메소드를 사용하여 Session 오브젝트를 가져올 수도 있습니다.

```
/**
* This returns a Session object that can be used by a single thread at a time.
* It's not allowed to share this Session object between threads without placing a
* critical section around it. While the core framework allows the object to move
* between threads, the TransactionCallback and Loader may prevent this usage,
* especially in J2EE environments. When security is enabled, this will use the
* SubjectSource to get a Subject object.
*
* If the initialize() method has not been invoked prior to the first
* getSession invocation, then an implicit initialization will occur. This ensures
* that all of the configuration is complete before any runtime usage is required.
*
* @see #initialize()
* @return An instance of Session
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws IllegalStateException if this method is called after the
* destroy() method is called.
```

```
*/
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

API 문서는 보안 사용이 가능한 시기를 지정하므로 이 메소드는 SubjectSource 플러그인을 사용하여 Subject 오브젝트를 가져옵니다. SubjectSource 플러그인은 Subject 오브젝트를 지원하기 위해 오브젝트 그리드에 정의된 보안 플러그인 중 하나입니다. 자세한 정보는 『보안 관련 플러그인』을 참조하십시오.

getSession(Subject) 메소드는 로컬 오브젝트 그리드 인스턴스에서만 호출될 수 있습니다. 분산 오브젝트 그리드 구성의 클라이언트측에서 getSession(Subject) 메소드를 호출할 경우 예외가 발생합니다.

보안 관련 플러그인

오브젝트 그리드는 대상(subject) 전달 메커니즘과 관련된 두 개의 보안 플러그인(SubjectSource 및 SubjectValidation 플러그인)을 제공합니다.

SubjectSource 플러그인

com.ibm.websphere.objectgrid.security.plugins.SubjectSource 인터페이스로 표시되는 SubjectSource 플러그인은 오브젝트 그리드 실행 환경에서 Subject 오브젝트를 가져오는 데 사용되는 플러그인입니다. 이 오브젝트 그리드 환경은 오브젝트 그리드를 사용하는 응용프로그램 또는 응용프로그램을 호스트하는 Application Server일 수 있습니다. 인터페이스는 다음과 같습니다.

```
/**
* This plug-in can be used to get a Subject object which represents the
* ObjectGrid client.
* This subject is then used for ObjectGrid authorization. The method
* getSubject is called by the ObjectGrid runtime when the
* ObjectGrid.getSession() method is used to get a session and the
* security is enabled.
*
* This plug-in is useful for an already authenticated client: it
* can retrieve the authenticated Subject object and then pass to the
* ObjectGrid instance. Therefore, there is no need for another
* authentication.
*
* For example, use
* Subject.getSubject(AccessControlContext)
* to get the subject associated with the AccessControlContext and
* then return it in the getSubject implementation.
*
* This plug-in can only be used in a secure domain, such as in a
* ObjectGrid server.
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface SubjectSource {
/**
* Get a Subject object which can represent the ObjectGrid client.
*
* @return a Subject object
* @throws ObjectGridSecurityException any exception during the subject
```

```
* retrieving
*/
Subject getSubject() throws ObjectGridSecurityException;
}
```

SubjectSource 플러그인을 대상(subject) 전달 메커니즘의 대안으로 고려하십시오. 대상(subject) 전달 메커니즘을 사용하면 응용프로그램이 Subject 오브젝트를 검색하고 이를 사용하여 ObjectGrid Session 오브젝트를 가져옵니다. SubjectSource 플러그인을 사용하면 오브젝트 그리드 런타임이 Subject 오브젝트를 검색하고 이를 사용하여 Session 오브젝트를 가져옵니다. 대상(subject) 전달 메커니즘이 Subject 오브젝트에 대한 제어를 응용프로그램에 제공하는 반면, SubjectSource 플러그인 메커니즘은 응용프로그램이 Subject 오브젝트를 검색하지 않도록 합니다.

이 SubjectSource 플러그인을 사용하여 오브젝트 그리드 권한 부여에 사용되는 오브젝트 그리드 클라이언트를 나타내는 Subject 오브젝트를 가져올 수 있습니다. ObjectGrid.getSession() 메소드가 호출될 때 보안 사용이 가능한 경우 오브젝트 그리드 런타임은 Subject getSubject() throws ObjectGridSecurityException() 메소드를 호출합니다.

오브젝트 그리드는 이 플러그인의 기본 구현인 com.ibm.websphere.objectgrid.security.plugins.builtins. WSSubjectSourceImpl을 제공합니다. WebSphere Application Server에서 응용프로그램을 실행 중인 경우 이 구현을 사용하여 호출자 대상(subject) 또는 RunAs 대상(subject)을 검색할 수 있습니다. WebSphere Application Server에서 오브젝트 그리드를 사용 중인 경우 이 클래스를 SubjectSource 구현 클래스로 구성할 수 있습니다. 다음은 WSSubjectSourceImpl.getSubject()의 기본 플로우를 표시하는 스니펫입니다.

```
Subject s = null;
try {
 if (finalType == RUN_AS_SUBJECT) {
  // get the RunAs subject
  s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
}
 else if (finalType == CALLER_SUBJECT) {
  // get the callersubject
  s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
}
}
catch (WSSecurityException wse) {
 throw new ObjectGridSecurityException(wse);
}
return s;
```

기타 세부사항은 SubjectSource 플러그인 및 WSSubjectSourceImpl 구현에 대한 API 문서를 참조하십시오.

SubjectValidation 플러그인

com.ibm.websphere.objectgrid.security.plugins.SubjectValidation 인터페이스

로 표시되는 SubjectValidation 플러그인은 또다른 보안 플러그인입니다. SubjectValidation 플러그인을 사용하여 오브젝트 그리드로 전달되거나 SubjectSource 플러그인에서 검색된 javax.security.auth.Subject가 변경되지 않은 유효한 Subject인지 유효성을 검증할 수 있습니다. 다음은 인터페이스입니다.

```
/**
* This plug-in can be used to validate a javax.security.auth.Subject
* passed to the ObjectGrid is a valid subject which has not been
* tampered with.
*
* An implementation of this plug?in needs support from the Subject
* object creator, because only the creator knows whether the Subject object
* has been tampered with. However, a subject creator may
* not know whether the Subject has been tampered with. In this
* case, this plug-in should not be used.
*
* This plug-in can only be used in a secure domain, such as in a
* application server. Do not put this plug-in on the client side, it is
* ignored.
*
* @ibm-api
*
* @since WAS XD 6.0
*/
public interface SubjectValidation {
/**
* Validate the Subject has not been tampered with.
* @param subject a subject to be validated
* @return the validated Subject object
* @throws InvalidSubjectException
*/
Subject validateSubject(Subject subject) throws
InvalidSubjectException;
}
```

SubjectValidation 인터페이스의 Subject validateSubject(Subject subject) thorws InvalidSubjectException; 메소드는 Subject 오브젝트를 갖고 Subject 오브젝트를 리턴합니다. Subject 오브젝트가 유효한 것으로 간주되는지 여부 및 리턴되는 Subject 오브젝트는 사용자의 구현에 따라 다릅니다. Subject 오브젝트가 유효하지 않은 경우, `InvalidSubjectException`이 발생합니다.

이 메소드로 전달된 Subject 오브젝트를 신뢰하지 않는 경우에는 이 플러그인을 사용할 수 있습니다. 이 경우, Subject 오브젝트를 검색하기 위한 코드를 개발하는 응용프로그램 개발자들을 신뢰할 수 있음을 고려합니다.

Subject 오브젝트가 변경되었는지 여부를 작성자만 알고 있기 때문에 이 플러그인을 구현하려면 Subject 오브젝트 작성자의 지원이 필요합니다. 그러나 일부 대상(subject) 작성자가 Subject가 변경되었는지 여부를 모를 수도 있습니다. 이 경우, 이 플러그인은 유용하지 않습니다.

오브젝트 그리드는 SubjectValidation의 기본 구현인 com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl을 제공합니다. 이 구현을 사용하여 WebSpher가 인증한 대상(subject)의 유효성을 검증할 수 있

습니다. WebSphere Application Server에서 오브젝트 그리드를 사용 중인 경우 사용자는 이 클래스를 SubjectValidation 구현 클래스로 구성할 수 있습니다. WSSubjectValidationImpl 구현은 이 Subject와 연관된 신임 토큰이 변경되지 않은 경우에만 Subject 오브젝트를 유효한 것으로 간주합니다. 즉, Subject 오브젝트의 기타 파트를 변경할 수 있습니다. WSSubjectValidationImpl 구현은 신임 토큰에 해당하는 원래 Subject의 WebSphere Application Server를 요청하고 원래 Subject 오브젝트를 유효성이 검증 Subject 오브젝트로 리턴합니다. 따라서 신임 토큰 이외에 Subject 컨텐츠에 작성된 변경사항은 영향을 주지 않습니다. 아래의 코드 스니펫은 WSSubjectValidationImpl.validateSubject (Subject)의 기본 플로우를 표시합니다.

```
// Create a LoginContext with scheme WSLogin and
// pass a Callback handler.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));
// When this method is called, the callback handler methods
// will be called to log the user in.
lc.login();
// Get the subject from the LoginContext
return lc.getSubject();
```

이전 코드 스니펫에서 신임 토큰 콜백 핸들러 오브젝트인 WSCredTOkenCallbackHandlerImpl은 유효성이 검증될 Subject 오브젝트를 사용하여 작성됩니다. 그런 다음, LoginContext가 로그인 설계 ″WSLogin″을 사용하여 작성됩니다. lc.login() 메소드가 호출될 때 WebSphere Application Server 보안은 Subject 오브젝트에서 신임 토큰을 검색한 후 해당 Subject를 유효성이 검증된 Subject 오브젝트로 리턴합니다.

기타 세부사항은 SubjectValidation 및 WSSubjectValidationImpl의 API 문서를 참조하십시오.

플러그인 구성

SubjectValidation 플러그인 및 SubjectSource 플러그인을 다음 두 가지 방법으로 구성할 수 있습니다.

- **구성**. XML 파일을 사용하여 오브젝트 그리드를 정의하고 이 두 개의 플러그인을 설정할 수 있습니다. 다음 예에서 WSSubjectSourceImpl 클래스는 SubjectSource 플러그인으로 구성되고 WSSubjectValidation 클래스는 SubjectValidation 플러그인으로 구성됩니다.

  ```
  <objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
  <bean id="SubjectSource"
  className="com.ibm.websphere.objectgrid.security.plugins.builtins.
  WSSubjectSourceImpl" />
  <bean id="SubjectValidation"
  className="com.ibm.websphere.objectgrid.security.plugins.builtins.
  WSSubjectValidationImpl" />
  ```

```
<bean id="TransactionCallback"
className="com.ibm.websphere.samples.objectgrid.
HeapTransactionCallback" />
...
</objectGrids>
```

- **프로그램 방식.** API를 통해 오브젝트 그리드를 작성하려는 경우, 다음 메소드를 호출하여 SubjectSource 또는 SubjectValidation 플러그인을 설정할 수 있습니다.

```
/**
* Set the SubjectValidation plug-in for this ObjectGrid instance. A
* SubjectValidation plug-in can be used to validate the Subject object
* passed in is a valid Subject. Refer to {@link SubjectValidation}
* for more details.
* @param subjectValidation the SubjectValidation plug-in
*/
void setSubjectValidation(SubjectValidation subjectValidation);
/**
* Set the SubjectSource plug-in. A SubjectSource plug-in can be used
* to get a Subject object from the environment to represent the
* ObjectGrid client.
*
* @param source the SubjectSource plug-in
*/
void setSubjectSource(SubjectSource source);
```

사용자 고유의 JAAS 인증 코드 작성

사용자 고유의 JAAS 인증 코드를 작성하여 인증을 처리할 수 있습니다. 사용자 고유의 로그인 모듈을 작성한 후 인증 모듈에 맞게 로그인 모듈을 구성해야 합니다.

로그인 모듈은 사용자에 대한 정보를 수신하고 사용자를 인증합니다. 이 정보는 사용자를 식별할 수 있는 임의의 내용일 수 있습니다. 예를 들어, 사용자 ID 및 암호, 클라이언트 인증서 등을 포함할 수 있습니다. 정보를 수신한 후에 로그인 모듈은 정보가 유효한 대상(subject)을 나타내는지 확인한 후 Subject 오브젝트를 작성합니다. 현재 여러 개의 로그인 모듈 구현을 공용으로 사용할 수 있습니다.

로그인 모듈을 작성한 후에는 런타임이 사용할 수 있도록 이 로그인 모듈을 구성하십시오. JAAS 로그인 모듈 구성 파일을 구성해야 합니다. 이 로그인 모듈은 로그인 모듈 및 인증 설계를 포함합니다. 예를 들면, 다음과 같습니다.

```
FileLogin
{
com.acme.auth.FileLoginModule required
};
```

인증 설계는 ″FileLogin″이고 로그인 모듈은 com.acme.auth.FileLoginModule입니다. 필수 토큰은 FileLoginModule 모듈이 이 로그인의 유효성을 검증해야 함을 표시하며, 그렇지 않으면 전부 실패합니다.

다음 방법 중 하나로 JAAS 로그인 모듈 구성 파일 설정을 수행할 수 있습니다.

- java.security 파일의 **login.config.url**에 JAAS 로그인 모듈 구성 파일을 설정합니다(예:login.config.url.1=file:${java.home}/lib/security/file.login).
- JVM 인수 **-Djava.security.auth.login.config**를 사용하여 명령행에서 JAAS 로그인 모듈 구성 파일을 설정합니다(예: -Djava.security.auth.login.config==$JAVA_HOME/lib/security/file.login).

로그인 모듈 작성 및 구성에 대한 자세한 정보는 JAAS 인증 학습서를 참조하십시오.

WebSphere Application Server에서 코드가 실행 중인 경우 관리 콘솔에서 JAAS 로그인을 구성하고 이 로그인 구성을 Application Server 구성에 저장해야 합니다. 자세한 정보는 Java 인증 및 권한 서비스에 대한 로그인 구성을 참조하십시오.

권한 부여

클라이언트가 인증된 후 오브젝트 그리드 인증 메커니즘을 사용하여 오브젝트 그리드 맵 데이터 및 관리 타스크에 대한 액세스 권한을 부여할 수 있습니다. 오브젝트 그리드 권한 부여는 Subject 오브젝트를 기반으로 합니다. 오브젝트 그리드는 두 가지 종류의 권한 부여 메커니즘 즉, JAAS(Java Authentication and Authorization Service) 권한 부여 및 사용자 정의 권한 부여를 지원합니다.

권한 클래스

오브젝트 그리드 권한에는 맵의 데이터에 대한 권한과 관리 타스크에 대한 권한, 두 가지 종류가 있습니다. 각 권한은 권한 클래스를 사용합니다. 맵에 액세스하는 권한은 MapPermission 클래스로 나타나고 관리 타스크를 실행하는 권한은 AdminPermission 클래스로 나타납니다.

MapPermission 클래스

오브젝트 그리드에서 com.ibm.websphere.objectgrid.security.MapPermission 공용 클래스는 오브젝트 그리드 자원, 특히 ObjectMap 또는 JavaMap 인터페이스의 메소드에 대한 권한을 나타냅니다. 오브젝트 그리드는 ObjectMap 및 JavaMap의 메소드에 액세스하기 위해 다음 권한 문자열을 정의합니다.

- **read:** 맵에서 데이터를 읽을 수 있는 권한을 부여합니다. 정수 상수는 MapPermission.READ로 정의됩니다.
- **write:** 맵의 데이터를 갱신할 수 있는 권한을 부여합니다. 정수 상수는 MapPermission.WRITE로 정의됩니다.
- **insert:** 맵에 데이터를 삽입할 수 있는 권한을 부여합니다. 정수 상수는 MapPermission.INSERT로 정의됩니다.
- **remove:** 맵에서 데이터를 제거할 수 있는 권한을 부여합니다. 정수 상수는 MapPermission.REMOVE로 정의됩니다.

- **invalidate:** 맵에서 데이터를 무효화할 수 있는 권한을 부여합니다. 정수 상수는 MapPermission.INVALIDATE로 정의됩니다.
- **all**: 모든 권한(read, write, insert, remote 및 invalidate)을 부여합니다. 정수 상수는 MapPermission.ALL로 정의됩니다.

완전한 오브젝트 그리드 맵 이름([ObjectGrid_name].[ObjectMap_name] 형식) 및 권한 문자열 또는 정수 값을 전달하여 MapPermission 오브젝트를 구성할 수 있습니다. 권한 문자열은 ″read, insert″와 같이 쉼표로 구분된 권한 문자열이거나 모든 권한이 부여됨을 의미하는 ″all″일 수 있습니다. 권한 정수 값은 위의 정수 상수 또는 DGMapPermission.GET|DGMapPermission.PUT과 같은 여러 정수 권한 상수의 수학적 “or” 값일 수 있습니다.

클라이언트가 ObjectMap 또는 JavaMap의 메소드를 호출할 때 권한 부여가 발생합니다. 오브젝트 그리드 런타임은 다른 메소드에 대해 다른 권한을 검사합니다. 필수 권한이 클라이언트에 부여되지 않은 경우, AccessControlException이 발생합니다.

표 14. 메소드 및 메소드의 필수 권한 목록

	com.ibm.websphere.objectgrid.ObjectMap com.ibm.websphere.objectgrid.JavaMap
read	boolean containsKey(Object)
	boolean equals(Object)
	Object get(Object)
	Object get(Object, Serializable)
	List getAll(List)
	List getAll(List keyList, Serializable)
	List getAllForUpdate(List, Serializable)
	Object getForUpdate(Object)
	Object getForUpdate(Object, Serializable)
write	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insert	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
	remove Object remove (Object)
	void removeAll(Collection)
invalidate	public void invalidate (Object, boolean)
	void invalidateAll(Collection, boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

권한은 실제로 메소드가 수행하는 작업 대신 사용되는 메소드에 따라 달라집니다. 예를 들어, PUT 메소드는 레코드의 존재 여부에 따라 레코드를 삽입하거나 갱신할 수 있습니다. 그러나, 이때 삽입 또는 갱신 사례는 구분되지 않습니다.

또한 기타 유형으로 조합하여 조작 유형을 수행할 수 있습니다. 예를 들어, 제거한 다음 삽입으로 갱신을 수행할 수 있습니다. 이러한 내용을 기억하고 권한 정책을 설계하십시오.

AdminPermission

관리 권한은 com.ibm.websphere.objectgrid.security.AdminPermission 클래스로 표시됩니다. 오브젝트 그리드는 관리 권한에 대한 두 가지 권한 조치를 정의합니다.

- **admin**: 관리 타스크를 수행하는 권한을 부여합니다.
- **monitor**: 읽기 액세스 전용 관리 타스크인 조치에 대한 권한을 부여합니다.

다음 표에서 서로 다른 권한을 가지는 사용자에게 부여한 세부 조작을 나열합니다. 이러한 조작은 ManagementMBean 인터페이스의 메소드에 해당합니다.

표 15. 관리 타스크와 관리 권한 간의 관계

조작	**admin** 권한	**monitor** 권한
startServer	Y	N
stopServer	Y	N
forceStopServer	Y	N
setServerTrace	Y	N
retrieveServerStatus	Y	Y
getMapStats	Y	Y
getOGStats	Y	Y
getReplicationStats	Y	Y

클라이언트가 admin 권한을 가지면 startServer 타스크를 실행할 수 있고 클라이언트가 monitor 권한을 가지면 startServer 타스크를 실행할 수 없습니다.

권한 부여 메커니즘

오브젝트 그리드는 두 가지 종류의 권한 부여 메커니즘 즉, JAAS authorization 및 사용자 정의 권한 부여를 지원합니다. 이것은 맵 데이터 액세스 권한 및 관리(admin) 권한 모두에 적용됩니다. JAAS 권한은 사용자 중심의 액세스 제어를 사용하여 Java 보안 정책을 확대합니다. 실행 중인 코드와 코드를 실행 중인 사용자를 기준으로 권한을 부여할 수 있습니다. JDK 1.4의 파트입니다.

또한 오브젝트 그리드는 com.ibm.websphere.objectgrid.security.plugins.MapAuthorization 플러그인 및 com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization 플러

그인으로 사용자 정의 권한을 지원합니다. JAAS 권한 부여를 사용하지 않으려는 경우에는 사용자 고유의 권한 부여 메커니즘을 구현할 수 있습니다. 사용자 정의 권한 부여 메커니즘을 사용하면 정책 데이터베이스, Policy Server 또는 Tivoli Access Manager를 사용하여 오브젝트 그리드 권한 부여를 관리할 수 있습니다.

다음 두 가지 방법으로 오브젝트 그리드 권한 부여 메커니즘을 구성할 수 있습니다.

- **구성**. XML 파일을 사용하여 ObjectGrid를 정의하고 권한 부여 메커니즘을 AUTHORIZATION_MECHANISM_JAAS 또는 AUTHORIZATION_MECHANISM_CUSTOM으로 설정할 수 있습니다. 다음은 ObjectGridSample 엔터프라이즈 응용프로그램 샘플에서 사용하는 secure-objectgrid-definition.xml 파일입니다.

```
<objectGrids>
<objectGrid name="secureClusterObjectGrid" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
<bean id="TransactionCallback"
classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
...
</objectGrids>
```

- **프로그래밍**. API를 사용하여 오브젝트 그리드를 작성하려는 경우, 다음 메소드를 호출하여 권한 부여 메커니즘을 설정할 수 있습니다. 이것은 오브젝트 그리드 인스턴스를 직접 인스턴스화할 때 로컬 오브젝트 그리드 프로그래밍 모델에만 적용됩니다.

```
/**
* Set the authorization Mechanism. The default is
* com.ibm.websphere.objectgrid.security.SecurityConstants.
* AUTHORIZATION_MECHANISM_JAAS.
* @param authMechanism the map authorization mechanism
*/
void setAuthorizationMechanism(int authMechanism);
```

JAAS 권한 부여

javax.security.auth.Subject 오브젝트는 인증된 사용자를 나타냅니다. Subject는 프린시펄 세트로 구성되며, 각 프린시펄은 해당 사용자의 ID를 나타냅니다. 예를 들어, Subject는 이름 프린시펄(″Joe Smith″) 및 그룹 프린시펄(″manager″)을 가질 수 있습니다.

JAAS 권한 부여 정책을 사용하여 특정 프린시펄에 권한을 부여할 수 있습니다. 오브젝트 그리드는 Subject를 현재의 액세스 제어 컨텍스트와 연관시킵니다. ObjectMap 또는 Javamap에 대한 각 메소드 호출의 경우, Java 런타임이 정책이 특정 프린시펄에만 필수 권한을 부여하는지 여부를 자동으로 판별합니다. 그러한 경우, 액세스 제어 컨텍스트와 연관된 Subject가 지정된 프린시펄을 포함하는 경우에만 조작이 허용됩니다.

정책 파일의 정책 구문에 친숙해야 합니다. JAAS 권한에 대한 자세한 설명은 JAAS 인증 학습서를 참조하십시오.

오브젝트 그리드에는 ObjectMap 및 JavaMap 메소드 호출에 대한 JAAS 권한 부여를 검사하는 데 사용되는 특수 코드 기본이 있습니다. 이 특수 코드 기

본은 http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction입니다. 프린시펄에 ObjectMap 또는 JavaMap 권한을 부여할 때 이 코드 기본을 사용하십시오. 오브젝트 그리드 JAR(Java Archive) 파일에는 모든 권한이 부여되기 때문에 이 특수 코드가 작성되었습니다.

MapPermission을 부여하기 위한 정책 템플리트는 다음과 같습니다.

```
grant codeBase "http://www.ibm.com/com/ibm/ws/objectgrid/security/
PrivilegedAction"
<Principal field(s)>{
permission com.ibm.websphere.objectgrid.security.MapPermission
"[ObjectGrid_name].[ObjectMap_name]", "action";
....
permission com.ibm.websphere.objectgrid.security.MapPermission
"[ObjectGrid_name].[ObjectMap_name]", "action";
};
```

프린시펄 필드는 다음과 같습니다.

```
Principal Principal_class "principal_name"
```

즉, ″Principal″ 단어 다음에 프린시펄 클래스의 완전한 이름과 프린시펄 이름이 옵니다. map_name은 [ObjectGrid Name].[Map Name] 형식의 완전한 맵 이름입니다(예: ″secureClusterObjectGrid.employees″). 조치는 MapPermission 클래스에 정의된 쉼표로 구분된 권한 문자열입니다(예: “read, insert” 또는 “all”).

와일드 카드 기능이 제한적으로 지원됩니다. 오브젝트 그리드 이름 또는 맵 이름을 바꿔 “any”를 표시할 수 있습니다. 그러나 오브젝트 그리드에는 오브젝트 그리드 이름 또는 맵 이름의 일부를 “*”로 바꾸는 것은 지원하지 않습니다. 따라서, “*.employees”, “clusterObjectGrid.*” 및 “*.*”는 모두 유효한 이름이지만 “cluster*.employees”는 유효하지 않습니다.

예를 들어, ObjectGridSample.ear 샘플 응용프로그램에는 두 개의 권한 부여 정책 파일(fullAccessAuth.policy 및 readInsertAccessAuth.policy)이 정의됩니다. readInsertAccessAuth.policy의 컨텐츠는 다음과 같습니다.

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/
PrivilegedAction"
Principal com.ibm.ws.security.common.auth.WSPrincipalImpl
"principal_name" {
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.employees", "read,insert";
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.offices", "read,insert";
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.sites", "read,insert";
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.counters", "read,insert";
};
```

이 정책에서는 네 개의 맵에 대한 ″insert″ 및 ″read″ 권한만 특정 프린시펄에 부여됩니다. 기타 정책 파일(fullAccessAuth.policy)은 이 맵에 대한 ″all″ 권한을 프린시펄에 부여합니다. 응용프로그램을 실행하기 전에 principal_name 및 principal 클래스를 적절한 값으로 변경하십시오. principal_name 값은 사용자 레지스트리에 따라 다릅니다. 예를 들어, 사용자 레지스트리로 로컬 OS를 사용하는 경우, 시스템 이름은 MACH1이고 사용자 ID는 user1이며 principal_name은 ″MACH1/user1″입니다.

JAAS 권한 부여 정책을 Java 정책 파일에 직접 넣거나 별도의 JAAS 권한 부여 파일에 넣은 후 -Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE] JVM 인수를 사용하거나 java.security 파일의 auth.poliyc.url.x=file:[JAAS_AUTH_POLICY_FILE]을 사용하여 설정할 수 있습니다.

JAAS 권한 설명은 관리 타스크에 대한 액세스 권한을 부여하기 위한 정책을 쓰고 구성할 경우에만 적용됩니다. 유일한 차이점은 ″com.ibm.websphere.objectgrid.security.MapPermission map name, actions;″ 형식을 사용하는 대신 ″com.ibm.websphere.objectgrid.security.AdminPermission action;″을 사용한다는 것입니다. 조치는 ″admin″ 또는 ″monitor″일 수 있습니다.

사용자 정의 맵 권한 부여

오브젝트 그리드는 MapAuthorization 플러그인을 통해 사용자 정의 맵 권한 부여도 지원합니다. 인터페이스는 다음과 같습니다.

```
/**This plugin can be used to authorize ObjectMap/JavaMap accesses to the
* principals represented by the Subject object.
*
* A typical implementation of this plug-in is to retrieve the
* principals from the Subject object, and then check whether
* the specified permissions are granted to the principals.
*
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface MapAuthorization {
/**
* Check whether the principals represented by the Subject object
* in the subject has the specified MapPermission. If
* the permissions are granted, true is returned; otherwise a false
* is returned.
*
* @param subject the subject
* @param permission the permission to access ObjectMap
*
* @return true if the permission is granted; false otherwise.
*/
boolean checkPermission(Subject subject, MapPermission permission);
}
```

이 플러그인을 사용하여 Subject 오브젝트에 포함된 프린시펄에 대한 ObjectMap 및 JavaMap 액세스 권한을 부여할 수 있습니다. 다음 메소드에서

```
boolean checkPermission(Subject subject, MapPermission permission)
```

MapAuthorization 인터페이스는 오브젝트 그리드 런타임에서 호출하여 전달된 (passed-in) Subject 오브젝트에 전달(passed-in) 권한이 있는지 여부를 확인합니다. MapAuthorization 인터페이스가 구현되면 true를 리턴하고, 그렇지 않으면 false를 리턴합니다.

이 플러그인의 일반 구현은 Subject 오브젝트에서 프린시펄을 검색하고 특정 정책을 컨설팅하여 프린시펄에 지정된 권한이 부여되었는지 여부를 확인하는 것입니다. 사용자가 이런 정책을 정의합니다. 예를 들어, 데이터베이스, 일반 파일 또는 Tivoli Access Manager Policy Server에 정책을 정의할 수 있습니다.

오브젝트 그리드는 이 플러그인에 대한 두 개의 기본 구현을 제공합니다. com.ibm.websphere.objectgrid.security.plugins.builtins.JAASMapAuthorizationImpl 클래스는 권한 부여에 JAAS 메커니즘을 사용하는 MapAuthorization의 구현입니다. 또다른 구현 클래스는 com.ibm.websphere.objectgrid.security.plugins.builtins입니다. TAMMapAuthorizationImpl 클래스는 Tivoli Access Manager를 사용하여 오브젝트 그리드 권한 부여를 관리하는 방법을 표시합니다. 다음은 JAASMapAuthorizationImpl.checkPermission(Subject, MapPermission)의 기본 플로우를 표시하는 코드 스니펫입니다.

```
// Creates a PrivilegedExceptionAction to check the permssions.
PrivilegedExceptionAction action =
  MapPermissionCheckAction.getInstance(permission);
Subject.doAsPrivileged(subject, action, null);
```

자세한 정보는 IBM Tivoli Access Manager 권한 Java 클래스 개발자 참조서를 참조하십시오.

설치 후 시나리오에는 이 TAMMapAuthorizationImpl 플러그인을 사용하지 마십시오. 테스트 목적으로만 이 플러그인을 사용하십시오. 특정한 제한적인 전제조건이 필요합니다.

- Subject 오브젝트는 com.tivoli.mts.PDPrincipal 프린시펄을 포함합니다.
- TAM Policy Server는 ObjectMap 또는 JavaMap 이름 오브젝트에 대해 다음 권한을 정의했습니다. Policy Server에 정의된 오브젝트는 [ObjectGrid_name].[ObjectMap_name] 형식으로 ObjectMap 또는 JavaMap 이름과 동일한 이름을 가져야 합니다. 권한은 MapPermission에 정의된 권한 문자열의 첫 번째 문자입니다. 예를 들어, Policy Server에 정의된 ″r″ 권한은 ObjectMap에 대한 ″read″ 권한을 나타냅니다.

다음 스니펫에서는 checkPermission 메소드를 구현하는 방법을 보여줍니다.

```
/**
* @see com.ibm.websphere.objectgrid.security.plugins.
* MapAuthorization#checkPermission
* (javax.security.auth.Subject, com.ibm.websphere.objectgrid.security.
* MapPermission)
*/
public boolean checkPermission(final Subject subject,
MapPermission permission) {
String[] str = permission.getParsedNames();
StringBuffer pdPermissionStr = new StringBuffer(5);
for (int i=0; i<str.length; i++) {
pdPermissionStr.append(str[i].substring(0,1));
}
PDPermission pdPerm = new PDPermission(permission.getName(),
pdPermissionStr.toString());
Set principals = subject.getPrincipals();
Iterator iter= principals.iterator();
while(iter.hasNext()) {
try {
PDPrincipal principal = (PDPrincipal) iter.next();
if (principal.implies(pdPerm)) {
  return true;
}
}
catch (ClassCastException cce) {
// Handle exception
}
}
 return false;
}
```

다음 방법으로 MapAuthorization 플러그인을 구성할 수 있습니다.

- 구성. ObjectGrid XML 파일을 사용하여 MapAuthorization 플러그인을 정의할 수 있습니다. 예를 들면 다음과 같습니다.

  ```
  <objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
  ...
  <bean id="MapAuthorization"
  className="com.ibm.websphere.objectgrid.security.plugins.builtins.
  JAASMapAuthorizationImpl" />
  </objectGrids>
  ```

- 프로그래밍. API를 사용하여 오브젝트 그리드를 작성하려는 경우, 다음 메소드를 호출하여 권한 부여 플러그인을 설정할 수 있습니다. 이것은 오브젝트 그리드 인스턴스를 직접 인스턴스화할 때 로컬 오브젝트 그리드 프로그래밍 모델에만 적용됩니다.

  ```
  /**
  * Sets the MapAuthorization plug-in for this ObjectGrid instance.
  *
  * A {@link MapAuthorization} plug-in can be used to authorize
  * access to the maps. Refer to {@link MapAuthorization}
  * for more details.
  * @param mapAuthorization the MapAuthorization plug-in
  */
  void setMapAuthorization(MapAuthorization mapAuthorization);
  ```

사용자 정의 관리 권한 부여

사용자 정의 맵 데이터 액세스 권한 지원과 같이 오브젝트 그리드는 사용자 정의 관리 권한을 지원합니다. 플러그인은 com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization입니다.

```
/**
* This plug-in can be used to authorize management operations to the
* principals contained in the Subject object. The permissions for the
* management operations are represented by AdminPermission
* objects.
*
* This plug-in is used in an ObjectGrid server. It can be configured in the
* ObjectGrid cluster XML file.
*
* A typical implementation of this plug-in is to retrieve the
* Principal set from the Subject object, and then
* check whether the specified permissions are granted to these principals.
*
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see AdminPermission
*/
public interface AdminAuthorization {
/**
* Checks whether the user represented by the Subject object has the
* specified AdminPermission or not.
*
* If the permissions are granted, true is returned; otherwise
* false is returned.
*
* @param subject the Subject object representing the user
* @param permission the administration permission to check
*
* @return true if the permission is granted; false otherwise.
*
* @see AdminPermission
*/
boolean checkPermission(Subject subject, AdminPermission permission);
}
```

이 플러그인을 사용하여 Subject 오브젝트에 포함된 프린시펄에 대한 관리 액세스 권한을 부여할 수 있습니다. AdminAuthorization 인터페이스의

```
boolean checkPermission(Subject subject, AdminPermission permission)
```

메소드는 오브젝트 그리드 런타임에서 호출하여 전달(passed-in) Subject 오브젝트가 전달(assed-in) 관리 권한을 보유하는지 여부를 확인합니다. AdminAuthorization 인터페이스가 구현되면 true를 리턴하고, 그렇지 않으면 false를 리턴합니다.

보안 요구사항에 따라 이 인터페이스를 구현할 수 있습니다. 오브젝트 그리드는 이 인터페이스의 구현 클래스를 제공하지 않습니다.

클러스터 XML의 클러스터 레벨에 AdminAuthorization 플러그인을 설정할 수 있습니다. 예를 들면 다음과 같습니다.

```
<cluster name="cluster1" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300"
statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost"
clientAccessPort="12503" peerAccessPort="12500" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost"
clientAccessPort="12504" peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator className ="com.ibm.websphere.objectgrid.security.plugins.
builtins.WSTokenAuthenticator"></authenticator>
<adminAuthorization className= "com.ibm.ws.objectgrid.test.security.util.
TestAdminAuthorization"></adminAuthorization>
</cluster>
```

권한 검사 기간

오브젝트 그리드는 성능상의 이유로 맵 권한 검사 결과 캐싱을 지원합니다. 이 메커니즘을 사용하지 않으면 184 페이지의 표 14에 나열된 메소드가 호출될 때 오브젝트 그리드 런타임이 구성된 권한 부여 메커니즘을 호출하여 액세스 권한을 부여합니다. 이 권한 검사 기간이 설정된 경우, 권한 부여 메커니즘이 권한 검사 기간을 기준으로 주기적으로 호출됩니다.

Subject 오브젝트를 기반으로 권한 부여 정보를 캐시합니다. 클라이언트가 메소드에 액세스하려고 시도할 때 오브젝트 그리드 런타임은 Subject 오브젝트에 따라 캐시를 검색합니다. 캐시에서 찾을 수 없으면 런타임은 이 Subject 오브젝트에 부여된 권한을 확인한 다음 권한을 캐시에 저장합니다.

오브젝트 그리드가 초기화되기 전에 권한 검사 기간을 정의해야 합니다. 다음 두 가지 방법으로 권한 검사 기간을 구성할 수 있습니다.

- **구성**. XML 파일을 사용하여 오브젝트 그리드를 정의하고 권한 검사 기간을 설정할 수 있습니다. 다음은 권한 검사 기간을 45초로 설정하기 위한 예입니다.

  ```
  <objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
  permissionCheckPeriod="45">
  <bean id="bean id="TransactionCallback"
  className="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- **프로그래밍**. API를 사용하여 오브젝트 그리드를 작성하려는 경우, 다음 메소드를 호출하여 권한 검사 기간을 설정하십시오. 이 메소드는 오브젝트 그리드 인스턴스가 초기화되기 전에만 호출될 수 있습니다. 이 메소드는 오브젝트 그리드 인스턴스를 직접 초기화할 때 로컬 오브젝트 그리드 프로그래밍 모델에만 적용됩니다.

  ```
  /**
  * This method takes a single parameter indicating how often the customer
  * wants to check the permission used to allow a client access. If the
  * parameter is 0 then every single get/put/update/remove/evict call will
  * ask the authorization mechanism, either JAAS authorization or custom
  ```

```
* authorization to check if the current subject has permission. This may be
* prohibitively expensive from a performance point of view depending on
* the authorization implementation but if this is required then you can do it.
* Alternatively, if the parameter is > 0 then it indicates the number
* of seconds to cache a set of permissions before returning to
* the authorization mechanism to refresh them. This provides much
* better performance but you run the risk that if the backend
* permissions are changed during this time then the ObjectGrid will
* possibly allow or prevent access even though the backend security
* provider has been modified.
*
* @param period the permission check period in seconds.
*/
void setPermissionCheckPeriod(int period);
```

오브젝트 그리드 클러스터 보안

오브젝트 그리드 클러스터 보안은 결합 서버에 올바른 신임이 있는지 확인하여 악성 서버가 클러스터를 결합할 수 없도록 합니다. 오브젝트 그리드는 이러한 목적을 위해 공유 비밀 문자열 메커니즘을 사용합니다.

모든 오브젝트 그리드 서버는 공유 비밀 문자열에 동의합니다. 서버는 클러스터를 결합할 때 비밀 문자열을 제시하도록 인증 확인됩니다. 결합하는 서버의 비밀 문자열이 의장 서버의 비밀 문자열과 일치할 경우 결합 서버는 클러스터를 결합할 수 있고, 그렇지 않으면 결합 요청은 거부됩니다.

텍스트 비밀 지우기 전송은 보안되지 않습니다. 오브젝트 그리드 보안 하부 구조는 서버가 이 비밀을 전송하기 전에 보안할 수 있도록 하는 보안 토큰 관리자 플러그인을 제공합니다. "secure" 조작을 구현하는 방법을 결정해야 합니다. 오브젝트 그리드는 "보안" 작업이 구현되어 비밀을 암호화 및 서명하는 독창적(out-of-box) 구현을 제공합니다.

비밀 문자열(authenticationSecret)은 `security.ogserver.props` 파일에 설정됩니다.

- `authenticationSecret`: 서버를 인증 확인할 비밀 문자열. 서버는 시작할 때 이 문자열을 의장 서버에 제시해야 합니다. 비밀 문자열이 의장 서버의 문자열과 일치하면 이 서버를 클러스트에 결합할 수 있습니다.

SecureTokenManager 플러그인

보안 토큰 관리자 플러그인은 com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager 인터페이스로 표시됩니다. 인터페이스는 다음과 같습니다.

```
package com.ibm.websphere.objectgrid.security.plugins;
import com.ibm.websphere.objectgrid.security.ObjectGridSecurityException;
import com.ibm.websphere.objectgrid.security.SecurityConstants;
/**
* This interface is used by ObjectGrid servers to transform an object to a
* secure token and vice versa. A secure token is a byte array.
* Here is one example of a possible usage: When a server joins the cluster,
* the joining server needs to present a password to the president server in the
* cluster. Before sending the password out, the joining server calls the
* generateToken(Object) method to generate a token for this
* password. The token should be hard to break so the password can be protected
```

```
* securely. The token will then be sent across the wire. Usually the token is
* associated with a time stamp so the malicious replay attack will be difficult.
* On the receving side, the server calls the verifyToken(byte[])
* method to verfiy the token and reconstruct the corresponding object from the
* token.
*
*ObjectGrid utilizes JCE to provide a default implementation of this
* interface. In this implementation, when generating the token, the object is
* encrypted with a time stamp and then signed. To verify a token, the token's
* signature is verified and then decrypted. This implementation will need a key
* store configured in the ObjectGrid servers to support the data
* encrypting and decrypting and signature signing and verifying. Please use
* security.ogserver.props for the secure token key settings.
*
* An implementation class should have a default constructor. Users can set the
* CustomSecureTokenManagerProps property in the server security configuration
* property file. This property will be set on the object using the
* setProperties(String) method.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see SecurityConstants#SECURE_TOKEN_MANAGER_CUSTOM_STRING
* @see SecurityConstants#SECURE_TOKEN_MANAGER_DEFAULT_STRING
*/
public interface SecureTokenManager {
/**
*
Set the user defined properties to the factory
*
*
This method is used to set addtional SecureTokenManager properties
* to the object. These properties can be set using the SecureTokenManagerProps
* property in the server security configuration property file.
* This way, you can customize your factory.
*
* @param properties user defined properties
*/
void setProperties(String properties);
/**
* Generates the token for the specified object.
*
* The generated token should be hard to break.
*
* @param o the object to be protected
*
* @return a token representing the object to be protected
*
* @throws ObjectGridSecurityException if any exception occurs during
* generation of the token byte array
*/
byte[] generateToken(Object o) throws ObjectGridSecurityException;
/**
* Verifies the token and reconstruct the object.
*
* @param bytes the token byte array representing the protected object.
*
* @return the protected object
*
* @throws ObjectGridSecurityException if any exception occurs during
* verification of the token byte array
*/
Object verifyToken(byte[] bytes) throws ObjectGridSecurityException;
}
```

generateToken(Object) 메소드는 보호할 오브젝트를 가져온 다음 다른 사용자가 이해할 수 없는 토큰을 생성합니다. verifyTokens(byte[]) 메소드는 역 프로세스를 수행합니다. 즉, 토큰을 원래 오브젝트로 다시 변환합니다.

단순 SecureTokenManager 구현은 단순 인코딩 알고리즘(예: XOR 알고리즘)을 사용하여 직렬화 형태로 오브젝트를 인코드한 다음 해당 디코딩 알고리즘을 사용하여 토큰을 디코드하는 것입니다. 이러한 구현은 보안이 적용되지 않아 보안 파기가 용이합니다.

오브젝트 그리드는 이 인터페이스에 대한 독창적(out-of-box) 구현을 제공합니다. 구현은 공용 API가 아니고 사용자에게 투명합니다.

기본 구현은 키 쌍을 사용하여 서명하고 서명을 확인하며 비밀 키를 사용하여 컨텐츠를 암호화합니다. 이 작업을 수행하기 위해 모든 서버는 JCKES 유형 키 스토어를 가지고 키 쌍(개인용 키와 공용 키) 및 비밀 키를 저장합니다. 키 스토어는 비밀 키를 저장하려면 JCKES 유형이어야 합니다.

이 키는 암호화 및 서명하거나 전송 끝의 비밀 문자열을 확인하는 데 사용됩니다. 또한 토큰은 만기 시간과 연관되어 있으므로 특정 시간 후에는 만료됩니다. 수신 끝에서 데이터는 확인, 암호화되고 수신기의 비밀 문자열과 비교됩니다. 개인용 키와 공용 키는 동일한 목적을 위해 제공되므로 인증을 위한 서버 쌍 사이에 SSL과 같은 통신 프로토콜이 필요하지 않습니다. 그러나, 서버 통신이 암호화되지 않으면 통신 시 포킹(poking)으로 데이터를 도난 당할 수 있습니다. 토큰이 곧 만료되므로 재생 공격 위협은 최소화됩니다. 이 가능성은 모든 서버가 방화벽 뒤에 전개될 경우 크게 줄어듭니다.

이러한 접근 방식의 단점은 오브젝트 그리드 관리자가 키를 생성하고 키를 보안 문제를 일으킬 수 있는 모든 서버에 전송해야 한다는 것입니다.

구성

보안 토큰 관리자를 사용하려면 `security.ogserver.props` 파일에 다음 특성을 구성하십시오.

- **secureTokenManagerType** 특성: 이 특성은 사용할 보안 토큰 관리자를 표시합니다.
  - 값이 **없음**이면 보안 토큰 관리자가 사용되지 않습니다.
  - 값이 **기본값**이면 기본값으로 제공된 독창적(out-of-box) 보안 토큰 관리자가 사용됩니다.
  - 값이 **사용자 정의**이면 사용자가 제공한 보안 토큰 관리자가 사용됩니다.
- **customSecureTokenManagerClass** 특성: 이 특성은 SecureTokenManager 구현 클래스를 지정합니다. 이것은 secureTokenManagerType 값이 "사용자 정의"인 경우에만 사용됩니다. 구현 클래스에는 인스턴스화할 기본 생성자가 있어야 합니다.

- **customSecureTokenManagerProps** 특성: 이 특성은 사용자 정의 SecureTokenManager 특성을 지정합니다. 이것은 secureTokenManagerType 값이 "사용자 정의"인 경우에만 사용됩니다. 값은 setProperties(String) 메소드를 사용하여 SecureTokenManager 오브젝트로 설정됩니다.
- secureTokenManagerType 값이 기본값으로 설정되면 이 서명 및 암호 키에 다음 구성이 필요합니다.
  - secureTokenKeyStore: 공용-개인용 키 쌍과 비밀 키를 저장하는 키 스토어의 파일 경로 이름을 지정합니다.
  - secureTokenKeyStoreType: 키 스토어 유형을 지정합니다(예: JCKES). 사용하는 JSSE(Java Secure Socket Extension) 프로바이더에 기반하여 이 값을 설정할 수 있습니다. 그러나, 이 키 스토어는 비밀 키를 지원할 수 있어야 합니다.
  - secureTokenKeyStorePassword: 키 스토어를 보호할 암호를 지정합니다.
  - secureTokenKeyPairAlias: 서명 및 확인에 사용되는 공용-개인용 키 쌍의 별명을 지정합니다.
  - secureTokenKeyPairPassword: 서명 및 확인에 사용되는 키 쌍 별명을 보호하는 암호를 지정합니다.
  - secureTokenSecretKeyAlias: 암호에 사용되는 비밀 키 별명을 지정합니다.
  - secureTokenSecretKeyPassword: 비밀 키를 보호하는 암호를 지정합니다.
  - secureTokenCipherAlgorithm: 암호에 사용되는 알고리즘을 지정합니다. 사용하는 JSSE 프로바이더에 기반하여 이 값을 설정할 수 있습니다.
  - secureTokenSignAlgorithm: 오브젝트를 서명하는 데 사용되는 알고리즘을 지정합니다. 사용하는 JSSE 프로바이더에 기반하여 이 값을 설정할 수 있습니다.

게이트웨이 보안

오브젝트 그리드 관리 게이트웨이는 클라이언트 관리 요청을 오브젝트 그리드 서버로 위임하는 위치 역할을 수행합니다. 이 주제에서는 게이트웨이 액세스 보안 방법을 설명합니다.

다음 다이어그램은 예입니다. 오브젝트 그리드 클라이언트는 클러스터에서 통계를 가져올 경우 먼저 요청을 게이트웨이로 전송합니다. 게이트웨이는 이 요청을 두 개 서버로 전송하여 통계를 가져온 다음 통계를 결합합니다. 결합된 통계는 다시 클라이언트로 전송됩니다.

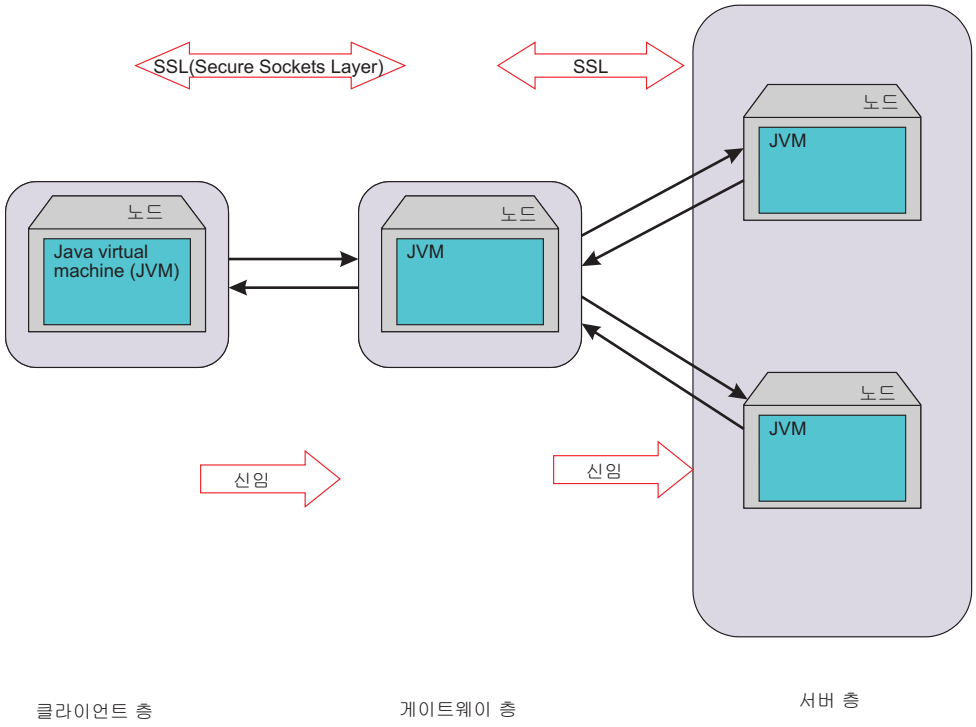


그림 18. 게이트웨이 보안

게이트웨이 및 서버 통신은 오브젝트 그리드 클라이언트 서버 통신 메커니즘을 사용합니다. 게이트웨이는 오브젝트 그리드 클라이언트로 취급됩니다. 클라이언트 및 게이트웨이 통신은 SSL에 의해 보안될 수 있습니다. 이 성능은 개방형 소스 프로젝트 mx4j인 JMX 커넥터 레이어에 의해 제공됩니다. 오브젝트 그리드는 게이트웨이를 작동시키기 위해 mx4j를 필요로 합니다.

인증을 위해 게이트웨이는 클라이언트가 제시한 신임(예: 사용자 ID와 암호)을 서버로 전달합니다. 인증과 권한 모두 오브젝트 그리드 서버에서 강제 실행됩니다.

게이트웨이 클라이언트의 클라이언트 증명 인증은 지원되지 않습니다.

게이트웨이 서버 보안

게이트웨이 서버는 오브젝트 그리드 클라이언트입니다. 모든 보안 요소는 오브젝트 그리드 클라이언트와 동일합니다. 명령행에서 게이트웨이 서버를 시작하는 방법에 대한 자세한 정보는 95 페이지의 『관리 게이트웨이 서버 시작』을 참조하십시오.

다음 코드 스니펫은 보안 게이트웨이를 프로그램 방식으로 시작하는 방법을 설명합니다.

```
// Get the ClientSecurityConfiguration from the client security property file
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration("etc/test/security/security.client.props");
CredentialGenerator creGen = new UserPasswordCredentialGenerator("admin",
"xxxxxx");
csConfig.setCredentialGenerator(credGen);
```

```
// Initialize the gateway
ManagementGateway gateway = ManagementGatewayFactory.getManagementGateway();
gateway.setConnectorPort(namingPort);
gateway.setClusterName("cluster1");
gateway.setHost("localhost");
gateway.setPort("12503");
gateway.setTraceEnabled(true);
gateway.setTraceSpec("ObjectGrid=all=enabled");
gateway.setTraceFile("logs/GatewayTrace.log");
// Set the ClientSecurityConfiguration object
gateway.setCsConfig(csConfig);
// Start the gateway
gateway.startConnector();
```

이전 코드에서는 ClientSecurityConfiguration 오브젝트가 작성되고 ManagementGateway 인스턴스에 설정됩니다.

게이트웨이 클라이언트 보안

게이트웨이 클라이언트는 연결 시 게이트웨이 서버로 신임을 전달해야 합니다. 다음 코드 스니펫은 신임을 전달하는 방법을 설명합니다.

```
/**
* retrieve the server status from the gateway
*/
public boolean retrieveServerStatus()
throws Exception {
String serverProtocol = "rmi";
String serverHost = "host";
String namingHost = "localhost";
String jndiPath = "/jmxconnector";
JMXServiceURL url = new JMXServiceURL("service:jmx:" + serverProtocol + "://"
+ serverHost + "/jndi/rmi://" + namingHost + ":" + namingPort + jndiPath);
// Create the JMXCconnectorServer
JMXConnector cntor = JMXConnectorFactory.newJMXConnector(url, null);
// The connection environment map
Map environment = new HashMap();
// create a credential
UserPasswordCredential gatewayClientCred =
new UserPasswordCredential("admin", "admin1");
environment.put(JMXConnector.CREDENTIALS, gatewayClientCred);
// Connect and invoke an operation on the remote MBeanServer
try {
cntor.connect(environment);
}
catch (SecurityException x) {
// Uh-oh ! Bad credentials !
throw x;
}
// Obtain a stub for the remote MBeanServer
mbsc = cntor.getMBeanServerConnection();
Iterator it = mbsc.queryMBeans(
new ObjectName("ManagementServer:type=ObjectGrid,S=server1"),
null).iterator();
ObjectInstance oi = (ObjectInstance) it.next();
server1MBean = oi.getObjectName();
boolean status = ((Boolean) mbsc.invoke(
server1MBean,
"retrieveServerStatus",
```

```
new Object[] {},
new String[] {})).booleanValue();
return status;
}
```

이 코드 스니펫에서는 gatewayClientCred 오브젝트가 작성되고 환경에 배치됩니다. 그런 다음 이 환경을 사용하여 게이트웨이 서버에 연결합니다.

SSL을 사용하여 게이트웨이 클라이언트에서 게이트웨이 서버로 연결하려면 시스템 특성을 사용하여 신뢰 저장 및 신뢰 저장 암호를 저장해야 합니다. 예를 들어, 게이트웨이 클라이언트를 시작할 때 다음 특성을 전달할 수 있습니다.

- `-Djavax.net.ssl.trustStore=etc/test/security/client.public`
- `-Djavax.net.ssl.trustStorePassword=public`

자세한 정보는 MX4J - 개방형 소스 Java 관리 확장 웹 사이트를 참조하십시오.

WebSphere Application Server에서 보안 통합

오브젝트 그리드는 여러 가지 보안 기능을 제공하여 WebSphere Application Server 보안 하부 구조와 통합합니다.

WebSphere Application Server에서 분산 오브젝트 그리드 보안 통합

분산 오브젝트 그리드 모델의 경우 다음 클래스를 사용하여 보안 통합을 수행할 수 있습니다.

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator.`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential`

이 클래스는 155 페이지의 『클라이언트 서버 보안』에 설명되어 있습니다. 다음은 WSTokenCredentialGenerator 클래스를 사용하는 방법에 대한 예입니다.

```
/**
* connect to the ObjectGrid Server.
*/
protected ClientClusterContext connect() throws ConnectException {
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration(proFile);
CredentialGenerator gen = getWSCredGen();
csConfig.setCredentialGenerator(gen);
return objectGridManager.connect(csConfig, null);
}
/**
* Get a WSTokenCredentialGenerator
*
private CredentialGenerator getWSCredGen() {
```

```
WSTokenCredentialGenerator gen = new WSTokenCredentialGenerator(
WSTokenCredentialGenerator.RUN_AS_SUBJECT);
return gen;
}
```

서버측에서 WSTokenAuthentication을 인증기로 사용하여 WSTokenCredential 오브젝트를 인증할 수 있습니다.

WebSphere Application Server에서 로컬 오브젝트 그리드 보안 통합

로컬 오브젝트 그리드 모델의 경우 다음 두 개 클래스를 사용하여 보안 통합을 수행할 수 있습니다.

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl`

이 클래스에 대한 자세한 정보는 176 페이지의 『로컬 오브젝트 그리드 보안』을 참조하십시오. WSSubjectSourceImpl 클래스를 SubjectSource 플러그인으로 구성하고 WSSubjectValidationImpl 클래스를 SubjectValidation 플러그인으로 구성할 수 있습니다.

리스너

오브젝트 그리드는 사용자가 확장할 수 있는 두 개의 리스너 유형 인터페이스를 제공합니다. 확장 기능은 확장 인터페이스를 통해 사용자를 안내하고 오브젝트 그리드 인스턴스 또는 Map 인스턴스에서 실행되는 조작을 설명합니다.

ObjectGridEventListener 인터페이스

오브젝트 그리드에서 중요 이벤트가 발생할 때 알림을 수신하려면 ObjectGridEventListener 인터페이스를 사용하십시오. 이런 이벤트에는 오브젝트 그리드 초기화, 트랜잭션 시작, 트랜잭션 종료 및 오브젝트 그리드 제거가 포함됩니다. 이런 이벤트를 청취하려면 ObjectGridEventListener 인터페이스를 구현하는 클래스를 작성하고 이를 오브젝트 그리드에 추가하십시오.

ObjectGridEventListener 인터페이스

ObjectGridEventListener 인터페이스에는 다음 메소드가 있습니다. 이 메소드는 오브젝트 그리드에서 특정 중요 이벤트가 발생할 때 호출됩니다.

```
/**
* This method is invoked when the ObjectGrid itself is initialized.
* A usable Session instance is passed into this Listener to allow the
* optional replaying of a received LogSequence into a Map.
*
* @param session The Session instance that this Listener is associated with.
*/
void initialize(Session session);
```

```
/**
* This event signals the beginning of a transaction (session).
* A stringified version of the TxID is provided for
* correlating with the end of the transaction
* (session), if you want to use this version. The type of
* transaction (session) is
* also provided via the isWriteThroughEnabled boolean parameter.
*
* @param txid Stringified version of the TxID
* @param isWriteThroughEnabled Boolean flag indicating whether the
* Session was started via beginNoWriteThrough
*/
void transactionBegin(String txid, boolean isWriteThroughEnabled);
/**
* This signals the ending of a transaction (session). A stringified
* version of the TxID is provided for correlating with the
* begin of the transaction
* (session), if so desired. Changes are also reported. Typical uses of
* this event are for custom peer invalidation
* or peer commit push. This event
* listener outputs the changes. Calls to this method are made
* after commit and are sequenced so that they are delivered one by one,
* not in parallel. The event order is the commit order.
*
* @param txid Stringified version of the TxID
* @param isWriteThroughEnabled a boolean flag indicating
* whether the Sesison was
* started via beginNoWriteThrough
* @param committed a boolean flag indicating whether the Session
* was committed
* (true) or rolled back (false)
* @param changes A Collection of LogSequences that have been
* processed for the current Session. */
void transactionEnd(String txid, boolean isWriteThroughEnabled,
boolean committed, Collection /*/* <LogSequence> */*/ changes);
/**
* This method will be invoked when the ObjectGrid is destroyed. It's the
* opposite of initialize. When this method is called, the
* ObjectGridEventListener can free up any resource it uses.
*/
void destroy();
```

ObjectGridEventListener 오브젝트 추가 및 제거

하나의 오브젝트 그리드가 다중 ObjectGridEventListener를 가질 수 있습니다. 오브젝트 그리드에는 ObjectGridEventListener를 추가할 수 있도록 허용하는 두 개의 메소드가 존재합니다. 추가된 ObjectGridEventListener를 오브젝트 그리드에서 제거할 수도 있습니다.

addEventListener 메소드를 사용하여 오브젝트 그리드에 ObjectGridEventListener를 추가할 수 있습니다.

```
/**
* Add an EventListener to the Session. Significant events
* will be communicated to interested listeners via this callback.
* Multiple event listeners are allowed to be registered, with no
* implied ordering of event notifications.
*
* Note, this method is allowed to be invoked before and after the
* {@link ObjectGrid#initialize()} method.
*
```

```
* @param cb An instance of ObjectGridEventListener
*/
void addEventListener(ObjectGridEventListener cb);
```

ObjectGridEventListeners 목록을 추가하려면 setEventListeners 메소드를 사용하십시오.

```
/**
* This overwrites the current list of callbacks and replaces it with the
* supplied list of callbacks.
*
* Note, this method is allowed to be invoked before and after the
* {@link ObjectGrid#initialize()} method.
* @param callbacks
*/
void setEventListeners(List callbacks);
```

오브젝트 그리드에서 ObjectGridEventListener를 제거하려면 removeEventListener 메소드를 사용하십시오.

```
/**
* Removes an EventListener from the Session. If the desired EventListener
*is not found on the Session, no error will be returned.
*
* Note, this method is allowed to be invoked before and after the
* {@link ObjectGrid#initialize()} method.
* @param cb An instance of ObjectGridEventListener
*/
void removeEventListener(ObjectGridEventListener cb);
```

사용자 정의 오브젝트 그리드 이벤트 리스너 작성

사용자 정의 오브젝트 그리드 이벤트 리스너를 사용하려면 먼저 com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener 인터페이스를 구현하는 클래스를 작성하십시오. 중요 이벤트에 대한 알림을 수신하려면 사용자 정의 리스너를 오브젝트 그리드에 추가하십시오. 프로그램 방식으로 또는 XML을 사용하여 ObjectGridEventListener를 구성할 수 있습니다.

- **프로그램 방식.** 오브젝트 그리드 이벤트 리스너의 클래스 이름이 com.company.org.MyObjectGridEventListener 클래스라고 가정하십시오. 이 클래스는 ObjectGridEventListener 인터페이스를 구현합니다. 다음 코드 스니펫은 사용자 정의 ObjectGridEventListener를 작성하고 이를 오브젝트 그리드에 추가합니다.

  ```
  ObjectGridManager objectGridManager =
   ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
  MyObjectGridEventListener myListener = new MyObjectGridEventListener();
  myGrid.addEventListener(myListener);
  ```

- **XML 사용.** XML을 사용하여 ObjectGridEventListener를 구성할 수도 있습니다. 다음 XML은 설명한 프로그램 방식으로 작성된 오브젝트 그리드 이벤트 리스너와 같은 구성을 작성합니다. 다음 텍스트는 myGrid.xml 파일에 있어야 합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation=
"http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="myGrid">
<bean id="ObjectGridEventListener"
className="com.company.org.MyObjectGridEventListener" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

이 구성을 쉽게 작성하려면 이 파일을 ObjectGridManager에 제공하십시오. 다음 코드 스니펫은 이 XML 파일을 사용하여 오브젝트 그리드를 작성하는 방법을 설명합니다. 작성된 오브젝트 그리드에는 myGrid ObjectGrid에 ObjectGridEventListener가 설정되어 있습니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
objectGridManager.createObjectGrid("myGrid", new URL(
"file:etc/test/myGrid.xml"), true, false);
```

Map 변경사항 감시

ObjectGridEventListener 인터페이스의 transactionEnd 메소드는 로컬 Map의 항목을 감시하는 응용프로그램에 매우 유용합니다. 응용프로그램은 이 리스너 중 하나를 추가한 후 transactionEnd 메소드를 사용하여 항목이 변경되는 시기를 알 수 있습니다. 예를 들어, 오브젝트 그리드가 분산 모드에서 작업 중인 경우, 응용프로그램은 수신 변경사항을 감시할 수 있습니다. 복제된 항목이 최신 주가에 대한 것이었다고 가정하십시오. 이 리스너는 도착하는 이 변경사항을 감시하고 포트폴리오에 위치 값을 보관하는 두 번째 Map을 갱신할 수 있습니다. 리스너는 리스너에 제공된 Loader를 사용하여 ObjectGridEventListener 인터페이스의 initialize 메소드에 모든 변경사항을 작성해야 합니다. 일반적으로 리스너는 트랜잭션이 동시 기록인지 여부를 확인하여 로컬 변경사항과 수신 원격 변경사항을 구별할 수 있습니다. 피어 오브젝트 그리드로부터의 수신 변경사항은 항상 동시 기록됩니다.

MapEventListener 인터페이스

맵에 대한 중요 이벤트를 수신하려면 MapEventListener 인터페이스를 사용하십시오. 맵에서 항목이 제거될 때 및 맵 사전 로드가 완료될 때 이벤트가 MapEventListener로 전송됩니다.

MapEventListener 인터페이스

MapEventListener 인터페이스에는 다음 메소드가 있습니다. 사용자 정의 MapEventListener를 작성하려면 com.ibm.websphere.objectgrid.plugins.MapEventListener 인터페이스를 구현하십시오.

```
/**
* This method is invoked when the specified entry is evicted from
* the map. The eviction could have occurred either by Evictor
* processing or by invoking one of the invalidate methods on the
* ObjectMap.
*
* @param key The key for the map entry that was evicted.
* @param value The value that was in the map entry evicted. The value
* object should not be modified.
*
*/
void entryEvicted(Object key, Object value);
/**
* This method is invoked when preload of this map has completed.
*
* @param t A Throwable object that indicates if preload completed without
* any Throwable occurring during the preload of the map. A null reference
* indicates preload completed without any Throwable objects occurring
* during the preload of the map.
*/
void preloadCompleted( Throwable t );
```

MapEventListener 추가 및 제거

다음 BackingMap 메소드는 맵에 MapEventListener를 추가하고 제거할 수 있도록 합니다.

```
/**
* Adds a MapEventListener to this BackingMap.
*
* Note, this method is allowed to be invoked before and after the
* ObjectGrid.initialize() method.
* @param eventListener A non?null reference to a MapEventListener to add
* to the list.
*
* @throws IllegalArgumentException if eventListener is null.
*
* @see MapEventListener
*/
public void addMapEventListener(MapEventListener
 eventListener );
/**
* Sets the list of MapEventListener objects.
*
* If this BackingMap already has a List of
* MapEventListeners, that list is replaced by the
* List passed as an argument to the current invocation
* of this method. This method can be called before and
* after the ObjectGrid.initialize() method.
*
* @param eventListenerList A non?null reference to a List of
* MapEventListener objects.
*
* @throws IllegalArgumentException is thrown if
* eventListenerList is null
* or the eventListenerList contains either a null
* reference or an object that is not an instance of
* MapEventListener.
*
* @see MapEventListener
*/
public void setMapEventListeners( List /*MapEventListener*/
 eventListenerList );
/**
```

```
* Removes a MapEventListener from this BackingMap.
*
* Note, this method is allowed to be invoked before and after the
* ObjectGrid.initialize() method.
*
* @param eventListener A non?null reference to an event listener
* that was previously added by invoking either the
* addMapEventListener(MapEventListener) or
* setMapEventListeners(List) method of this interface.
*
* @throws IllegalArgumentException if eventListener is null.
*
* @see MapEventListener
*/
public void removeMapEventListener(MapEventListener eventListener );
```

MapEventListener 작성

사용자 정의 MapEventListener를 작성하려면 com.ibm.websphere.objectgrid.plugins.MapEventListener 인터페이스를 구현하십시오. MapEventListener를 사용하려면 BackingMap에 이를 추가하십시오. 프로그램 방식으로 또는 XML을 사용하여 MapEventListener를 작성하고 구성할 수 있습니다.

- **프로그램 방식**. 사용자 정의 MapEventListener의 클래스 이름은 com.company.org.MyMapEventListener 클래스입니다. 이 클래스는 MapEventListener 인터페이스를 구현합니다. 다음 코드 스니펫은 사용자 정의 MapEventListener를 작성하고 이를 BackingMap에 추가합니다.

  ```
  ObjectGridManager objectGridManager =
   ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
  BackingMap myMap = myGrid.defineMap("myMap");
  MyMapEventListener myListener = new MyMapEventListener();
  myMap.addMapEventListener(myListener);
  ```

- **XML 작성**. XML을 사용하여 MapEventListner를 구성할 수도 있습니다. 다음 XML은 이전 프로그램 방식 작성과 같은 구성을 수행합니다. 다음 XML은 myGrid.xml 파일에 있어야 합니다.

  ```
  <?xml version="1.0" encoding="UTF-8" ?>
  <objectGridconfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config
  ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="myGrid">
  <backingMap name="myMap" pluginCollectionRef="myPlugins" />
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="myPlugins">
  <bean id="MapEventListener"
  classname="com.company.org.MyMapEventListener" />
  </backingMapPluginCollection>
  </backingMapPluginCollection>
  </objectGridConfig>
  ```

  이 파일을 ObjectGridManager에 제공하면 이 구성을 쉽게 작성할 수 있습니다. 다음 코드 스니펫은 이 XML 파일을 사용하여 오브젝트 그리드를 작

성하는 방법을 표시합니다. 새로 작성된 오브젝트 그리드에는 myMap BackingMap에 MapEventListener가 설정되어 있습니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
objectGridManager.createObjectGrid("myGrid", new URL(
"file:etc/test/myGrid.xml"), true, false);
```

축출기

오브젝트 그리드는 기본 축출기 메커니즘을 제공합니다. 플러그 가능 축출기 메커니즘을 제공할 수도 있습니다.

축출기는 각 BackingMap에서 항목 멤버쉽을 제어합니다. 기본 축출기는 각 BackingMap에 *TTL(Time to Live)* 축출 정책을 사용합니다. 플러그 가능 축출기 메커니즘을 제공하는 경우, 축출기는 일반적으로 시간 대신 항목 수를 기반으로 하는 축출 정책을 사용합니다. 이 주제에서는 두 축출기 유형을 모두 설명합니다.

기본 TTL(Time to Live) 축출기

오브젝트 그리드는 각 BackingMap에 TTL(Time to Live) 축출기를 제공합니다. TTL 축출기는 작성된 각 항목에 대해 만기 시간을 유지보수합니다. 항목의 만기 시간이 도래하면 축출기가 BackingMap에서 항목을 제거합니다. 성능에 미치는 영향을 최소화하기 위해 TTL 축출기는 항목이 만기되기 전에 축출하는 대신 만기 시간 이후에 항목을 축출하도록 기다릴수 있습니다.

BackingMap에는 TTL(Time to Live) 축출기가 각 항목의 만기 시간을 계산하는 방법을 제어하는 데 사용되는 속성이 있습니다. 응용프로그램은 ttlType 속성을 설정하여 TTL 축출기가 만기 시간을 계산하는 방법을 지정합니다. ttlType 속성을 다음 값 중 하나로 설정할 수 있습니다.

- **없음**은 BackingMap의 항목이 만기되지 않음을 표시합니다. TTL 축출기는 이런 항목은 축출하지 않습니다.
- **작성 시간**은 만기 시간 계산 시 항목이 작성되는 시간을 사용함을 표시합니다.
- **마지막 액세스 시간**은 만기 시간 계산 시 항목에 마지막 액세스한 시간을 사용함을 표시합니다.

ttlType 속성이 BackingMap에 설정되지 않은 경우, TTL 축출기가 항목을 축출하지 않도록 기본 유형인 **없음**이 사용됩니다. ttlType 속성이 **작성 시간** 또는 **마지막 액세스 시간**을 설정된 경우, BackingMap의 TTL(Time to Live) 속성 값이 작성 시간 또는 마지막 액세스 시간에 추가되어 만기 시간이 계산됩니다. TTL(Time to Live) 맵 속성의 시간 정밀도는 초 단위입니다. TTL(Time to Live) 속성 값 0은 맵 항목이 영원히 존재함, 즉, 응용프로그램이 명시적으로 맵 항목을 제거하거나 무효화할 때까지 항목이 맵에 남아 있음을 표시하는 데 사용되는 특수 값입니다.

TTL 축출기의 속성 지정

TTL 축출기는 BackingMap 인스턴스와 연관되어 있습니다. 다음 코드 스니펫은 각 항목이 작성될 때 항목이 작성된 이후에 만기 시간이 10분으로 설정되도록 ackingMap 인터페이스를 사용하여 필요한 속성을 설정할 수 있는 방법을 설명합니다.

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.TTLType;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "myMap" );
bm.setTtlEvictorType( TTLType.CREATION_TIME );
bm.setTimeToLive( 600 );
```

setTimeToLive 메소드 인수는 TTL(Time to Live) 값이 초 단위임을 표시하기 때문에 600입니다. initialize 메소드가 오브젝트 그리드 인스턴스에서 호출되기 전에 위 코드를 실행해야 합니다. 오브젝트 그리드 인스턴스가 초기화된 후에는 이런 BackingMap 속성을 변경할 수 없습니다. 코드가 실행된 후에 myMap BackingMap에 삽입되는 항목은 만기 시간을 갖습니다. 만기 시간에 도달한 후에 TTL 축출기는 항목을 제거합니다.

응용프로그램이 만기 시간을 마지막 액세스 시간에 10분을 더한 값으로 설정해야 하는 경우, 위 코드의 한 행을 변경해야 합니다. setTtlEvictorType 메소드로 전달되는 인수가 TTLType.CREATION_TIME에서 TTLType.LAST_ACCESS_TIME으로 변경됩니다. 이 값을 사용하면 마지막 액세스 시간에 10분을 더한 값으로 만기 시간이 계산됩니다. 항목이 처음 작성된 경우, 마지막 액세스 시간은 작성 시간입니다.

TTLType.LAST_ACCESS_TIME이 사용되는 경우, ObjectMap 및 JavaMap 인터페이스를 사용하여 BackingMap TTL(Time to Live) 값을 대체할 수 있습니다. 이 메커니즘은 응용프로그램이 작성되는 각 항목에 다른 TTL(Time to Live) 값을 사용하도록 허용합니다. 이전 코드 스니펫이 ttlType 속성을 LAST_ACCESS_TIME으로 설정하는 데 사용되고 TTL(Time to Live) 값은 BackingMap에서 10분으로 설정되었다고 가정하십시오. 응용프로그램은 항목을 작성하거나 수정하기 전에 다음 코드를 실행하여 각 항목의 TTL(Time to Live) 값을 대체할 수 있습니다.

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.ObjectMap;
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
int oldTimeToLive1 = om.setTimeToLive( 1800 );
om.insert("key1", "value1" );
int oldTimeToLive2 = om.setTimeToLive( 1200 );
om.insert("key2", "value2" );
```

위 코드 스니펫에서 key1 키를 가진 항목은 ObjectMap에서 setTimeToLive(1800) 메소드 호출의 삽입 시간에 30분을 더한 만기 시간을 갖습니다. ObjectMap에서

setTimeToLive 메소드가 이전에 호출되지 않은 경우 BackingMap의 TTL(Time to Live) 값이 기본값으로 사용되기 때문에 oldTimeToLive1 변수는 600으로 설정됩니다.

key2 키를 가진 항목은 ObjectMap에서 setTimeToLive(1200) 메소드 호출의 결과로 삽입 시간에 20분을 더한 만기 시간을 갖습니다. 이전 ObjectMap.setTimeToLive 메소드 호출의 TTL(Time to Live) 값이 TTL(Time to Live)을 1800으로 설정했으므로 oldTimeToLive2 변수는 1800으로 설정됩니다.

이전 예에서는 키 값 key1과 key2의 myMap 맵에 삽입되는 두 개의 맵 항목을 표시합니다. 나중에 새 스레드의 응용프로그램은 이 맵 항목을 새로운 맵 값으로 갱신하려고 합니다. 그러나, 응용프로그램은 각 맵 항목에 대한 삽입 시 사용되는 TTL(Time-To-Live) 값을 보유하고자 합니다. 다음 예는 바로 이 목적을 위해 ObjectMap 인터페이스에 정의된 상수를 사용하여 TTL(Time-To-Live) 값을 보유하는 방법을 나타냅니다.

```
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
om.setTimeToLive( ObjectMap.USE_DEFAULT );
session.begin();
om.update("key1", "updated value1" );
om.update("key2", "updated value2" );
om.insert("key3", "value3" );
session.commit();
```

ObjectMap.USE_DEFAULT 특수 값은 setTimeToLive 메소드 호출에서 사용되므로 key1은 TTL(Time-To-Live) 값을 1800초로 유지하고 이전 트랜잭션에서 이 맵 항목을 삽입할 때 이 값이 사용되므로 key2는 TTL 값을 1200초로 유지합니다.

또한 앞의 예는 삽입 중인 key3의 새 맵 항목을 표시합니다. 이 경우 USE_DEFAULT 특수 값은 이 맵에 대한 TTL(Time to Live) 값의 기본 설정을 사용하는 것으로 나타냅니다. 기본값은 TTL(Time-to-Live) BackingMap 속성으로 정의됩니다. TTL(Time-to-Live) 속성이 BackingMap에 정의된 방법에 대한 자세한 정보는 120 페이지의 『BackingMap 속성』을 참조하십시오.

ObjectMap 및 JavaMap 인터페이스의 setTimeToLive 메소드에 대해서는 API 문서를 참조하십시오. 이는 메소드가 TTLType.LAST_ACCESS_TIME 값 이외의 값을 리턴하면 IllegalStateException 예외가 발생함을 경고합니다. LAST_ACCESS_TIME TTL 축출기 유형을 사용 중인 경우에만 ObjectMap 및 JavaMap을 사용하여 TTL(Time to Live) 값을 대체할 수 있습니다. CREATION_TIME TTL 축출기 유형 또는 NONE TTL 축출기 유형을 사용 중인 경우에는 이 메소드를 사용하여 TTL(Time to Live) 값을 대체할 수 없습니다.

XML 파일을 사용하여 TTL 축출기의 속성 지정

BackingMap 인터페이스를 사용하여 TTL 축출기가 사용할 BackingMap 속성을 프로그램 방식으로 설정하는 대신 XML 파일을 사용하여 각 BackingMap을 구성할 수 있습니다. 다음 코드는 세 개의 서로 다른 BackingMap에 이런 속성을 설정하는 방법을 설명합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="grid1">
<backingMap name="map1" ttlEvictorType="NONE" />
<backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="1800" />
<backingMap name="map3" ttlEvictorType="CREATION_TIME" timeToLive="1200" />
</objectgrid>
</objectGrids>
```

위 예는 map1 BackingMap이 NONE TTL 축출기 유형을 사용함을 표시합니다. map2 BackingMap은 LAST_ACCESS_TIME TTL 축출기 유형을 사용하며, TTL(Time to Live) 값은 1800초 또는 30분입니다. map3 BackingMap은 CREATION_TIME TTL 축출기 유형을 사용하도록 정의되며, TTL(Time to Live) 값은 1200초 또는 20분입니다.

선택적 플러그 가능 축출기

기본 TTL 축출기는 시간을 기준으로 하는 축출 정책을 사용하며, BackingMap의 항목 수는 항목의 만기 시간에 영향을 주지 않습니다. 선택적 플러그 가능 축출기를 사용하여 시간을 기준으로 하는 대신 존재하는 항목 수를 기준으로 항목을 축출할 수 있습니다. 다음 선택적 플러그 가능 축출기는 BackingMap이 일부 *크기* 제한을 초과하면 축출할 항목을 결정하기 위해 일반적으로 사용되는 알고리즘을 제공합니다.

- **LRUEvictor**는 BackingMap이 최대 항목 수를 초과할 때 *가장 오래 전에 사용* 알고리즘을 사용하여 축출할 항목을 결정하는 축출기입니다.
- **LFUEvictor**는 BackingMap이 최대 항목 수를 초과할 때 *가장 적게 사용* 알고리즘을 사용하여 축출할 항목을 결정하는 축출기입니다.

BackingMap은 트랜잭션에서 항목이 작성, 수정 또는 제거될 때 축출기에 알립니다. BackingMap은 이런 항목을 추적하고 BackingMap에서 하나 이상의 항목을 축출할 시기를 선택합니다.

BackingMap에는 최대 *크기*에 대한 구성 정보가 없습니다. 대신, 축출기 동작을 제어하도록 축출기 특성이 설정됩니다. LRUEvictor와 LFUEvictor 둘 다에는 최대 *크기*를 초과한 후에 축출기가 항목 축출을 시작하도록 하는 데 사용되는 최대 *크기* 특성이 있습니다. TTL 축출기와 마찬가지로 LRU 및 LFU 축출기는 최대 항목 수에 도달하면 성능에 미치는 영향을 최소화하기 위해 항목을 즉시 축출하지 않을 수도 있습니다.

LRU 또는 LFU 축출 알고리즘이 특정 응용프로그램에 적합하지 않은 경우, 사용자 고유의 축출기를 작성하여 원하는 축출 계획을 달성할 수 있습니다.

플러그 가능 축출기 지정

축출기가 BackingMap과 연관되어 있기 때문에 BackingMap 인터페이스는 사용할 플러그 가능 축출기를 지정하는 데 사용됩니다. 다음 코드 스니펫은 map1 BackingMap에 LRUEvictor 축출기를, map2 BackingMap에 LFUEvictor 축출기를 지정하는 예입니다.

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor;
import com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
LRUEvictor evictor = new LRUEvictor();
evictor.setMaxSize(1000);
evictor.setSleepTime( 15 );
evictor.setNumberOfLRUQueues( 53 );
bm.setEvictor(evictor);
bm = og.defineMap( "map2" );
LFUEvictor evictor2 = new LFUEvictor();
evictor2.setMaxSize(2000);
evictor2.setSleepTime( 15 );
evictor2.setNumberOfHeaps( 211 );
bm.setEvictor(evictor2);
```

이전 스니펫은 최대 항목 수가 1000인 map1 BackingMap에 사용되는 LRUEvictor 축출기를 표시합니다. LFUEvictor 축출기는 최대 항목 수가 2000인 map2 BackingMap에 사용됩니다. 축출기가 작동되어 항목을 축출해야 하는지 여부를 확인하기 전에 일시 정지(sleep) 상태를 유지하는 기간을 표시하는 일시 정지(sleep) 시간 특성이 LRU 및 LFU 축출기 둘 다에 있습니다. 일시 정지(sleep) 시간은 초 단위로 지정됩니다. 값이 15초면 성능에 미치는 영향을 줄이면서, BackingMap이 너무 커지지 않도록 합니다. 목적은 BackingMap의 *크기*가 지나치게 커지지 않도록 하면서 가능한 최대 일시 정지(sleep) 시간을 사용하는 것입니다.

setNumberOfLRUQueues 메소드는 축출기가 LRU 정보를 관리하는 데 사용하는 LRU 대기열 수를 표시하는 LRUEvictor 특성을 설정합니다. 모든 항목이 동일한 대기열에 LRU 정보를 보관하지 않도록 대기열 콜렉션이 사용됩니다. 이 접근 방식은 동일한 대기열 오브젝트에서 동기화해야 하는 맵 항목 수를 최소화하여 성능을 향상시킬 수 있습니다. 대기열 수를 늘리면 LRU 축출기가 성능에 미칠 수 있는 영향을 최소화할 수 있습니다. 최대 항목 수의 10퍼센트를 대기열 수로 사용하여 시작하는 것이 좋습니다. 일반적으로 소수를 사용하는 것이 소수 이외의 수를 사용하는 것보다 좋습니다.

setNumberOfHeaps 메소드는 LFUEvictor가 LFU 정보를 관리하는 데 사용하는 2진 힙 오브젝트 수를 설정하기 위한 LFUEvictor 특성을 설정합니다. 성능 향상을 위해 콜렉션이 다시 사용됩니다. 최대 항목 수의 10퍼센트를 사용하여 시작하는 것이 좋으며, 일반적으로 소수를 사용하는 것이 소수 이외의 수를 사용하는 것보다 좋습니다.

XML을 사용하여 플러그 가능 축출기 지정

다양한 API를 사용하여 프로그램 방식으로 축출기를 플러그인하고 특성을 설정하는 대신, 아래 샘플에 설명된 대로 XML 파일을 사용하여 각 BackingMap을 구성할 수 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="grid">
<backingMap name="map1" ttlEvictorType="NONE" pluginCollectionRef="LRU" />
<backingMap name="map2" ttlEvictorType="NONE" pluginCollectionRef="LFU" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPlugincollection id="LRU">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
<property name="maxSize" type="int" value="1000"
description="set max size for LRU evictor">
<property name="sleepTime" type="int" value="15"
description="evictor thread sleep time" />
<property name="numberOfLRUQueues" type="int" value="53"
description="set number of LRU queues" />
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="LFU">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
<property name="maxSize" type="int" value="2000"
description="set max size for LFU evictor">
<property name="sleepTime" type="int" value="15"
description="evictor thread sleep time" />
<property name="numberOfHeaps" type="int" value="211"
description="set number of LFU heaps" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

사용자 정의 축출기 작성

오브젝트 그리드를 확장하여 임의의 축출기 알고리즘을 사용할 수 있습니다. com.ibm.websphere.objectgrid.plugins.Evictor 인터페이스를 구현하는 사용자 정의 축출기를 작성해야 합니다. 인터페이스는 다음과 같습니다.

```
public interface Evictor
{
void initialize(BackingMap map, EvictionEventCallback callback);
void destroy();
void apply(LogSequence sequence);
}
```

- initialize 메소드는 BackingMap 오브젝트를 초기화하는 중에 호출됩니다. 이 메소드는 BackingMap에 대한 참조 및 com.ibm.websphere.objectgrid.plugins.EvictionEventCallback 인터페이스를 구현하는 오브젝트에 대한 참조를 사용하여 Evictor 플러그인을 초기화합니다.
- apply 메소드는 하나 이상의 BackingMap 항목에 액세스하는 트랜잭션이 확약될 때 호출됩니다. apply 메소드는 com.ibm.websphere.objectgrid.plugins.LogSequence 인터페이스를 구현하는 오브젝트에 대한 참조를 전달합니다. LogSequence 인터페이스는 Evictor 플러그인이 트랜잭션이 작성, 수정 또는 제거한 BackingMap 항목을 판별할 수 있도록 합니다. Evictor는 이 정보를 사용하여 축출할 항목과 시기를 결정합니다.
- destroy 메소드는 BackingMap이 제거될 때 호출됩니다. 이 메소드는 Evictor가 작성되었을 수 있는 스레드를 종료할 수 있도록 합니다.

EvictionEventCallback 인터페이스에는 다음 메소드가 있습니다.

```
public interface EvictionEventCallback
{
void evictMapEntries(List evictorDataList) throws ObjectGridException;
void evictEntries(List keysToEvictList) throws ObjectGridException;
 void setEvictorData(Object key, Object data);
 Object getEvictorData(Object key);
}
```

EvictionEventCallback 메소드는 다음과 같이 Evictor 플러그인이 오브젝트 그리드 프레임워크로 콜백하는 데 사용됩니다.

- setEvictorData 메소드는 축출기가 작성하는 일부 축출기 오브젝트를 저장하고 키 인수로 표시된 항목과 연관시키는 데 사용되는 프레임워크를 요청하는 데 사용됩니다. 데이터는 축출기별로 다르며, 축출기가 사용 중인 알고리즘를 구현하는 데 필요한 정보로 판별됩니다. 예를 들어, 가장 적게 사용 알고리즘에서 축출기는 주어진 키의 항목을 참조하는 LogElement를 사용하여 apply 메소드가 호출되는 횟수를 추적하기 위해 축출기 데이터 오브젝트에 계수를 유지보수합니다.
- getEvictorData 메소드는 이전 apply 메소드 호출 중에 축출기가 setEvictorData 메소드로 전달한 데이터를 검색하는 데 사용됩니다. 지정된 키 인수의 축출기 데이터를 찾을 수 없는 경우, EvictorCallback 인터페이스에 정의된 특수 KEY_NOT_FOUND 오브젝트가 리턴됩니다.
- evictMapEntries 메소드는 축출기가 하나 이상의 맵 항목 축출을 요청하는 데 사용됩니다. evictorDataList 매개변수의 각 오브젝트는 com.ibm.websphere.objectgrid.plugins.EvictorData 인터페이스를 구현해야 합니다. 또한 setEvictorData 메소드에 전달된 동일한 EvictorData 인스턴스가 이 메소드의 축출기 데이터 목록 매개변수에 있어야 합니다. EvictorData 인스턴스의 getKey 메소드를 사용하여 축출할 맵 항목을 판별합니다. 맵 항목은 캐시 항목이 현재 이 캐시 항목의 축출기 데이터 목록에 있는 EvictorData 인스턴스를 포함할 경우 축출됩니다.

- evictEntries 메소드는 축출기가 하나 이상의 맵 항목 축출을 요청하는 데 사용됩니다. 이 메소드는 setEvictorData 메소드로 전달된 오브젝트가 com.ibm.websphere.objectgrid.plugins.EvictorData 인터페이스를 구현하는 경우에만 사용됩니다.

오브젝트 그리드는 트랜잭션 완료 *후에* Evictor 인터페이스의 apply 메소드를 호출합니다. 완료된 트랜잭션에서 확보한 모든 트랜잭션 잠금은 더 이상 보유되지 않습니다. 잠재적으로 다중 스레드가 apply 메소드를 동시에 호출할 수 있고 각 스레드는 고유 트랜잭션을 완료할 수 있습니다. 트랜잭션 잠금은 완료된 트랜잭션에서 이미 해제했으므로 apply 메소드는 고유의 동기화를 제공하여 apply 메소드가 스레드 문제를 발생시키지 않도록 해야 합니다.

EvictorData 인터페이스를 구현하고 evictEntries 메소드 대신 evictMapEntries 메소드를 사용하는 이유는 잠재적인 타이밍 창을 닫기 위해서입니다. 다음 이벤트 순서를 고려하십시오.

1. 트랜잭션 1이 완료되고 키 1의 맵 항목을 삭제하는 LogSequence와 함께 apply 메소드를 호출합니다.
2. 트랜잭션 2가 완료되고 키 1의 새 맵 항목을 삽입하는 LogSequence와 함께 apply 메소드를 호출합니다. 즉, 트랜잭션 2는 트랜잭션 1로 삭제된 맵 항목을 다시 작성합니다.

축출기는 트랜잭션을 실행하는 스레드에서 비동기식으로 실행되므로 축출기가 키 1을 축출하기로 결정할 때 트랜잭션 1 완료 전에 있었던 맵 항목을 축출하거나 트랜잭션 2에서 다시 작성한 맵 항목을 축출할 수 있습니다. 타이밍 창을 없애고 축출기가 축출하려는 키 1 맵 항목의 버전에 대한 불확실성을 없애려면 setEvictorData 메소드로 전달된 오브젝트로 EvictorData 인터페이스를 구현하십시오. 맵 항목 수명 동안 동일한 EvictorData 인스턴스를 사용하십시오. 맵 항목이 삭제된 다음 다른 트랜잭션에 의해 다시 작성될 때 축출기는 EvictorData 구현의 새 인스턴스를 사용해야 합니다. 축출기는 EvictorData 구현과 evictMapEntries 메소드를 사용하여 맵 항목과 연관된 캐시 항목이 올바른 EvictorData 인스턴스를 포함할 경우, 그리고 이 경우에만 맵 항목이 축출되도록 할 수 있습니다.

Evictor 및 EvictonEventCallback 인터페이스는 응용프로그램이 축출을 위해 사용자 정의 알고리즘를 구현하는 축출기를 플러그인할 수 있도록 합니다. 다음 코드 스니펫은 Evictor 인터페이스의 initialize 메소드를 구현할 수 있는 방법을 설명합니다.

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
import java.util.LinkedList;
// Instance variables
private BackingMap bm;
private EvictionEventCallback evictorCallback;
private LinkedList queue;
```

```
private Thread evictorThread;
public void initialize(BackingMap map, EvictionEventCallback callback)
{
 bm = map;
 evictorCallback = callback;
queue = new LinkedList();
 // spawn evictor thread
 evictorThread = new Thread( this );
String threadName = "MyEvictorForMap?" + bm.getName();
 evictorThread.setName( threadName );
 evictorThread.start();
}
```

위 코드는 맵 및 콜백 오브젝트에 대한 참조를 인스턴스 변수로 저장하여 apply 및 destroy 메소드에 사용할 수 있도록 합니다. 이 예에서는 링크 목록이 작성되어 가장 적게 사용됨(LRU) 알고리즘을 구현하기 위해 *FIFO(First In, First Out)* 대기열로 사용됩니다. 스레드가 파생되고 스레드에 대한 참조는 인스턴스 변수로 보관됩니다. 이 참조를 보관하면 destroy 메소드가 파생된 스레드를 인터럽트 및 종료할 수 있습니다.

스레드 안전이 보장되는 코드를 작성하기 위한 동기화 요구사항을 무시하고 apply 메소드를 구현할 수 있는 방법을 설명합니다.

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.EvictorData;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
public void apply(LogSequence sequence)
{
 Iterator iter = sequence.getAllChanges();
 while ( iter.hasNext() )
{
  LogElement elem = (LogElement)iter.next();
  Object key = elem.getCacheEntry().getKey();
  LogElement.Type type = elem.getType();
  if ( type == LogElement.INSERT )
{
   // do insert processing here by adding to front of LRU queue.
   EvictorData data = new EvictorData(key);
   evictorCallback.setEvictorData(key, data);
   queue.addFirst( data );
}
  else if ( type == LogElement.UPDATE || type == LogElement.FETCH ||
   type == LogElement.TOUCH )
{
   // do update processing here by moving EvictorData object to
   // front of queue.
   EvictorData data = evictorCallback.getEvictorData(key);
   queue.remove(data);
   queue.addFirst(data);
}
  else if ( type == LogElement.DELETE || type == LogElement.EVICT )
{
   // do remove processing here by removing EvictorData object
   // from queue.
```

```
   EvictorData data = evictorCallback.getEvictorData(key);
   if ( data == EvictionEventCallback.KEY_NOT_FOUND )
{
    // Assumption here is your asynchronous evictor thread
    // evicted the map entry before this thread had a chance
    // to process the LogElement request. So you probably
    // need to do nothing when this occurs.
}
else
{
    // Key was found. So process the evictor data.
    if ( data != null )
{
     // Ignore null returned by remove method since spawned
     // evictor thread may have already removed it from queue.
     // But we need this code in case it was not the evictor
     // thread that caused this LogElement to occur.
     queue.remove( data );
}
else
{
     // Depending on how you write you Evictor, this possibility
     // may not exist or it may indicate a defect in your evictor
     // due to improper thread synchronization logic.
}
}
}
}
}
```

일반적으로 apply 메소드에서 삽입 처리는 EvictionEventCallback 인터페이스의 setEvictorData 메소드로 전달되는 축출기 데이터 오브젝트 작성을 처리합니다. 이 축출기는 LRU 구현을 설명하므로 initialize 메소드로 작성된 대기열 맨 앞에 EvictorData도 추가됩니다. 일반적으로 apply 메소드에서 갱신 처리는 apply 메소드의 일부 이전 호출(예: apply 메소드의 삽입 처리)로 작성된 축출기 데이터 오브젝트를 갱신합니다. 이 축출기는 LRU 구현이므로 EvictorData 오브젝트를 현재 대기열 위치에서 대기열 맨 앞으로 이동해야 합니다. 마지막 대기열 요소는 가장 오래 전에 사용된 항목을 나타내므로 파생된 축출기 스레드는 대기열에서 마지막 EvictorData 오브젝트를 제거합니다. EvictorData 오브젝트에는 축출기 스레드가 축출해야 할 항목의 키를 알 수 있도록 getKey 메소드가 있다고 가정합니다. 이 예는 스레드 안전이 보장되는 코드를 작성하기 위한 동기화 요구사항을 무시하고 있음을 주의하십시오. 실제 사용자 정의 축출기는 동기화 지점의 결과로 발생하는 동기화 및 성능 병목 현상을 처리하기 때문에 보다 복잡합니다.

다음 코드 스니펫은 initialize 메소드가 파생시킨 실행 가능한 스레드의 destroy 메소드 및 run 메소드를 설명합니다.

```
// Destroy method simply interrupts the thread spawned by the initialize method.
public void destroy()
{
 evictorThread.interrupt();
}
// Here is the run method of the thread that was spawned by the initialize method.
```

```
public void run()
{
 // Loop until destroy method interrupts this thread.
 boolean continueToRun = true;
 while ( continueToRun )
{
try
{
   // Sleep for a while before sweeping over queue.
   // The sleepTime is a good candidate for a evictor
   // property to be set.
   Thread.sleep( sleepTime );
   int queueSize = queue.size();
   // Evict entries if queue size has grown beyond the
   // maximum size. Obviously, maximum size would
   // be another evictor property.
   int numToEvict = queueSize ? maxSize;
if ( numToEvict > 0 )
{
    // Remove from tail of queue since the tail is the
    // least recently used entry.
    List evictList = new ArrayList( numToEvict );
while( queueSize > ivMaxSize )
{
     EvictorData data = null;
try
{
      EvictorData data = (EvictorData) queue.removeLast();
evictList.add( data );
      queueSize = queue.size();
}
     catch ( NoSuchElementException nse )
{
      // The queue is empty.
      queueSize = 0;
}
}
    // Request eviction if key list is not empty.
    if ( ! evictList.isEmpty() )
{
evictorCallback.evictMapEntries( evictList );
}
}
}
  catch ( InterruptedException e )
{
   continueToRun = false;
}
 } // end while loop
} // end run method.
```

선택적 RollBackEvictor 인터페이스

com.ibm.websphere.objectgrid.plugins.RollbackEvictor 인터페이스는 축출기 플러그인에 의해 선택적으로 구현될 수 있습니다. 이 인터페이스를 구현함으로써 축출기는 트랜잭션이 확약될 때 뿐만 아니라 트랜잭션이 롤백될 때에도 호출할 수 있습니다.

```
public interface RollbackEvictor
{
void rollingBack( LogSequence ls );
}
```

apply 메소드는 트랜잭션이 확약되는 경우에만 호출됩니다. 트랜잭션이 롤백되고 RollbackEvictor 인터페이스가 축출기에 의해 구현되면 rollingBack 메소드가 호출됩니

다. RollbackEvictor 인터페이스가 구현되지 않고 트랜잭션이 롤백된다면 apply 메소드 및 rollingBack 메소드는 호출되지 않습니다.

로더

오브젝트 그리드 로더는 오브젝트 그리드 맵이 동일한 시스템 또는 일부 다른 시스템의 지속적 저장에 일반적으로 보관되는 데이터에 대한 메모리 캐시로 작동하도록 허용하는 플러그 가능 컴포넌트입니다.

일반적으로 데이터베이스 또는 파일 시스템을 지속적 저장으로 사용합니다. 원격 JVM(Java Virtual Machine)은 오브젝트 그리드를 사용하여 허브 기반 캐시를 빌드할 수 있도록 하는 데이터 소스로 사용될 수도 있습니다. 로더에는 지속적 저장에서 데이터를 읽고 쓰기 위한 논리가 있습니다.

Loader는 오브젝트 그리드 BackingMap의 플러그인입니다. 하나의 Loader만 주어진 BackingMap과 연관시킬 수 있습니다. 각 BackingMap에는 고유의 Loader 인스턴스가 있습니다. BackingMap은 로더로부터 포함하고 있지 않은 데이터를 요청합니다. 맵 변경사항이 로더에 푸시됩니다. Loader 플러그인을 사용하면 BackingMap이 맵과 지속적 저장 간에 데이터를 이동할 수 있습니다.

Loader 플러그인

다음 코드 스니펫은 응용프로그램 제공 Loader가 오브젝트 그리드 API를 사용하여 map1에 대한 BackingMap으로 플러그인되는 방법을 설명합니다.

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
MyLoader loader = new MyLoader();
loader.setDataBaseName("testdb");
loader.setIsolationLevel("read committed");
bm.setLoader( loader );
```

MyLoader는 com.ibm.websphere.objectgrid.plugins.Loader 인터페이스를 구현하는 응용프로그램 제공 클래스라고 가정합니다. 오브젝트 그리드가 초기화된 후에는 BackingMap과의 Lodaer 연관을 변경할 수 없기 때문에 호출 중인 오브젝트 그리드 인터페이스의 initialize 메소드를 호출하기 전에 코드를 실행해야 합니다. 초기화가 발생한 후에 호출되는 경우 setLoader 메소드 호출 시 IllegalStateException 예외가 발생합니다.

응용프로그램 제공 Loader는 설정된 특성을 가질 수 있습니다. 예에서는 MyLoader 로더가 관계형 데이터베이스의 테이블에서 데이터를 읽고 쓰는 데 사용됩니다. 로더는 사

용할 데이터베이스 이름과 SQL 분리 레벨을 가져야 합니다. MyLoader 로더에는 응용프로그램이 이 두 Loader 특성을 설정할 수 있도록 허용하는 setDataBaseName 및 setIsolationLevel 메소드가 있습니다.

XML 파일을 사용하여 응용프로그램 제공 Loader를 플러그인할 수도 있습니다. 다음 예는 동일한 데이터베이스 이름과 분리 레벨 Loader 특성을 설정하여 MyLoader 로더를 map1 BackingMap에 플러그인하는 방법을 설명합니다.

```
<?xml version="1.0" encoding="UTF-8" ?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="grid">
<backingMap name="map1" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="map1">
<bean id="Loader" className="com.myapplication.MyLoader">
<property name="dataBaseName" type="java.lang.String" value="testdb"
description="database name" />
<property name="isolationLevel" type="java.lang.String"
value="read committed" description="iso level" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Loader 인터페이스 구현

응용프로그램 제공 Loader는 *com.ibm.websphere.objectgrid.plugins.Loader* 인터페이스를 구현합니다. Loader 인터페이스는 다음 정의를 갖습니다.

```
public interface Loader
{
 static final SpecialValue KEY_NOT_FOUND;
 List get(TxID txid, List keyList, boolean forUpdate)
 throws LoaderException;
void batchUpdate(TxID txid, LogSequence sequence)
 throws LoaderException, OptimisticCollisionException;
 void preloadMap(Session session, BackingMap backingMap)
 throws LoaderException;
}
```

다음 각 섹션은 Loader 인터페이스의 각 메소드를 구현할 때의 고려사항과 설명을 제공합니다.

get 메소드

BackingMap은 Loader get 메소드를 호출하여 **keyList** 인수로 전달되는 키 목록과 연관된 값을 가져옵니다. get 메소드는 키 목록에 있는 각 키에 대해 하나의 java.lang.util.List 값 목록을 리턴해야 합니다. 값 목록에 있는 리턴된 첫 번째 값은 키 목록의 첫 번째 키에 대응하고 값 목록에 있는 리턴된 두 번째 값은 키 목록의 두 번째 키에 대응하는 식입니다. 로더가 키 목록에서 키 값을 찾지 못할 경우, Loader가 Loader 인터페이스에 정의된 특수

KEY_NOT_FOUND 값 오브젝트를 리턴해야 합니다. null을 유효값으로 허용하도록 BackingMap을 구성할 수 있기 때문에, Loader가 키를 찾을 수 없는 경우 Loader가 특수 KEY_NOT_FOUND 오브젝트를 리턴하는 것이 매우 중요합니다. 이 값을 사용하여 BackingMap은 null 값과 키를 찾을 수 없기 때문에 존재하지 않는 값을 구별할 수 있습니다. BackingMap이 null 값을 지원하지 않는 경우, 존재하지 않는 키에 대해 KEY_NOT_FOUND 오브젝트 대신 널을 리턴하는 Loader는 예외를 발생시킵니다.

forUpdate 인수는 응용프로그램이 맵에서 get 메소드를 호출했는지 맵에서 getForUpdate 메소드를 호출했는지 여부를 Loader에 알립니다. 자세한 정보는 com.ibm.websphere.objectgrid.ObjectMap 인터페이스를 참조하십시오. Loader는 지속적 저장에 대한 동시 액세스를 제어하는 동시성 제어 정책 구현을 책임지고 있습니다. 예를 들어, 다수의 관계형 데이터베이스 관리 시스템은 관계형 테이블에서 데이터를 읽는 데 사용되는 SQL select 문에서 for update 구문을 지원합니다. Loader는 부울 true가 이 메소드의 **forUpdate** 매개변수에 대한 인수 값으로 전달되는지 여부를 기준으로 SQL select 문에서 for update 구문을 사용하도록 선택할 수 있습니다. 일반적으로 Loader는 변경이 예상되는 동시성 제어 정책을 사용하는 경우에만 for update 구문을 사용합니다. 변경이 예상되지 않는 동시성 제어의 경우, Loader는 SQL select 문에서 for update 구문을 사용하지 않습니다. Loader는 Loader가 사용 중인 동시성 제어 정책을 기준으로 forUpdate 인수 사용을 결정할 책임을 지고 있습니다.

txid 매개변수에 대한 설명은 234 페이지의 『TransactionCallback 플러그인』 주제를 참조하십시오.

batchUpdate 메소드

batchUpdate 메소드는 Loader 인터페이스에서 중요합니다. 이 메소드는 오브젝트 그리드가 Loader에 현재 변경사항을 모두 적용해야 할 때마다 호출됩니다. Loader에는 이 Map에 대한 변경사항 목록이 제공됩니다. 변경사항은 반복되고 백엔드에 적용됩니다. 메소드는 현재 TxID 값 및 적용할 변경사항을 수신합니다. 다음 샘플은 변경사항 세트에 대해 반복되며 세 개의 JDBC(Java Database Connectivity) 문(하나는 insert를, 다른 하나는 update를, 나머지 하나는 delete를 사용)을 일괄처리합니다.

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
 // Get a SQL connection to use.
 Connection conn = getConnection(tx);
try
```

```
{
  // Process the list of changes and build a set of prepared
  // statements for executing a batch update, insert, or delete
  // SQL operation.
  Iterator iter = sequence.getPendingChanges();
  while ( iter.hasNext() )
{
   LogElement logElement = (LogElement)iter.next();
   Object key = logElement.getCacheEntry().getKey();
   Object value = logElement.getCurrentValue();
switch ( logElement.getType().getCode() )
{
    case LogElement.CODE_INSERT:
    buildBatchSQLInsert( tx, key, value, conn );
break;
    case LogElement.CODE_UPDATE:
    buildBatchSQLUpdate( tx, key, value, conn );
break;
    case LogElement.CODE_DELETE:
    buildBatchSQLDelete( tx, key, conn );
break;
}
}
  // Execute the batch statements that were built by above loop.
  Collection statements = getPreparedStatementCollection( tx, conn );
  iter = statements.iterator();
  while ( iter.hasNext() )
{
   PreparedStatement pstmt = (PreparedStatement) iter.next();
   pstmt.executeBatch();
}
}
catch (SQLException e)
{
  LoaderException ex = new LoaderException(e);
  throw ex;
}
}
```

위 샘플은 LogSequence 인수를 처리하는 상위 레벨 논리를 설명하지만 SQL insert, update 또는 delete 문 빌드 방법에 대한 세부사항은 설명되어 있지 않습니다. 설명된 몇 가지 핵심 내용은 다음과 같습니다.

- getPendingChanges 메소드는 Loader가 처리해야 하는 LogElement 목록에 대한 반복기를 얻기 위해 LogSequence 인수에서 호출됩니다.
- LogElement.getType().getCode() 메소드는 LogElement가 SQL insert, update 또는 delete 조작에 대한 것인지 여부를 판별하는 데 사용됩니다.
- SQLException 예외가 발견되고 일괄처리 갱신 중에 예외가 발생했음을 보고하기 위해 인쇄되는 LoaderException 예외에 연결됩니다.
- JDBC 일괄처리 갱신 지원은 작성해야 하는 백엔드 조회 수를 최소화하는 데 사용됩니다.

preloadMap 메소드

오브젝트 그리드를 초기화하는 동안 정의된 각 BackingMap이 초기화됩니다. Loader가 BackingMap에 플러그인되면, BackingMap은 Loader 인터페이스의 preloadMap 메소드를 호출하여 로더가 백엔드에서 데이터를 프리페치하고 맵에 데이터를 로드할 수 있도록 합니다. 다음 샘플은 Employee 테이블의 처음

100개 행이 데이터베이스에서 읽혀지고 맵에 로드된다고 가정합니다. EmployeeRecord 클래스는 Employee 테이블에서 읽은 직원 데이터를 보유하는 응용프로그램 제공 클래스입니다.

```
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
 boolean tranActive = false;
 ResultSet results = null;
 Statement stmt = null;
 Connection conn = null;
try
{
session.beginNoWriteThrough();
  tranActive = true;
  ObjectMap map = session.getMap( backingMap.getName() );
TxID tx = session.getTxID();
  // Get a auto?commit connection to use that is set to
  // a read committed isolation level.
  conn = getAutoCommitConnection(tx);
  // Preload the Employee Map with EmployeeRecord
  // objects. Read all Employees from table, but
  // limit preload to first 100 rows.
  stmt = conn.createStatement();
  results = stmt.executeQuery( SELECT_ALL );
  int rows = 0;
  while ( results.next() && rows < 100 )
{
   int key = results.getInt(EMPNO_INDEX);
   EmployeeRecord emp = new EmployeeRecord( key );
   emp.setLastName( results.getString(LASTNAME_INDEX) );
   emp.setFirstName( results.getString(FIRSTNAME_INDEX) );
   emp.setDepartmentName( results.getString(DEPTNAME_INDEX) );
   emp.updateSequenceNumber( results.getLong(SEQNO_INDEX) );
   emp.setManagerNumber( results.getInt(MGRNO_INDEX) );
   map.put( new Integer(key), emp );
++rows;
}
  // Commit the transaction.
session.commit();
  tranActive = false;
}
 catch (Throwable t)
{
throw new LoaderException("preload failure: " + t, t);
}
finally
{
  if ( tranActive )
{
try
```

```
{
    session.rollback();
}
   catch ( Throwable t2 )
{
    // Tolerate any rollback failures and
    // allow original Throwable to be thrown.
}
}
  // Be sure to clean up other databases resources here
  // as well such a closing statements, result sets, etc.
}
}
```

이 샘플은 다음 핵심 사항을 설명합니다.

- preloadMap BackingMap은 session 인수로 맵에 전달된 Session 오브젝트를 사용합니다.
- Session.beginNoWriteThrough() 메소드는 begin 메소드보다 오히려 트랜잭션을 시작하는 데 사용됩니다. 맵을 로드하기 위해 이 메소드에서 발생하는 각 put 조작에 대해 Loader를 호출할 수는 없습니다.
- Loader는 EmployeeRecord Java 오브젝트의 필드에 Employee 테이블의 열을 맵핑할 수 있습니다.
- Loader는 발생하는 throwable 예외를 모두 발견하고 발견한 throwable 예외를 Loader에 연결시켜 LoaderException 예외를 발생시킵니다.
- finally 블록은 beginNoWriteThrough 메소드가 호출되는 시간과 commit 메소드가 호출되는 시간 사이에 발생하는 throwable 예외로 finally 블록에서 활성 트랜잭션을 롤백하도록 합니다. 이 조치는 preloadMap 메소드로 시작된 트랜잭션이 호출자에게 리턴되기 전에 완료되도록 하는 데 중요합니다. finally 블록은 필요한 기타 정리 조치(예: JDBC 연결 및 기타 JDBC 오브젝트 닫기)를 수행하기에 적합한 위치입니다.

preloadMap 샘플은 테이블의 행을 모두 선택하는 SQL select 문을 사용하고 있습니다. 응용프로그램 제공 Loader에서는 맵에 사전 로드해야 하는 테이블의 양을 제어하려면 하나 이상의 Loader 특성을 설정해야 할 수도 있습니다.

preloadMap 메소드는 BackingMap 초기화 중에 한 번만 호출되므로 Loader 초기화 코드를 한 번 실행하기에도 적합한 위치입니다. Loader가 백엔드에서 데이터를 프리페치하지 않고 맵에 데이터를 로드하지 않기로 선택하더라도 Loader의 기타 메소드를 보다 효율적으로 만들기 위한 일부 기타 초기화를 한 번 수행해야 할 수도 있습니다. 다음은 TransactionCallback 오브젝트 및 OptimisticCallback 오브젝트를 Loader의 인스턴스 변수로 캐싱하여 Loader의 기타 메소드가 이 오브젝트에 액세스하기 위해 메소드 호출을 작성할 필요가 없도록 하는 예입니다. BackingMap이 초기화된 후에는 TransactionCallback 및

OptimisticCallback 오브젝트를 변경하거나 바꿀 수 없으므로 이 오브젝트 그리드 플러그인 값 캐싱을 수행할 수 있습니다. 이 오브젝트 참조를 Loader의 인스턴스 변수로 캐시할 수 있습니다.

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.OptimisticCallback;
import com.ibm.websphere.objectgrid.plugins.TransactionCallback;
// Loader instance variables.
MyTransactionCallback ivTcb; // MyTransactionCallback
// extends TransactionCallback
MyOptimisticCallback ivOcb; // MyOptimisticCallback
// implements OptimisticCallback
...
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
 // Cache TransactionCallback and OptimisticCallback objects
 // in instance variables of this Loader.
 ivTcb = (MyTransactionCallback)
 session.getObjectGrid().getTransactionCallback();
 ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
 // The remainder of preloadMap code (such as shown in prior example).
}
```

사전 로드, 그리고 복제 실패복구와 관련될 때 복구 가능 사전 로드에 관한 정보는 246 페이지의 『복제 프로그래밍』을 참조하십시오.

로더 고려사항

로더 구현 시 다음 고려사항을 사용하십시오.

사전 로드 고려사항

각 BackingMap에는 맵 사전 로드가 비동기로 완료되는지 여부를 표시하기 위해 설정할 수 있는 부울 preloadMode 속성이 있습니다. 기본적으로, preloadMode 속성은 맵 사전 로드가 완료된 후에 BackingMap 초기화가 완료됨을 표시하는 `false`입니다. 예를 들어, BackingMap 초기화는 `preloadMap` 메소드가 리턴된 후에 완료됩니다. `preloadMap` 메소드가 백엔드에서 대량의 데이터를 읽고 맵에 데이터를 로드할 경우, 완료하려면 비교적 오랜 시간이 걸릴 수 있습니다. 이 경우, preloadMode 속성을 true로 설정하여 맵 비동기 사전 로드를 사용하도록 BackingMap을 구성할 수 있습니다. 이 설정은 BackingMap 초기화 코드가 `preloadMap` 메소드를 호출하는 스레드를 파생시키도록 하여 맵 사전 로드가 진행되는 중에 BackingMap 초기화가 완료될 수 있도록 합니다.

다음 코드 스니펫은 비동기 사전 로드가 사용 가능하도록 preloadMode 속성을 설정하는 방법을 설명합니다.

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

다음 예에 설명된 바와 같이 XML 파일을 사용하여 preloadMode 속성을 설정할 수도 있습니다.

```
<backingMap name="map1" preloadMode="true"
pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
```

TxID 및 TransactionCallback 인터페이스 사용

Loader 인터페이스의 get 메소드와 batchUpdate 메소드 둘 다 get 또는 batchUpdate 조작을 수행해야 하는 Session 트랜잭션을 나타내는 TxID 오브젝트로 전달됩니다. get 및 batchUpdate 메소드가 트랜잭션당 두 번 이상 호출될 수 있습니다. 따라서 Loader가 필요로 하는 트랜잭션 범위의 오브젝트는 일반적으로 TxID 오브젝트 슬롯에 보관됩니다. JDBC(Java Database Connectivity) Loader는 Loader가 TxID 및 TransactionCallback 인터페이스를 사용하는 방법을 설명하는 데 사용됩니다.

여러 개의 오브젝트 그리드 맵을 동일한 데이터베이스에 저장할 수도 있습니다. 각 맵마다 고유 Loader가 있으며 각 Loader는 동일한 데이터베이스에 연결해야 할 수 있습니다. 동일한 데이터베이스에 연결할 때 각 테이블 변경사항이 동일한 데이터베이스 트랜잭션의 일부로 확약되도록 각 Loader는 동일한 JDBC 연결을 사용하려고 합니다. 일반적으로 Loader 구현을 작성하는 동일한 사용자가 TransactionCallback 구현도 작성합니다. TransactionCallback 인터페이스가 확장된 경우 최상의 메소드는 Loader가 데이터베이스 연결을 가져오고 준비된 명령문을 캐싱하는 데 필요한 메소드를 추가하는 것입니다. Loader가 TransactionCallback 및 TxID 인터페이스를 사용하는 방법을 살펴보면 이 방법론이 필요한 이유를 명백히 알 수 있습니다.

한 예로, Loader가 TransactionCallback 인터페이스를 다음과 같이 확장해야 할 수도 있습니다.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
public interface MyTransactionCallback extends TransactionCallback
{
 Connection getAutoCommitConnection(TxID tx, String databaseName)
 throws SQLException;
 Connection getConnection(TxID tx, String databaseName,
 int isolationLevel ) throws SQLException;
 PreparedStatement getPreparedStatement(TxID tx, Connection conn,
 String tableName, String sql) throws SQLException;
 Collection getPreparedStatementCollection( TxID tx, Connection conn,
 String tableName );
}
```

이 새로운 메소드를 사용하면 Loader get 및 batchUpdate 메소드로 다음과 같이 연결을 가져올 수 있습니다.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getConnection(TxID tx, int isolationLevel)
```

```
{
 Connection conn = ivTcb.getConnection(tx, databaseName, isolationLevel );
 return conn;
}
```

위 예와 다음 예에서 *ivTcb* 및 *ivOcb*는 223 페이지의 『사전 로드 고려사항』 섹션에 설명된 대로 초기화된 Loader 인스턴스 변수입니다. *ivTcb* 변수는 MyTransactionCallback 인스턴스에 대한 참조이며, *ivOcb*는 MyOptimisticCallback 인스턴스에 대한 참조입니다. *databaseName* 변수는 BackingMap 초기화 중에 Loader 특성으로 설정된 Loader의 인스턴스 변수입니다. isolationLevel 인수는 JDBC를 지원하는 다양한 분리 레벨에 대해 정의된 JDBC 연결 상수 중 하나입니다. Loader가 변경이 예상되지 않는 구현을 사용 중인 경우, 일반적으로 get 메소드는 JDBC 자동 확약 연결을 사용하여 데이터베이스에서 데이터를 페치합니다. 이 경우, Loader는 다음과 같이 구현되는 getAutoCommitConnection 메소드를 가질 수 있습니다.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getAutoCommitConnection(TxID tx)
{
 Connection conn = ivTcb.getAutoCommitConnection(tx, databaseName);
 return conn;
}
```

batchUpdate 메소드에는 다음 switch 문이 있음을 상기하십시오.

```
switch ( logElement.getType().getCode() )
{
 case LogElement.CODE_INSERT:
 buildBatchSQLInsert( tx, key, value, conn );
break;
 case LogElement.CODE_UPDATE:
 buildBatchSQLUpdate( tx, key, value, conn );
break;
 case LogElement.CODE_DELETE:
 buildBatchSQLDelete( tx, key, conn );
break;
}
```

buildBatchSQL 메소드는 각각 MyTransactionCallback 인터페이스를 사용하여 준비된 명령문을 가져옵니다. 다음은 EmployeeRecord 항목을 갱신하고 일괄처리 갱신을 위해 이를 추가할 수 있도록 SQL update 문을 빌드하는 buildBatchSQLUpdate 메소드를 표시하는 코드 스니펫입니다.

```
private void buildBatchSQLUpdate( TxID tx, Object key, Object value, Connection
conn )
throws SQLException, LoaderException
{
String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?, DEPTNO = ?,
 SEQNO = ?, MGRNO = ? where EMPNO = ?";
PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn, "employee",
sql );
 EmployeeRecord emp = (EmployeeRecord) value;
```

```
 sqlUpdate.setString(1, emp.getLastName());
 sqlUpdate.setString(2, emp.getFirstName());
 sqlUpdate.setString(3, emp.getDepartmentName());
 sqlUpdate.setLong(4, emp.getSequenceNumber());
 sqlUpdate.setInt(5, emp.getManagerNumber());
 sqlUpdate.setInt(6, key);
 sqlUpdate.addBatch();
}
```

batchUpdate 루프는 준비된 명령문을 모두 빌드한 후에 getPreparedStatementCollection 메소드를 호출합니다. 다음과 같이 이 메소드를 구현할 수 있습니다.

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

응용프로그램이 Session의 commit 메소드를 호출할 때 Session 코드는 트랜잭션이 변경한 각 맵에 대해 트랜잭션이 작성한 변경사항을 모두 Loader에 푸시한 후에 TransactionCallback 메소드의 commit 메소드를 호출합니다. 모든 Loader가 MyTransactionCallback 메소드를 사용하여 필요한 준비된 명령문과 연결을 가져왔기 때문에 TransactionCallback 메소드는 백엔드가 변경사항 확약을 요청하기 위해 사용할 연결을 알고 있습니다. 따라서 각 Loader가 필요로 하는 메소드가 있는 TansactionCallback 인터페이스를 확장하면 다음과 같은 장점이 있습니다.

- TransactionCallback 오브젝트가 트랜잭션 범위의 데이터에 대한 TxID 슬롯 사용을 캡슐화하며, Loader는 TxID 슬롯에 대한 정보를 필요로 하지 않습니다. Loader는 Loader가 필요로 하는 기능을 지원하기 위해 MyTransactionCallback 인터페이스를 사용하여 TransactionCallback에 추가되는 메소드에 대해서만 알면 됩니다.
- TransactionCallback 오브젝트는 동일한 백엔드에 연결하는 각 Loader 사이에 연결 공유가 발생하도록 하므로 2단계 확약 프로토콜을 피할 수 있습니다.
- TransactionCallback 오브젝트는 필요한 경우 연결 시 호출되는 확약 또는 롤백을 통해 백엔드 연결이 완료되도록 합니다.
- TransactionCallback은 트랜잭션이 완료될 때 데이터베이스 자원 정리가 발생하도록 합니다.
- TransactionCallback은 WebSphere Application Server와 같은 관리 환경 또는 일부 기타 J2EE(Java 2 Platform, Enterprise Edition) 호환 Application Server에서 관리 중인 연결을 얻고 있는지 여부를 숨길 수 있습니다. 이런 장점으로 인해 동일한 Loader 코드를 관리 및 비관리 환경 모두에서 사용할 수 있습니다. TransactionCallback 플러그인만 변경해야 합니다.

TransactionCallback 구현이 트랜잭션 범위의 데이터에 TxID 슬롯을 사용하는 방법에 대한 자세한 정보는 234 페이지의 『TransactionCallback 플러그인』을 참조하십시오.

OptimisticCallback

앞서 언급한 바와 같이 Loader는 동시성 제어에 변경이 예상되지 않는 접근을 사용하도록 결정할 수 있습니다. 이 경우, 변경이 예상되지 않는 접근을 구현하려면 buildBatchSQLUpdate 메소드 예를 약간 수정해야 합니다. 변경이 예상되지 않는 접근을 사용하기 위한 몇 가지 방법이 있습니다. 일반적인 방법은 행의 각 갱신사항을 버전화하기 위해 시간소인 열 또는 순서 번호 카운터 열을 사용하는 것입니다. Employee 테이블에 행이 갱신될 때마다 증분되는 순서 번호 열이 있다고 가정하십시오.

buildBatchSQLUpdate 메소드의 서명을 수정하여 키 및 값 쌍 대신 LogElement 오브젝트를 전달하도록 할 수 있습니다. 또한 이 메소드는 초기 버전 오브젝트를 가져오고 버전 오브젝트를 갱신하기 위해 BackingMap에 플러그인된 OptimisticCallback 오브젝트를 사용해야 합니다. 다음은 preloadMap 섹션에 설명된 대로 초기화된 *ivOcb* 인스턴스 변수를 사용하는 수정된 buildBatchSQLUpdate 메소드의 예입니다.

```
private void buildBatchSQLUpdate( TxID tx, LogElement le,
 Connection conn )throws SQLException, LoaderException
{
 // Get the initial version object when this map entry was last read
 // or updated in the database.
 Employee emp = (Employee) le.getCurrentValue();
 long initialVersion = ((Long) le.getVersionedValue()).longValue();
 // Get the version object from the updated Employee for the SQL update
//operation.
 Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
 long nextVersion = currentVersion.longValue();
 // Now build SQL update that includes the version object in where clause
 // for optimistic checking.
String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?,
 DEPTNO = ?,SEQNO = ?, MGRNO = ? where EMPNO = ? and SEQNO = ?";
 PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
"employee", sql );
 sqlUpdate.setString(1, emp.getLastName());
 sqlUpdate.setString(2, emp.getFirstName());
 sqlUpdate.setString(3, emp.getDepartmentName());
 sqlUpdate.setLong(4, nextVersion );
 sqlUpdate.setInt(5, emp.getManagerNumber());
 sqlUpdate.setInt(6, key);
 sqlUpdate.setLong(7, initialVersion);
 sqlUpdate.addBatch();
}
```

위 예는 LogElement가 초기 버전 값을 얻는 데 사용됨을 표시합니다. 트랜잭션이 맵 항목에 처음 액세스할 때 맵에서 얻은 초기 Employee 오브젝트를 사용하여 LogElement가 작성됩니다. 또한 초기 Employee 오브젝트는 OptimisticCallback 인터페이스의 getVersionedObjectForValue 메소드로 전달되고 결과는 LogElement에 저장됩니다. 이 처리는 초기 Employee 오브젝트에 대한 참조가 응용프로그램에 주어지기 전에 발생하며, 초기 Employee 오브젝트의 상태를 변경하는 일부 메소드를 호출할 가능성이 있습니다.

예에서는 로더가 getVersiondObjectForValue 메소드를 사용하여 현재 갱신된 Employee 오브젝트의 버전 오브젝트를 확보했다고 표시합니다. Loader 인터페이스의 batchUpdate 메소드를 호출하기 전에, 오브젝트 그리드는 OptimisticCallback 인터페이스의 updateVersionedObjectForValue 메소드를 호출하여 갱신된 Employee 오브젝트에 생성할 새 버전 오브젝트를 생성합니다. batchUpdate 메소드가 오브젝트 그리드로 리턴되면 LogElement는 현재 버전 오브젝트로 갱신되어 새 초기 버전 오브젝트가 됩니다. 응용프로그램이 Session에서 commit 메소드 대신 맵에서 flush 메소드를 호출했을 수도 있으므로 이 단계가 필요합니다. 동일한 키에 대해 단일 트랜잭션으로 Loader를 여러 번 호출할 수 있습니다. 해당 이유로 오브젝트 그리드는 행이 직원 테이블에서 갱신될 때마다 LogElement가 새 버전 오브젝트를 통해 갱신됩니다.

이제 Loader에 초기 버전 오브젝트와 다음 버전 오브젝트가 둘 다 있으므로 SEQNO 열을 다음 버전 오브젝트 값으로 설정하고 `where` 절에 초기 버전 오브젝트 값을 사용하는 SQL update 문을 실행할 수 있습니다. 이 접근 방식을 과규정화 `update` 문이라도고 합니다. 과규정화 `update` 문을 사용하면 관계형 데이터베이스가 이 트랜잭션이 데이터베이스에서 데이터를 읽는 시간과 이 트랜잭션이 데이터베이스를 갱신하는 시간 사이에 일부 다른 트랜잭션에 의해 행이 변경되었는지 확인할 수 있습니다. 다른 트랜잭션이 행을 수정한 경우, 일괄처리 갱신이 리턴하는 계수 배열은 이 키에 대해 0개의 행이 갱신되었음을 표시합니다. Loader는 SQL `update` 조작이 실제로 행을 갱신했는지 확인할 책임을 지고 있습니다. 갱신되지 않은 경우, Loader는 `com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException` 예외를 표시하여 동시에 두 개 이상의 트랜잭션이 데이터베이스 테이블에서 동일한 행을 갱신하려고 시도하기 때문에 batchUpdate 메소드가 실패했음을 Session에 알립니다. 이 예외로 인해 Session은 롤백되며, 응용프로그램은 전체 트랜잭션을 재시도해야 합니다. 이론적으로 재시도가 성공하기 때문에, 이 접근을 변경이 예상되지 않는 접근이라고 합니다. 변경이 예상되지 않는 접근은 데이터를 자주 수정하지 않거나 트랜잭션이 동시에 동일한 행을 갱신하려고 시도하지 않으면 실제로 성능을 향상시킵니다.

로더가 OptimisticCollisionException 생성자의 *key* 매개변수를 사용하여 변경이 예상되지 않는 batchUpdate 메소드가 실패하게 된 키 또는 키 세트를 식별해야 합니다. key 매개변수는 키 오브젝트 자체이거나 둘 이상의 키로 인해 갱신 중 변경이 예상되지 않는 장애가 발생한 경우 키 오브젝트 배열이 될 수 있습니다. 오브젝트 그리드는 OptimisticCollisionException 생성자의 getKey 메소드를 사용하여 무효 데이터를 포함하고 예외를 발생시킨 맵 항목을 판별합니다. 롤백 처리의 일부분에서 각 무효 맵 항목을 맵에서 축출합니다. 동일한 키에 액세스하는 후속 트랜잭션이 호출 중인 Loader 인터페이스의 get 메소드가 데이터베이스의 현재 데이터로 맵 항목을 새로 고치므로 무효 항목을 추출하는 것이 중요합니다.

Loader가 변경이 예상되지 않는 접근을 구현하기 위한 기타 방법은 다음과 같습니다.

- 시간소인 또는 순서 번호 열이 없습니다. 이 경우, OptimisticCallback 인터페이스의 getVersionObjectForValue 메소드는 값 오브젝트 자체만 버전으로 리턴합니다.

이 접근 방식을 사용하면 Loader가 초기 버전 오브젝트의 각 필드를 포함하는 where 절을 빌드해야 합니다. 이 접근 방식은 아주 효율적이지는 않으며, 과규정화 SQL update 문의 where 절에서 모든 열 유형을 사용할 수는 없습니다. 일반적으로는 이 접근 방식을 사용하지 않습니다.

- 시간소인 또는 순서 번호 열이 없습니다. 그러나 이전 접근 방식과 달리 where 절이 트랜잭션이 수정한 값 필드만 포함하고 있습니다. 수정된 필드를 발견하는 한 가지 방법은 BackingMap의 복사 모드를 CopyMode.COPY_ON_WRITE 모드로 설정하는 것입니다. 이 복사 모드에서는 BackingMap 인터페이스의 setCopyMode 메소드로 값 인터페이스를 전달해야 합니다. BackingMap은 제공된 값 인터페이스를 구현하는 동적 프록시 오브젝트를 작성합니다. 이 복사 모드를 사용하면 Loader가 com.ibm.websphere.objectgrid.plugins.ValueProxyInfo 오브젝트로 각 값을 캐스트할 수 있습니다. ValueProxyInfo 인터페이스에는 Loader가 트랜잭션이 변경한 속성 이름 목록을 얻을 수 있도록 하는 메소드가 있습니다. Loader는 이 메소드를 사용하여 속성 이름에 대해 값 인터페이스의 get 메소드를 호출하여 변경된 데이터를 얻고 변경된 속성만 설정하는 SQL update 문을 빌드할 수 있습니다. 이제 where 절을 빌드하여 1차 키와 변경된 각각의 속성 열을 가질 수 있습니다. 이 접근 방식은 이전 접근 방식에 비해 효율적이지만 Loader에 더 많은 코드를 작성해야 하며, 다른 순열을 처리하려면 준비된 명령문 캐시를 늘려야 합니다. 그러나 일반적으로 트랜잭션은 약간의 속성만 수정하므로 이 제한사항은 문제가 되지 않을 수 있습니다.
- 일부 관계형 데이터베이스에는 변경이 예상되지 않는 버전화에 유용한 열 데이터 자동 유지보수를 지원하기 위한 API가 있을 수 있습니다. 데이터베이스 문서를 참조하여 이런 가능성이 있는지 여부를 판별하십시오.

ObjectTransformer 플러그인

고성능이 필요한 경우, ObjectTransformer 플러그인을 사용하십시오. CPU 사용 시 성능 문제점이 있는 경우, 각 맵에 ObjectTransformer 플러그인을 추가하십시오. ObjectTransformer 플러그인을 제공하지 않을 경우, 총 CPU 시간의 최대 60-70%가 항목을 직렬화하고 복사하는 데 사용됩니다.

목적

ObjectTransformer 플러그인의 목적은 응용프로그램에서 다음 조작을 위한 사용자 정의 메소드를 제공할 수 있도록 하는 것입니다.

- 항목의 키 직렬화 또는 직렬화 해제
- 항목의 값 직렬화 또는 직렬화 해제
- 항목의 키 또는 값 복사

ObjectTransformer 플러그인이 제공되지 않는 경우, 오브젝트 그리드가 직렬화 및 직렬화 해제 순서를 사용하여 오브젝트를 복사하므로 키 및 값을 직렬화할 수 있어야 합

니다. 이 메소드에는 많은 비용이 드므로 높은 성능이 반드시 필요한 경우에 ObjectTransformer 플러그인을 사용하십시오. 응용프로그램이 트랜잭션에 처음으로 오브젝트를 검색할 때 복사가 발생합니다. 맵의 복사 모드를 NO_COPY로 설정하여 이 복사를 방지하거나 복사 모드를 COPY_ON_READ로 설정하여 복사를 줄일 수 있습니다. 응용프로그램에 필요한 경우 이 플러그인에 사용자 정의 메소드를 제공하여 복사 조작을 최적화하십시오. 이런 플러그인은 복사 오버헤드를 총 CPU 시간의 65?70%에서 2/3%로 줄일 수 있습니다.

기본 copyKey 및 copyValue 메소드 구현은 clone() 메소드가 제공된 경우 먼저 이 메소드 사용을 시도합니다. clone() 메소드 구현이 제공되지 않으면 구현은 직렬화로 기본 지정됩니다.

또한 오브젝트 그리드가 분배 모드에서 실행 중인 경우에는 오브젝트 직렬화가 직접 사용됩니다. LogSequence는 ObjectTransformer 플러그인을 사용하여 오브젝트 그리드의 피어로 변경사항을 전송하기 전에 키 및 값을 직렬화하도록 돕습니다. 내장 JDK 직렬화 대신 사용자 정의 직렬화 메소드를 제공할 경우에는 유의해야 합니다. 오브젝트 버전화는 복잡한 문제이며, 버전화를 위한 사용자 정의 메소드가 설계되지 않은 경우에는 버전 호환성 문제점이 발생할 수 있습니다.

다음 목록은 오브젝트 그리드로 키와 값을 둘 다 직렬화하는 방법을 자세히 설명합니다.

- 사용자 정의 ObjectTransformer 플러그인이 작성되어 플러그인된 경우, 오브젝트 그리드는 ObjectTransformer 메소드의 메소드를 호출하여 키 및 값을 직렬화하고 오브젝트 키 및 값 사본을 가져옵니다.
- 사용자 정의 ObjectTransformer 플러그인을 사용하지 않는 경우, 오브젝트 그리드는 기본값에 따라 직렬화 및 직렬화 해제를 수행합니다. 기본값을 사용하는 경우, 각 오브젝트는 외부화 가능(externalizable)으로 구현되거나 직렬화 가능(serializable)으로 구현됩니다.
  - 오브젝트가 Externalizable 인터페이스를 지원하는 경우, writeExternal 메소드가 호출됩니다. 외부화 가능으로 구현되는 오브젝트는 성능을 향상시킵니다.
  - 오브젝트가 Externalizable 인터페이스를 지원하지 않고 Serializable을 구현하는 경우, 오브젝트는 ObjectOutputStream 메소드를 사용하여 저장됩니다.

ObjectTransformer 인터페이스

ObjectTransformer 인터페이스에 대한 자세한 정보는 API 문서를 참조하십시오. ObjectTransformer 인터페이스에는 키 또는 값을 직렬화 및 직렬화를 해제하고 키 또는 값을 복사하는 다음 메소드가 있습니다.

```
public interface ObjectTransformer
{
 void serializeKey(Object key, ObjectOutputStream stream)
  throws IOException;
```

```
 void serializeValue(Object value, ObjectOutputStream stream)
  throws IOException;
 Object inflateKey(ObjectInputStream stream)
  throws IOException, ClassNotFoundException;
 Object inflateValue(ObjectInputStream stream)
  throws IOException, ClassNotFoundException;
 Object copyKey(Object value);
 Object copyValue(Object value);
}
```

ObjectTransformer 인터페이스 사용법

다음 상황에서 ObjectTransformer 인터페이스를 사용할 수 있습니다.

- 직렬화 불가능 오브젝트
- 직렬화 가능 오브젝트지만 직렬화 성능 향상
- 키 또는 값 복사

다음 예에서는 오브젝트 그리드를 사용하여 Stock 클래스를 저장합니다.

```
/**
* Stock object for ObjectGrid demo
*
*
*/
public class Stock implements Cloneable {
 String ticket;
 double price;
 String company;
 String description;
 int serialNumber;
 long lastTransactionTime;
/**
 * @return Returns the description.
*/
 public String getDescription() {
  return description;
}
/**
 * @param description The description to set.
*/
 public void setDescription(String description) {
  this.description = description;
}
/**
 * @return Returns the lastTransactionTime.
*/
 public long getLastTransactionTime() {
  return lastTransactionTime;
}
/**
 * @param lastTransactionTime The lastTransactionTime to set.
*/
 public void setLastTransactionTime(long lastTransactionTime) {
  this.lastTransactionTime = lastTransactionTime;
}
```

```
/**
 * @return Returns the price.
*/
 public double getPrice() {
  return price;
}
/**
 * @param price The price to set.
*/
 public void setPrice(double price) {
  this.price = price;
}
/**
 * @return Returns the serialNumber.
*/
 public int getSerialNumber() {
  return serialNumber;
}
/**
 * @param serialNumber The serialNumber to set.
*/
 public void setSerialNumber(int serialNumber) {
  this.serialNumber = serialNumber;
}
/**
 * @return Returns the ticket.
*/
 public String getTicket() {
  return ticket;
}
/**
 * @param ticket The ticket to set.
*/
 public void setTicket(String ticket) {
  this.ticket = ticket;
}
/**
 * @return Returns the company.
*/
 public String getCompany() {
  return company;
}
/**
 * @param company The company to set.
*/
 public void setCompany(String company) {
  this.company = company;
}
//clone
 public Object clone() throws CloneNotSupportedException
{
  return super.clone();
}
}
```

Stock 클래스의 사용자 정의 Object Transformer 클래스를 작성할 수 있습니다.

```
/**
* Custom implementation of ObjectGrid ObjectTransformer for stock object
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
/* (non?Javadoc)
* @see
* com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
 * (java.lang.Object,
 * java.io.ObjectOutputStream)
*/
 public void serializeKey(Object key, ObjectOutputStream stream)
 throws IOException {
  String ticket= (String) key;
  stream.writeUTF(ticket);
}
/* (non?Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
 ObjectTransformer#serializeValue(java.lang.Object,
java.io.ObjectOutputStream)
*/
 public void serializeValue(Object value, ObjectOutputStream stream)
 throws IOException {
  Stock stock= (Stock) value;
  stream.writeUTF(stock.getTicket());
  stream.writeUTF(stock.getCompany());
  stream.writeUTF(stock.getDescription());
  stream.writeDouble(stock.getPrice());
  stream.writeLong(stock.getLastTransactionTime());
  stream.writeInt(stock.getSerialNumber());
}
/* (non?Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateKey(java.io.ObjectInputStream)
*/
 public Object inflateKey(ObjectInputStream stream) throws IOException,
ClassNotFoundException {
  String ticket=stream.readUTF();
  return ticket;
}
/* (non?Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateValue(java.io.ObjectInputStream)
*/
 public Object inflateValue(ObjectInputStream stream) throws IOException,
ClassNotFoundException {
  Stock stock=new Stock();
  stock.setTicket(stream.readUTF());
  stock.setCompany(stream.readUTF());
  stock.setDescription(stream.readUTF());
  stock.setPrice(stream.readDouble());
  stock.setLastTransactionTime(stream.readLong());
  stock.setSerialNumber(stream.readInt());
  return stock;
}
/* (non?Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyValue(java.lang.Object)
```

```
*/
 public Object copyValue(Object value) {
  Stock stock = (Stock) value;
try{
   return stock.clone();
}
  catch (CloneNotSupportedException e)
{
//streamize one
}
}
/* (non?Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
*/
 public Object copyKey(Object key) {
  String ticket=(String) key;
  String ticketCopy= new String (ticket);
  return ticketCopy;
}
}
```

그런 다음, 이 사용자 정의 MyStockObjectTransformer 클래스를 BackingMap에 플러그인하십시오.

```
ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);
```

TransactionCallback 플러그인

일반적으로 응용프로그램은 TransactionCallback 플러그인 및 Loader 모두에 쌍으로 플러그인됩니다. Loader는 백엔드에서 데이터를 페치할 뿐만 아니라 백엔드에 변경사항을 적용해야 합니다. 일반적으로 이 페치 및 비우는 작업은 오브젝트 그리드 트랜잭션의 컨텍스트에서 수행됩니다.

TransactionCallback 플러그인은 다음을 수행해야 합니다.

- 트랜잭션 및 Loader에 필요한 트랜잭션 특정 상태의 슬롯 예약
- 플랫폼 트랜잭션으로 오브젝트 그리드 트랜잭션 변환 또는 맵핑
- 오브젝트 그리드에서 트랜잭션을 시작할 때 개별 트랜잭션 상태 설정
- 오브젝트 그리드 트랜잭션 확약 시 트랜잭션 확약
- 오브젝트 그리드 트랜잭션 롤백 시 트랜잭션 롤백

오브젝트 그리드는 XA 트랜잭션 조정자가 아닙니다. 오브젝트 그리드는 해당 성능을 제공하는 플랫폼에 따라 다릅니다. Session에 표시되는 오브젝트 그리드 `begin`, `commit` 및 `rollback` 메소드는 라이프 사이클 호출입니다. TransactionCallback 플러그인은 이 이벤트를 수신하여 플랫폼에서 Loader가 사용하는 자원에서 트랜잭션 방식의 성능을 제

공하게 해야 합니다. 이 주제에서는 다양한 시나리오를 점검하고 이 시나리오에서 사용할 TransactionCallback 플러그인을 작성하는 방법을 논의합니다.

TransactionCallback 플러그인 개요

TransactionCallback 플러그인은 TransactionCallback 인터페이스를 구현하는 POJO입니다. TransactionCallback 인터페이스는 다음 샘플과 유사합니다.

```
public interface TransactionCallback
{
 void initialize(ObjectGrid objectGrid) throws TransactionCallbackException;
 void begin(TxID id) throws TransactionCallbackException;
 void commit(TxID id) throws TransactionCallbackException;
 void rollback(TxID id) throws TransactionCallbackException;
 boolean isExternalTransactionActive(Session session);
}
```

initialize 메소드

initialize 메소드는 오브젝트 그리드를 초기화할 때 호출됩니다. 콜백에서는 필요한 TxID 오브젝트의 슬롯을 예약합니다. 일반적으로 트랜잭션을 시작할 때 begin 메소드에서 작성할 오브젝트 또는 상태의 각 부분에서 사용할 슬롯을 예약합니다. 예를 들어 오브젝트 그리드에서 Loader로 지속 관리자를 사용하려고 합니다. 이 지속 관리자에 세션 및 트랜잭션 상태 오브젝트가 있다고 가정하면 TransactionCallback은 세션 및 트랜잭션을 얻어 TxID의 슬롯에서 이 두 오브젝트에 대한 참조를 보관합니다. 이 경우 initialize 메소드는 다음 샘플과 유사합니다.

```
/**
* This is called when the grid first initializes. We'll just
* reserve our slots in the TxID.
*/
public void initialize(ObjectGrid objectGrid) throws
TransactionCallbackException
{
 // reserve a slot for the persistence manager transaction
 TXslot = objectGrid.reserveSlot(TxID.SLOT_NAME);
 // reserve a slot for the persistence manager session
 SessionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
}
```

TxID에는 슬롯이 있습니다. 슬롯은 ArrayList 배열의 항목입니다. 플러그인은 ObjectGrid.reserveSlot 메소드를 호출하고 TxID 오브젝트에서 슬롯을 사용함을 표시하여 ArrayList 배열의 항목을 예약할 수 있습니다. 그 다음 메소드는 응용프로그램에 다음 항목 색인을 리턴합니다. 그러면 응용프로그램이 이 슬롯에 정보를 저장할 수 있습니다. 다음 메소드에서는 이 기술을 보여줍니다.

begin 메소드

오브젝트 그리드는 새 트랜잭션을 시작할 때 이 메소드를 호출합니다. 플러그인은 이 이벤트를 실제 트랜잭션에 맵핑합니다. 그러면 Loader가 commit 메소드를 호출하기 전에 도달하는 get 및 update 메소드 호출에서 이 트랜잭션

을 사용할 수 있습니다. 다음은 오브젝트 그리드 시작을 지속 관리자 트랜잭션 시작에 맵핑하는 begin 메소드 예입니다.

```
/**
* This is called when the grid starts a new transaction. We just create a
* persistence manager transaction and call begin on it. We then store
* the transaction in the TxID slot so we can get it again later
* without needing ThreadLocal etc.
*/
public void begin(TxID id) throws TransactionCallbackException
{
 Session PMsession = getPMcurrentSession();
 Transaction tx = PMsession.beginTransaction();
 id.putSlot(TXslot, tx);
 id.putSlot(SessionSlot, PMsession);
}
```

이 샘플은 initialize 메소드가 TxID 오브젝트에서 두 개의 슬롯을 예약한 사실에 의존합니다. 슬롯 하나는 지속 관리자 세션용이고 다른 하나의 슬롯은 지속 관리자 트랜잭션용입니다. begin 메소드는 지속 관리자를 호출하여 세션을 확보하고 색인화된 SessionSlot 슬롯에서 세션을 저장한 후 세션에서 트랜잭션을 작성하고 색인화된 TXSlot 슬롯을 사용하여 이 트랜잭션에 대한 참조를 저장합니다.

commit 메소드

commit 메소드는 오브젝트 그리드 트랜잭션을 확약할 때 호출됩니다. 모든 Loader는 이미 비워져 있습니다. 플러그인은 플랫폼에 이 확약 이벤트를 전달해야 합니다.

```
/**
* This is called when the grid wants to commit a transaction.
* We just pass it on to persistence manager.
*/
public void commit(TxID id) throws TransactionCallbackException
{
 Transaction tx = (Transaction)id.getSlot(TXslot);
 tx.commit();
}
```

이 메소드는 슬롯에 저장된 지속 관리자 트랜잭션을 찾은 후 commit 메소드를 호출합니다.

rollback 메소드

이 메소드는 오브젝트 그리드 트랜잭션에서 트랜잭션을 롤백할 경우 호출됩니다. 플러그인은 이 트랜잭션을 플랫폼 트랜잭션 관리자에 전달합니다. 다음은 코드 스니펫입니다.

```
/**
* This is called when the grid wants to rollback a transaction.
* We just pass it on to persistence manager.
*/
public void rollback(TxID id) throws TransactionCallbackException
```

```
{
 Transaction tx = (Transaction)id.getSlot(TXslot);
 tx.rollback();
}
```

이 메소드는 commit 메소드와 매우 유사합니다. 이 메소드는 슬롯에서 지속 관리자 트랜잭션에 대한 참조를 확보한 후 rollback 메소드를 호출합니다.

isExternalTransactionActive 메소드

보통 오브젝트 그리드 세션은 자동 확약 모드 또는 트랜잭션 모드에서 작동합니다. 자동 확약 모드는 세션의 ObjectMap 인스턴스에 대한 모든 메소드 호출에서 암시적 트랜잭션을 작성함을 의미합니다. 활성 트랜잭션이 없고 응용프로그램에서 ObjectMap 메소드를 호출한 경우 프레임워크는 TransactionCallback 플러그인에서 이 메소드를 호출하여 적절한 트랜잭션이 활성화되었는지 확인합니다. 이 메소드에서 true를 리턴한 경우 프레임워크는 자동 시작을 수행하고 그렇지 않으면 자동 확약을 수행합니다. 이 메소드를 사용하면 응용프로그램이 오브젝트 그리드 API 대신 플랫폼 API에서 begin, commit 또는 rollback 메소드를 호출하는 환경에서 오브젝트 그리드를 통합할 수 있습니다.

시나리오: 단순 JDBC(Java Database Connectivity) 기반 J2SE(Java 2 Platform, Standard Edition) 환경

이 예에서는 응용프로그램에 JDBC 기반 Loader가 있는 J2SE 환경을 사용합니다. 두 개의 Map이 있습니다. 각 Map에는 데이터베이스의 서로 다른 테이블에서 각각을 지원하는 Loader가 있습니다. TransactionCallback 플러그인은 JDBC 연결을 확보한 후 연결 시 begin, commit 및 rollback 메소드를 호출합니다. 다음은 샘플 TransactionCallback 구현입니다.

```
public class JDBCTCB implements TransactionCallback
{
 DataSource datasource;
 int connectionSlot;
 public JDBCTCB(DataSource ds)
{
  datasource = ds;
}
 public void initialize(ObjectGrid objectGrid)
throws TransactionCallbackException
{
   connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
}
public void begin(TxID id) throws TransactionCallbackException
{
try
{
   Connection conn = datasource.getConnection();
   conn.setAutoCommit(false);
   id.putSlot(connectionSlot, conn);
}
  catch(SQLException e)
{
```

```
throw new TransactionCallbackException("Cannot start transaction", e);
}
}
public void commit(TxID id) throws TransactionCallbackException
{
  Connection conn = null;
try
{
   conn = (Connection)id.getSlot(connectionSlot);
   conn.commit();
    conn.close();
}
  catch(SQLException e)
{
throw new TransactionCallbackException("Cannot commit transaction", e);
}
finally {
   if (conn!=null) {
try {
    conn.close();
}
    catch (SQLException closeE) {
}
}
}
}
public void rollback(TxID id) throws TransactionCallbackException
{
  Connection conn = null;
try
{
   conn = (Connection)id.getSlot(connectionSlot);
   conn.rollback();
    conn.close();
}
  catch(SQLException e)
{
throw new TransactionCallbackException("Cannot rollback transaction", e);
}
finally {
   if (conn!=null) {
try {
    conn.close();
}
    catch (SQLException closeE) {
}
}
}
}
 public boolean isExternalTransactionActive(Session session)
{
                                  return false;
}
 public int getConnectionSlot()
{
  return connectionSlot;
}
}
```

이 예에서는 오브젝트 그리드 트랜잭션 이벤트를 JDBC 연결로 변환하는 TransactionCallback 플러그인을 표시합니다. 플러그인이 초기화되면 JDBC 연결 참조를 보관하도록 단일 슬롯을 예약합니다. 그 다음 begin 메소드로 새 트랜잭션에서 사용할 JDBC 연결을 얻고 자동 확약을 끈 후 TxID 슬롯에 연결에 대한 참조를 저장합니다. commit 및 rollback 메소드는 TxID 슬롯에서 연결을 검색하고 연결 시 적절한 메소드를 호출합니다. isExternalTransaction 메소드는 응용프로그램이 트랜잭션을 명시적으로 제어하도록 오브젝트 그리드 트랜잭션 API를 사용해야 함을 표시하는 false를 항상 리턴합니다. 이 플러그인과 쌍으로 제공되는 Loader는 TxID에서 JDBC 연결을 얻습니다. Loader는 다음 예와 유사합니다.

```
public class JDBCLoader implements Loader
{
 JDBCTCB tcb;
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
 tcb = (JDBCTCB)session.getObjectGrid().getTransactionCallback();
}
 public List get(TxID txid, List keyList, boolean forUpdate)
throws LoaderException
{
  Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
  // implement get here
return null;
}
 public void batchUpdate(TxID txid, LogSequence sequence)
 throws LoaderException, OptimisticCollisionException
{
  Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
  // TODO implement batch update here
}
}
```

Loader는 initialize 메소드를 호출할 때 JDBCTCB 인스턴스에 대한 참조를 얻습니다. 그 다음 get 및 batchUpdate 메소드에 연결이 필요한 경우 JDBCTCB에서 얻은 연결을 얻습니다. 일반적으로 TransactionCallback 구현 및 Loader는 서로 협력하는 쌍으로 작성됩니다. TransactionCallback 구현은 트랜잭션을 처리하고 Loader에 필요한 오브젝트를 TxID 슬롯에 저장합니다. 그러면 로더는 일반적으로 TCB에서 확보한 자원을 사용하여 TransactionCallback이 관리하는 트랜잭션의 컨텍스트에서 get 및 batchUpdate 메소드를 구현합니다.

시나리오: Servlet 엔진 환경

이 시나리오에서 오브젝트 그리드는 관리 Servlet 엔진을 제외하고 JDBC 기반 Loader를 사용합니다. 컨테이너는 사용자가 UserTransaction 메소드를 사용하여 트랜잭션을 시작 및 확약할 것을 예상합니다. TxID 슬롯에 JDBC 연결에 대한 참조를 저장하지 않아도 되므로 J2SE와는 조금 다릅니다. 컨테이너는 JDBC 연결을 관리합니다. 컨테이너 트랜잭션이 활성화되면 데이터 소스를 사용하여 찾은 연결이 항상 동일합니다. 컨테이너가 이 트랜잭션에서 사용하는 연결을 기억하여 DataSource.getConnection

메소드를 호출할 때마다 동일한 연결을 리턴하기 때문입니다. 다음 예에서는 데이터 소스 참조가 Shareable로 구성되었다고 가정합니다.

```
public class ManagedJDBCTCB implements TransactionCallback {
 UserTransaction tx;
 public void initialize(ObjectGrid objectGrid)
throws TransactionCallbackException
{
try
{
   InitialContext ic = new InitialContext();
tx = (UserTransaction)ic.lookup("java:comp/UserTransaction");
}
  catch(NamingException e)
{
throw new TransactionCallbackException("Cannot find UserTransaction", e);
}
}
public void begin(TxID id) throws TransactionCallbackException
{
try
{
   tx.begin();
}
  catch(SystemException e)
{
throw new TransactionCallbackException("Cannot begin tx", e);
}
  catch(NotSupportedException e)
{
throw new TransactionCallbackException("Cannot begin tx", e);
}
}
public void commit(TxID id) throws TransactionCallbackException
{
try
{
 tx.commit();
}
  catch(SystemException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
  catch(HeuristicMixedException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
  catch(RollbackException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
  catch(HeuristicRollbackException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
}
public void rollback(TxID id) throws TransactionCallbackException
{
```

```
try
{
 tx.rollback();
}
  catch(SystemException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
}
 public boolean isExternalTransactionActive(Session session) {
                                  return false;
}
}
```

이 예에서는 initialize 메소드에서 UserTransaction 메소드에 대한 참조를 얻은 후 적절한 UserTransaction 메소드로 begin, commit 및 rollback을 맵핑합니다. 컨테이너가 이 트랜잭션에서 올바른 연결 정보를 검색했는지 확인하므로 슬롯은 필요하지 않습니다. 다음은 이 TransactionCallback 구현과 함께 사용하는 JDBC Loader입니다.

```
public class ManagedJDBCLoader implements Loader
{
 DataSource myDataSource;
 ManagedJDBCLoader(DataSource ds)
{
  myDataSource = ds;
}
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
}
 public List get(TxID txid, List keyList, boolean forUpdate)
throws LoaderException
{
try
{
   Connection conn = myDataSource.getConnection();
   // TODO implement get here with this connection
return null;
}
  catch(SQLException e)
{
throw new LoaderException("Cannot get objects", e);
}
}
 public void batchUpdate(TxID txid, LogSequence sequence)
 throws LoaderException, OptimisticCollisionException
{
try
{
   Connection conn = myDataSource.getConnection();
   // TODO implement update here using this connection
}
  catch(SQLException e)
{
```

```
throw new LoaderException("Cannot update objects", e);
}
}
}
```

이 예는 기본 JDBC 버전보다 더 단순할 수 있습니다. 컨테이너가 연결을 관리하고 매번 동일한 활성 트랜잭션에서 DataSource.getConnection 메소드가 호출될 때 동일한 트랙잭션 내에서 항상 동일한 연결을 리턴하는지 확인하기 때문입니다. 결과적으로 캐시하려고 선택하면 응용프로그램에서 연결을 캐시할 수 있지만, 슬롯의 연결은 캐시하지 마십시오.

OptimisticCallback 인터페이스

com.ibm.websphere.objectgrid.plugins.OptimisticCallback 인터페이스를 구현하는, 변경이 예상되지 않는 플러그 가능 콜백 오브젝트를 제공할 수 있습니다.

목적

OptimisticCallback 인터페이스는 맵 값에 대한 변경이 예상되지 않는 비교 조작을 제공하는 데 사용됩니다. 147 페이지의 『잠금 중 변경이 예상되지 않음(Optimistic)』에 설명된 대로 잠금 중 변경이 예상되지 않는 계획을 사용 중일 때 OptimisticCallback이 필요합니다. 오브젝트 그리드는 기본 OptimisticCallback 구현을 제공합니다. 그러나 대개 응용프로그램은 OptimisticCallback 인터페이스의 자체 구현을 플러그인해야 합니다.

응용프로그램 제공 OptimisticCallback 오브젝트 플러그인

다음 예는 응용프로그램이 grid1 오브젝트 그리드 인스턴스의 employee BackingMap에 대한 OptimisticCallback 오브젝트를 플러그인할 수 있는 방법을 설명합니다.

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

위 예의 EmployeeOptimisticCallbackImpl 오브젝트는 OptimisticCallback 인터페이스를 구현해야 합니다. 응용프로그램은 다음 예에 표시된 대로 XML 파일을 사용하여 OptimisticCallback 오브젝트를 플러그인할 수도 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="grid1">
```

```
<backingMap name="employees" pluginCollectionRef="employees"
lockStrategy="OPTIMISTIC" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="employees">
<bean id="OptimisticCallback"
className="com.xyz.EmployeeOptimisticCallbackImpl" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

기본 구현

오브젝트 그리드 프레임워크는 이전 섹션에 설명된 대로 응용프로그램이 응용프로그램 제공 OptimisticCallback 오브젝트를 플러그인하지 않는 경우 사용되는 OptimisticCallback 인터페이스의 기본 구현을 제공합니다. 기본 구현은 항상 특수 값 NULL_OPTIMISTIC_VERSION을 값에 대한 버전 오브젝트로 리턴하며 버전 오브젝트를 갱신하지 않습니다. 이 조치로 인해 변경이 예상되지 않는 비교는 ″조작 없음″ 기능이 됩니다. 대부분의 경우, 잠금 중 변경이 예상되지 않는 계획을 사용 중일 때 ″조작 없음″ 기능이 발생하기를 원치 않습니다. 응용프로그램은 OptimisticCallback 인터페이스를 구현하고 자체 OptimisticCallback 구현을 플러그인하여 기본 구현이 사용되지 않도록 해야 합니다. 그러나 기본 제공된 OptimisticCallback 구현이 유용한 시나리오가 존재합니다. 다음 상황을 고려하십시오.

- 로더가 BackingMap에 대해 플러그인되어 있습니다.
- 로더가 OptimisticCallback 플러그인의 지원 없이 변경이 예상되지 않는 비교를 수행하는 방법을 알고 있습니다.

Loader는 OptimisticCallback 오브젝트의 지원 없이 변경이 예상되지 않는 버전화를 처리하는 방법을 어떻게 알 수 있습니까? Loader는 value 클래스 오브젝트에 대해 알고 있으며, 변경이 예상되지 않는 버전화 값으로 사용되는 값 오브젝트의 필드를 알고 있습니다. 예를 들어, 다음 인터페이스가 employees 맵의 값 오브젝트에 사용된다고 가정하십시오.

```
public interface Employee
{
// Sequential sequence number used for optimistic versioning.
public long getSequenceNumber();
public void setSequenceNumber(long newSequenceNumber);
// Other get/set methods for other fields of Employee object.
}
```

이 경우, Loader는 getSequenceNumber 메소드를 사용하여 Employee 값 오브젝트의 현재 버전 정보를 가져올 수 있음을 알고 있습니다. 새 Employee 값으로 지속적 기억장치를 갱신하기 전에 리턴된 값을 증분하여 새 버전 번호를 생성합니다. JDBC(Java Database Connectivity) Loader의 경우, 과규정화 SQL update 문의 where 절에 있는 현재 순서 번호가 사용되며 새로 생성된 순서 번호를 사용하여 순서 번호 열을 새

순서 번호 값으로 설정합니다. 또다른 가능성은 Loader가 변경이 예상되지 않는 버전화에 사용할 수 있는 숨겨진 열을 자동으로 갱신하는 일부 백엔드 제공 기능을 사용하는 것입니다. 일부 경우, 스토어드 프로시저 또는 트리거를 사용하여 버전화 정보를 보유하는 열을 유지보수할 수 있습니다. Loader가 이런 기술 중 하나를 사용하여 변경이 예상되지 않는 버전화 정보를 유지보수하고 있는 경우, 응용프로그램은 OptimisticCallback 구현을 제공하지 않아도 됩니다. Loader가 OptimisticCallbac 오브젝트의 지원 없이 변경이 예상되지 않는 버전화를 처리할 수 있기 때문에 이 경우에는 기본 OptimisticCallback을 사용할 수 있습니다.

OptimisticCallback 인터페이스 구현

OptimisticCallback 인터페이스에는 다음 메소드와 특수 값이 포함되어 있습니다.

```
public interface OptimisticCallback
{
 final static Byte NULL_OPTIMISTIC_VERSION;
 Object getVersionedObjectForValue(Object value);
 void updateVersionedObjectForValue(Object value);
 void serializeVersionedValue(Object versionedValue,
 ObjectOutputStream stream) throws IOException;
 Object inflateVersionedValue(ObjectInputStream stream) throws
 IOException, ClassNotFoundException;
}
```

다음 목록은 OptimisticCallback 인터페이스의 각 메소드에 대한 설명 또는 고려사항을 제공합니다.

NULL_OPTIMISTIC_VERSION

응용프로그램 제공 OptimisticCallback 구현 대신 기본 OptimisticCallback 구현을 사용하는 경우 getVersionedObjectForValue 메소드가 이 특수 값을 리턴합니다.

getVersionedObjectForValue 메소드

이 메소드는 값 사본을 리턴하거나 버전화 목적에 사용할 수 있는 값 속성을 리턴할 수 있습니다. 이 메소드는 오브젝트가 트랜잭션과 연관될 때마다 호출됩니다. Loader가 BackingMap에 플러그인되지 않은 경우, BackingMap은 확약 시 이 값을 사용하여 변경이 예상되지 않는 버전 비교를 수행합니다. BackingMap은 변경이 예상되지 않는 버전 비교를 사용하여 이 트랜잭션이 수정한 맵 항목에 처음 액세스한 이후로 버전이 변경되지 않았는지 확인합니다. 또다른 트랜잭션이 이 맵 항목에 대한 버전을 이미 수정한 경우, 버전 비교에 실패하고 BackingMap은 트랜잭션 롤백을 강제 실행하기 위해 OptimisticCollisionException 예외를 표시합니다. Loader가 플러그인된 경우, BackingMap은 변경이 예상되지 않는 버전화 정보를 사용하지 않습니다. 대신 Loader는 변경이 예상되지 않는 버전화 비교를 수행하고 필요하면 버전화 정보를 갱신하는 책임을 지고 있습니다. Loader는 일반적으로 비우기 조작이 발

생하거나 트랜잭션이 확약될 때 호출되는 Loader의 batchUpdate 메소드로 전달된 LogElement에서 초기 버전화 오브젝트를 가져옵니다.

다음 코드는 EmployeeOptimisticCallbackImpl 오브젝트가 사용하는 구현을 표시합니다.

```
public Object getVersionedObjectForValue(Object value)
{
 if (value == null)
{
  return null;
}
else
{
  Employee emp = (Employee) value;
  return new Long( emp.getSequenceNumber() );
}
}
```

위 예에 설명된 것처럼 Loader가 예상한 대로 sequenceNumber 속성은 java.lang.Long 오브젝트에 리턴됩니다. 이는 Loader를 작성한 동일한 사용자가 EmployeeOptimisticCallbackImpl 구현을 작성했거나 EmployeeOptimisticCallbackImpl을 구현한 사용자와 긴밀히 협력했음을 의미합니다(예를 들어, getVersionedObjectForValue 메소드가 리턴한 값에 동의).

앞서 설명한 대로 기본 OptimisticCallback은 특수 값 NULL_OPTIMISTIC_VERSION을 버전 오브젝트로 리턴합니다.

updateVersionedObjectForValue 메소드

이 메소드는 트랜잭션이 값을 갱신했거나 버전화된 새 오브젝트가 필요할 때마다 호출됩니다. getVersionedObjectForValue가 값 속성을 리턴하는 경우, 이 메소드는 일반적으로 속성 값을 새 버전 오브젝트로 갱신합니다. getVersionedObjectForValue가 값 사본을 리턴하는 경우, 이 메소드는 일반적으로 아무것도 수행하지 않습니다. getVersionedObjectForValue 기본 구현이 항상 특수 값 NULL_OPTIMISTIC_VERSION을 값의 버전 오브젝트로 리턴하기 때문에 기본 OptimisticCallback은 아무것도 수행하지 않습니다.

다음은 OptimisticCallback 섹션에서 사용되는 EmployeeOptimisticCallbackImpl 오브젝트가 사용하는 구현을 표시합니다.

```
public void updateVersionedObjectForValue(Object value)
{
 if ( value != null )
{
  Employee emp = (Employee) value;
  long next = emp.getSequenceNumber() + 1;
  emp.updateSequenceNumber( next );
}
}
```

이전 예에 설명된 대로 sequenceNumber 속성은 하나씩 증분되어 다음 번에 getVersionedObjectForValue 메소드가 호출될 때 리턴되는 java.lang.Long 값이 원래 순서 번호 값에 1을 더한 긴 값을 갖게 되어 이 employee 인스턴스의 다음 버전 값이 되도록 합니다. 다시, 이 예는 Loader를 작성한 동일한 사용자가 EmployeeOptimisticCallbackImpl을 작성했거나 EmployeeOptimisticCallbackImpl을 구현한 사용자와 긴밀히 협력했음을 의미합니다.

serializeVersionedValue 메소드

이 메소드는 버전화된 값을 지정된 스트림에 기록합니다. 구현에 따라 버전화된 값을 사용하여 갱신 중 변경이 예상되지 않음 충돌을 식별할 수 있습니다. 일부 구현에서는 버전화된 값이 원래 값의 사본입니다. 기타 구현에는 값의 버전을 표시하기 위한 순서 번호 또는 일부 기타 오브젝트가 있을 수 있습니다. 실제 구현을 알 수 없기 때문에 이 메소드는 적절한 직렬화를 수행하기 위해 제공됩니다. 기본 구현은 writeObject 호출을 수행합니다.

inflateVersionedValue 메소드

이 메소드는 버전화된 값의 직렬화된 버전을 취하고 실제 버전화된 값 오브젝트를 리턴합니다. 구현에 따라 버전화된 값을 사용하여 갱신 중 변경이 예상되지 않음 충돌을 식별할 수 있습니다. 일부 구현에서는 버전화된 값이 원래 값의 사본입니다. 기타 구현에는 값의 버전을 표시하기 위한 순서 번호 또는 일부 기타 오브젝트가 있을 수 있습니다. 실제 구현을 알 수 없기 때문에 이 메소드는 적절한 직렬화 해제를 수행하기 위해 제공됩니다. 기본 구현은 readObject를 수행합니다.

복제 프로그래밍

MapSet를 ReplicationGroup 및 복제 정책 속성과 연관시켜 복제를 구성합니다. ReplicationGroup은 1차 및 연관된 복제본과 대기에 사용되는 서버 구성원을 정의합니다. 또한 이 구성에 필요한 최소 및 최대 복제본 수를 정의합니다. 복제 정책 속성은 동기 또는 비동기 복제가 필요한지 여부, 복제본에 대한 읽기 액세스의 허용 여부 및 복제본으로 복제 데이터 전송 시 압축 사용 여부를 표시합니다. 복제는 프로그래밍 모델에 최소한의 영향을 주고 맵에 데이터를 사전 로드하는 응용프로그램에 주로 영향을 줍니다.

맵 사전 로드

Loader를 각 Map과 연관시킬 수 있습니다. 맵에서 오브젝트를 찾을 수 없을 때 로더를 사용하여 오브젝트를 페치하고 트랜잭션이 확약할 때 변경사항을 벡엔드에 작성합니다. 또한 데이터를 맵에 사전 로드하는 데 로더를 사용할 수 있습니다. JVM(Java Virtual Machine)이 복제 그룹의 1차가 될 때 Loader 인터페이스의 preload 메소드가 호출됩니다. 복제본 또는 대기에는 preload 메소드가 호출되지 않습니다. preload 메소

드는 제공된 세션을 사용하여 백엔드에서 맵으로 의도된 모든 참조 데이터를 로드합니다. 사용할 맵은 preload 메소드로 전달된 BackingMap 인수로 식별됩니다.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

파티션된 MapSet에서 사전 로드

맵을 *N*개 파티션으로 파티션할 수 있습니다. 맵은 다중 서버 전반에 걸쳐 저장될 수 있으며, 각 항목은 해당 서버 중 하나에만 저장된 키로 식별됩니다. 응용프로그램은 더 이상 모든 맵 항목을 보유하는 데 단일 JVM의 힙 크기로 제한되지 않으므로 대용량 맵을 오브젝트 그리드에 보유할 수 있습니다. Loader 인터페이스의 preload 메소드로 사전 로드할 응용프로그램은 사전 로드해야 하는 데이터의 서브세트를 식별해야 합니다. 항상 고정된 수의 파티션이 존재합니다. 이 수는 다음 코드 스니펫을 사용하여 판별할 수 있습니다.

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

이 코드 스니펫은 응용프로그램이 데이터베이스에서 사전 로드할 데이터의 서브세트를 식별할 수 있는 방법을 표시합니다. 맵이 초기에 파티션되지 않더라도 응용프로그램은 항상 이 메소드를 사용해야 합니다. 이 메소드는 유연합니다. 관리자가 나중에 맵을 파티션하면 로더는 계속해서 올바르게 작동합니다.

응용프로그램은 백엔드에서 myPartition 서브세트를 검색하는 조회를 실행해야 합니다. 데이터베이스가 사용 중이면 테이블의 데이터를 쉽게 파티션할 수 있는 일부 자연어 조회가 없을 경우 지정된 레코드의 파티션 ID가 있는 열을 포함하는 것이 더 용이할 수 있습니다.

성능

사전 로드 구현은 다중 오브젝트를 단일 트랜잭션의 맵에 저장하여 백엔드에서 맵에 데이터를 복사해야 합니다. 다음 질문은 "트랜잭션당 저장할 레코드 수는 얼마입니까?"입니다. 안타깝게도 답변은 "이 수는 상황에 따라 다릅니다."입니다. 트랜잭션이 100개 항목의 블록보다 더 많이 포함하면 성능이 저하됩니다. 최적의 수는 오브젝트 복잡도와 크기를 포함하여 여러 요소에 따라 달라집니다. 100개 항목으로 시작한 다음 더 이상 성능이 저하되지 않을 때까지 수를 늘리십시오. 트랜잭션이 많을수록 복제 성능은 향상됩니다. 1차만 사전 로드 코드를 실행합니다. 이점을 기억하십시오. 사전 로드된 데이터는 1차에서 온라인 상태인 복제본으로 복제됩니다.

MapSets 사전 로드

응용프로그램이 다중 맵을 보유하는 MapSet를 사용하면 각 맵은 고유 로더를 보유하게 됩니다. 각 로더는 preload 메소드를 보유합니다. 각 맵은 오브젝트 그리드에서 직렬로 로드합니다. 단일 맵을 사전 로드 맵으로 지정하여 모든 맵을 사전 로드하는 것이 더욱 효과적일 수 있습니다. 이것은 응용프로그램 규칙일 뿐입니다. 예를 들어, 두 개의

맵(부서와 직원)이 부서 로더를 사용하여 부서 및 직원 맵을 모두 사전 로드할 수 있습니다. 이것은 트랜잭션으로 볼 때, 응용프로그램이 부서를 원하면 해당 부서의 직원이 캐시되도록 합니다. 물론, 부서 로더가 백엔드에서 부서를 사전 로드할 때 해당 부서의 직원도 페치된다는 의미입니다. 이것이 true가 되려면 부서 오브젝트와 연관된 직원 오브젝트가 단일 트랜잭션을 사용하여 맵에 추가되어야 합니다.

복구 가능한 사전 로드

일부 고객은 매우 큰 데이터 세트를 캐시해야 합니다. 이 데이터를 사전 로드하는 데에는 많은 시간이 소요될 수 있습니다. 때때로, 응용프로그램이 온라인 상태로 되려면 사전 로드가 먼저 완료되어야 합니다. 사전 로드를 복구 가능하게 한다는 의미이기도 합니다. 사전 로드할 백만 개의 레코드가 있다고 가정하십시오. 1차가 이 레코드를 사전 로드하다가 8십만 번째 레코드에서 실패합니다. 일반적으로 새 1차로 선택된 복제본은 복제된 상태를 지우고 처음부터 시작합니다. 오브젝트 그리드는 ReplicaPreloadController를 사용하여 작업을 해당 상황보다 더욱 효율적으로 수행할 수 있습니다. 또한 응용프로그램의 로더가 ReplicaPreloadController 인터페이스를 구현해야 합니다. 이것은 로더에 단일 메소드를 추가합니다.

```
Status checkPreloadStatus(Session session, BackingMap bmap);
```

일반적으로 Loader 인터페이스의 preload 메소드가 호출되기 전에 오브젝트 그리드 런타임에서 이 메소드를 호출합니다. 오브젝트 그리드는 이 메소드의 결과(상태)를 테스트하여 복제본이 1차로 승격될 때마다 오브젝트 그리드의 동작을 판별합니다.

리턴된 상태 값	반응 시 오브젝트 그리드 동작
Status.PRELOADED_ALREADY	이 상태 값은 맵이 완전히 사전로드되었음을 나타내므로 오브젝트 그리드는 preload 메소드를 전혀 호출하지 않습니다.
Status.FULL_PRELOAD_NEEDED	일반적으로 오브젝트 그리드는 맵을 지우고 preload 메소드를 호출합니다.
Status.PARTIAL_PRELOAD_NEEDED	오브젝트 그리드는 맵을 현재 상태로 두고 사전 로드를 호출합니다. 이 계획을 사용하여 응용프로그램의 로더는 해당 지점에서 사전 로드를 계속 수행할 수 있습니다.

분명히 1차는 맵을 사전 로드하는 동안 복제본이 리턴할 상태를 파악할 수 있도록 복제 중인 MapSet의 맵에 있는 일부 상태를 그대로 두어야 합니다. 호출된 추가 맵(예: RecoveryMap)을 사용할 수 있습니다. 이 RecoveryMap은 사전 로드 중인 동일한 MapSet의 파트여야 합니다. 따라서 사전 로드 중인 데이터와 일관되게 복제되어야 합니다.

제안된 구현은 다음과 같습니다. 사전 로드는 레코드의 각 블록을 확약할 때 RecoveryMap의 카운터/값도 해당 트랜잭션의 파트로 갱신해야 합니다. 사전 로드된 데이터 및 RecoveryMap 데이터가 상세하게 복제본으로 복제된다는 의미입니다. 복제본

이 1차로 승격되면 이제 RecoveryMap을 확인하여 발생한 상황을 확인할 수 있습니다. RecoveryMap은 단순히 'state' 키를 가지는 단일 항목을 보유할 수 있습니다. 이 키의 오브젝트가 없으면 전체 사전 로드가 필요합니다(checkPreloadStatus는 FULL_PRELOAD_NEEDED를 리턴함). 이 'state' 키의 오브젝트가 있으면 값이 'COMPLETE'일 경우 사전 로드가 완료되고 checkPreloadStatus는 PRELOADED_ALREADY를 리턴합니다. 그렇지 않으면, 값 오브젝트는 사전 로드가 다시 시작되어야 하는 위치를 표시하고 checkPreloadStatus 메소드는 PARTIAL_PRELOAD_NEEDED를 리턴해야 합니다. 로더는 사전 로드가 호출될 때 시작점을 알 수 있도록 인스턴스 변수에 복구 지점을 저장할 수 있습니다. 또한 각 맵이 개별적으로 사전 로드된 경우 RecoveryMap은 맵마다 항목을 보유할 수 있습니다.

로더에서 동기 복제 모드로 복구 처리

오브젝트 그리드 런타임은 1차가 실패할 때 확약된 데이터가 유실되지 않도록 설계되었습니다. 다음 섹션은 이 작업을 수행하는 데 사용된 알고리즘을 표시합니다. 이 알고리즘은 복제 그룹이 동기 복제를 사용할 때에만 적용됩니다. Loader는 선택적입니다.

1차의 모든 변경사항을 복제본으로 동기식으로 복제하도록 오브젝트 그리드 런타임을 구성할 수 있습니다. JVM이 복제본으로 승격될 때 먼저 1차는 맵의 스냅샷을 복제본으로 전송합니다. 복제본이 이 스냅샷을 처리했으면 1차는 스냅샷 생성 이후 모든 변경사항(완료된 트랜잭션) 전송을 시작합니다. 결국 복제본은 1차와 함께 캐치됩니다. 이 초기 복제 처리는 비동기식입니다. 복제본이 1차와 함께 캐치되면 해당 쌍은 피어 모드로 들어가고 마지막으로 동기 복제가 시작됩니다. 이 지점에서 1차에 확약된 각 트랜잭션이 피어 모드에서 모든 복제본으로 전송되고 1차는 수신확인 메시지를 대기합니다. 따라서 수신확인 메시지 수신과 관련한 대기시간 때문에 비동기 복제 시나리오와 비교할 때 1차 속도가 낮아집니다. 1차의 동기 확약 순서는 다음과 같습니다.

로더가 있는 단계	로더가 없는 단계
항목의 잠금 가져오기	동일
로더로 변경사항 비우기	NOOP
캐시에 변경사항 저장	동일
복제본으로 변경사항 전송 및 수신확인 대기	동일
TransactionCallback 플러그인을 통해 로더에 확약	TransactionCallBack 플러그인 확약은 여전히 호출되지만 일반적으로 수행되는 작업은 없습니다.
항목의 잠금 해제	동일

변경사항은 로더로 확약되기 전에 복제본으로 전송됩니다. 언제 변경사항을 복제본에 확약합니까? 이 순서를 검토하십시오.

초기화 시 1차의 tx 목록을 초기화하십시오.

- Set CommitedTx = {}, RolledBackTx = {}

동기 확약 처리 중:

로더가 있는 단계	로더가 없는 단계
항목의 잠금 가져오기	동일
로더로 변경사항 비우기	NOOP
캐시에 변경사항 저장	동일
확약된 트랜잭션 및 롤백된 트랜잭션과 함께 변경사항을 복제본으로 전송하고 수신확인 대기	동일
확약된 트랜잭션 및 롤백된 트랜잭션 목록 지우기	동일
TransactionCallBack 플러그인을 통해 로더 확약	TransactionCallBack 플러그인 확약은 여전히 호출되지만 일반적으로 수행되는 작업은 없습니다.
확약이 성공하면 트랜잭션을 확약된 트랜잭션에 추가하고, 그렇지 않으면 롤백된 트랜잭션에 추가하십시오.	NOOP
항목의 잠금 해제	동일

복제본 처리

- 변경사항 수신
- 수신된 모든 트랜잭션을 확약된 트랜잭션 목록에 확약
- 수신된 모든 트랜잭션을 롤백된 트랜잭션 목록에 롤백
- 트랜잭션 또는 세션 시작
- 트랜잭션 또는 세션에 변경사항 적용
- 보류 목록에 트랜잭션 또는 세션 저장
- 응답 다시 전송

복제본 모드에 있는 동안 복제본에는 로더 상호 작용이 없습니다. 1차는 모든 변경사항을 로더를 통해 푸시해야 합니다. 복제본이 삭제됩니다.

이 알고리즘의 부작용은 복제본은 항상 트랜잭션을 가지지만 다음 1차 트랜잭션이 해당 트랜잭션의 확약 상태를 전송할 때까지 확약되지 않는다는 것입니다. 전송되고 나면 복제본에 확약되거나 롤백됩니다. 그러나, 그때까지 트랜잭션은 확약되지 않습니다. 잠시(몇 초) 후 트랜잭션 결과를 전송하는 타이머를 1차에 추가할 수 있습니다. 이것은 무효를 해당 시간 창으로 제한하지만 완전히 제거하지는 않습니다. 이 무효는 복제본 읽기 모드를 사용할 때에만 문제점이 됩니다. 그렇지 않으면, 표시되지 않고 응용프로그램에 영향을 주지 않습니다.

1차가 실패할 때 1차에 확약/롤백되었지만 이 결과와 함께 복제본에 대한 메시지가 작성되지 않은 소수의 트랜잭션이 존재할 수 있습니다. 복제본이 새 1차로 승격될 때 첫 번째 조치 중 하나는 이 조건을 처리하는 것입니다. 새 1차 맵 세트에 대해 각 보류 트랜잭션이 다시 처리됩니다. 로더가 있으면 각 트랜잭션이 로더에 제공됩니다. 이 트랜잭션은 엄격한 선입선출(FIFO) 순서의 적용을 받습니다. 트랜잭션이 실패하면 이 순서는 무시됩니다. 보류 중인 세 개의 트랜잭션인 A, B 및 C가 있으면 A가 확약되고

B가 롤백되며 C 또한 확약될 수 있습니다. 트랜잭션은 다른 트랜잭션에 영향을 주지 않습니다. 이 트랜잭션은 개별적인 것으로 가정합니다.

'실패복구' 모드와 '표준' 모드에서 비교할 때 로더는 약간 다른 로직을 사용하기를 원할 수 있습니다. 로더는 ReplicaPreloadController 인터페이스를 구현하여 실패복구 모드에 있는 시기를 쉽게 알 수 있습니다. checkPreloadStatus 메소드는 실패복구가 완료될 때에만 호출됩니다. 따라서, checkPreloadStatus 전에 Loader 인터페이스의 apply 메소드가 호출되면 이것은 복구 트랜잭션입니다. checkPreloadStatus 메소드가 호출되면 실패복구가 완료됩니다.

복제를 사용하는 Stateful 싱글톤

WebSphere Extended Deployment는 파티션 지정 기능과 함께 첫 번째 릴리스의 싱글톤 지원을 추가했습니다. 이 지원을 통해 응용프로그램은 클러스터에 싱글톤을 작성할 수 있었습니다. 오브젝트 그리드 런타임에서는 복제된 MapSets를 통해 유사한 기능을 사용할 수 있습니다. 오브젝트 그리드 싱글톤 패턴에는 많은 장점이 있지만 몇 가지 단점도 있습니다. 파티션 기능은 싱글톤/파티션이 로컬로 활성화될 때 응용프로그램에 이벤트를 제공하고 파티션 기능의 partitionLoad 메소드를 사용하여 이벤트를 전달합니다. 복제된 MapSet에도 1차인 싱글톤이 있습니다. 로더의 ReplicaPreloadController#checkPreloadStatus 메소드로 싱글톤이 1차가 될 때 응용프로그램에 통지됩니다. 이것은 파티션 지정 기능과 유사한 방법으로 사용될 수 있지만 WebSphere Application Server의 다양한 버전 또는 경쟁 제품의 Application Server에서 이식 가능하다는 장점이 있습니다.

파티션 기능에는 비활성화 이벤트가 있지만 오브젝트 그리드 런타임은 이 기능을 제공하지 않습니다. 일반적으로 오브젝트 그리드의 1차는 실패할 때까지 실행됩니다. 이 1차를 이동할 수는 없습니다. 이것이 오브젝트 그리드 상에서 파티션 지정 기능의 장점입니다. 다음은 기능 테이블입니다.

표 16.

기능	파티션 지정 기능	오브젝트 그리드 싱글톤
싱글톤 시작 이벤트	예	예
싱글톤 중지 이벤트	예	아니오
싱글톤 상태 복제	아니오	예
복제의 가변 서비스 품질(QoS)	아니오	예
유연한 싱글톤 배치	예	아니오
런타임 시 싱글톤 이동 가능	예	아니오
싱글톤으로 작업 IIOP 라우팅	예	아니오
J2EE(Java 2 Platform, Enterprise Environment) 서버 필요	예	아니오
WebSphere Extended Deployment의 전체 버전 필요	예	아니오
EJB(Enterprise JavaBeans) 필요	예	아니오

표 16. (계속)

기능	파티션 지정 기능	오브젝트 그리드 싱글톤
응용프로그램을 다른 Application Server로 이식 가능	아니오	예

싱글톤 상태

파티션 기능이 상태 관리에 내장 지원을 제공하지 않았습니다. 싱글톤이 상태를 필요로 하는 경우 응용프로그램이 고유의 장치로 제공되었습니다. 일반적으로 상태가 데이터베이스에 푸시되었다는 의미입니다. 파티션이 실패했으면 파티션을 호스트 및 복구하도록 선출된 서버가 해당 데이터베이스에서 이 상태를 검색해야 했습니다. 대신 응용프로그램이 오브젝트 그리드를 사용하면 싱글톤은 이 싱글톤을 관리하는 ReplicaPreloadController와 연관된 맵의 상태를 유지할 수 있습니다. 1차 또는 싱글톤이 실패하면 새 1차로 선출된 복제본이 복제 때문에 상태를 로컬로 이미 보유하게 됩니다. 응용프로그램에서 데이터 유실을 승인할 수 없는 경우 동기 복제를 사용하십시오.

유연한 싱글톤 배치

파티션 지정 기능은 고가용성 관리자 정책 메커니즘을 사용하여 호스트될 파티션을 판별하고 런타임 시 이 정책을 효과적으로 변경할 수 있습니다. 오브젝트 그리드 복제 그룹 정책은 고가용성 관리자를 포함한 정책만큼 유연하지 않고 모든 서버를 다시 시작하지 않으면 변경될 수 없습니다. 오브젝트 그리드를 사용할 경우 런타임 시 싱글톤을 이동시킬 수 없습니다.

가변 QoS 복제

파티션 기능은 상태 관리를 제공하지 않습니다. 오브젝트 그리드는 다양한 복제 접근 방식을 제공합니다.

- 복제 없음
- 비동기 복제
- 동기 복제

상태에 사용 중인 맵과 연관된 MapSet의 복제 정책으로 해당 정책을 결정합니다. 동기 복제는 데이터 손실은 없지만 속도가 느립니다. 비동기 복제는 빠르지만 1차가 실패할 경우 1차에 확약된 하나 이상의 트랜잭션이 유실될 수 있습니다.

복제본 상에서 로드 밸런스

특별히 구성되지 않았다면 오브젝트 그리드는 모든 읽기 및 쓰기 요청을 지정된 복제 그룹의 1차 서버로 전송합니다. 1차만으로 클라이언트의 모든 요청을 서비스해야 한다

는 의미입니다. 읽기 요청을 1차의 복제본으로 전송하기를 원할 수 있습니다. 이렇게 하면 읽기 요청의 로드를 여러 JVM에서 공유할 수 있지만 복제본으로 읽기 요청을 전송할 때 일관성이 유지되지 않습니다.

일반적으로 이것은 클라이언트가 항상 변경하는 데이터를 캐시할 때 또는 클라이언트가 잠금 중 변경이 예상됨을 사용할 때에만 사용됩니다.

데이터가 계속해서 변경되면 클라이언트 주변 캐시와 1차에 무효화되어가는 과정에서 클라이언트의 상대적으로 높은 가져오기 요청율이 결과로 나타나야 합니다. 마찬가지로 잠금 중 변경이 예상됨 모드에서는 로컬 캐시가 없어 모든 요청이 1차로 전송됩니다.

데이터가 상대적으로 정적이거나 변경이 예상됨 모드가 사용되지 않으면 웜 캐시를 포함한 클라이언트에서 가져오기 요청 빈도가 높지 않으므로 복제본 읽기는 성능에 큰 영향을 주지 않습니다.

그러나, 클라이언트가 처음 시작할 때 클라이언트의 주변 캐시는 비어 있고 빈 해당 캐시에 대한 캐시 요청은 1차로 전달됩니다. 클라이언트 캐시는 시간이 경과됨에 따라 데이터를 가져오므로, 이 요청 로드는 제거됩니다. 다수의 클라이언트가 있고 이 클라이언트 대부분이 동시에 시작되면 이 로드는 상당히 클 수 있으므로 복제본 읽기를 선택하는 것이 성능에 적합할 수 있습니다.

복제본 및 비동기 복제에서 읽기

복제 그룹의 데이터가 자주 변경되지 않으면 일반적으로 이것은 적절한 절충입니다. 따라서 클라이언트에서 가져오기 요청을 온라인 상태인 복제본의 데이터로 지정할 수 있습니다. 가져오기 요청을 복사본이 없는 복제본으로 전송할 수 있고 이 시점에서 키/값을 복제본에 복제하지 않을 수 있습니다. 데이터가 복제본에 없으면 가져오기 요청은 1차로 재지정됩니다.

데이터가 변경되면 복제본에서 가져오기가 무효 데이터를 리턴할 수 있습니다. 응용프로그램에서는 이 무효 데이터를 허용할 수도 있고 허용하지 않을 수도 있습니다. 허용되지 않으면 복제본에서 읽기를 사용 가능하게 할 수 없습니다.

동기 복제 모드의 복제본에서 읽기

동기 복제는 복제본을 1차와 완전히 동일하게 보관하려고 시도합니다. 1차가 실패하면 1차에 확약된 모든 데이터가 장애 발생 시 피어 모드에 있었던 모든 복제본에서 사용가능하게 됩니다. 장애가 발생할 때 복제본에서 읽기를 허용하면 사용된 알고리즘에서 일부 부작용이 나타납니다.

1차가 트랜잭션을 확약하려고 할 때 변경사항의 복사본은 복제본으로 전송되고 복제본은 다음 두 가지 경우에 이 트랜잭션을 확약합니다.

- 1차가 실패할 경우

- 1차의 다음 트랜잭션이 전송되는 경우

1차가 실패할 때 복제본의 모든 보류 트랜잭션은 확약됩니다.

보류 트랜잭션은 후속 트랜잭션이 1차에서 확약될 때에만 확약됩니다. 1차는 확약의 결과를 이 복제 메시지에 피기백(piggy back) 방식으로 전송합니다. 복제본이 이 메시지 중 하나를 수신할 때 복제본은 해당 메시지에 지정된 결과를 보유한 보류 트랜잭션을 확약하거나 롤백합니다.

보류 트랜잭션은 확약될 때에만 복제본에 읽을 수 있게 표시됩니다. 명백하게 1차가 로드되고 일반적인 수정 작업이 있었다면 이러한 보류 트랜잭션은 매우 신속하게 확약됩니다. 1차의 수정 로드가 낮으면 다음 1차 수정이 이루어질 때까지 보류 트랜잭션이 확약되지 않는 기간이 있습니다.

명백하게 수정을 수행하는 1차의 복제본은 일반적으로 보기의 읽기 지점부터 1차 뒤에 최소한 하나의 트랜잭션이 있습니다. 데이터 유실이 없고 이 트랜잭션은 실제로 복제본에 있으며 이 보류 트랜잭션의 결과가 1차에서 전송될 때까지 확약되지 않습니다. 이 확약은 다음 읽기 및 쓰기 트랜잭션이 실행될 때 발생합니다.

요약

복제본에서 읽기가 가능하면 응용프로그램은 무효 데이터를 리턴하는 일부 가져오기를 허용하도록 준비해야 합니다. 이러한 문제는 동기 복제가 사용 중이든 비동기 복제가 사용 중이든 상관 없이 해당됩니다.

파티션

MapSet의 오브젝트가 단일 JVM(Java Virtual Machine)에서 사용 가능한 것보다 더 많은 메모리를 요구하거나 JVM이 갱신을 위해 필요한 처리량을 제공할 수 없는 경우 파티션 지정을 사용합니다.

항목 보유 위치

해싱 알고리즘은 각 항목을 보유하는 서버를 판별합니다. 관리자는 사용할 파티션 수를 지정하여 PartitionSet 정의에 사용합니다. 이 구성은 JVM이 시작되고 나면 변경될 수 없습니다. 단순 해시 값은 항목의 키에서 확보하고 이 값 모듈로(%) 파티션 수의 결과는 해당 항목을 ″소유하는″ 서버를 나타냅니다.

일반적으로 키 오브젝트의 Java hashCode 메소드가 사용됩니다. hashCode 구현을 대체하여 이 값을 대체하십시오.

때때로, 응용프로그램은 표준 해싱의 값을 수정하기를 원하지 않지만 여전히 항목 분배를 위해 다른 해시 알고리즘을 사용하기를 원할 수 있습니다. com.ibm.websphere.objectgrid.plugins.PartitionableKey 인터페이스에서 이 상황이 허용됩니다. 이 인터페이스에는 단일 메소드가 있습니다.

```
Object ibmGetPartition();
```

키가 이 인터페이스를 구현할 경우 오브젝트 그리드 런타임은 키 오브젝트의 해시가 아닌 이 메소드에서 리턴한 오브젝트의 해시를 사용합니다.

런타임 시 파티션

com.ibm.websphere.objectgrid.PartitionManager 인터페이스는 응용프로그램이 런타임 시 파티션 지정에 대한 정보를 판별할 수 있도록 하는 API를 제공합니다. 응용프로그램은 BackingMap 인터페이스의 getPartitionManager 메소드를 사용하여 이 인터페이스의 인스턴스에 대한 참조를 확보할 수 있습니다. 맵의 BackingMap 참조는 오브젝트 그리드 인스턴스에서 오브젝트 그리드 인터페이스의 getMap(String) 메소드를 사용하여 확보할 수 있습니다. 또는 Loader 인터페이스의 preload(Session, BackingMap)와 같은 일부 플러그인 콜백의 매개변수로 전달됩니다.

PartitionManager 인터페이스의 메소드

PartitionManager 인스턴스를 사용하여 응용프로그램은 파티션 지정에 대한 다음 사실을 판별할 수 있습니다.

메소드 이름	설명
int getNumOfPartitions()	맵을 분할 중인 파티션 수를 리턴합니다.
int getPartition(Object key)	이것은 지정된 키를 보유한 항목에 사용된 0 기본 파티션 수를 리턴합니다.
List /*Integer*/ getPartitions(List /*Object*/ keys)	이 메소드는 getPartition 메소드와 동일하지만 대신 키 목록에서 작동됩니다. 리턴된 정수 목록은 각 해당 입력 키의 파티션 수를 포함합니다.
List /*List Integer*/ getPartitionLists(List /* Object */ keys)	이 메소드는 getPartitions 메소드와 동일하지만 파티션 목록의 정렬된 목록을 리턴합니다. 예를 들어, 리턴된 목록의 첫 번째 항목은 파티션 0에 해당하는 입력 키의 목록을 포함합니다. 다음 항목은 파티션 1에 해당하는 입력 키의 목록을 포함합니다. 이후도 이러한 형태로 계속 진행됩니다.
List /*LogSequence*/ partitionLogSequence (LogSequence ls)	이 메소드는 LogSequence를 지정된 파티션의 LogSequences 목록으로 분할합니다. 입력 LogSequence가 조사되고 여기에 있는 각 LogElement의 해당 파티션이 판별됩니다. LogElement를 보유한 각 파티션에 대해 순서가 조사되면 해당 LogElements의 LogSequence가 리턴됩니다.

파티션 지정 제한사항

트랜잭션에서 트랜잭션당 한 개의 파티션에 있는 항목만 수정할 수 있습니다. 트랜잭션이 MapSet에 있는 여러 개의 항목을 수정하고 해당 항목이 다른 파티션으로 해시되면 트랜잭션을 확약하려고 시도될 때 해당 트랜잭션이 롤백됩니다. 트랜잭션은 다른 파티션의 오브젝트를 읽을 수 있습니다. 그러나, 트랜잭션은 단일 파티션 내에 있는 항목만 수정할 수 있습니다.

파티션의 1 차 선출 시 응용프로그램 이벤트

맵에 로더가 제공되고 로더에서 ReplicaPreloadController도 구현하면 응용프로그램은 checkPreloadStatus 콜백을 사용하여 해당 메소드 호출을 수신하는 JVM이 이제 해당 파티션의 1차임을 표시하는 이벤트를 수신할 수 있습니다. BackingMap 인터페이스의 getPartitionId 메소드를 사용하여 파티션 ID를 식별할 수 있습니다. 사전 로드에 대한 자세한 정보는 217 페이지의 『로더』를 참조하십시오.

클라이언트에서 파티션 지정과 서버에서 실행 비교

파티션 지정은 응용프로그램이 오브젝트 그리드 인터페이스의 connect 메소드를 사용하여 확보한 오브젝트 그리드를 사용할 때에만 작동합니다. 플러그인의 콜백으로 오브젝트 그리드가 응용프로그램에 제공되면 이것은 라우팅하지 않는 로컬 오브젝트 그리드가 됩니다. 서버에서 실행 중이고 파티션 지정 기능을 투명하게 이용하려면 모든 트랜잭션의 connect 메소드를 사용하여 확보한 오브젝트 그리드를 사용하십시오. 그러나, 프레임워크에서 제공한 로컬 오브젝트 그리드 참조를 사용하는 경우와 비교할 때 성능 손실이 있습니다. 파티션 지정 기능이 필요 없으면 가능한 경우 플러그인에 제공된 로컬 참조를 사용하십시오.

색인 지정

색인 지정 기능을 사용하여 BackingMap에 한 개 또는 여러 개의 색인을 빌드할 수 있습니다. 색인은 BackingMap의 오브젝트 속성으로부터 빌드됩니다. 이 기능은 응용프로그램이 특정 오브젝트를 더욱 신속하게 찾는 방법을 제공합니다. 색인이 없으면 응용프로그램은 오브젝트 키로 오브젝트를 찾아야 합니다. 응용프로그램은 색인 지정 기능을 사용하여 특정 값을 가지거나 값 범위 내에 있는 오브젝트를 찾을 수 있습니다. 이것은 지정된 기준으로 조회하여 EJB(Enterprise JavaBeans)를 찾을 수 있는 EJB 조회와 유사합니다. 색인 지정은 응용프로그램에 오브젝트를 더 쉽게 찾는 편리함과 오브젝트 검색 프로세스에서 성능 향상을 제공합니다.

두 가지 유형(정적 및 동적)의 색인 지정이 있습니다. *정적* 색인 지정을 사용하면 오브젝트 그리드 인스턴스를 초기화하기 전에 BackingMap에 색인 플러그인을 구성해야 합니다. BackingMap의 XML 또는 프로그램 방식 구성으로 이 구성을 수행할 수 있습니다. 정적 색인 지정은 오브젝트 그리드 초기화 중에 색인 빌드를 시작합니다. 색인은

항상 BackingMap과 동기화되고 사용할 준비가 되어 있습니다. 정적 색인 지정 프로세스가 시작된 후 색인 유지보수는 오브젝트 그리드 트랜잭션 관리 프로세스의 일부분입니다. 트랜잭션이 변경사항을 확약할 때 이러한 변경사항은 정적 색인도 갱신합니다. 트랜잭션이 롤백되면 색인 변경사항도 롤백됩니다.

동적 색인 지정을 사용하여 포함되는 오브젝트 그리드 인스턴스 초기화 전이나 후에 BackingMap에 색인을 작성할 수 있습니다. 응용프로그램은 동적 색인 지정 프로세스 전반에 걸쳐 라이프 사이클 제어를 수행합니다. 동적 색인이 더 이상 필요하지 않으면 제거할 수 있습니다. 응용프로그램이 동적 색인을 작성할 경우 색인 빌드 프로세스를 완료하는 데 소요되는 시간 때문에 색인이 즉시 사용하도록 준비되지 않을 수 있습니다. 시간은 색인 지정되는 데이터 양에 따라 달라지므로 특정 색인 지정 이벤트가 발생할 때 알림을 수신할 응용프로그램에 DynamicIndexCallback 인터페이스가 제공됩니다. 이러한 이벤트에는 준비, 오류 및 파기가 있습니다. 응용프로그램은 이 콜백 인터페이스를 구현하고 동적 색인 지정 프로세스로 등록할 수 있습니다.

색인 지정 기능은 MapIndexPlugin 플러그인으로 또는 간단히 색인으로 나타냅니다. MapIndexPlugin은 BackingMap 플러그인입니다. BackingMap은 색인 구성 규칙을 따를 경우 다중 색인 플러그인을 구성할 수 있습니다.

BackingMap이 색인 플러그인을 구성했으면 해당 ObjectMap에서 색인 프록시 오브젝트를 검색할 수 있습니다. ObjectMap에서 getIndex 메소드를 호출하고 색인 플러그인의 이름을 전달하면 색인 프록시 오브젝트가 리턴됩니다. 색인 프록시 오브젝트는 MapIndex, MapRangeIndex 또는 사용자 정의된 색인 인터페이스와 같은 해당 인터페이스 응용프로그램 색인 인터페이스로 캐스트되어야 합니다.

현재 색인 지정 기능은 분산 캐시가 아닌 로컬 캐시에서만 지원됩니다. 색인 지정 조작을 분산 캐시에 대해 시도하면 `UnsupportedOperationException` 예외가 발생합니다.

색인 플러그인 구현

com.ibm.websphere.objectgrid.plugins.index 패키지의 HashIndex 클래스가 내장 응용프로그램 색인 인터페이스인 MapIndex와 MapRangeIndex를 모두 지원할 수 있는 색인 플러그인 구현 시 빌드됩니다.

응용프로그램은 고유의 색인 플러그인 구현을 제공하여 더욱 복잡한 색인을 프로그래밍하게 할 수 있습니다. index 구현 클래스는 com.ibm.websphere.objectgrid.plugins.index.MapIndexPlugin 인터페이스를 구현해야 합니다. MapIndexPlugin은 다음 정의를 갖습니다.

```
/**
* An index implementation must implement this interface so that modifications
* to the Map are propagated to it so that it can maintain the index as
* transactions are committed. Only attributes that implement the
* {@link java.lang.Comparable} interface are eligible to be indexed.
*
```

```
* @see com.ibm.websphere.objectgrid.plugins.index.MapIndex
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex
*/
public interface MapIndexPlugin
{
/**
* This should be the name of the attribute to be indexed. If the object
* has an attribute called EmployeeName then the index will call the
* "getEmployeeName" method. The attribute name must be the name
* as that in the get method and the attribute must implement the
* {@link java.lang.Comparable} interface.
*
* @param attributeName
* The name of the attribute to set.
*/
public void setAttributeName(String attributeName);
/**
* This index name.
*
* @return The name of the index.
*
* @see com.ibm.websphere.objectgrid.ObjectMap#getIndex
*/
 String getName();
/**
* Gets an index proxy object for performing index lookup operations. The
* caller must cast the object returned to either a MapIndex or MapRangeIndex
* object to perform the lookup operations.
*
* @param map The MapIndexInfo object required for maintaining the index.
* .
* @return a proxy to either an object that implements MapIndex or MapRangeIndex.
*/
Object getIndexProxy(MapIndexInfo map);
/**
* This is called by the core to allow the index to be updated as the result
* of changes applied to map during the commit cycle of a transaction.
* Use the {@link LogElement#getType()} method to determine what operation is
* required to for updating the index. Use the {@link LogElement#getBeforeImage()}
* to get the value object that existed prior to committing transaction applying
* a change to the map and the {@link LogElement#getAfterImage()} to get the value
* object after the committing transaction applied the change to the map entry.
*
* Note, the {@link #undoBatchUpdate(TxID, LogSequence)} method may be called
* later to undo these changes if an exception occurs that causes committing
* transactions to be rolled back instead.
*
* @param txid The transaction for the changes.
* @param sequence The log sequence that contains changes from transaction.
*
* @throws ObjectGridRuntimeException is a failure occurs that requires transaction
* to be rolled back.
*/
void doBatchUpdate(TxID txid, LogSequence sequence) throws
ObjectGridRuntimeException;
/**
* This is called by the core to undo any changes made to the index as a result of
* a prior call to the {@link #doBatchUpdate(TxID, LogSequence)} method. This
* method is called when an exception or error condition that requires all
* changes made by transaction to be rolled back. For this reason, the
* implementation of this method should catch all Throwable and continue with
* next LogElement in the LogSequence until all LogElements are processed so that
* as many changes to the index is undone as possible. An ObjectGridException
* should only be thrown after processing the entire LogSequence and this method
* was unable to successfully undo 1 or more changes in the LogSequence.
*
* Use the {@link LogElement#getUndoType()} method to determine what operation is
* required to undo any change made to the index. Use the
* {@link LogElement#getBeforeImage()} to get the value object that existed prior
```

```
* to committing transaction applying a change to the map and the {@link
* LogElement#getAfterImage()} to get the value object after the committing
* transaction applied the change to the map entry.
*
* @param txid The transaction for the changes.
* @param sequence The log sequence that contains changes from transaction.
*
*/
void undoBatchUpdate( TxID txid, LogSequence sequence) throws ObjectGridException;
}
```

setAttributeName 및 getName 메소드는 간단하며 이름으로 나타냅니다. 다른 메소드는 좀 더 많은 주의가 필요합니다.

getIndexProxy 메소드

getIndexProxy 메소드는 MapIndex 인터페이스, MapRangeIndex 인터페이스 또는 사용자 정의 색인 인터페이스를 구현하는 색인 프록시 오브젝트를 리턴해야 합니다. 색인 프록시 오브젝트의 구현은 색인 플러그인의 코어 부분입니다.

MapIndexInfo 오브젝트가 이 메소드에 전달되어 트랜잭션 변경 정보를 제공합니다. 이것은 getIndexProxy 메소드를 호출하는 현재 트랜잭션에만 표시되는 데이터입니다. 색인 프록시 오브젝트는 이 MapIndexInfo 오브젝트를 사용하여 이 트랜잭션 데이터를 검색할 수 있습니다.

다음은 MapIndexInfo 인터페이스의 정의입니다.

```
/**
* This interface is used to provide an index with detailed change information
* for a specific Map in a transaction.
*/
public interface MapIndexInfo
{
/**
* An index contains the key values of a set of map entries that have a
* a specific attribute value. This method returns the ObjectMap the
* index is referring to ObjectMap that the index is associated with.
*
* @return ObjectMap this index is associated with.
*/
ObjectMap getMap();
/**
* Returns the set of all changes made by the current transaction to the
* ObjectMap that is returned by the {@link #getMap()} method.
*
* @param includeRemoved must be set to true to include LogElement.DELETE types
* in the list returned by this method.
*
* @return a List of LogElement created for each ObjectMap entry that was
* either inserted, updated, or removed by current transaction.
*
* @throws ObjectGridRuntimeException
*/
List getTransactionChanges(boolean includeRemoved) throws
ObjectGridRuntimeException;
/**
* This returns the set of changes as they apply to a particular set of keys
```

```
* in the current transaction for the ObjectMap that is returned by the
* {@link #getMap()} method. If a key has not been referenced
* in the transaction then null is returned.
*
* @param keys The list of keys for which the data is required.
* @return a List of LogElement corresponding to the keys or null if the keys
* was not referenced.
*
* @throws ObjectGridRuntimeException
*
* @see com.ibm.websphere.objectgrid.plugins.LogElement
* @see com.ibm.websphere.objectgrid.ObjectMap
*/
List getTransactionChanges(List keys) throws ObjectGridRuntimeException;
}
```

getIndexProxy 메소드는 ObjectMap 인터페이스의 getIndex(String name) 메소드를 지원하기 위해 설계되었습니다. 리턴된 색인 프록시 오브젝트는 ObjectMap의 getIndex 메소드에 의해 리턴되는 오브젝트입니다. 예를 들어, 응용프로그램이 ObjectMap의 getIndex 메소드를 호출하면, 이 메소드가 이 getIndexProxy 메소드를 호출하고 이 getIndexProxy 메소드에 의해 리턴되는 오브젝트를 리턴합니다. 응용프로그램은 리턴된 색인 프록시 오브젝트를 MapIndex, MapRangeIndex 또는 다른 사용자 정의된 색인 인터페이스와 같은 응용프로그램 색인 인터페이스로 캐스트해야 합니다.

다음 코드 예는 getIndexProxy 메소드에서 리턴할 수 있는 일부 색인 프록시 오브젝트 구현을 나타낸 것입니다.

```
/**
* A class used to return a proxy to this map index
* so that applications can perform query operations
* using MapIndex interface.
*/
class Proxy implements MapIndex
{
/**
* The MapIndexInfo object associated with this index proxy object.
*/
protected MapIndexInfo ivMap;
/**
* Maximum number of retries when concurrent transactions
* modify index during a query operation.
*/
protected static final int RETRY_LIMIT = 10;
/**
* EQUAL comparator to use.
*/
final protected ProxyEQComparator ivEQ = new ProxyEQComparator();
final protected ProxyGTComparator ivGT = new ProxyGTComparator();
final protected ProxyRangeComparator ivRange = new ProxyRangeComparator();
/**
* Construct a proxy object for a given ObjectMap.
*
* @param map
* is the MapIndexInfo object.
*/
Proxy(MapIndexInfo map)
{
ivMap = map;
}
/**
*
```

```
* @see com.ibm.websphere.objectgrid.plugins.index.MapIndex#findAll
*/
public Iterator findAll(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivEQ.ivAttribute = (Comparable) attributeValue;
ArrayList resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variables that need to be re-initialize each time thru loop.
retry = false;
resultList = new ArrayList();
// Use index to obtains the Set of keys for map entries that
// contain the specified attribute value.
Set s = (Set) index.get( attributeValue );
Set keySet = processSet( s, ivEQ );
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
// Whoops, another transaction modified Set obtained from index
// while the above was iterating over the Set to perform the
// addAll operation. Therefore, we need to retry by starting
// over beginning with getting Set from index to pickup changes
// from the transaction that just modified the Set.
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
retry = true;
}
} while ( retry );
// Return iterator for result list created by above loop.
Iterator result = resultList.iterator();
  return result;
}
/**
* Process a Set obtained from index to determine if which of the keys
* are for map entries that meet the query select criteria.
*
* @param s
* is the Set of key values for entries in BackingMap
* this index is built over. A null reference indicates only
* changes from current transaction needs to be processed.
*
* @param comparator
* is the comparator to use for making range check.
*
* @return Set of keys that met the select criteria or a null reference
* if a Exception occurs while iterating over the Set.
*
* @throws FinderException
* if an error condition prevents processing of Set
* from being performed.
*/
protected Set processSet(Set s, ProxyComparator comparator)
throws FinderException {
HashSet resultSet = new HashSet();
//...
```

```
//process the s Set, use comparator and prepare the resultSet.
//...
return resultSet;
}
} // end class Proxy
/**
* A class used to return a proxy to this map index so that applications can
* perform query operations using MapRangeIndex interface.
*/
class RangeProxy extends Proxy implements MapRangeIndex
{
/**
* Various comparators needed by proxy to perform the
* the range check of attribute value.
*/
final private ProxyLTComparator ivLT = new ProxyLTComparator();
final private ProxyLEComparator ivLE = new ProxyLEComparator();
final private ProxyGEComparator ivGE = new ProxyGEComparator();
/**
* Index is a synchronized SortedMap.
*/
final SortedMap ivIndexSortedMap;
/**
* Construct a MapRangeIndex proxy.
*/
RangeProxy(MapIndexInfo map)
{
super( map );
ivIndexSortedMap = (SortedMap) index;
}
/**
* Execute query operation on a specified Map and ProxyComparator object.
*
* @param map
* is a subset of the index to perform finder operation on.
* @param proxyComparator
* is a comparator used to perform range check on attribute value.
*
* @return Set of keys required to be returned by finder method.
*
* @throws FinderException
* is any failure occurs during execution of the query.
*/
private Set executeQuery(Map map, ProxyComparator proxyComparator)
throws FinderException {
HashSet resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variables that need to be re-initialize each time thru loop.
retry = false;
resultList = new HashSet();
// Use index to obtains the Set of keys for map entries that
// contain the specified attribute value.
SortedMap treeMap = (SortedMap) index;
Collection values = map.values();
if ( values.isEmpty() )
{
// Nothing in range currently in index, so we only
// need to check changes from current transaction.
Set keySet = processSet( null, proxyComparator );
if ( keySet != null )
{
resultList.addAll( keySet );
}
}
else
{
```

```
// Index does contains some keys in range, so we need to query
// both index entries as well as current transaction changes.
Iterator iter = values.iterator();
 while ( iter.hasNext() )
{
Set keySet;
try
{
Set s = (Set) iter.next();
keySet = processSet( s, proxyComparator );
}
catch (ConcurrentModificationException e)
{
// Indicate unable to get keySet.
keySet = null;
}
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
retry = true;
}
}
}
} while ( retry );
return resultList;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findGreater
*/
public Iterator findGreater(Object attributeValue)
throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivGT.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGT );
Iterator result = resultSet.iterator();
  return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findGreaterEqual
*/
public Iterator findGreaterEqual(Object attributeValue)
throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivGE.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGE );
```

```
Iterator result = resultSet.iterator();
  return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLess
*/
public Iterator findLess(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivLT.ivAttribute = (Comparable) attributeValue;
SortedMap headMap = ivIndexSortedMap.headMap( attributeValue );
Set resultSet = executeQuery( headMap, ivLT );
Iterator result = resultSet.iterator();
  return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLessEqual
*/
public Iterator findLessEqual(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivLE.ivAttribute = (Comparable) attributeValue;
Set resultSet;
int retryCount = 0;
boolean retry;
do
{
// re-initialize for each retry that occurs.
retry = false;
SortedMap headMap = ivIndexSortedMap.headMap( attributeValue );
resultSet = executeQuery( headMap, ivLE );
Set s = (Set) ivIndexSortedMap.get( attributeValue );
ivEQ.ivAttribute = (Comparable) attributeValue;
Set equalSet = processSet( s, ivEQ );
if ( equalSet != null )
{
    if ( ! equalSet.isEmpty() )
{
resultSet.addAll( equalSet );
}
}
else
{
// Whoops, another transaction modified index while processSet
// was executing. Therefore, we need to retry the entire query.
++retryCount;
retry = true;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
}
} while ( retry );
// Return iterator for result list created by above loop.
Iterator result = resultSet.iterator();
```

```
  return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findRange
*/
public Iterator findRange(Object lowAttributeValue, Object highAttributeValue)
throws FinderException {
if ( lowAttributeValue == null )
{
throw new IllegalArgumentException(
"the lowAttributeValue must be a non null reference" );
}
if ( highAttributeValue == null )
{
throw new IllegalArgumentException(
"the highAttributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivRange.ivLowAttribute = (Comparable) lowAttributeValue;
ivRange.ivHighAttribute = (Comparable) highAttributeValue;
SortedMap subMap = ivIndexSortedMap.
subMap( lowAttributeValue, highAttributeValue );
Set resultSet = executeQuery( subMap, ivRange );
Iterator result = resultSet.iterator();
  return result;
}
}
/**
* Abstract base class used for determining if attribute value is in range.
*/
abstract class ProxyComparator
{
abstract boolean inRange(Object attribute);
}
/**
* Performs less than range check.
*/
class ProxyLTComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
 return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) < 0 );
}
}
}
/**
* Performs less than or equal range check.
*/
class ProxyLEComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
 return false;
}
else
{
```

```
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) <= 0 );
}
}
}
/**
* Performs equal range check.
*/
class ProxyEQComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
 return false;
}
else
{
return ( ivAttribute.compareTo( attribute ) == 0 );
}
}
}
/**
* Performs greater than range check.
*/
class ProxyGTComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
 return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) > 0 );
}
}
}
/**
* Performs greater than or equal range check.
*/
class ProxyGEComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
 return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) >= 0 );
}
}
}
/**
* Performs lowAttribute <= attribute < highAttribute range check.
*/
class ProxyRangeComparator extends ProxyComparator
{
Comparable ivLowAttribute;
Comparable ivHighAttribute;
boolean inRange(Object o)
```

```
{
if ( o == null )
{
 return false;
}
Comparable attribute = (Comparable) o;
if ( attribute.compareTo( ivLowAttribute ) < 0 )
{
return false; // attribute < ivLowAttribute
}
else
{
// ivLowAttribute <= attribute
if ( attribute.compareTo( ivHighAttribute ) < 0 )
{
return true; // ivLowAttribute <= attribute < ivHighAttribute
}
else
{
return false; // attribute >= ivHighAttribute
}
}
}
}
```

doBatchUpdate 및 undoBatchUpdate 메소드

doBatchUpdate 및 undoBatchUpdate 메소드는 MapIndexPlugin 인터페이스에서 중요한 메소드입니다. doBatchUpdate 메소드는 트랜잭션의 확약 주기 중에 맵에 적용된 변경사항의 결과로 호출됩니다. doBatchUpdate 메소드에 대한 이전 호출의 결과로 색인에 작성된 변경사항을 실행 취소하려면 undoBatchUpdate 메소드를 사용합니다. 이 메소드는 트랜잭션에서 작성한 모든 변경사항을 롤백해야 하는 예외 또는 오류 조건이 발생할 때 호출됩니다. 두 메소드에는 모두 현재 TxID 및 이 Map에 대한 변경사항 목록이 제공됩니다. 이 메소드가 변경사항을 반복하고 처리해야 합니다.

다음 코드 예는 이러한 두 개의 메소드 및 지원 메소드를 구현하는 방법을 표시합니다.

```
/**
* The synchronized Map used as the index implementation where
* the attribute value object is the key and a Java Set is the value.
* A Set member is the key of a BackingMap entry that matches attribute value.
*/
Map index; //<Object attribute, Set keys>
public void doBatchUpdate(TxID txid, LogSequence sequence)
throws ObjectGridRuntimeException
{
 Iterator iter = sequence.getAllChanges();
 while ( iter.hasNext() )
{
LogElement elem = (LogElement) iter.next();
  Object key = elem.getCacheEntry().getKey();
LogElement.Type doType = elem.getType();
if ( doType == LogElement.INSERT )
{
Object newAttribute = getAttribute( elem.getAfterImage() );
insertIntoIndex( key, newAttribute );
}
else if ( doType == LogElement.UPDATE )
{
Object newAttribute = getAttribute( elem.getAfterImage() );
Object oldAttribute = getAttribute( elem.getBeforeImage() );
updateIndex( key, oldAttribute, newAttribute );
```

```
}
else if ( doType == LogElement.DELETE )
{
Object oldAttribute = getAttribute( elem.getBeforeImage() );
removeFromIndex( key, oldAttribute );
}
else if ( doType == LogElement.EVICT )
{
Object beforeImage = elem.getBeforeImage();
if ( beforeImage != null )
{
Object oldAttribute = getAttribute( beforeImage );
removeFromIndex( key, oldAttribute );
}
}
}
}
public void undoBatchUpdate(TxID txid, LogSequence sequence)
throws ObjectGridException
{
int errors = 0;
 Iterator iter = sequence.getAllChanges();
 while ( iter.hasNext() )
{
try
{
LogElement elem = (LogElement) iter.next();
  Object key = elem.getCacheEntry().getKey();
LogElement.Type undoType = elem.getUndoType();
if ( undoType == LogElement.INSERT )
{
Object newAttribute = getAttribute( elem.getBeforeImage() );
insertIntoIndex( key, newAttribute );
}
else if ( undoType == LogElement.UPDATE )
{
Object oldAttribute = getAttribute( elem.getAfterImage() );
Object newAttribute = getAttribute( elem.getBeforeImage() );
updateIndex( key, oldAttribute, newAttribute );
}
else if ( undoType == LogElement.DELETE )
{
Object oldAttribute = getAttribute( elem.getAfterImage() );
removeFromIndex( key, oldAttribute );
}
}
catch ( Throwable t )
{
++errors;
}
}
if ( errors > 0 )
{
throw new ObjectGridException( errors
+ " exceptions occurred during rollback of index changes.");
}
}
/**
* Extracts the attribute from a specified value Object.
*
* @param value The value Object.
*
* @return attribute from the value Object, which may be a null reference.
*
* @throws ObjectGridRuntimeException is thrown if any exception occurs
* attempting to extract the attribute value from the value Object.
*/
private Object getAttribute(Object value) throws ObjectGridRuntimeException
{
```

```
try
{
Object attribute = null;
 if ( value != null )
{
Method m = getAttributeMethod( value );
attribute = getAttributeMethod.invoke( value, emptyArray );
}
return attribute;
}
catch ( InvocationTargetException e )
{
Throwable t = e.getTargetException();
throw new ObjectGridRuntimeException( "Caught unexpected Throwable", t );
}
catch ( Throwable t )
{
throw new ObjectGridRuntimeException( "Caught unexpected Throwable", t );
}
}
private void updateIndex(Object key, Object oldAttribute, Object newAttribute)
{
// Was attributed changed by the update?
if ( newAttribute != null && oldAttribute != null &&
oldAttribute.equals( newAttribute ) )
{
// Nope, then nothing needs to be changed in index.
return;
}
// Unless we restrict Loader to only access tables with non-nullable columns,
// we have to handle the possibility that the attribute is null.
Set oldKeys = null;
if ( oldAttribute != null )
{
// Remove oldAttribute from index entry.
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
// Unless we restrict Loader to only access tables with non-nullable columns,
// we have to handle the possibility that the attribute is null.
Set keys = null;
if ( newAttribute != null )
{
keys = (Set) index.get( newAttribute );
// Add newAttribute to index.
if ( keys == null )
{
// Since different transactions can be updating different BackingMap
// entries and multiple map entries can have same attribute value,
// we need to use a synchronized Set object to ensure only
// one transaction at a time can make changes to the Set.
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Add key for this map entry to the Set of keys for the new attribute value.
keys.add( key );
}
}
private void insertIntoIndex( Object key, Object newAttribute )
{
// Unless we restrict Loader to only access tables with non-nullable columns,
// we have to handle the possibility that the attribute is null.
```

```
if ( newAttribute != null )
{
Set keys = (Set) index.get( newAttribute );
if ( keys == null )
{
// Since different transactions can be updating different
// Map entries and multiple map entries can have same attribute
// value, we need to use a synchronized Set object to ensure only
// one transaction at a time can make changes to the Set.
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Add key for this map entry to the Set of keys for the new attribute value.
keys.add( key );
}
}
private void removeFromIndex(Object key, Object oldAttribute )
{
// Extract the old attribute value
Set oldKeys = null;
// Unless we restrict Loader to only access tables with non-nullable columns,
// we have to handle the possibility that the attribute is null.
if ( oldAttribute != null )
{
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
}
```

응용프로그램 색인 인터페이스

응용프로그램 색인 인터페이스는 조회 메소드를 지원하기 위해 설계되었습니다. 현재 MapIndex와 MapRangeIndex인 두 개의 응용프로그램 색인 인터페이스가 정의되어 있습니다.

MapIndex

MapIndex는 속성 값으로 오브젝트를 검색하는 단순 색인입니다. 이것을 사용하여 맵의 속성 값을 색인화할 수 있습니다. 응용프로그램이 이 메소드를 사용하면 특정 속성 값을 가지는 모든 오브젝트를 맵에서 빨리 찾을 수 있습니다. 다음은 MapIndex 인터페이스의 정의입니다.

```
/**
* This is an abstract index that can be created on an empty Map. The
* index can be used to perform efficient look ups and possibly other
* operations such as relational operations on an attribute in a Map.
* The MapIndex is provided with all update events and maintains an
* index that can be used to issue simple queries against the index
* later. The index could use an index defined callback to make an
* index on composite attributes.
*/
public interface MapIndex
{
/**
* Returns the Keys for the entries that have the specified attribute
```

```
* value.
*
* @param attributeValue
* a non-null reference to the attribute value to search for.
*
* @return A list of the keys for the entries with that attribute.
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry limit is
* reached when concurrent transactions updating the index
* prevent findAll from completing.
*/
Iterator findAll(Object attributeValue) throws FinderException;
}
```

MapRangeIndex

MapRangeIndex는 특정 범위에서 속성 값을 가지는 오브젝트를 검색하는 단순 색인입니다. 이것을 사용하여 맵의 속성 값을 색인화할 수 있습니다. 이 색인은 값 범위 및 값 비교 조작을 사용하여 조회할 수 있다는 점에서 MapIndex와 다릅니다. 이 메소드를 조회에 사용하여 특정 값보다 작거나 큰 속성 값을 가지는 모든 오브젝트를 찾을 수 있습니다. 다음은 MapRangeIndex 인터페이스의 정의입니다.

```
/**
* This is an index that allows comparison type searches.
*/
public interface MapRangeIndex extends MapIndex
{
/**
* This find all keys with entries with an attribute greater than the
* specified value.
*
* @param attributeValue is the low endpoint of range excluding the
* low attribute value.
*
* @return The set of keys with values greater than the attribute.
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry
* limit is reached
* when concurrent transactions updating the index prevent
* findAll from completing.
*/
Iterator findGreater(Object attributeValue) throws FinderException;
/**
* This find all keys with entries with an attribute greater or equal
* to the specified value.
*
* @param attributeValue is the low endpoint of range including the
* low attribute value.
*
* @return The set of keys with attributes meeting the criteria
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry
* limit is reached
* when concurrent transactions updating the index prevent findAll
* from completing.
*/
Iterator findGreaterEqual(Object attributeValue) throws FinderException;
/**
* This find all keys with entries with an attribute less than the
* specified value.
```

```
*
* @param attributeValue is the high endpoint of range excluding high
* endpoint value.
*
* @return The set of keys with attributes meeting the criteria
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry limit
* is reached
* when concurrent transactions updating the index prevent
* findAll from completing.
*/
Iterator findLess(Object attributeValue) throws FinderException;
/**
* This find all keys with entries with an attribute less than or equal
* to the specified value.
*
* @param attributeValue is the high endpoint of range including high
* endpoint value.
*
* @return The set of keys with attributes meeting the criteria
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry limit
* is reached
* when concurrent transactions updating the index prevent
* findAll from completing.
*/
Iterator findLessEqual(Object attributeValue) throws FinderException;
/**
* This returns all keys for the entries with the attribute inclusively
* within the specified range such that lowAttributeValue <= attribute
* < highAattributeValue.
*
* @param lowAttributeValue is the low endpoint of range including the
* low attribute value.
* @param highAttributeValue is the high endpoint of range excluding
* high attribute value.
*
* @return The list of keys with entries in that range, in ascending order.
*
* @throws IllegalArgumentException if either lowAttributeValue or
* highAttributeValue
* argument is null or lowAttributeValue > highAttributeValue.
* @throws FinderException is thrown if exception or retry limit is reached
* when concurrent transactions updating the index prevent
* findAll from completing.
*/
Iterator findRange(Object lowAttributeValue, Object highAttributeValue)
throws FinderException;
}
```

응용프로그램은 ObjectMap 인스턴스의 getIndex 메소드에서 확보한 색인 오브젝트를 해당되는 응용프로그램 색인 인터페이스로 캐스트해야 합니다. 색인 플러그인이 MapRangeIndex 인터페이스를 지원하도록 설계된 경우 색인 오브젝트를 MapRangeIndex 유형으로 캐스트할 수 있으며, 그렇지 않으면 MapIndex 유형으로 캐스트되어야 합니다.

사용자 정의 응용프로그램 색인 인터페이스를 정의할 수 있습니다. 사용자 정의 응용프로그램 색인을 MapIndexPlugin의 getIndexProxy 메소드에서 리턴할

수 있는 색인 프록시 오브젝트로 구현하십시오. ObjectMap 인스턴스의 getIndex 메소드에서 확보한 색인 오브젝트를 이 사용자 정의된 응용프로그램 색인 인터페이스로 캐스트하고 사용하십시오.

정적 색인 플러그인 추가

정적 색인 플러그인을 BackingMap 구성에 추가하는 접근 방식으로는 XML 구성과 프로그램 방식 구성 두 가지가 있습니다. 다음 예는 XML 구성 접근 방식을 설명합니다.

```
<backingMapPluginCollection id="person">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.
plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="CODE"
description="index name" />
<property name="RangeIndex" type="boolean" value="true" description="true
for MapRangeIndex" />
<property name="AttributeName" type="java.lang.String" value="employeeCode"
description="attribute name" />
</bean>
</backingMapPluginCollection>
```

BackingMap 인터페이스에는 정적 색인 플러그인을 추가하는 데 사용할 수 있는 두 개의 메소드인 addMapIndexPlugin 메소드와 setMapIndexPlugins 메소드가 있습니다. 다음은 이 두 메소드의 정의입니다.

```
/**
* This method adds an index plugin to this Map. We assume the index implementation
* was constructed
* with the name of the attribute to index. The name of the index is specified when
* the index is constructed.
*
* Note, to avoid an {@link IllegalStateException}, this method must be called
* prior to {@link ObjectGrid#initialize()} method. Also, keep in mind that the
* {@link ObjectGrid#getSession()} method implicitly calls the
* {@link ObjectGrid#initialize()} method if it has yet to be called by the
* application. *
* @param index The index implementation.
*
* @throws IndexAlreadyDefinedException This index already exists.
* @throws IllegalStateException if this method is called after the
* {@link ObjectGrid#initialize()} method is called.
*/
public void addMapIndexPlugin(MapIndexPlugin index)
throws IndexAlreadyDefinedException;
/**
* This method sets the list of MapIndexPlugin objects for this BackingMap.
* If the BackingMap already has a List of MapIndexPlugin objects,
* that list is replaced by the List passed as
* an argument to the current invocation of this method.
*
* Note, to avoid an {@link IllegalStateException}, this method must be called
* prior to {@link ObjectGrid#initialize()} method. Also, keep in mind that the
* {@link ObjectGrid#getSession()} method implicitly calls the
* {@link ObjectGrid#initialize()} method if it has yet to be called by the
* application. *
* @param indexList A non-null reference to a List of {@link MapIndexPlugin}
* objects.
*
* @throws IllegalArgumentException is thrown if indexList is null
* or the indexList contains an object that is not
```

```
* an instanceof {@link MapIndexPlugin}.
*/
public void setMapIndexPlugins(List indexList );
```

다음 코드의 스니펫은 프로그램 방식의 구성 접근 방식을 나타냅니다.

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid ivObjectGrid = ogManager.createObjectGrid( "grid" );
BackingMap personBackingMap = ivObjectGrid.getMap("person");
//use the builtin HashIndex class as the index plugin class.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
personBackingMap.addMapIndexPlugin(mapIndexPlugin);
```

정적 색인 사용

정적 색인 플러그인이 BackingMap 구성에 추가되고 포함하는 오브젝트 그리드 인스턴스가 초기화되었다면 응용프로그램은 BackingMap의 ObjectMap 인스턴스에서 이름으로 색인 오브젝트를 가져올 수 있습니다. 색인 오브젝트를 응용프로그램 색인 인터페이스로 캐스트하십시오. 이제 응용프로그램 색인 인터페이스에서 지원하는 색인 조작이 실행될 수 있습니다. 다음은 ObjectMap 인터페이스의 getIndex 메소드 정의입니다.

```
/**
* This returns a reference to the named index that can be used with this Map.
* This index cannot be shared between threads and works on the same rules as
* Session. The returned value should be cast to the right index interface
* such as MapIndex or MapRangeIndex or a custom index interface such as geo
* spatial index.
*
* @param name The index name
*
* @return A reference to the index, it must be cast to the appropriate index
* interface.
*
* @throws IndexUndefinedException if the index is not defined on the BackingMap
* @throws IndexNotReadyException if the index is not ready
* @throws UnsupportedOperationException if the map is a distributed map
*/
Object getIndex(String name)
throws IndexUndefinedException, IndexNotReadyException,
UnsupportedOperationException;
```

다음 코드의 스니펫은 정적 색인을 가져오고 사용하는 방법을 나타냅니다.

```
Session session = ivObjectGrid.getSession();
ObjectMap map = session.getMap("person ");
MapRangeIndex codeIndex = (MapRangeIndex) m.getIndex("CODE");
Iterator iter = codeIndex.findLessEqual(new Integer(15));
    while (iter.hasNext()) {
Object key = iter.next();
Object value = map.get(key);
}
```

동적 색인 추가 및 제거

BackingMap 인스턴스에서 언제든지 프로그램 방식으로 동적 색인을 작성하고 제거할 수 있습니다. 동적 색인은 포함되는 오브젝트 그리드 인스턴스가 초기화된 후에도 동적 색인을 작성할 수 있다는 점에서 정적 색인과 다릅니다. 정적 색인 지정과 달리 동적 색인 지정은 비동기 프로세스이며 목적을 수행하기 전에 준비 상태에 있어야 합니다. 동적 색인을 가져오고 사용하는 방법은 정적 색인과 동일합니다. 동적 색인이 더 이상 필요 없으면 제거될 수 있습니다. BackingMap 인터페이스에는 동적 색인을 작성 및 제거하는 메소드가 있습니다. 다음은 메소드의 정의입니다.

```
/**
* Create a dynamic index on the BackingMap
*
* @param name the name of the index. The name can not be null.
* @param isRangeIndex Indicate whether to create a MapRangeIndex or a MapIndex.
* If set to true, the index will be type of MapRangeIndex.
* @param attributeName The name of the attribute to be indexed.
* The attributeName can not be null.
* @param dynamicIndexCallback The callback that will invoke upon dynamic
* index events.
* The dynamicIndexCallback is optional and can be null.
*
* @throws IndexAlreadyDefinedException if a MapIndexPlugin with the specified
* name already exists.
* @throws UnsupportedOperationException if the map is a distributed map.
*
*/
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb)
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* Create a dynamic index on the BackingMap.
*
* @param index The index implementation. The index can not be null.
* @param dynamicIndexCallback The callback that will invoke upon dynamic
* index events.
* The dynamicIndexCallback is optional and can be null.
*
* @throws IndexAlreadyDefinedException if a MapIndexPlugin with the
* specified name already exists.
* @throws UnsupportedOperationException if the map is a distributed map.
*/
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback
dynamicIndexCallback)
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* remove a dynamic index from the BackingMap
*
* @param name the name of the index. The name can not be null.
*
* @throws IndexUndefinedException if a MapIndexPlugin with the specified name
* does not exists.
* @throws OperationNotSupportedException if the map is a distributed map.
*/
public void removeDynamicIndex(String name) throws IndexUndefinedException;
```

다음 코드의 스니펫은 동적 색인을 작성, 사용 및 제거하는 프로그램 방식의 접근 방식을 나타냅니다.

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.getMap("person");
  og.initialize();
//create index after ObjectGrid initialization without DynamicIndexCallback.
bm.createDynamicIndex("CODE", true, "employeeCode", null);
try {
//If not using DynamicIndexCallback, need to wait for the Index to be ready.
//The waiting time depends on the current size of the map
Thread.sleep(3000);
} catch (Throwable t) {
//...
}
//Once the index is ready, applications can try to get application index
//interface instance.
//Applications have to find a way to ensure the index is ready to use,
//if not using DynamicIndexCallback interface.
//The following demonstrates the way to wait for the index to be ready
//The total waiting time should consider the size of the map
Session session = og.getSession();
ObjectMap m = session.getMap("person");
MapRangeIndex codeIndex = null;
int counter = 0;
int maxCounter = 10;
boolean ready = false;
while(!ready && counter < maxCounter){
try {
counter++;
codeIndex = (MapRangeIndex) m.getIndex("CODE");
ready = true;
} catch (IndexNotReadyException e) {
//implies index is not ready, ...
System.out.println("Index is not ready. continue to wait.");
try {
Thread.sleep(3000);
} catch (Throwable tt) {
//...
}
} catch (Throwable t) {
//unexpected exception
t.printStackTrace();
}
}
if(!ready){
System.out.println("Index is not ready. Need to handle this situation.");
}
//use the index to peform queries
//Refer to MapIndex or MapRangeIndex interface for supported operations.
//The object attribute that the index created on is the EmployeeCode.
//Assuming the EmployeeCode attribute is Integer type, which should be
```

```
//the data type of the parameter passed into index operations.
Iterator iter = codeIndex.findLessEqual(new Integer(15));
//remove the dynamic index when no longer needed
bm.removeDynamicIndex("CODE");
```

DynamicIndexCallback 인터페이스

DynamicIndexCallback 인터페이스는 색인 지정 이벤트(준비, 오류 또는 파기) 발생 시 알림을 받을 응용프로그램을 위해 설계되었습니다. 이것은 BackingMap의 createDynamicIndex() 메소드에 대한 선택적 매개변수입니다. 등록된 DynamicIndexCallback 인스턴스를 사용할 경우 응용프로그램은 색인 지정 이벤트 알림을 수신할 때 비즈니스 로직을 실행할 수 있습니다. 예를 들어, 준비 이벤트는 색인이 사용될 준비가 되었음을 의미합니다. 이 이벤트 알림을 수신하면 응용프로그램은 응용프로그램 색인 인터페이스 인스턴스를 가져와서 사용하려고 시도할 수 있습니다. 다음은 DynamicIndexCallback 인터페이스의 정의입니다.

```
/**
* This is the callback interface for dynamic indexing process.
* If applications wish to get notification at the event of ready, error,
* or destroy, they can implement this callback interface and register with
* the dynamic indexing process when creating dynamic index.
*
*/
public interface DynamicIndexCallback {
/**
* This callback method will be invoked when the dynamic index is ready.
*
* @param indexName
* The index name
*
*/
public void ready(String indexName);
/**
* Invoked when the dynamic indexing process encounters an unexpected error.
*
* @param indexName the index name
* @param t A Throwable object that causes the error situation in dynamic
* indexing process.
*/
public void error(String indexName, Throwable t);
/**
* This callback method will be invoked when the dynamic index is removed
*
* @param indexName
* The index name
*/
public void destroy(String indexName);
}
```

다음 코드 스니펫은 DynamicIndexCallback 인터페이스 사용을 나타냅니다.

```
BackingMap personBackingMap = ivObjectGrid.getMap("person");
DynamicIndexCallback callback = new DynamicIndexCallbackImpl();
personBackingMap.createDynamicIndex("CODE", true, "employeeCode", callback);
class DynamicIndexCallbackImpl implements DynamicIndexCallback {
public DynamicIndexCallbackImpl() {
}
```

```
public void ready(String indexName) {
System.out.println("DynamicIndexCallbackImpl.ready() -> indexName = " +
indexName);
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
Session session = og.getSession();
ObjectMap map = session.getMap("person");
MapIndex codeIndex = (MapIndex) map.getIndex("CODE");
Iterator iter = codeIndex.findAll(codeValue);
}
public void error(String indexName, Throwable t) {
System.out.println("DynamicIndexCallbackImpl.error() ->
indexName = " + indexName);
t.printStackTrace();
}
public void destroy(String indexName) {
System.out.println("DynamicIndexCallbackImpl.destroy() -> indexName = " +
indexName);
}
}
```

성능 고려

색인 지정 기능의 기본 목적 중 하나가 전체 BackingMap 성능을 향상시키는 것이지만 이 기능을 사용하기 전에 고려해야 할 일부 요소가 있습니다. 색인 지정이 적절하게 사용되지 않으면 응용프로그램의 성능이 저하될 수 있습니다.

- 동시 쓰기 트랜잭션 수. 트랜잭션이 데이터를 BackingMap에 쓸 때마다 색인 처리가 발생할 수 있습니다. 응용프로그램이 색인 조회 조작을 시도할 때 동시에 데이터를 맵에 쓰는 트랜잭션이 많으면 성능이 저하될 수 있습니다.
- 조회 조작으로 리턴된 결과 세트의 크기. 결과 세트의 크기가 증가하면 조회 성능은 저하됩니다. 결과 세트의 크기가 BackingMap의 15% 이상일 때 성능이 저하되는 한 가지 실험적인 경우가 나타나 있습니다.
- 동일한 BackingMap 상에 빌드된 색인 수. 각 색인은 시스템 자원을 이용합니다. BackingMap 상에 빌드된 색인 수가 증가하면 성능이 저하됩니다.

따라서 색인 지정 기능은 특정 환경에서 BackingMap 성능을 크게 향상시킬 수 있습니다. 색인 지정을 위한 이상적인 경우는 BackingMap이 거의 읽혀지고 조회 결과 세트가 BackingMap 항목의 작은 백분율을 차지하며 몇 개의 색인만 BackingMap에 빌드될 때입니다.

오브젝트 그리드 구성

오브젝트 그리드를 분산 환경에서 실행하거나 단일 JVM 내에서만 사용 가능한 로컬 캐시로 구성할 수 있습니다. 프로그램 방식으로 또는 ObjectGrid XML을 사용하여 로컬 오브젝트 그리드를 구성할 수 있습니다. 오브젝트 그리드 XML 파일에서 오브젝트 그리드, BackingMaps 및 해당 플러그인을 정의할 수 있습니다.

로컬 오브젝트 그리드를 분산 환경으로 이주할 수 있습니다. 분산 오브젝트 그리드를 구성하려면 오브젝트 그리드 XML과 함께 클러스터 XML을 제공해야 합니다. 클러스터 XML 파일은 오브젝트 그리드 토폴로지의 서버와 오브젝트 그리드 데이터가 여러 서버에서 파티션 및 복제되는 방법을 정의합니다. 이 섹션에서는 로컬 및 분산 오브젝트 그리드를 구성하는 방법에 대해 자세히 설명합니다.

로컬 오브젝트 그리드 구성

로컬 오브젝트 그리드를 구성하려면 『로컬 오브젝트 그리드 구성』을 참조하십시오.

분산 오브젝트 그리드

분산 오브젝트 그리드를 구성하려면 294 페이지의 『분산 오브젝트 그리드 구성』을 참조하십시오.

로컬 오브젝트 그리드 구성

프로그램 방식으로 또는 XML을 사용하여 로컬 오브젝트 그리드를 구성할 수 있습니다. ObjectGridManager는 두 가지 구성 방법 모두의 시작점입니다.

ObjectGridManager에는 로컬 오브젝트 그리드를 작성하는 데 사용할 수 있는 여러 메소드가 존재합니다. 각 메소드에 대한 자세한 설명은 99 페이지의 『ObjectGridManager 인터페이스』를 참조하십시오.

다음 주제를 사용하여 로컬 오브젝트 그리드를 구성하십시오.

- 『기본 오브젝트 그리드 구성』에서는 정의된 하나의 오브젝트 그리드와 하나의 BackingMap을 사용하여 아주 간단한 XML 파일을 작성하는 방법을 설명합니다.
- 280 페이지의 『전체 ObjectGrid 구성』에서는 XML 파일의 각 요소와 속성을 정의하고 프로그램 방식으로 XML 파일과 동일한 결과를 얻는 방법을 설명합니다.
- 292 페이지의 『혼합 모드 오브젝트 그리드 구성』에서는 XML과 프로그램 방식 구성 메소드를 조합하여 사용하는 방법을 설명합니다.

기본 오브젝트 그리드 구성

이 주제에서는 정의된 하나의 오브젝트 그리드와 하나의 BackingMap을 사용하여 아주 간단한 ObjectGrid XML 파일인 `bookstore.xml` 파일을 작성하는 방법을 설명합니다.

파일의 처음 몇 행은 각 ObjectGrid XML 파일의 필수 헤더입니다. 다음 XML은 *book* BackingMap을 사용하여 *bookstore* 오브젝트 그리드를 정의합니다.

bookstore.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

XML 파일이 ObjectGridManager 인터페이스에 전송하여 파일을 기반으로 오브젝트 그리드 인스턴스를 작성합니다. 다음 코드 스니펫은 XML 스키마에 대해 bookstore.xml 파일 유효성을 검증하고 *bookstore* 오브젝트 그리드를 작성합니다. 새로 작성된 오브젝트 그리드 인스턴스는 캐시되지 않습니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/bookstore.xml"), true, false);
```

다음 코드는 XML을 사용하지 않고 동일한 타스크를 수행합니다. 오브젝트 그리드에 프로그램 방식으로 BackingMap을 정의하려면 이 코드를 사용하십시오. 이 코드는 bookstoreGrid 오브젝트 그리드에 book BackingMap을 작성합니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
```

전체 ObjectGrid 구성

이 주제는 오브젝트 그리드를 구성하기 위한 전체 안내서입니다. XML 파일의 각 요소와 속성이 정의됩니다. 동일한 타스크를 프로그램 방식으로 수행하는 코드와 함께 샘플 XML 파일이 제공됩니다.

이 주제 전체에서는 다음 XML 파일(bookstore.xml)을 참조합니다. 이 예 다음에 이 파일의 요소와 속성이 자세히 설명되어 있습니다.

bookstore.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<bean id="ObjectGridEventListener"
className="com.company.organization.MyObjectGridEventListener" />
<backingMap name="books" pluginCollectionRef="collection1" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
```

```
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
classname="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
<property name="maxSize" type="int" value="321" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

objectGridConfig 요소

발생 횟수: 1

하위 요소: objectGrids 요소 및 backingMapPluginCollections 요소

objectGridConfig 요소는 XML 파일의 최상위 레벨 요소입니다. 위 예에 표시된 대로 XML 문서로 작성되어야 합니다. 이 요소는 파일의 네임 스페이스 및 스키마 위치를 설정합니다. 스키마는 `objectGrid.xsd` 파일에 정의됩니다. 오브젝트 그리드는 오브젝트 그리드 JAR(Java archive) 파일의 루트 디렉토리에서 이 파일을 찾습니다.

objectGrids 요소

발생 횟수: 1

하위 요소: objectGrid 요소

objectGrids 요소는 모든 objectGrid 요소의 컨테이너입니다. `sample1.xml` 파일에서 objectGrids 요소는 이름이 `bookstore`인 하나의 objectGrid를 포함합니다.

objectGrid 요소

발생 횟수: 1 - 다수

하위 요소: Bean 요소 및 backingMap 요소

XML 파일에 오브젝트 그리드를 정의하려면 objectGrid 요소를 사용하십시오. objectGrid 요소의 각 속성은 오브젝트 그리드 인터페이스의 메소드에 대응합니다.

```
<objectGrid
(1) name="objectGridName"
(2) securityEnabled="true|false"
(3) authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS|
AUTHORIZATION_MECHANISM_CUSTOM"
(4) permissionCheckPeriod="permission check period"
(5) txTimeout="seconds"
/>
```

속성:

1. **name** 속성(필수): 오브젝트 그리드에 지정되는 이름을 지정합니다. 이 속성이 누락된 경우, XML 유효성 검증에 실패합니다.

2. **securityEnabled** 속성(선택적, 기본값은 false): 이 속성을 true로 설정하면 오브젝트 그리드에 보안 사용이 가능합니다. 오브젝트 그리드 레벨의 보안을 사용 가능으로 설정한다는 것은 맵의 데이터에 대한 액세스 권한을 사용 가능하게 한다는 의미입니다. 기본적으로 보안은 사용 불가능합니다.
3. **authorizationMechanism** 속성(선택적, 기본값은 AUTHORIZATION_MECAHNISM_JAAS로 설정됨): 이 오브젝트 그리드의 권한 부여 메커니즘을 설정합니다. 이 속성을 두 개 값 중 하나(AUTHORIZATION_MECHANISM_JAAS 또는 AUTHORIZATION_MECHANISM_CUSTOM)로 설정할 수 있습니다. 사용자 정의 MapAuthorization 플러그인을 사용 중인 경우 AUTHORIZATION_MECHANISM_CUSTOM으로 설정하십시오. securityEnabled 속성이 true로 설정된 경우 이 설정이 적용됩니다.
4. **permissionCheckPeriod**(선택적, 기본값은 0으로 설정됨): 클라이언트 액세스를 허용하는 데 사용되는 권한을 검사하는 빈도를 표시하는 정수 값을 초 단위로 지정합니다. 속성 값이 0인 경우, 모든 get, put, update, remove 또는 evict 메소드 호출은 권한 부여 메커니즘(JAAS 권한 부여 또는 사용자 정의 권한 부여)에 현재 대상(subject)에 권한이 있는지 여부를 검사하도록 요청합니다. 0보다 큰 값은 새로 고치기 위해 권한 부여 메커니즘으로 리턴하기 전에 권한 세트를 캐시할 초 수를 표시합니다. securityEnabled 속성이 true로 설정된 경우 이 설정이 적용됩니다. 자세한 내용은 149 페이지의 『오브젝트 그리드 보안』을 참조하십시오.
5. **txTimeout**(선택적, 기본값은 0으로 설정됨): 트랜잭션을 완료하기 위해 허용된 시간(초). 트랜잭션이 이 시간 내에 완료되지 않으면 트랜잭션은 롤백으로 표시되고 TransactionTimeoutException 예외가 발생합니다. 값이 0으로 설정되면 트랜잭션은 제한시간을 초과하지 않습니다.

다음 XML 파일 예(bookstoreObjectGridAttr.xml 파일)는 오브젝트 그리드의 속성을 구성하기 위한 한 가지 방법을 설명합니다. 이 예에서는 보안 사용이 가능하고 권한 부여 메커니즘은 JAAS로 설정되며, 권한 검사 기간은 45초로 설정됩니다.

bookstoreObjectGridAttr.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
permissionCheckPeriod="45">
</objectGrid>
</objectGrids>
</objectGridConfig>
```

다음 코드는 위 예의 bookstoreObjectGridAttr.xml 파일과 동일한 구성을 수행하기 위한 프로그램 방식 접근을 설명합니다.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory
.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);
bookstoreGrid.setSecurityEnabled();
bookstoreGrid.setAuthorizationMechanism(
SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
bookstoreGrid.setPermissionCheckPeriod(45);
```

backingMap 요소

발생 횟수: 0 - 다수

하위 요소: 없음

backingMap 요소는 오브젝트 그리드에 BackingMap을 정의하는 데 사용됩니다. backingMap 요소의 각 속성은 BackingMap 인터페이스의 메소드에 대응합니다.

```
<backingMap
(1) name="backingMapName"
(2) readOnly="true|false"
(3) pluginCollectionRef="reference to backingMapPluginCollection"
(4) numberOfBuckets="number of buckets"
(5) preloadMode="true|false"
(6) lockStrategy="OPTIMISTIC|PESSIMISTIC|NONE"
(7) numberOfLockBuckets="number of lock buckets"
(8) lockTimeout="lock timeout"
(9) copyMode="COPY_ON_READ_AND_COMMIT|COPY_ON_READ|
COPY_ON_WRITE|NO_COPY"
(10) valueInterfaceClassName="value interface class name"
(11) copyKey="true|false"
(12) nullValuesSupported="true|false"
(13) ttlEvictorType="CREATION_TIME|LAST_ACCESS_TIME|NONE"
(14) timeToLive="time to live"
/>
```

속성:

1. **name** 속성(필수): BackingMap에 지정되는 이름을 지정합니다. 이 속성이 누락된 경우, XML 유효성 검증에 실패합니다.
2. **readOnly** 속성(선택적, 기본값은 `false`로 설정됨): 이 속성을 `true`로 설정하면 읽기 전용 BackingMap이 됩니다. 속성을 `false`로 설정하면 읽기-쓰기 BackingMap이 됩니다. 값을 지정하지 않은 경우, 기본값인 읽기-쓰기가 됩니다.
3. **pluginCollectionRef** 속성(선택적): backingMapPluginCollection 플러그인에 대한 참조를 지정합니다. 이 속성 값은 backingMapCollection 플러그인의 id 속성과 일치해야 합니다. 일치하는 ID가 없으면 유효성 검증에 실패합니다. 이 참조는 BackingMap 플러그인을 재사용할 수 있도록 쉬운 방법으로 설계됩니다.
4. **numberOfBuckets** 속성(선택적, 기본값은 `503`으로 설정됨): BackingMap이 사용할 버켓 수. BackingMap은 구현 시 해시 맵을 사용합니다. BackingMap에 많은 항목이 존재하는 경우, 버켓 수가 늘어날수록 충돌 위험이 줄어들기 때문에 성능이 향상됩니다. 또한 버켓이 많을수록 동시성이 향상됩니다.

5. **preloadMode** 속성(선택적, 기본값은 false로 설정됨): 이 BackingMap에 Loader 플러그인이 설정된 경우 사전 로드 모드를 설정합니다. 속성이 true로 설정된 경우, Loader.preloadMap(Session, BackingMap) 메소드가 비동기로 호출됩니다. 그렇지 않으면 사전 로드가 완료될 때까지 캐시를 사용할 수 없도록 데이터 로드 시 메소드 실행을 차단합니다. 사전 로드는 오브젝트 그리드 초기화 중에 발생합니다.
6. **lockStrategy** 속성(선택적, 기본값은 OPTIMISTIC으로 설정됨): BackingMap에 사용되는 LockStrategy를 설정합니다. 잠금 계획은 트랜잭션이 맵 항목에 액세스할 때마다 내부 오브젝트 그리드 잠금 관리자가 사용되지는지 여부를 결정합니다. 이 속성을 세 개 값(OPTIMISTIC, PESSIMISTIC 또는 NONE) 중 하나로 설정할 수 있습니다.

   OPTIMISTIC은 일반적으로 Loader 플러그인이 없는 맵에 사용되며, 해당 맵은 대개 읽혀지고 objectGrid를 사이드 캐시로 사용하는 지속 관리자 또는 응용프로그램이 잠금을 제공하지 않습니다. 잠금 중 변경이 예상되지 않는 계획의 경우, 확약 시 삽입, 갱신 또는 제거되는 맵 항목에 대한 독점 잠금을 얻습니다. 잠금은 확약 중인 트랜잭션이 변경이 예상되지 않는 버전화 검사를 수행 중인 동안 다른 트랜잭션에 의해 버전 정보가 변경되지 않도록 합니다.

   PESSIMISTIC은 일반적으로 Loader 플러그인이 없는 맵에 사용되며, objectGrid를 사이드 캐시로 사용하는 지속 관리자, Loader 플러그인 또는 응용프로그램이 잠금을 제공하지 않습니다. 잠금 중 변경이 예상되는 계획은 갱신 트랜잭션이 동일한 맵 항목에서 자주 충돌하기 때문에 변경이 예상되지 않는 접근이 너무 자주 실패하는 경우에 사용됩니다. 변경이 예상되지 않는 접근은 맵을 대부분 읽을 수 없거나 대다수의 클라이언트가 소형 맵에 액세스하는 경우 실패할 수 있습니다.

   NONE은 오브젝트 그리드를 사이드 캐시로 사용하는 지속 관리자 또는 데이터베이스 잠금을 사용하여 동시성을 제어하는 Loader 플러그인에 의해 동시성 제어가 오브젝트 그리드 외부에서 제공되므로 내부 LockManager를 사용하지 않아도 됨을 표시합니다.
7. **numberOfLockBuckets** 속성(선택적, 기본값은 383으로 설정됨): 이 BackingMap에 잠금 관리자가 사용하는 잠금 버켓 수를 설정합니다. lockStrategy 속성이 OPTIMISTIC 또는 PESSIMISTIC으로 설정된 경우, BackingMap에 대해 잠금 관리자가 작성됩니다. 잠금 관리자는 해시 맵을 사용하여 하나 이상의 트랜잭션에서 잠근 항목을 추적합니다. 항목이 많은 경우, 버켓 수가 증가할수록 충돌 위험은 낮아지므로 버켓이 많을수록 성능이 향상됩니다. 또한 잠금 버켓이 많을수록 동시성이 향상됩니다. lockStrategy 속성이 NONE으로 설정된 경우, 이 BackingMap은 잠금 관리자를 사용하지 않습니다. 이 경우, numberOfLockBuckets 속성을 설정하지 않아도 됩니다.
8. **lockTimeout** 속성(선택적, 기본값은 15로 설정됨): 이 BackingMap에 잠금 관리자가 사용하는 잠금 제한시간을 설정합니다. lockStrategy 속성이 OPTIMISTIC 또

는 PESSIMISTIC으로 설정된 경우, BackingMap에 대해 잠금 관리자가 작성됩니다. 교착 상태를 방지하려면 잠금 관리자에서 잠금 부여를 대기하는 기본 제한시간 값을 설정합니다. 이 제한시간을 초과하면 LockTimeoutException 예외가 발생합니다. 대부분의 응용프로그램의 경우 기본값인 15초면 충분하지만 과부하 상태의 시스템에서는 교착 상태가 없을 때 제한시간 초과가 발생할 수 있습니다. 이 경우, 잘못된 제한시간 초과 예외가 발생하지 않도록 이 메소드를 사용하여 잠금 제한시간 값을 기본값에서 늘릴 수 있습니다. 잠금 계획이 NONE으로 설정된 경우, 이 BackingMap은 잠금 관리자를 사용하지 않습니다. 이 경우, lockTimeout 속성을 설정하지 않아도 됩니다.

9. **copyMode** 속성(선택적, 기본값은 COPY_ON_READ_AND_COMMIT로 설정됨): copyMode 속성은 BackingMap에서 항목에 대한 get 조작이 실제 값, 값의 사본 또는 값에 대한 프록시를 리턴하는지 여부를 결정합니다. copyMode 속성을 네 개 값(COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE 또는 NO_COPY) 중 하나로 설정할 수 있습니다.

   COPY_ON_READ_AND_COMMIT 모드는 BackingMap에 있는 값 오브젝트에 대한 참조가 응용프로그램에 없으며, 대신 응용프로그램이 항상 BackingMap에 있는 값의 사본을 사용하여 작업 중임을 보장합니다.

   COPY_ON_READ 모드는 트랜잭션이 확약될 때 발생하는 복사를 제거하여 COPY_ON_READ_AND_COMMIT 모드에 비해 성능을 향상시킵니다. BackingMap 데이터의 무결성을 보존하기 위해 응용프로그램은 트랜잭션이 확약된 후에 항목에 대해 갖고 있는 모든 참조를 제거할 것을 약속합니다. 이 모드에서는 값에 대한 참조 대신 값의 사본을 리턴하는 ObjectMap.get 메소드가 발생하여 응용프로그램이 값에 대해 작성하는 변경사항이 트랜잭션이 확약될 때까지는 BackingMap 값에 영향을 주지 않도록 합니다.

   COPY_ON_WRITE 모드는 주어진 키에 대해 트랜잭션이 ObjectMap.get 메소드가 처음 호출될 때 발생하는 복사를 제거하여 COPY_ON_READ_AND_COMMIT 모드에 비해 성능을 향상시킵니다. 대신 ObjectMap.get 메소드는 값 오브젝트에 대한 직접 참조 대신 값에 대한 프록시를 리턴합니다. 프록시는 응용프로그램이 값 인터페이스의 set 메소드를 호출하지 않으면 값 사본이 작성되지 않도록 합니다.

   NO_COPY 모드에서 응용프로그램은 성능 향상을 위해 ObjectMap.get 메소드를 사용하여 얻은 값 오브젝트를 수정하지 않도록 약속할 수 있습니다. 이 모드를 사용하는 경우, 값은 복사되지 않습니다.

10. **valueInterfaceClassName** 속성(선택적): copyMode 속성이 COPY_ON_WRITE로 설정된 경우, valueInterfaceClassName 속성이 필요합니다. 기타 모든 모드의 경우에는 무시됩니다. Copy on write는 ObjectMap.get 메소드 호출이 이루어질 때

프록시를 사용합니다. 프록시는 응용프로그램이 valueInterfaceClassName 속성으로 지정된 클래스에서 set 메소드를 호출하지 않으면 값 사본이 작성되지 않도록 합니다.

11. **copyKey** 속성(선택적, 기본값은 `false`로 설정됨): 이 속성은 맵 항목이 작성될 때 키를 복사해야 하는지 여부를 결정합니다. 키 오브젝트를 복사하면 응용프로그램이 각 ObjectMap 조작에 동일한 키 오브젝트를 사용할 수 있습니다. `true`로 설정하면 맵 항목이 작성될 때 키 오브젝트를 복사합니다.
12. **nullValuesSupported** 속성(선택적, 기본값은 `true`로 설정됨): 널값 지원은 맵에 널값을 넣을 수 있음을 의미합니다. `true`로 설정된 경우, ObjectMap에서 널값이 지원되며 그렇지 않으면 널값이 지원되지 않습니다. 널값이 지원되는 경우, `null`을 리턴하는 `get` 조작은 값이 널이거나 맵이 전달(passed-in) 키를 포함하지 않음을 의미할 수 있습니다.
13. **ttlEvictorType** 속성(선택적, 기본값은 `NONE`으로 설정됨): ttlEvictorType 속성은 BackingMap 항목의 만기 시간 계산 방법을 결정합니다. 이 속성을 세 개 값(`CREATION_TIME`, `LAST_ACCESS_TIME` 또는 `NONE`) 중 하나로 설정할 수 있습니다.

    `NONE`은 항목에 만기 시간이 없고 응용프로그램이 명시적으로 항목을 제거하거나 무효화할 때까지 BackingMap에 존재할 수 있음을 표시합니다.

    `CREATION_TIME`은 항목 만기 시간이 항목 작성 기간과 timeToLive 속성 값의 합계임을 표시합니다.

    `LAST_ACCESS_TIME`은 항목 만기 시간이 항목의 마지막 액세스 시간과 timeToLive 속성 값의 합계임을 표시합니다.
14. **timeToLive** 속성(선택적, 기본값은 `0`으로 설정됨): 각 맵 항목의 TTL(Time to Live)로, 초 단위입니다. 기본값 `0`은 맵 항목이 영원히 또는 응용프로그램이 항목을 명시적으로 제거하거나 무효화할 때까지 존재함을 의미합니다. 속성이 `0`이 아닌 경우, TTL 축출기가 이 값을 기준으로 맵 항목을 축출하는 데 사용됩니다.

다음 XML 파일(`bookstoreBackingMapAttr.xml` 파일)은 샘플 backingMap 구성을 설명합니다. 이 예에서는 pluginCollectionRef 속성을 제외한 모든 선택적 속성을 사용합니다. pluginCollectionRef 사용 방법을 표시하는 예는 291 페이지의 『backingMapPluginCollection 요소』를 참조하십시오.

bookstoreBackingMapAttr.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" readOnly="true" numberOfBuckets="641"
```

```
preloadMode="false" lockStrategy="OPTIMISTIC"
numberOfLockBuckets="409" lockTimeout="30" copyMode="COPY_ON_WRITE"
valueInterfaceClassName=
"com.ibm.websphere.samples.objectgrid.CounterValueInterface"
copyKey="true" nullValuesSupported="false"
ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3000" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

다음 샘플 코드는 위 예의 bookstoreBackingMapAttr.xml 파일과 동일한 구성을 수행하기 위한 프로그램 방식 접근을 설명합니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
bookMap.setReadOnly(true);
bookMap.setNumberOfBuckets(641);
bookMap.setPreloadMode(false);
bookMap.setLockStrategy(LockStrategy.OPTIMISTIC);
bookMap.setNumberOfLockBuckets(409);
bookMap.setLockTimeout(30);
// when setting copy mode to COPY_ON_WRITE, a valueInterface class is required
bookMap.setCopyMode(CopyMode.COPY_ON_WRITE,
com.ibm.websphere.samples.objectgrid.CounterValueInterface.class);
bookMap.setCopyKey(true);
bookMap.setNullValuesSupported(false);
bookMap.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);
bookMap.setTimeToLive(3000); // set time to live to 50 minutes
```

Bean 요소

발생 횟수(objectGrid 요소 내에서): 0 - 다수

발생 횟수(backingMapPluginCollection 요소 내에서): 0 - 다수

하위 요소: property 요소

플러그인을 정의하려면 Bean 요소를 사용하십시오. 플러그인을 오브젝트 그리드 및 BackingMap에 첨부할 수 있습니다.

오브젝트 그리드 플러그인:

- TransactionCallback 플러그인
- ObjectGridEventListener 플러그인
- SubjectSource 플러그인
- MapAuthorization 플러그인
- SubjectValidation 플러그인

BackingMap 플러그인:

- Loader 플러그인
- ObjectTransformer 플러그인

- OptimisticCallback 플러그인
- Evictor 플러그인
- MapEventListener 플러그인
- MapIndex 플러그인

Bean 요소 속성

```
<bean
(1) id="TransactionCallback|ObjectGridEventListener|SubjectSource|
MapAuthorization|SubjectValidation|Loader|ObjectTransformer|
OptimisticCallback|Evictor|MapEventListener|MapIndexPlugin"
(2) className="class name"
/>
```

1. **id** 속성(필수): 작성할 플러그인의 유형을 지정합니다. objectGrid 요소의 하위 요소인 Bean의 경우, 유효값은 `TransactionCallback`, `ObjectGridEventListener`, `SubjectSource`, `MapAuthorization` 및 `SubjectValidation` 플러그인입니다. backingMapPluginCollection 요소의 하위 요소인 Bean의 경우, 유효값은 `Loader`, `ObjectTransformer`, `OptimisticCallback`, `Evictor` 및 `MapEventListener` 플러그인입니다. id 속성의 유효값은 각각 인터페이스를 나타냅니다.
2. **className** 속성(필수): 플러그인을 작성하기 위해 인스턴스화할 클래스의 이름을 지정합니다. 클래스는 플러그인 유형 인터페이스를 구현해야 합니다.

다음 `bean.xml` 파일 샘플은 Bean 요소를 사용하여 플러그인을 구성하는 방법을 설명합니다. 이 XML 파일에서는 ObjectGridEventListener 플러그인이 bookstore 오브젝트 그리드에 추가됩니다. 이 Bean의 className 속성은 com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener 클래스입니다. 이 클래스는 필요할 경우 com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener 인터페이스를 구현합니다.

BackingMap 플러그인도 `bookstoreBean.xml` 파일 샘플에 정의되어 있습니다. evictor 플러그인은 book BackingMap에 추가됩니다. Bean ID가 *Evictor*이기 때문에 className 속성은 com.ibm.websphere.objectgrid.plugins.Evictor 인터페이스를 구현하는 클래스를 지정해야 합니다. com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor 클래스는 이 인터페이스를 구현합니다. backingMap은 pluginCollectionRef 속성을 사용하여 플러그인을 참조합니다. BackingMap에 플러그인을 추가하는 방법에 관한 자세한 정보는 119 페이지의 『BackingMap 인터페이스』를 참조하십시오.

bookstoreBean.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<bean id="ObjectGridEventListener"
className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener" />
```

```
<backingMap name="book" pluginCollectionRef="bookPlugins" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="bookPlugins">
<bean id="Evictor"
classnName="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

다음 코드는 위 bookstoreBean.xml 파일과 동일한 구성을 수행하기 위한 프로그램 방식 접근을 설명합니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
TranPropListener tranPropListener = new TranPropListener();
bookstoreGrid.addEventListener(tranPropListener);
BackingMap bookMap = bookstoreGrid.defineMap("book");
Evictor lruEvictor = new
 com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
bookMap.setEvictor(lruEvictor);
```

property 요소

발생 횟수: 0 - 다수

하위 요소: 없음

property 요소는 플러그인에 특성을 추가하는 데 사용됩니다. 특성 이름은 특성을 포함하는 Bean의 className 속성에 대한 set 메소드에 대응합니다.

property 요소 속성

```
<property
(1) name="name"
(2) type="java.lang.String|boolean|java.lang.Boolean|int|
java.lang.Integer|double|java.lang.Double|byte|
java.lang.Byte|short|java.lang.Short|long|
java.lang.Long|float|java.lang.Float|char|
java.lang.Character"
(3) value="value"
(4) description="description"
/>
```

1. **name** 속성(필수): 특성 이름을 지정합니다. 이 속성에 지정된 값은 Bean 요소에 className 속성으로 제공된 클래스의 set 메소드에 대응해야 합니다. 예를 들어, Bean의 className 속성이 com.ibm.MyPlugin으로 설정되고 제공된 특성 이름이 size인 경우 com.ibm.MyPlugin 클래스에는 setSize 메소드가 있어야 합니다.
2. **type** 속성(필수): 특성 유형을 지정합니다. set 메소드로 전달된 매개변수의 유형이며, name 속성으로 식별됩니다. 유효값은 Java 기본 요소, 해당 java.lang 및 java.lang.String입니다. name 및 type 속성은 Bean의 className 속성에 대

한 메소드 서명과 일치해야 합니다. 예를 들어, 이름이 size이고 유형이 int이면 setSize(int) 메소드가 Bean의 className 속성으로 지정된 클래스에 존재해야 합니다.

3. **value** 속성(필수): 특성 값을 지정합니다. 이 값은 type 속성으로 지정된 유형으로 변환된 후 name 및 type 속성으로 식별되는 set 메소드 호출 시 매개변수로 사용됩니다. 이 속성 값은 어떤 방식으로든 유효성 검증이 되지 않습니다. 플러그인 구현자는 전달된 값이 유효한지 확인해야 합니다. 구현자는 매개변수가 유효하지 않은 경우 set 메소드에 IllegalArgumentException 예외를 표시할 수 있습니다.
4. **description** 속성(선택적): 특성에 대한 설명을 기록하려면 이 속성을 사용하십시오.

다음 bookstoreProperty.xml 파일은 Bean에 property 요소를 추가하는 방법을 설명합니다. 이 예에서는 이름이 maxSize이고 유형이 int인 특성이 축출기에 추가됩니다. com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor 축출기에는 setMaxSize(int) 메소드와 일치하는 메소드 서명이 있습니다. 정수 값 499가 com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor 클래스의 setMaxSize(int) 메소드로 전달됩니다.

bookstoreProperty.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" pluginCollectionRef="bookPlugins" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="bookPlugins">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
<property name="MaxSize" type="int" value="449"
description="The maximum size of the LRU Evictor" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

다음 코드는 bookstoreProperty.xml 파일과 동일한 구성을 수행합니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
LRUEvictor lruEvictor =
new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
```

```
// if the XML file were used instead,
// the property that was added would cause the following call to be made
lruEvictor.setMaxSize(449);
bookMap.setEvictor(lruEvictor);
```

backingMapPluginCollections 요소

발생 횟수: 0 - 다수

하위 요소: backingMapPluginCollection 요소

backingMapPluginCollections 요소는 모든 backingMapPluginCollection 요소의 컨테이너입니다. bookstore.xml 파일에서 backingMapPluginCollections 요소는 ID가 collection1인 하나의 backingMapPluginCollection 요소를 포함합니다.

backingMapPluginCollection 요소

발생 횟수: 0 - 다수

하위 요소: Bean 요소

backingMapPluginCollection 요소는 BackingMap 플러그인을 정의합니다. 각 backingMapPluginCollection 요소는 id 속성으로 식별됩니다. 각 backingMap 요소는 backingMap 요소의 pluginCollectionRef 속성을 사용하여 플러그인을 참조해야 합니다. 유사하게 구성된 플러그인이 있어야 하는 BackingMap이 여러 개 존재하는 경우, 각각 동일한 backingMapPluginCollection 요소를 참조할 수 있습니다.

backingMapPluginCollection 요소 속성

```
<backingMapPluginCollection
(1) id="id"
/>
```

1. **id** 속성(필수): backingMapPluginCollection의 ID. 각 ID는 고유해야 합니다. backingMap 요소의 pluginCollectionRef 속성이 ID를 참조합니다. pluginCollectionRef 속성 값이 한 backingMapPluginCollection 요소의 ID와 일치하지 않는 경우, XML 유효성 검증에 실패합니다. 임의 수의 backingMap 요소가 단일 backingMapPluginCollection 요소를 참조할 수 있습니다.

다음 bookstoreCollection.xml 파일은 backingMapPluginCollection 요소 사용 방법을 설명합니다. 이 파일에서 세 개의 backingMap 요소가 정의됩니다. book 및 customer BackingMap은 둘 다 collection1 backingMapPluginCollection을 사용합니다. 이 두 개의 BackingMap에는 각각 고유의 LRUEvictor 축출기가 있습니다. employee BackingMap은 collection2 backingMapPluginCollection을 참조합니다. 이 BackingMap에는 LFUEvictor 축출기가 Evictor 플러그인으로 설정되어 있고 EmployeeOptimisticCallbackImpl 클래스가 OptimisticCallback 플러그인으로 설정되어 있습니다.

bookstoreCollection.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" pluginCollectionRef="collection1" />
<backingMap name="customer" pluginCollectionRef="collection1" />
<backingMap name="employee" pluginCollectionRef="collection2" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
<backingMapPluginCollection id="collection2">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor" />
<bean id="OptimisticCallback"
className="com.ibm.websphere.samples.objectgrid.
EmployeeOptimisticCallBackImpl" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

다음 코드는 프로그램 방식으로 bookstoreCollection.xml 파일과 동일한 구성을 수행하는 방법을 설명합니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
LRUEvictor bookEvictor = new LRUEvictor();
bookMap.setEvictor(bookEvictor);
BackingMap customerMap = bookstoreGrid.defineMap("customer");
LRUEvictor customerEvictor = new LRUEvictor();
customerMap.setEvictor(customerEvictor);
BackingMap employeeMap = bookstoreGrid.defineMap("employee");
LFUEvictor employeeEvictor = new LFUEvictor();
employeeMap.setEvictor(employeeEvictor);
OptimisticCallback employeeOptCallback =
 new EmployeeOptimisticCallbackImpl();
employeeMap.setOptimisticCallback(employeeOptCallback);
```

혼합 모드 오브젝트 그리드 구성

XML 구성과 프로그램 방식 구성을 조합 사용하여 오브젝트 그리드를 구성할 수 있습니다.

혼합 구성을 수행하려면 먼저 ObjectGridManager 인터페이스로 전달할 XML 파일을 작성하십시오. XML 파일을 기반으로 오브젝트 그리드를 작성한 후에 ObjectGrid.initialize() 메소드가 호출되지 않는 한 오브젝트 그리드를 프로그램 방식으로 조작할

수 있습니다. 응용프로그램이 ObjectGrid.initialize() 메소드를 호출하지 않은 경우 ObjectGrid.getSession() 메소드가 내부적으로 이를 호출합니다.

예

다음은 혼합 모드 구성을 수행하는 방법을 설명합니다. 다음 mixedBookstore.xml 파일이 사용됩니다.

mixedBookstore.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/ ..objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" readOnly="true" numberOfBuckets="641"
pluginCollectionRef="bookPlugins" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="bookPlugins">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

다음 코드 스니펫은 ObjectGridManager로 전달되는 XML을 표시하며, 새로 작성된 오브젝트 그리드가 추가로 조작됩니다.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/document/mixedBookstore.xml"), true, false);
// at this point, we have the ObjectGrid that was defined in the XML
// now modify the BackingMap that was created and configured
BackingMap bookMap = bookstoreGrid.getMap("book");
// the XML set readOnly to true
// here the readOnly attribute is changed to false
bookMap.setReadOnly(false);
// the XML did not set nullValuesSupported, so
// it would default to true. Here the
// value is set to false
bookMap.setNullValuesSupported(false);
// get the Evictor that was set in the XML,
// and set its maxSize
LFUEvictor lfuEvictor = (LFUEvictor) bookMap.getEvictor();
lfuEvictor.setMaxSize(443);
bookstoreGrid.initialize();
// further configuration is not allowed
// to this ObjectGrid after the initialize call
```

분산 오브젝트 그리드 구성

분산 오브젝트 그리드를 작성하려면 클러스터 XML 파일을 작성하고 오브젝트 그리드 XML 파일과 쌍을 지정해야 합니다.

클러스터 XML 및 오브젝트 그리드 XML 파일을 사용하여 오브젝트 그리드 서버를 시작할 수 있습니다.

클러스터 XML 파일을 작성하기 전에 로컬 오브젝트 그리드에 대한 것처럼 오브젝트 그리드 XML 파일을 작성하십시오. 오브젝트 그리드 XML 파일을 구성하는 방법에 대한 자세한 정보는 279 페이지의 『로컬 오브젝트 그리드 구성』을 참조하십시오. 다음 university.xml 파일은 295 페이지의 『클러스터 구성』의 예에 대한 오브젝트 그리드 XML로 사용됩니다.

university.xml 파일

```
<?xml version="1.0" encoding="UTF-8">
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="academics">
<backingMap name="faculty" />
<backingMap name="student" />
<backingMap name="course" />
</objectGrid>
<objectGrid name="athletics">
<backingMap name="athlete" />
<backingMap name="equipment" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

다음은 오브젝트 그리드 서버를 시작하기 위해 university.xml 파일과 함께 사용할 수 있는 universityCluster.xml 클러스터 XML 파일입니다. universityCluster.xml 파일은 모든 선택적 XML 속성을 제거한 매우 기본적인 클러스터 XML 파일입니다.

universityCluster.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
</cluster>
<objectgridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
```

```
</objectgridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

많은 선택적 XML 요소 및 속성의 샘플 사용법이 『클러스터 구성』 섹션에 있습니다.

클러스터 구성

클러스터 XML의 각 요소 및 속성은 이 섹션에 설명되어 있습니다. 또한 클러스터 XML을 오브젝트 그리드 XML과 함께 사용하여 구성을 수행하는 방법을 보여주는 예도 제공되어 있습니다. `university.xml` 파일은 이러한 예의 오브젝트 그리드 XML로 사용됩니다.

clusterConfig 요소

발생 횟수: 1

하위 요소: cluster, objectgridBinding, partitionSet 및 replicationGroup 요소

clusterConfig 요소는 클러스터 XML 파일의 최상위 레벨 요소입니다. 이것은 `universityCluster.xml` 파일에 나타난 대로 파일의 맨 위에 있어야 합니다. 이 요소는 파일의 네임 스페이스 및 스키마 위치를 설정합니다. 스키마는 `objectGridCluster.xsd` 파일에 정의됩니다.

cluster 요소

발생 횟수: 1

하위 요소: serverDefinition, authenticator 및 adminAuthorization 요소

cluster 요소는 오브젝트 그리드 클러스터를 정의하는 데 사용됩니다. 클러스터의 각 서버는 cluster 요소 내에 정의됩니다. 또한 cluster 요소는 보안 및 네트워크 관련 속성을 정의하는 데 사용됩니다.

```
<cluster
(1) name="clusterName"
(2) securityEnabled="true|false"
(3) statisticsEnabled="true|false"
(4) statisticsSpec="statisticsSpecification"
(5) singleSignOnEnabled="true|false"
(6) loginSessionExpirationTime="seconds"
```

```
(7) adminAuthorizationEnabled="true|false"
(8) adminAuthorizationMechanism=""
(9) clientMaxRetries="numberOfRetries"
(10) clientMaxForwards="numberOfForwards"
(11) clientStartupRetries="numberOfRetries"
(12) clientRetryInterval="seconds"
(13) tcpConnectionTimeout="seconds"
(14) tcpMinConnections="numberOfConnections"
(15) tcpMaxConnections="numberOfConnections"
(16) tcpInactivityTimeout="seconds"
(17) tcpMaxWaitTime="seconds"
(18) peerHeartbeatInterval="seconds"
(19) peerTransportBufferSize="sizeInMBs"
(20) threadPoolMinSize="minThreads"
(21) threadPoolMaxSize="maxThreads"
(22) threadPoolInactivityTimeout="seconds"
(23) managementTimeout="seconds"
(24) threadsPerClientConnect="numberOfThreads"
/>
```

속성:

1. **name** 속성(필수): 이 속성은 클러스터에 지정되는 이름입니다. 이 속성이 누락된 경우, XML 유효성 검증에 실패합니다.
2. **securityEnabled** 속성(선택적, 기본값은 **false**로 설정됨): true로 설정될 때 클러스터에 대한 보안을 사용 가능하게 합니다. 이것이 false로 설정되면 클러스터급 보안은 사용 불가능합니다. 자세한 정보는 149 페이지의 『오브젝트 그리드 보안』을 참조하십시오.
3. **statisticsEnabled** 속성(선택적, 기본값은 **false**로 설정됨): true로 설정될 때 클러스터에 대한 보안을 사용 가능하게 합니다. 통계가 사용 가능하면 statisticsSpec 속성을 사용하여 통계 스펙을 설정합니다.
4. **statisticsSpec** 속성(선택적): 통계 스펙을 설정하는 데 사용되는 문자열을 지정합니다. 이 문자열은 수집되는 통계를 결정합니다.
5. **singleSignOnEnabled** 속성(선택적, 기본값이 false로 설정): singleSignOnEnabled 속성을 true로 설정하면 클라이언트는 서버 중 하나로부터 인증을 받은 후 모든 서버에 연결할 수 있습니다. 이 속성이 false로 설정될 때 클라이언트가 연결할 수 있으려면 클라이언트가 각 서버에서 인증되어야 합니다.
6. **loginSessionExpirationTime** 속성(선택적): 로그인 세션이 만료되기 전까지 남은 시간(초). 로그인 세션이 만료되면 클라이언트는 다시 인증해야 합니다.
7. **adminAuthorizationEnabled** 속성(선택적, 기본값은 false로 설정됨): 이 값은 관리 권한 부여를 사용 가능하게 하는 데 사용됩니다. 값이 true이면 모든 관리 타스크는 권한이 필요합니다. 사용되는 권한 메커니즘은 adminAuthorizationMechanism 속성의 값으로 지정됩니다.
8. **adminAuthorizationMechanism** 속성(선택적, 기본값이 AUTHORIZATION_MECHANISM_JAAS로 설정): 이 속성은 사용되는 권한 메

커니즘을 나타냅니다. 오브젝트 그리드는 두 가지 종류의 권한 부여 메커니즘 즉, JAAS(Java Authentication and Authorization Service) 및 사용자 정의를 지원합니다. JAAS 권한 메커니즘은 표준 JAAS 정책 기반 접근 방식을 사용합니다. 권한 부여 메커니즘으로 JAAS를 지정하려면 값을 AUTHORIZATION_MECHANISM_JAAS로 설정하십시오. 사용자 정의 권한 메커니즘은 사용자 플러그인 AdminAuthorization 구현을 사용합니다. 사용자 정의 권한 부여 메커니즘을 지정하려면 값을 AUTHORIZATION_MECHANISM_CUSTOM으로 설정하십시오. 이러한 두 개 메커니즘이 사용되는 방법에 대한 자세한 정보는 149 페이지의 『오브젝트 그리드 보안』을 참조하십시오.

9. **clientMaxRetries** 속성(선택적, 기본값이 4로 설정): 서비스를 사용할 수 없을 때 서버의 요청을 자동으로 재시도할 수 있는 최대 횟수.
10. **clientMaxForwards** 속성(선택적, 기본값이 5로 설정): 실패한 요청이 다른 서버로 전달되는 최대 횟수.
11. **clientStartupRetries** 속성(선택적, 기본값이 8로 설정): 서버 시작이 완료되기를 대기하는 중에 요청이 자동으로 재시도되는 최대 횟수. 고가용성 관리자를 시작하려면 상당한 시간이 필요하므로 이 수를 충분히 높게 설정하십시오. 숫자가 충분히 크지 않으면 서버가 완전히 시작되기 전에 제출된 클라이언트 요청이 실패합니다.
12. **clientRetryInterval** 속성(선택적, 기본값이 10으로 설정): 클라이언트 재시도 간격(초). 이것은 clientMaxRetries 및 clientStartupRetries 속성 모두에 사용됩니다.
13. **tcpConnectionTimeout** 속성(선택적, 기본값이 180으로 설정): tcpConnectionTimeout 속성은 소켓 연결 제한시간입니다. 값은 초 단위입니다.
14. **tcpMinConnections** 속성(선택적, 기본값은 2로 설정됨): 연결 풀에 대한 최소 연결 수
15. **tcpMaxConnections** 속성(선택적, 기본값은 20으로 설정됨): 연결 풀의 최대 연결 수
16. **tcpInactivityTimeout** 속성(선택적, 기본값은 무한대로 설정됨): 연결이 연결 풀에서 제거되기 전에 경과해야 하는 연결 비활성 시간(초).
17. **tcpMaxWaitTime** 속성(선택적, 기본값은 120으로 설정됨): 모든 연결이 사용 중이고 연결 수가 tcpMaxConnections 속성 값에 도달했을 때 시스템이 사용 가능한 연결을 대기하는 최대 시간(초).
18. **peerHeartbeatInterval** 속성(선택적, 기본값은 120으로 설정됨): peerHeartbeatInterval 속성은 고가용성 관리자에서 사용되는 하트비트 간격입니다. 값은 초 단위입니다.
19. **peerTransportBufferSize** 속성(선택적, 기본값은 10으로 설정됨): peerTransportBufferSize 속성은 고가용성 관리자에서 사용되는 전송 메시지를 나타냅니다. 값은 MB 단위로 지정됩니다.

20. **threadPoolMinSize** 속성(선택적, 기본값은 6으로 설정됨): 고가용성 관리자 스레드 풀의 최소 크기를 지정합니다.
21. **threadPoolMaxSize** 속성(선택적, 기본값은 20으로 설정됨): 고가용성 관리자 스레드 풀의 최대 크기를 지정합니다.
22. **threadPoolInactivityTimeout** 속성(선택적, 기본값은 6000으로 설정됨): 고가용성 관리자 스레드 풀에 대한 스레드 비활성 제한시간을 지정합니다. 제한시간 값은 초 단위입니다.
23. **managementTimeout** 속성(선택적, 기본값은 30으로 설정됨): 여러 오브젝트 그리드 MBean 함수는 클러스터의 서버로 메시지를 전송하여 정보를 수집하거나 서버에서 조작을 수행합니다. managementTimeout 값은 클라이언트가 서버에서 다시 메시지를 수신하려는 기간을 명시합니다. 클라이언트와 서버 간에 통신 문제가 있거나 서버가 사용 중이면 클라이언트는 managementTimeout 값으로 지정된 시간 동안 재시도합니다. managementTimeout 값은 초 단위로 지정됩니다.
24. **threadsPerClientConnect** 속성(선택적, 기본값은 5로 설정됨): ClientClusterContext 당 작성되는 스레드 수. ObjectGridManager 인터페이스의 connect 메소드에 대한 호출이 있을 때마다 새 ClientClusterContext가 생성됩니다.

다음 universityClusterAttr.xml 파일은 클러스터 요소의 다양한 선택적 속성을 이용하는 샘플 구성입니다. 이 예에서 보안은 사용 불가능합니다. 다양한 클라이언트, TCP, 피어 및 스레드 관련 속성도 변경됩니다. universityClusterAttr.xml은 속성에 지정할 값으로 권장되지 않습니다. 다음은 속성 값을 설정하는 방법에 대한 예입니다.

universityClusterAttr.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster" securityEnabled="false" statisticsEnabled="true"
statisticsSpec="map.all=enabled" singleSignOnEnabled="false"
loginSessionExpirationTime="1800" adminAuthorizationEnabled="false"
adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_JAAS" clientMaxRetries="2"
clientMaxForwards="2" clientStartupRetries="2" clientRetryInterval="5"
tcpConnectionTimeout="160" tcpMinConnections="2" tcpMaxConnections="15"
tcpInactivityTimeout="3600" tcpMaxWaitTime="160" peerHeartbeatInterval="130"
peerTransportBufferSize="15" threadPoolMinSize="8" threadPoolMaxSize="25"
threadPoolInactivityTimeout="6050" managementTimeout="60">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectGridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
```

```
<map ref="equipment" />
</mapSet>
</objectGridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

serverDefinition 요소

발생 횟수: 1 - 다수

하위 요소: 없음

serverDefinition 요소는 오브젝트 그리드 서버를 정의하는 데 사용됩니다. 각 서버는 고유의 JVM(Java Virtual Machine)에서 실행되고 두 개의 포트를 필요로 합니다.

속성:

```
<serverDefinition
(1) name="serverName"
(2) host="hostname"
(3) clientAccessPort="portNumber"
(4) peerAccessPort="portNumber"
(5) traceSpec="traceSpecification"
(6) systemStreamToFileEnabled="true|false"
(7) workingDirectory="logsDirectory"
/>
```

1. **name** 속성(필수): 이 속성은 서버에 지정되는 이름입니다. 이 속성이 누락된 경우, XML 유효성 검증에 실패합니다.
2. **host** 속성(필수): 서버 JVM이 실행되는 시스템의 호스트 이름. 각 시스템은 다중 오브젝트 그리드 서버를 호스트할 수 있습니다. 이 속성이 누락된 경우, XML 유효성 검증에 실패합니다.
3. **clientAccessPort** 속성(필수): 클라이언트 연결에 사용되는 서버의 포트. 이 속성이 누락된 경우, XML 유효성 검증에 실패합니다.
4. **peerAccessPort** 속성(필수): 오브젝트 그리드 서버 간 통신에 사용되는 서버의 포트. 이 속성이 누락된 경우, XML 유효성 검증에 실패합니다.
5. **traceSpec** 속성(선택적, 기본값은 *=all=disabled로 설정됨): traceSpec 속성을 설정하면 지정된 문자열을 사용하여 서버를 추적할 수 있습니다.
6. **systemStreamToFileEnabled** 속성(선택적, 기본값은 **true**로 설정됨): 이 속성이 true로 설정되면, System.out, System.err 및 추적 출력 스트림이 파일로 이동합니다. 이 속성이 false로 설정될 때, System.out은 stdout 스트림으로 이동하고 System.err은 stderr 스트림으로 이동합니다. 추적이 사용 가능하면 추적 출력은 systemStreamToFileEnabled 속성 값과 상관 없이 파일로 이동합니다.

7. **workingDirectory** 속성(선택적): workingDirectory 속성은 로그 파일이 작성되는 위치를 지정합니다. workingDirectory 속성이 지정되지 않으면 로그는 현재 디렉토리에 작성됩니다.

universityClusterServerAttr.xml 파일은 serverDefinition 속성을 사용하는 경우를 보여줍니다. 이 XML 파일에서 server1 서버는 lion.ibm.com 호스트에서 실행하도록 구성되었습니다. 포트 12501은 클라이언트가 서버에 액세스할 때 사용되고 포트 12502는 서버 간 통신에 사용됩니다. systemStreamToFileEnabled 속성이 true로 설정되어 있으므로, System.out, System.err 및 추적은 workingDirectory 속성으로 지정된 디렉토리 내 파일로 출력됩니다. 이 예에서 파일은 /objectgrid/ 디렉토리에 있습니다. traceSpec 속성은 ″ObjectGrid=all=enabled″로 설정되어 있으므로 모든 오브젝트 그리드 관련 추적은 캡처되고 파일로 출력됩니다.

universityClusterServerAttr.xml 파일

```
<?xml version="1.0" encoding="UTF-8" ?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" systemStreamToFileEnabled="true"
workingDirectory="/objectgrid/" traceSpec="ObjectGrid=all=enabled" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectGridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectGridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

objectgridBinding 요소

발생 횟수: 1 - 다수

하위 요소: mapSet 요소

objectgridBinding 요소는 오브젝트 그리드 XML의 오브젝트 그리드 요소를 클러스터 XML에 정의된 토폴로지로 바인드하는 데 사용됩니다. ref 속성에 지정된 값은 오브젝

트 그리드 XML에 있는 오브젝트 그리드 요소 중 하나의 이름 속성과 일치해야 합니다. 오브젝트 그리드 XML의 오브젝트 그리드 요소는 클러스터 XML의 한 objectgridBinding에서만 참조될 수 있습니다.

속성

```
<objectgridBinding
(1) ref="objectGridReference"
(2) minThreadPoolSize="minSize"
(3) maxThreadPoolSize="maxSize"
/>
```

1. **ref** 속성(필수): ref 속성은 오브젝트 그리드 XML 파일에 정의된 오브젝트 그리드 요소를 참조하는 데 사용됩니다. 각 objectgridBinding 요소는 오브젝트 그리드 XML의 오브젝트 그리드 요소 중 하나를 참조해야 합니다. ref 속성은 오브젝트 그리드 XML에 있는 오브젝트 그리드 요소 중 하나의 이름 속성과 일치해야 합니다.
2. **minThreadPoolSize** 속성(선택적, 기본값은 3으로 설정됨): minThreadPoolSize 속성은 각 복제 그룹 구성원에 대해 스레드 풀에서 허용되는 최소 스레드 수입니다. 스레드 수는 스레드 풀 관리자에서 제어하지만 숫자는 minThreadPoolSize 값 이하로 설정할 수 없습니다. 일반적으로 스레드가 많을수록 클라이언트는 서버에서 더욱 빨리 응답을 받을 수 있습니다. 또한 스레드가 많을수록 회선 경합도 많아집니다. 그러나, 시스템은 속도가 빠를수록 추가되는 동시 스레드를 효과적으로 처리할 수 있습니다.
3. **maxThreadPoolSize** 속성(선택적, 기본값은 10으로 설정됨): maxThreadPoolSize 속성은 각 복제 그룹 구성원에 대한 스레드 풀에서 허용되는 최대 스레드 수입니다. 스레드 수는 스레드 풀 관리자에서 제어하지만 숫자는 maxThreadPoolSize 값 이상으로 설정할 수 없습니다. 일반적으로 스레드가 많을수록 클라이언트는 서버에서 더욱 빨리 응답을 받을 수 있습니다. 또한 스레드가 많을수록 회선 경합도 많아집니다. 그러나, 시스템은 속도가 빠를수록 추가되는 동시 스레드를 효과적으로 처리할 수 있습니다.

universityClusterOGBinding.xml 파일은 objectgridBinding 요소 및 속성 사용 방법을 설명합니다. 이 예에서 하나의 objectgridBinding 요소가 정의됩니다. objectgridBinding 요소는 university.xml 파일에 정의된 "academics"를 참조합니다. "athletics" 오브젝트 그리드가 university.xml 파일에 있지만 universityClusterOGBinding.xml 파일의 athletics 오브젝트 그리드를 참조하는 objectgridBinding 요소가 없음을 주목하십시오. "athletics" 오브젝트 그리드는 universityClusterOGBinding.xml 파일에 포함되어 있지 않으므로 클러스터되지 않습니다. "academics" 오브젝트 그리드는 university.xml 파일에 있고 universityClusterOGBinding.xml 파일에서 참조되므로 이 경우에는 이 오브젝트 그리드만 작성 및 클러스터됩니다.

이 예에서는 minThreadPoolSize 및 maxThreadPoolSize 속성도 설정되어 있습니다. minThreadPoolSize 값은 2로 설정되고 maxThreadPoolSize 값은 11로 설정되어 있습니다. 각 복제 그룹 구성원의 스레드 풀 관리자는 이 오브젝트 그리드의 모든 맵에 대해 이러한 경계 내 스레드 수를 유지합니다.

universityClusterOGBinding.xml 파일

```
<?xml version="1.0" encoding="UTF-8" ?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
</cluster>
<objectgridBinding ref="academics" minThreadPoolSize="2" maxThreadPoolSize="11">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

mapSet 요소

발생 횟수: 1 - 다수

하위 요소: map 요소

mapSet 요소는 맵을 함께 그룹화하는 데 사용됩니다. mapSet 내에 있는 맵은 유사하게 파티션됩니다. 분산 오브젝트 그리드에서 각 맵은 하나의 mapSet에만 속해야 합니다.

속성

```
<mapSet
(1) name="mapSetName"
(2) partitionSetRef="partitionSetReference"
(3) synchronousReplication="true|false"
(4) replicaReadEnabled="true|false"
(5) replicaDeliveryRate="deliveryRate"
(6) compression="true|false"
/>
```

1. **name** 속성(필수): mapSet에 지정되는 이름을 지정합니다.
2. **partitionSetRef** 속성(필수): 각 mapSet는 partitionSetRef 속성을 통해 partitionSet와 연관되어야 합니다. partitionSetRef 값은 partitionSet 요소 중 하나의 이름 속

성 값과 일치해야 합니다. partitionSetRef 속성 및 이 속성에 상응하는 partitionSet를 사용하여 mapSet의 맵을 파티션합니다.

3. **synchronousReplication** 속성(선택적, 기본값은 **false**로 설정됨): 이 속성이 true로 설정되면 복제 그룹 구성원 간 복제가 동시에 발생합니다. false로 설정되면 복제는 비동기식으로 발생합니다.
4. **replicaReadEnabled** 속성(선택적, 기본값은 **false**로 설정됨): synchronousReplication 값이 false로 설정되고 replicaReadEnabled 값이 true이면 클라이언트는 복제본에서 데이터를 읽을 수 있습니다. 1차와 복제본 간에 읽기 요청을 분배하는 최상의 노력이 이루어집니다. synchronousReplication 속성이 true로 설정된 경우, replicaReadEnabled 속성이 무시됩니다.
5. **replicaDeliveryRate**(선택적, 기본값은 **1000**으로 설정됨): replicaDeliveryRate 값은 각 본제본에 전달되는 LogSequence당 최대 레코드 수를 나타냅니다.
6. **compressReplicationEnabled** 속성(선택적, 기본값은 **true**로 설정됨): compressReplicationEnabled가 true로 설정되면 복제 메시지가 압축됩니다.

universityClusterMapSet.xml 파일은 이전 XML 파일 예보다 조금 더 복잡합니다. 이 파일에서 academics 오브젝트 그리드는 두 개의 맵 세트로 분할됩니다. academicsMapSet1 맵 세트는 학부와 강의 맵을 포함합니다. 이 두 맵은 partitionSet1 partitionSet에 따라 파티션됩니다. 이러한 맵은 동일한 mapSet에 있으므로 이 맵들의 복제 설정도 동일합니다.

academics objectgridBinding 또한 academicsMapSet2 mapSet를 포함합니다. 이 mapSet는 학생 맵만 포함합니다. 학생 맵은 academicsMapSet1 mapSet의 맵과 다르게 파티션됩니다. 학생 맵은 studentPSet partitionSet에 따라 파티션됩니다. academicsMapSet2는 synchronousReplication, replicaReadEnabled, replicaDeliveryRate, 및 compressReplicationEnabled 속성을 포함하여 복제 관련 속성의 값을 명시적으로 명시하지 않았으므로 이 속성에는 기본값이 지정됩니다. 이것은 academics objectgridBinding에 포함된 두 mapSet의 동작을 달리하는 또 다른 방법입니다.

athletics objectgridBinding은 athleticsMapSet mapSet를 포함합니다. academics objectgridBinding의 academicsMapSet1 mapSet와 마찬가지로 이것은 partitionSet1 partitionSet에 따라 파티션됩니다. 이 mapSet에 대한 복제 관련 속성은 값이 명확하게 명시되지 않았으므로 기본값으로 설정됩니다.

universityClusterMapSet.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
```

```
peerAccessPort="12504" />
</cluster>
<objectgridBinding ref="academics"
<mapSet name="academicsMapSet1" partitionSetRef="partitionSet1"
synchronousReplication="true" replicaReadEnabled="true"
replicaDeliveryRate="1500" compressReplicationEnabled="true">
<map ref="faculty" />
<map ref="course" />
</mapSet>
<mapSet name="academicsMapSet2" partitionSetRef="studentPSet">
<mapRef="student" />
</mapSet>
</objectGridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetREf="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<partitionSet name="studentPSet">
<partition name="studentPartition1" replicationGroupRef="replicationGroup1" />
<partition name="studentPartition2" replicationGroupRef="replicationGroup2" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
<replicationGroup name="replicationGroup2" minReplicas="1" maxReplicas="2">
<replicationGroupMember serverRef="server1" priority="2" />
<replicationGroupMember serverRef="server2" priority="1" />
</replicationGroup>
</clusterConfig>
```

map 요소

발생 횟수: 1 - 다수

하위 요소: 없음

mapSet의 각 맵은 오브젝트 그리드 XML 파일에 정의된 backingMap 요소 중 하나를 참조합니다. 분산 오브젝트 그리드를 정의할 때 오브젝트 그리드 XML의 오브젝트 그리드 요소에 포함된 각 backingMap은 클러스터 XML의 맵에 의해 참조되어야 합니다. 분산 오브젝트 그리드의 모든 맵은 한 mapSet에만 속해야 합니다.

속성

```
<map
(1) ref="backingMapReference"
/>
```

1. **ref** 속성(필수): 오브젝트 그리드 XML의 backingMap에 대한 참조. mapSet의 각 맵은 오브젝트 그리드 XML 파일에서 backingMap을 참조해야 합니다. ref에 지정된 값은 오브젝트 그리드 XML의 backingMap 요소 중 하나의 이름 속성과 일치해야 합니다.

맵 요소의 샘플 사용법에 대해서는 `universityClusterMapSet.xml` 파일을 참조하십시오. `university.xml`에서 academics 오브젝트 그리드의 모든 backingMap은 맵에

의해 참조되고 universityClusterMapSet.xml의 한 mapSet에 의해서만 참조됩니다. athletics 오브젝트 그리드에 대해서도 동일합니다. objectgridBinding이 오브젝트 그리드 XML에서 오브젝트 그리드를 참조하지만 mapSet의 모든 맵을 포함하지 않으면 ObjectGridException 예외가 발생합니다.

partitionSet 요소

발생 횟수: 1 - 다수

하위 요소: partition

partitionSet 요소는 mapSet의 파티션을 정의하는 데 사용됩니다. mapSet의 각 맵은 partitionSet의 파티션 상에 파티션됩니다. mapSet는 mapSet 요소에 대한 partitionSetRef 속성을 가지는 partitionSet와 연관됩니다. partitionSet에 한 파티션만 정의되어 있을 때 연관된 mapSet의 맵에 포함된 데이터는 파티션되지 않습니다.

속성

```
<partition-set
(1) name="partitionSetName"
/>
```

1. **name** 속성(필수): 이 속성은 partitionSet에 이름을 지정하는 데 사용됩니다. partitionSet의 이름은 mapSet의 partitionSetRef 속성에서 참조합니다.

partitionSet의 샘플 사용법에 대해서는 universityClusterMapSet.xml 파일을 참조하십시오. universityClusterMapSet.xml에는 partitionSet1과 studentPSet의 두 개 partitionSet가 정의되어 있습니다. partitionSet1 partitionSet에는 한 개의 파티션만 정의되어 있습니다. 한 개의 파티션만 정의되어 있으므로 partitionSet1 partitionSet를 사용하는 mapSet가 파티션된 데이터를 보유합니다. universityClusterMapSet.xml 파일에는 partitionSet1 partitionSet에 따라 파티션된 두 개의 mapSet가 있습니다. mapSet 요소의 partitionSetRef를 통해 academicsMapSet1과 athleticsMapSet mapSet는 partitionSet1 partitionSet로 바인드됩니다.

studentPSet partitionSet에는 두 개의 파티션이 있습니다. 이 partitionSet를 사용하는 mapSet는 맵 데이터를 두 개의 파티션으로 나누었습니다. universityClusterMapSet.xml 파일에서 academicsMapSet2 mapSet는 studentPSet partitionSet를 사용합니다.

partition 요소

발생 횟수: 1 - 다수

하위 요소: 없음

partition 요소는 partitionSet 내에 파티션을 정의하는 데 사용됩니다. 파티션은 mapSet의 맵에 있는 데이터를 분할하는 데 사용됩니다.

속성

```
<partition
(1) name="partitionName"
(2) replicationGroupRef="replicationGroupReference"
/>
```

1. **name** 속성(필수): name 속성은 partition에 이름을 지정하는 데 사용됩니다. 파티션 이름은 partitionSet 내에서 고유해야 합니다.
2. **replicationGroupRef** 속성(필수): replicationGroupRef 속성은 replicationGroup을 partition과 연관시키는 데 사용됩니다. replicationGroupRef는 replicationGroup 요소 중 하나의 이름 속성과 일치해야 합니다.

partition 요소의 샘플 사용법에 대해서는 `universityClusterMapSet.xml` 파일을 참조하십시오. `universityClusterMapSet.xml` 파일에 여러 개의 파티션이 정의되어 있습니다. partitionSet1 partitionSet에는 partition1이라는 한 개의 파티션이 있습니다. studentPSet partitionSet에는 두 개의 파티션(studentPartition1과 studentPartition2)이 있습니다.

각 파티션은 replicationGroupRef 속성을 통해 replicationGroup과 연관됩니다. `universityClusterMapSet.xml` 파일에서 studentPartition1 파티션은 replicationGroup1 replicationGroup 상에서 복제됩니다. studentPartition2 파티션은 replicationGroup2 replicationGroup 상에서 복제됩니다.

replicationGroup 요소

발생 횟수: 1 - 다수

하위 요소: replicationGroupMember 요소

replicationGroup은 맵 또는 맵의 파티션이 복제되는 방식을 정의하는 데 사용됩니다. 맵의 파티션은 replicationGroup 내 복제 그룹 구성원 간에 복제됩니다.

속성

```
<replicationGroup
(1) name="replicationGroupName"
(2) minReplicas="minNumberOfReplicas"
(3) maxReplicas="maxNumberOfReplicas"
```

1. **name** 속성(필수): name 속성은 replicationGroup에 이름을 지정하는 데 사용됩니다.
2. **minReplicas** 속성(선택적, 기본값은 **0**으로 설정됨, replicationGroup에 한 개의 replicationGroupMember만 있음. 기본값은 **1**로 설정됨, replicationGroup에 두 개 이상의 replicationGroupMember가 있음): minReplicas 속성은 이 replicationGroup

의 맵 데이터에 대한 쓰기 액세스가 허용되기 전에 사용 가능해야 하는 replicationGroupMember 수를 나타내는 데 사용됩니다. 사용 가능한 복제본 수가 지정된 minReplicas의 수 아래로 떨어지면 맵에 대해 읽기 액세스만 허용됩니다. minReplicas가 0으로 설정되면 모든 복제본이 사용 불가능하더라도 1차에서는 쓰기 액세스가 허용됩니다.

복제를 활성화하려면 최소 두 개의 복제 그룹 구성원이 사용 가능해야 하고 minReplicas 속성이 최소 1이어야 합니다. 오브젝트 그리드 클러스터의 ″bringup″ 단계 중 replicationGroup의 동작 방식을 알아두는 것이 중요합니다. 한 서버만 시작한 후 맵 데이터를 사용 가능하게 하려면 한 개의 replicationGroupMember만으로 replicationGroup을 정의하십시오. 한 개의 replicationGroupMember만 포함한 replicationGroup에서는 데이터가 복제되지 않습니다.

다음은 minReplicas 값을 설정하는 몇 가지 규칙입니다.

```
minReplicas >= 0
minReplicas <= maxReplicas
minReplicas <= # of members in the replicationGroup -1
```

3. **maxReplicas** 속성(선택적, replicationGroup에 한 개의 replicationGroupMember만 있을 경우 기본값은 **0**으로 설정됨, replicationGroup에 두 개 이상의 replicationGroupMember가 있을 경우 기본값은 **1**로 설정됨): maxReplicas 속성은 replicationGroup에서 활성화된 최대 복제본 수를 나타냅니다. replicationGroup 내에서 복제는 지정된 maxReplicas 수가 사용 가능해지면 이 수의 복제본 간에 발생합니다. maxReplicas가 그룹의 복제 그룹 구성원 수보다 적으면 추가 구성원이 대기합니다. 즉, 복제본 중 하나가 사용 불가능하게 될 때까지 추가 구성원은 유휴 상태가 됩니다.

   다음은 maxReplicas 값을 설정하는 몇 가지 규칙입니다.

```
maxReplicas >= 0
maxReplicas >= minReplicas
```

partition 요소의 샘플 사용법에 대해서는 `universityClusterMapSet.xml` 파일을 참조하십시오. `universityClusterMapSet.xml`l에는 replicationGroup1 및 replicationGroup2와 같은 두 개의 replicationGroup이 정의되어 있습니다. replicationGroup1 replicationGroup에는 한 개의 replicationGroupMember만 있습니다. 복제에는 두 개 이상의 replicationGroupMember가 필요하므로 이 replicationGroup으로 바인드된 파티션은 복제되지 않습니다.

replicationGroup2 replicationGroup에는 두 개의 replicationGroupMembers가 있습니다. studentPSet partitionSet의 studentPartition2 파티션은 이 replicationGroup을 사용합니다. 따라서, studentPartition2 파티션은 두 개의 replicationGroupMembers 상에서 복제됩니다. 또한 replicationGroup2 replicationGroup은 minReplicas 및 maxReplicas 속성을 설정했습니다. minReplicas가 1로 설정되었기 때문에 이

replicationGroup에 포함된 맵 데이터는 1차와 최소 한 개의 복제본이 사용 가능해질 때까지 읽기 전용입니다. maxReplicas 값 1은 이 replicationGroup의 1차가 데이터를 최대 한 개의 복제본에 복제함을 나타냅니다. replicationGroup2 replicationGroup의 경우 그룹에는 두 개의 구성원만 있으므로 한 개의 복제본을 초과할 수 없습니다. 한 구성원은 1차이고 다른 한 구성원은 복제본입니다.

replicationGroupMember 요소

발생 횟수: 1 - 다수

하위 요소: 없음

replicationGroupMember 요소는 서버 정의를 참조하는 데 사용됩니다. 또한 각 replicationGroupMember 요소는 연관된 우선순위를 가집니다. 이 우선순위는 1차 서버가 되는 replicationGroupMember와 복제본이 되는 구성원을 판별할 수 있습니다.

속성

```
<replicationGroupMember
(1) serverRef="serverDefinitionReference"
(2) priority="priority"
/>
```

1. **serverRef** 속성(필수): serverRef 속성은 서버 정의를 replicationGroupMember 요소와 연관시키는 데 사용됩니다. serverRef 속성은 각 replicationGroupMember를 특정 서버와 연관시킵니다.
2. **priority** 요소(필수): 우선순위 속성은 1차가 되는 복제 그룹 구성원을 결정하는 데 사용됩니다. 우선순위 값은 1에서 복제 그룹 구성원 수까지 범위입니다(1이 가장 높은 우선순위임). 오브젝트 그리드는 각 복제 그룹 구성원에 우선순위를 지정하기 위해 최상의 노력을 기울입니다. 환경에서 금지하는 경우가 아니라면 우선순위가 1인 replicationGroupMember 요소가 1차입니다. 모든 서버와 서버의 replicationGroupMembers가 거의 동시에 사용 가능해질 경우 우선순위 설정이 부여됩니다. 그러나, 다른 replicationGroupMember 이전에 우선순위가 2인 repliationGroupMember이 오랫동안 사용 가능하면 이것이 1차가 됩니다.

   1차가 성공적으로 시작하고 일정 기간 후 실패하면 새 1차를 선택해야 합니다. 다음으로 높은 우선순위를 가지는 replicationGroupMember 요소가 새 1차가 될 수 있습니다. 그러나, 다음으로 높은 우선순위를 가지는 복제본이 복제에서 숨김 대상으로 확인되면 다른 복제본이 새 1차로 선택될 수 있습니다.

partition 요소의 샘플 사용법에 대해서는 universityClusterMapSet.xml 파일을 참조하십시오. universityClusterMapSet.xml에서 replicationGroup1 replicationGroup에는 한 개의 replicationGroupMember만 있습니다. 이러한 정의 때문에, 이 replicationGroup에는 1차만 있습니다. 이 그룹에 복제본은 없습니다. 정의된 replicationGroupMember는 serverRef 값이 명시한 대로 server1 서버에서 활성입니다.

replicationGroup2 replicationGroup에는 두 개 이상의 icationGroupMember가 있습니다. 나열된 것 중에 첫 번째 replicationGroupMember가 server1 서버에서 활성화됩니다. 이 구성원은 우선순위 2를 가집니다. 나열된 것 중에 두 번째 replicationGroupMember가 sever2 서버에서 활성화됩니다. 두 번째 구성원이 우선순위 1을 보유하므로 그룹 구성원이 거의 동시에 사용 가능해질 경우에는 이 replicationGroup의 1차가 됩니다. 나열된 첫 번째 replicationGroupMember는 우선순위 2를 보유하므로 복제본의 역할을 수행합니다.

또한 minReplicas 값이 replicationGroup2 replicationGroup에 어떻게 영향을 주는지 이해해야 합니다. server1 서버와 server2 서버가 모두 실행 중인 시나리오를 고려하십시오. 이 경우, 두 replicationGroupMembers 모두 사용 가능합니다. 따라서, minReplicas 및 maxReplicas 값은 모두 만족되고 데이터는 이 그룹의 1차와 복제본 간에 복제됩니다. server1이 사용 불가능해지면 replicationGroupMembers 중 하나가 사용 불가능해집니다. 이러한 상황에서 minReplicas 값은 더 이상 만족되지 않으므로 replicationGroup2 replicationGroup의 데이터는 읽기 전용이 됩니다.

authenticator 요소

발생 횟수: 0 - 다수

하위 요소: property 요소

authenticator 요소는 클러스터의 오브젝트 그리드 서버에 클라이언트를 인증하는 데 사용됩니다. className 속성으로 지정되는 클래스는 com.ibm.websphere.objectgrid.security.plugins.Authenticator 인터페이스를 구현합니다. 인증기는 특성을 사용하여 className 속성으로 지정된 클래스의 메소드를 호출할 수 있습니다. 특성 사용에 대한 자세한 정보는 311 페이지의 『property 요소』를 참조하십시오.

속성

```
<authenticator
(1) className="authenticatorClassName"
/>
```

1. **className** 속성(필수): className 속성은 com.ibm.websphere.objectgrid.security.plugins.Authenticator 인터페이스를 구현하는 클래스를 지정하는 데 사용됩니다. 이 클래스는 오브젝트 그리드 클러스터의 서버에 클라이언트를 인증하는 데 사용됩니다.

다음 universityClusterSecurity.xml 파일은 authenticator 요소 사용 방법을 설명합니다. 이 예에서는 com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator 클래스가 인증기로 지정됩니다. 이 클래스는 com.ibm.websphere.objectgrid.security.plugins.Authenticator 인터페이스를 구현합니다.

universityClusterSecurity.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster" securityEnabled="true"
singleSignOnEnabled="true"
loginSessionExpirationTime="1800" adminAuthorizationEnabled="true"
adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
<serverDefinition name="server1" host="lion.ibm.com"
clientAccessPort="12501" peerAccessPort="12502" />
<authenticator
className ="com.ibm.websphere.objectgrid.security.plugins.builtins.
KeyStoreLoginAuthenticator" />
<adminAuthorization className= "com.ibm.MyAdminAuthorization">
<property name="interval" type="int" value="60" description="Set the
interval to 60 seconds" />
</adminAuthorization>
</cluster>
<objectgridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

adminAuthorization 요소

발생 횟수: 0 - 다수

하위 요소: property 요소

adminAuthorization 요소는 오브젝트 그리드 클러스터에 대한 관리 액세스를 설정하는 데 사용됩니다. 관리 액세스가 제공된 후 관리 타스크를 수행할 수 있습니다.

속성

```
<adminAuthorization
(1) className="adminAuthClassName"
/>
```

1. **className** 속성(필수): className 속성은 com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization 인터페이스를 구현하는 클래스를 지정하는 데 사용됩니다.

adminAuthorization 요소의 샘플 사용법에 대해서는 인증기 섹션의 universityClusterSecurity.xml 파일을 참조하십시오. universityClusterSecurity.xml에서 사용자 정의 adminAuthorization이 사용됩니다. com.ibm.MyAdminAuthorization 클래스가 adminAuthorization 클래스로 사용됩니다. 사용자 정의 adminAuthorization을 사용하려면 securityEnabled 속성이 true여야 하고 adminAuthorizationMechanism이 AUTHORIZATION_MECHANISM_CUSTOM으로 설정되고 adminAuthorization 요소가 제공되어야 합니다. 또한 이 adminAuthorization 요소는 특성을 사용합니다. 특성 사용 방법에 대한 자세한 정보는 『property 요소』를 참조하십시오.

property 요소

발생 횟수: 0 - 다수

하위 요소: 없음

property 요소는 authenticator 및 adminAuthorization에 대한 set 메소드를 호출하는 데 사용됩니다. 특성 이름은 특성을 포함하는 authenticator 또는 adminAuthorization 요소의 className에 대한 set 메소드에 대응합니다.

속성

```
<property
(1) name="propertyName"
(2) type="java.lang.String|boolean|java.lang.Boolean|int|
java.lang.Integer|double|java.lang.Double|byte|
java.lang.Byte|short|java.lang.Short|long|
java.lang.Long|float|java.lang.Float|char|
java.lang.Character"
(3) value="propertyValue"
(4) description="description"
/>
```

1. **name** 속성(필수): 특성의 이름. 이 속성에 지정된 값은 authenticator 또는 adminAuthorization의 className으로 제공된 클래스에 대한 set 메소드에 대응합니다. 예를 들어, authenticator의 className이 com.ibm.MyAuthenticator로 설정되고 제공된 특성 이름이 interval인 경우, com.ibm.MyAuthenticator 클래스에는 setInterval 메소드가 있어야 합니다.
2. **type** 속성(필수): 특성의 유형. set 메소드로 전달된 매개변수의 유형이며, name 속성으로 식별됩니다. 유효값은 Java 기본 요소, 해당 java.lang counterparts 및 java.lang.String입니다. name 및 type은 Bean의 className에 대한 메소드 서명과 일치해야 합니다. 예를 들어, 이름이 interval이고 유형이 int이면, setInterval(int) 메소드가 authenticator 또는 adminAuthorization의 className으로 지정된 클래스에 존재해야 합니다.

3. **value** 속성(필수): 특성의 값. 이 값은 type 속성으로 지정된 유형으로 변환된 후 name 및 type 속성으로 식별되는 set 메소드 호출 시 매개변수로 사용됩니다. 이 속성 값이 어떤 방식으로든 유효성 검증이 되지 않는다는 것을 알아야 합니다. 플러그인 구현자는 전달된 값이 유효한지 확인해야 합니다. 구현자는 매개변수가 유효하지 않은 경우 set 메소드에 `IllegalArgumentException` 예외를 표시할 수 있습니다.
4. **description** 속성(선택적): 특성에 대한 설명을 기록하려면 이 속성을 사용하십시오.

adminAuthorization 요소의 샘플 사용법에 대해서는 `universityClusterSecurity.xml` 파일을 참조하십시오. property 요소는 클러스터 XML의 authenticator 또는 adminAuthorization 요소 내에서 사용될 수 있습니다. `universityClusterSecurity.xml` 파일에서 특성은 adminAuthorization에서 set 메소드를 호출하는 데 사용됩니다. 이 경우, setInterval 메소드는 com.ibm.MyAdminAuthorization 클래스에서 호출됩니다. 정수 값 `60`이 전달되었습니다.

제 10 장 WebSphere Application Server에서 오브젝트 그리드 통합

WebSphere Application Server에서 제공하는 기능과 함께 오브젝트 그리드를 사용하여 오브젝트 그리드 성능을 통해 응용프로그램을 확장하십시오.

WebSphere Application Server 및 WebSphere Extended Deployment를 설치하십시오. WebSphere Extended Deployment를 설치하면 J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램에 오브젝트 그리드 기능을 추가할 수 있습니다.

오브젝트 그리드 API는 WebSphere Application Server 대상 J2EE 응용프로그램에서 사용할 수 있습니다. WebSphere Extended Deployment를 설치하면 wsobjectgrid.jar 파일은 ₩base₩lib 디렉토리에 있습니다. 오브젝트 그리드 API를 J2EE 응용프로그램 프로그래밍 모델에 통합하는 것 이외에도 분산 트랜잭션 전달 지원을 활용할 수 있습니다. 이 지원을 사용하면 WebSphere Application Server 클러스터에서 트랜잭션 확약 결과를 조정하도록 오브젝트 그리드 인스턴스를 구성할 수 있습니다.

1. 오브젝트 그리드를 사용하여 J2EE 응용프로그램을 사용 가능하게 하는 기본 프로그래밍 단계를 수행하십시오. 자세한 정보는 314 페이지의 『J2EE(Java 2 Platform, Enterprise Edition) 환경에서 오브젝트 그리드 통합』을 참조하십시오.
2. 오브젝트 그리드 응용프로그램의 성능 데이터를 모니터하십시오. 자세한 정보는 318 페이지의 『WebSphere Application Server PMI(Performance Monitoring Infrastructure)에서 오브젝트 그리드 성능 모니터링』을 참조하십시오.
3. 오브젝트 그리드가 임베드된 경우 외부 트랜잭션 조정자에서 트랜잭션을 시작 및 종료할 수 있습니다. 자세한 정보는 326 페이지의 『오브젝트 그리드 및 외부 트랜잭션 상호 작용』을 참조하십시오.
4. 오브젝트 그리드를 통해 파티션 기능을 사용하십시오. 오브젝트 그리드 기능에서는 트랜잭션 방식의 캐싱 키 용량 및 값 쌍 용량을 제공하고 파티션 기능에서는 오브젝트 특성에 따라 컨텍스트 기반 라우팅 용량을 제공합니다. 자세한 정보는 330 페이지의 『오브젝트 그리드 및 파티션 기능 통합』을 참조하십시오.
5. WebSphere Application Server 버전 6.0.2 이후에서는 CMP(Container-Managed Persistence) Bean을 사용할 수 있으므로 오브젝트그리드를 내장 캐시 대신 외부 캐시로 사용할 수 있습니다. 자세한 정보는 354 페이지의 『컨테이너 관리 Bean에 대해 작업하도록 오브젝트 그리드 구성』을 참조하십시오.

또한 플랫폼이 혼합된 환경 또는 서로 다른 층에 변경사항을 분배하기 위해 JMS에서 오브젝트 그리드를 사용할 수 있습니다. 자세한 정보는 373 페이지의 『트랜잭션 변경사항을 분배하기 위한 Java Message Service』를 참조하십시오.

J2EE(Java 2 Platform, Enterprise Edition) 환경에서 오브젝트 그리드 통합

오브젝트 그리드는 J2EE(Java 2 Platform, Enterprise Edition) 환경에서 Servlet 및 EJB(Enterprise JavaBeans) 프로그래밍 모델 모두를 지원합니다.

이 주제에서는 오브젝트 그리드를 사용하여 J2EE 응용프로그램을 사용 가능하게 하는 공통 프로그래밍 단계를 탐색합니다.

로컬 오브젝트 그리드 시나리오

1. 오브젝트 그리드 구성을 정의하십시오. XML 파일, 프로그램 방식의 인터페이스 또는 XML 파일 및 프로그램 방식의 구성을 혼합한 사용법을 통해 오브젝트 그리드 구성을 정의하십시오. 자세한 정보는 오브젝트 그리드 구성을 참조하십시오.
2. URL 오브젝트를 작성하십시오. 오브젝트 그리드 구성이 XML 파일 형식인 경우 해당 XML 파일을 지시하는 URL 오브젝트를 작성하십시오. 이 URL 오브젝트를 사용하여 ObjectGridManager API를 통해 오브젝트 그리드 인스턴스를 작성할 수 있습니다. 오브젝트 그리드 구성 XML 파일이 웹 아카이브(WAR) 또는 EJB(Enterprise JavaBeans) Java 아카이브(JAR) 파일에 포함된 경우 웹 및 EJB 모듈 모두에서 클래스 로더에 대한 자원으로 액세스 가능합니다. 예를 들어 오브젝트 그리드 구성 XML 파일이 웹 모듈 WAR 파일의 WEB-INF 폴더에 있는 경우 해당 WAR 파일에 있는 Servlet에서 다음 패턴으로 URL 오브젝트를 작성할 수 있습니다.

   ```
   URL url =className.class.getClassLoader().
   getResource("META-INF/objectgrid-definition.xml");
   URL objectgridUrl = ObjectGridCreationServlet.class.getClassLoader().
   getResource("WEB-INF/objectgrid-definition.xml");
   ```

3. 오브젝트 그리드 인스턴스를 작성 또는 확보하십시오. ObjectGridManager API를 사용하여 오브젝트 그리드 인스턴스를 확보 및 작성하십시오. ObjectGridManager API를 사용하면 오브젝트 그리드 인스턴스를 XML로 작성하고 유틸리티 메소드를 사용하여 단순 오브젝트 그리드 인스턴스를 쉽게 작성할 수 있습니다. 응용프로그램에서는 ObjectGridManager API에 대한 참조를 확보하도록 ObjectGridManagerFactory API를 사용해야 합니다. 다음 코드 예를 참조하십시오.

   ```
   import com.ibm.websphere.objectgrid.ObjectGridManager;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory ;
   ...
   ObjectGridManager objectGridManager = ObjectGridManagerFactory.
   getObjectGridManager();
   ObjectGrid ivObjectGrid = objectGridManager.
   createObjectGrid(objectGridName, objectgridUrl, true, true);
   ```

   ObjectGridManager API에 대한 자세한 정보는 ObjectGridManager 인터페이스 주제를 참조하십시오.

4. 오브젝트 그리드 인스턴스를 초기화하십시오. 오브젝트 그리드 인터페이스에서 initialize 메소드를 사용하여 오브젝트 그리드 및 Session 인스턴스의 부트스트랩을 시작하십시오. 이 initialize 메소드는 getSession 메소드의 첫 번째 호출에서 암시적 초기화를 수행하므로 선택사항으로 간주됩니다. 이 메소드를 호출한 후 오브젝트 그리드 구성은 완료된 것으로 간주되어 런타임 사용 준비가 되었습니다. 추가 구성 메소드 호출(예: defineMap(String mapName) 메소드 호출)은 예외를 발생시킵니다.

5. Session 및 ObjectMap 인스턴스를 확보하십시오. 세션은 ObjectMap 인스턴스의 컨테이너입니다. 스레드는 오브젝트 그리드 코어와 상호 작용하도록 고유한 Session 오브젝트를 확보해야 합니다. 이 기술은 한 번에 단일 스레드에서만 사용할 수 있는 세션으로 생각할 수 있습니다. 세션은 한 번에 하나의 스레드만 사용하는 경우 스레드에서 공유 가능합니다. 그러나 J2EE 연결 또는 트랜잭션 하부 구조를 사용하는 경우 세션 오브젝트는 스레드에서 공유 불가능합니다. 이 오브젝트와 유사한 항목으로 데이터베이스에 대한 JDBC(Java Database Connectivity) 연결이 있습니다.

   ObjectMap 맵은 이름 지정된 맵에 대한 핸들입니다. 맵에는 동종의 키 및 값이 있어야 합니다. ObjectMap 인스턴스는 이 ObjectMap 인스턴스를 확보할 때 사용한 세션과 현재 연관된 스레드에서만 사용할 수 있습니다. 다중 스레드는 Session 및 ObjectMap 오브젝트를 동시에 공유할 수 없습니다. 키워드는 트랜잭션에서 적용됩니다. 트랜잭션 롤백은 이 트랜잭션 중 적용된 모든 키워드 연관을 롤백합니다.코드 예는 다음과 같습니다.

   ```
   Session ivSession = ivObjectGrid.getSession();
   ObjectMap ivEmpMap = ivSession.getMap("employees");
   ObjectMap ivOfficeMap = ivSession.getMap("offices");
   ObjectMap ivSiteMap = ivSession.getMap("sites");
   ObjectMap ivCounterMap = ivSession.getMap("counters");
   ```

6. 세션을 시작하고 오브젝트를 읽거나 작성한 후 세션을 확약하거나 롤백하십시오. 맵 조작은 트랜잭션 컨텍스트에서 수행되어야 합니다. Session 오브젝트의 begin 메소드는 명시적인 트랜잭션 방식의 컨텍스트를 시작할 때 사용됩니다. 세션을 시작하면 응용프로그램이 맵 조작 수행을 시작할 수 있습니다. 가장 일반적인 조작으로는 맵의 오브젝트에 대한 get, update, insert 및 remove 메소드 호출이 있습니다. 맵 조작 종료 시 Session 오브젝트의 commit 또는 rollback 메소드가 명시적인 트랜잭션 방식의 컨텍스트를 확약하거나 명시적인 트랜잭션 방식의 컨텍스트를 롤백하도록 호출됩니다. 프로그래밍 예는 다음과 같습니다.

   ```
   ivSession.begin();
   Integer key = new Integer(1);
   if (ivCounterMap.containsKey(key) == false) {
    ivCounterMap.insert(key, new Counter(10));
   }
   ivSession.commit();
   ```

분산 오브젝트 그리드 시나리오

이 분산 오브젝트 그리드 시나리오는 오브젝트 그리드 인스턴스를 가져오는 방법에서만 로컬 오브젝트 그리드 시나리오와 다릅니다. 다음 코드 예는 분산 오브젝트 그리드 인스턴스를 가져오는 방법을 나타냅니다.

```
//Use ObjectGridManagerFactory to get the reference to the ObjectGridManager API
ObjectGridManager objectGridManager = ObjectGridManagerFactory.
getObjectGridManager();
//Obtain the ClientClusterContext represents the ObjectGrid cluster from
//ObjectGridManager
//Assuming the WebSphere server also host an ObjectGrid server
//that is a member of the ObjectGrid cluster.
ClientClusterContext context = objectGridManager.connect(null, null);
//Get the ObejctGrid instance from the ObjectGridManager a specific
//ClientClusterContext.
ObjectGrid ivObjectGrid= objectGridManager.getObjectGrid(context,
"objectgridName");
```

오브젝트 그리드를 사용하여 J2EE 응용프로그램을 사용 가능하게 하는 기본 프로그래밍 단계를 수행했습니다.

자세한 정보는 오브젝트 그리드 사용 J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램 빌드 및 J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램 및 오브젝트 그리드 통합 시 고려사항을 참조하십시오.

오브젝트 그리드 사용 J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램 빌드

이 타스크를 참고하여 오브젝트 그리드 사용 J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램의 빌드 경로 또는 클래스 경로를 구성하십시오. 클래스 경로는 `$install_root/lib` 디렉토리에 있는 `wsobjectgrid.jar` 파일을 포함해야 합니다.

오브젝트 그리드 사용 J2EE 응용프로그램을 개발하십시오. 자세한 정보는 J2EE 환경에서 오브젝트 그리드 통합을 참조하십시오.

이 타스크에서는 IBM Rational Software Development Platform 버전 6.0에서 `wsobjectgrid.jar` 파일을 포함하도록 빌드 경로를 설정하는 방법을 보여줍니다.

1. J2EE Perspective의 프로젝트 탐색기 보기에서 **웹** 또는 **EJB**(Enterprise JavaBeans) 프로젝트를 마우스 오른쪽 단추로 클릭한 후 **특성**을 선택하십시오. 특성 창이 표시됩니다.
2. 왼쪽 패널에서 **Java 빌드 경로**를 선택하고 오른쪽 패널에서 **라이브러리** 탭을 클릭한 후 **변수 추가**를 클릭하십시오. **새 변수 클래스 경로 항목** 창이 표시됩니다.
3. **변수 구성**을 클릭하여 **환경 설정** 창을 여십시오.
4. 새 변수 항목을 추가하십시오.
   a. **새로 작성**을 클릭하십시오.

b. **이름** 필드에 OBJECTGRID_JAR을 입력하십시오. **파일**을 클릭하여 **JAR 선택사항** 창을 여십시오.

c. /lib 디렉토리를 찾아보고 **wsobjectgrid.jar** 파일을 클릭한 후 **열기**를 클릭하여 **JAR 선택사항** 창을 닫으십시오.

d. **확인**을 클릭하여 **새 변수 항목** 창을 닫으십시오.

OBJECTGRID_JAR 변수가 클래스 경로 변수 목록에 표시됩니다.

5. **확인**을 클릭하여 **환경 설정** 창을 닫으십시오.
6. 변수 목록에서 OBJECTGRID_JAR 변수를 선택하고 **확인**을 클릭하여 **새 변수 클래스 경로 항목** 창을 닫으시십시오. OBJECTGRID_JAR 변수가 **라이브러리** 패널에 표시됩니다.
7. **확인**을 클릭하여 **특성** 창을 닫으십시오.

IBM Rational Software Development Platform 버전 6.0에서 wsobjectgrid.jar 파일을 포함하도록 빌드 경로를 설정했습니다.

J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램 및 오브젝트 그리드 통합 시 고려사항

J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램을 오브젝트 그리드와 통합할 때 다음 고려사항을 사용하십시오.

시작 Bean 및 오브젝트 그리드

응용프로그램에서 시작 Bean을 사용하여 응용프로그램을 시작할 때 오브젝트 그리드 인스턴스를 부트스트랩하고 응용프로그램을 중지할 때 오브젝트 그리드 인스턴스를 제거할 수 있습니다. 시작 Bean은 com.ibm.websphere.startupservice.AppStartUpHome 원격 홈 및 com.ibm.websphere.startupservice.AppStartUp 원격 인터페이스를 포함하는 Stateless 세션 Bean입니다. WebSphere Application Server는 EJB(Enterprise JavaBean)를 보면 시작 Bean으로 인식합니다. 원격 인터페이스에는 start 메소드 및 stop 메소드와 같은 두 개의 메소드가 있습니다. start 메소드를 사용하여 그리드를 부트스트랩하고 stop 메소드를 사용하여 grid destroy 메소드를 호출하십시오. 응용프로그램은 필요한 경우 참조를 확보하도록 ObjectGridManager.getObjectGrid 메소드를 사용하여 그리드에 대한 참조를 보관할 수 있습니다. 자세한 정보는 ObjectGridManager 인터페이스 주제를 참조하십시오.

클래스 로더 및 오브젝트 그리드 인스턴스

서로 다른 클래스 로더를 사용하는 응용프로그램 모듈 사이에서 단일 오브젝트 그리드 인스턴스를 공유하는 경우 주의해야 합니다. 서로 다른 클래스 로더를 사용하는 응용프로그램 모듈은 작동하지 않으므로 응용프로그램에서 클래스 캐스트 예외가 발생합니다. 오브젝트 그리드는 동일한 클래스 로더를 사용하는 응용프로그램 모듈에서만 또는 응

용프로그램 오브젝트(예: 플러그인, 키 및 값)이 공통 클래스 로더에 있는 경우에만 공유되어야 합니다.

Servlet에서 오브젝트 그리드 인스턴스의 라이프 사이클 관리

Servlet의 init 메소드 및 destroy 메소드를 사용하여 오브젝트 그리드 인스턴스의 라이프 사이클을 관리할 수 있습니다. init 메소드를 사용하여 응용프로그램에 필요한 오브젝트 그리드 인스턴스를 작성 및 초기화하십시오. 오브젝트 그리드 인스턴스를 작성 및 캐시하면 ObjectGridManager API에서 이름별로 인스턴스를 얻을 수 있습니다. destroy 메소드를 사용하여 이 오브젝트 그리드 인스턴스를 제거한 후 시스템 자원을 해제하십시오. 자세한 정보는 ObjectGridManager 인터페이스 주제를 참조하십시오.

WebSphere Application Server PMI(Performance Monitoring Infrastructure)에서 오브젝트 그리드 성능 모니터링

오브젝트 그리드는 WebSphere Application Server 또는 WebSphere Extended Deployment Application Server에서 실행될 때 PMI(Performance Monitoring Infrastructure)를 지원합니다. PMI는 런타임 응용프로그램에서 성능 데이터를 수집하고 성능 데이터를 모니터하도록 외부 응용프로그램을 지원하는 인터페이스를 제공합니다.

오브젝트 그리드에서 제공하는 통계에 대한 자세한 정보는 오브젝트 그리드 통계를 참조하십시오.

오브젝트 그리드는 WebSphere Application Server의 사용자 정의 PMI 기능을 사용하여 고유한 PMI 기능을 추가합니다. 이 접근 방식을 통해 관리 콘솔 또는 JMX(Java Management Extension) 인터페이스를 사용하여 오브젝트 그리드 PMI를 사용 가능 및 사용 불가능하게 할 수 있습니다. 또한 Tivoli Performance Viewer를 포함하여 모니터링 도구에서 사용하는 표준 PMI 및 JMX 인터페이스를 통해 오브젝트 그리드 통계에 액세스할 수 있습니다.

1. 오브젝트 그리드 PMI를 사용 가능하게 하십시오. PMI 통계를 보려면 PMI가 사용 가능해야 합니다. 자세한 정보는 322 페이지의 『오브젝트 그리드 PMI 사용 가능』을 참조하십시오.
2. 오브젝트 그리드 PMI 통계를 검색하십시오. Tivoli Performance Viewer를 통해 오브젝트 그리드 응용프로그램의 성능을 보십시오. 자세한 정보는 325 페이지의 『오브젝트 그리드 PMI 통계 검색』을 참조하십시오.

오브젝트 그리드 통계

오브젝트 그리드에서는 objectGridModule 모듈 및 mapModule 모듈과 같은 두 개의 PMI(Performance Monitoring Infrastructure) 모듈을 제공합니다.

objectGridModule 모듈

objectGridModule 모듈에는 하나의 시간 통계(트랜잭션 응답 시간)가 들어 있습니다. 오브젝트 그리드 트랜잭션은 Session.begin 메소드 호출 및 Session.commit 메소드 호출 사이의 지속 기간으로 정의됩니다. 이 지속 기간은 트랜잭션 응답 시간으로 추적됩니다.

objectGridModule 모듈의 루트 요소인 ObjectGrids 요소는 오브젝트 그리드 통계의 시작점으로 처리됩니다. 이 루트 요소에는 하위로 트랜잭션 유형이 들어 있는 오브젝트 그리드 인스턴스가 하위로 들어 있습니다. 응답 시간 통계를 각 트랜잭션 유형과 연관됩니다. objectGridModule 모듈 구조가 다음 다이어그램에 표시됩니다.

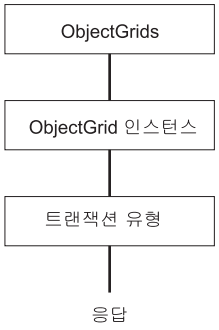


그림 19. ObjectGridModule 모듈 구조

다음 다이어그램에서는 ObjectGrid PMI 모듈 구조에 대한 예를 표시합니다. 이 예에서 시스템에는 objectGrid1 오브젝트 그리드 및 objectGrid2 오브젝트 그리드와 같은 두 개의 오브젝트 그리드 인스턴스가 있습니다. objectGrid1 인스턴스에는 갱신 및 읽기와 같은 두 개의 트랜잭션 유형이 있으며 objectGrid2 인스턴스에는 갱신과 같은 하나의 트랜잭션 유형이 있습니다.

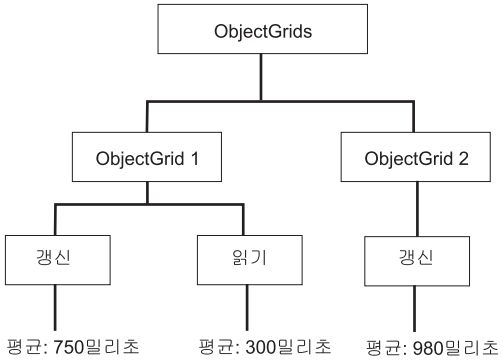


그림 20. 오브젝트 그리드 PMI 모듈 구조

응용프로그램 개발자는 응용프로그램에서 사용하는 트랜잭션 유형을 알고 있으므로 응용프로그램 개발자가 트랜잭션 유형을 정의합니다. 트랜잭션 유형은 다음 Session.setTransactionType(String) 메소드를 사용하여 설정됩니다.

```
/**
* Sets the transaction type for future transactions.
*
* After this method is called, all of the future transactions have the
* same type until another transaction type is set. If  no transaction
* type is set, the default  TRANSACTION_TYPE_DEFAULT transaction type
* is used.
*
* Transaction types are used mainly for statistical data tracking purpose.
* Users can predefine types of transactions that run in an
* application. The idea is to categorize transactions with the same characteristics
* to one category (type), so one transaction response time statistic can be
* used to track each transaction type.
*
* This tracking is useful when your application has different types of
* transactions.
* Among them, some types of transactions, such as update transactions, process
* longer than other transactions, such as read-only transactions. By using the
* transaction type, different transactions are tracked by different statistics,
* so the statistics can be more useful.
*
* @param tranType the transaction type for future transactions.
*/
void setTransactionType(String tranType);
```

다음 예에서는 트랜잭션 유형을 updatePrice로 설정합니다.

```
// Set the transaction type to updatePrice
// The time between session.begin() and session.commit() will be
// tracked in the time statistic for "updatePrice".
session.setTransactionType("updatePrice");
session.begin();
map.update(stockId, new Integer(100));
session.commit();
```

첫 번째 행에서는 후속 트랜잭션 유형이 updatePrice임을 표시합니다. updatePrice 통계는 예의 세션에 해당하는 오브젝트 그리드 인스턴스 아래 있습니다. JMX(Java Management Extensions) 인터페이스를 사용하여 updatePrice 트랜잭션의 트랜잭션 응답 시간을 확보할 수 있습니다. 또한 지정된 오브젝트 그리드의 모든 트랜잭션 유형에서 수집한 통계를 확보할 수 있습니다.

mapModule 모듈

mapModule PMI 모듈에는 오브젝트 그리드 맵과 관련된 세 개의 통계가 있습니다.

- **맵 히트 비율:** 이 BoundedRangeStatistic 통계에서는 맵의 히트 비율을 추적합니다. 히트 비율은 0 - 100 사이(경계값 포함)의 float 값으로, 맵의 get 조작과 관련된 맵 히트 백분율을 표시합니다.
- **항목 수**: 이 CountStatistic 통계는 맵에 있는 항목 수를 추적합니다.
- **로더 일괄처리 갱신 응답 시간:** 이 TimeStatistic 통계는 로더 일괄처리 갱신 조작에 사용되는 응답 시간을 추적합니다.

mapModule 모듈의 루트 요소인 오브젝트 그리드 Map 요소는 오브젝트 그리드 Map 통계의 시작점으로 처리됩니다. 이 루트 요소에는 하위로 Map 인스턴스가 들어 있는 오브젝트 그리드 인스턴스가 하위로 들어 있습니다. 모든 Map 인스턴스에는 세 개의 나열된 통계가 있습니다. mapModule 구조가 다음 다이어그램에 표시됩니다.

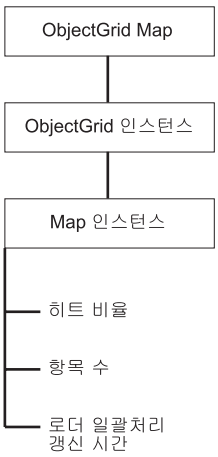


그림 21. mapModule 모듈 구조

다음 다이어그램에서는 mapModule 구조에 대한 예를 표시합니다.

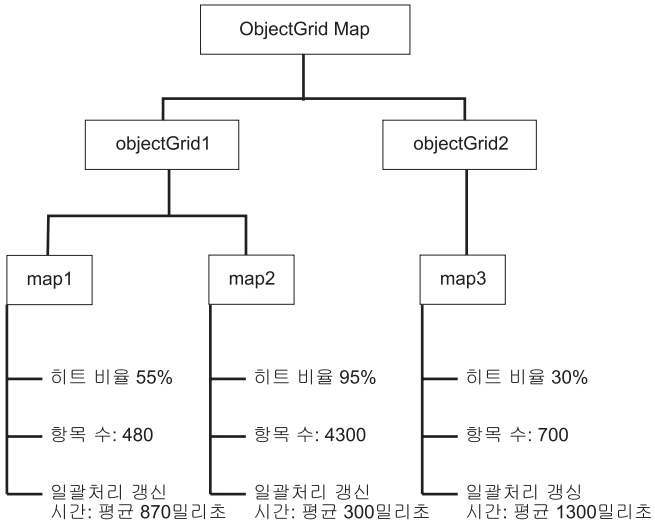


그림 22. mapModule 모듈 구조 예

오브젝트 그리드 PMI 사용 가능

WebSphere Application Server PMI(Performance Monitoring Infrastructure)를 사용하여 모든 레벨에서 통계를 사용 가능 또는 사용 불가능하게 할 수 있습니다. 예를 들어 항목 통계 수 또는 로더 일괄처리 갱신 시간 통계가 아닌 특정 맵의 맵 히트 비율 통계만 선택할 수 있습니다. 이 주제에서는 관리 콘솔 및 wsadmin 스크립트를 사용하여 오브젝트 그리드 PMI를 사용 가능하게 하는 방법을 표시합니다.

WebSphere Application Server PMI를 사용하여 모든 레벨에서 통계를 사용 가능 또는 사용 불가능하게 할 수 있는 세분화 메커니즘을 제공하십시오. 예를 들어 항목 통계 수 또는 로더 일괄처리 갱신 시간 통계가 아닌 특정 맵의 맵 히트 비율 통계를 선택할 수 있습니다. 이 섹션에서는 관리 콘솔 및 wsadmin 스크립트를 사용하여 오브젝트 그리드 PMI를 사용 가능하게 하는 방법을 표시합니다.

1. 관리 콘솔(예: http://localhost:9060/ibm/console)을 여십시오.
2. **모니터링 및 조정 > PMI(Performance Monitoring Infrastructure) > *server_name***을 클릭하십시오.
3. **PMI(Performance Monitoring Infrastructure) 사용 가능**을 선택했는지 확인하십시오. 이 설정은 기본적으로 사용 가능합니다. 설정이 사용 가능하지 않으면 선택란을 선택한 후 서버를 다시 시작하십시오.
4. **사용자 정의**를 클릭하십시오. 구성 트리에서 **오브젝트 그리드** 및 **오브젝트 그리드 Map** 모듈을 선택하십시오. 각 모듈의 통계를 사용 가능하게 하십시오.

오브젝트 그리드 통계의 트랜잭션 유형 카테고리는 런타임에 작성됩니다. 런타임 패널에서는 오브젝트 그리드 및 Map 통계의 하위 카테고리만 볼 수 있습니다.

예를 들어 다음 단계를 수행하여 샘플 응용프로그램에서 PMI 통계를 사용 가능하게 할 수 있습니다.

1. `http://host:port/ObjectGridSample`을 웹 주소로 사용하여 응용프로그램을 실행하십시오. 여기서 host 및 port는 샘플이 설치된 서버의 호스트 이름 및 HTTP 포트 번호입니다.
2. 샘플 응용프로그램에서 **ObjectGridCreationServlet**을 클릭한 후 조치 단추 1, 2, 3, 4 및 5를 클릭하여 오브젝트 그리드 및 맵에서 일부 조치를 생성하십시오. 현재 이 Servlet 페이지를 닫지 마십시오.
3. 관리 콘솔로 돌아가 **모니터링 및 조정 > PMI(Performance Monitoring Infrastructure) > *server_name***을 클릭하십시오. **런타임** 탭을 클릭하십시오.
4. **사용자 정의** 단일 선택 단추를 클릭하십시오.
5. 런타임 트리에서 **오브젝트 그리드 Map** 모듈을 펼친 후 **clusterObjectGrid** 링크를 클릭하십시오. **오브젝트 그리드 Map** 그룹 아래 clusterObjectGrid라고 하는 하나의 오브젝트 그리드 인스턴스가 있고 이 clusterObjectGrid 그룹 아래 카운터, 직원, 사무실 및 사이트라는 네 개의 맵이 있습니다. **ObjectGrids** 인스턴스에는 하나의 clusterObjectGrid 인스턴스가 있고 해당 인스턴스 아래에는 `DEFAULT`라고 하는 트랜잭션 유형이 있습니다.
6. 관심이 있는 통계를 사용 가능하게 할 수 있습니다. 설명 목적으로 직원 맵의 **맵 항목 수** 및 DEFAULT 트랜잭션 유형의 **트랜잭션 응답 시간**을 사용 가능하게 할 수 있습니다.

스크립트를 사용하여 PMI를 사용 가능하게 하는 타스크를 자동화할 수 있습니다. 자세한 정보는 스크립트를 사용하여 오브젝트 그리드 PMI 사용 가능을 참조하십시오.

스크립트를 사용하여 오브젝트 그리드 PMI 사용 가능

wsadmin 도구를 사용하여 오브젝트 그리드 PMI를 사용 가능하게 하는 타스크를 자동화하십시오.

Application Server가 시작되어 있고 오브젝트 그리드 사용 응용프로그램이 설치되어 있어야 합니다. 또한 로그인하여 wsadmin 도구를 사용할 수 있어야 합니다. wsadmin 도구에 대한 자세한 정보는 WebSphere Extended Deployment 버전 6.0.x Information Center에 있는 스크립트 사용(wsadmin)을 참조하십시오.

이 타스크를 참고하여 PMI 사용 가능을 자동화하십시오. 관리 콘솔에서 PMI를 사용 가능하게 하려면 오브젝트 그리드 PMI 사용 가능을 참조하십시오.

1. 명령행 프롬프트를 여십시오. *`install_root`*`/bin` 디렉토리를 탐색하십시오. `wsadmin`을 입력하여 wsadmin 명령행 도구를 시작하십시오.
2. 오브젝트 그리드 PMI 런타임 구성을 수정하십시오. 다음 명령으로 서버에서 PMI가 사용 가능한지 확인하십시오.

```
wsadmin>set s1 [$AdminConfig getid /Cell:CELL_NAME/Node:NODE_NAME/Server:
APPLICATION_SERVER_NAME/]
wsadmin>set pmi [$AdminConfig list PMIService $s1]
wsadmin>$AdminConfig show $pmi.
```

PMI가 사용 가능하지 않은 경우 다음 명령을 실행하여 PMI를 사용 가능하게 하십시오.

```
wsadmin>$AdminConfig modify $pmi {{enable true}}
wsadmin>$AdminConfig save
```

PMI를 사용 가능하게 해야 하는 경우 서버를 다시 시작하십시오.

3. 통계 세트를 사용자 정의 세트로 변경하는 변수를 설정하십시오. 다음 명령을 실행하십시오.

```
wsadmin>set perfName [$AdminControl completeObjectName type=
Perf,process=APPLICATION_SERVER_NAME,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
wsadmin>set params [java::new {java.lang.Object[]} 1]
wsadmin>$params set 0 [java::new java.lang.String custom]
wsadmin>set sigs [java::new {java.lang.String[]} 1]
wsadmin>$sigs set 0 java.lang.String
```

4. 통계 세트를 사용자 정의로 설정하십시오. 다음 명령을 실행하십시오.

```
wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
```

5. objectGridModule PMI 통계를 사용 가능하게 하는 변수를 설정하십시오. 다음 명령을 실행하십시오.

```
wsadmin>set params [java::new {java.lang.Object[]} 2]
wsadmin>$params set 0 [java::new java.lang.String objectGridModule=1]
wsadmin>$params set 1 [java::new java.lang.Boolean false]
wsadmin>set sigs [java::new {java.lang.String[]} 2]
wsadmin>$sigs set 0 java.lang.String
wsadmin>$sigs set 1 java.lang.Boolean
```

6. 통계 문자열을 설정하십시오. 다음 명령을 실행하십시오.

```
wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params $sigs
```

7. mapModule PMI 통계를 사용 가능하게 하는 변수를 설정하십시오. 다음 명령을 실행하십시오.

```
wsadmin>set params2 [java::new {java.lang.Object[]} 2]
wsadmin>$params2 set 0 [java::new java.lang.String mapModule=*]
wsadmin>$params2 set 1 [java::new java.lang.Boolean false]
wsadmin>set sigs2 [java::new {java.lang.String[]} 2]
wsadmin>$sigs2 set 0 java.lang.String
wsadmin>$sigs2 set 1 java.lang.Boolean
```

8. 통계 문자열을 설정하십시오. 다음 명령을 실행하십시오.

```
wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params2 $sigs2
```

이 단계에서는 오브젝트 그리드 런타임 PMI를 사용 가능하게 하지만 PMI 구성을 수정하지는 않습니다. Application Server를 재시작하는 경우 기본 PMI 사용 가능을 제외한 PMI 설정이 유실됩니다.

PMI가 사용 가능하면 관리 콘솔 또는 스크립트를 통해 PMI 통계를 볼 수 있습니다. 자세한 정보는 『오브젝트 그리드 PMI 통계 검색』 및 『스크립트에서 오브젝트 그리드 PMI 통계 검색』을 참조하십시오.

오브젝트 그리드 PMI 통계 검색

오브젝트 그리드 응용프로그램의 성능 통계를 참조하십시오.

오브젝트 그리드 통계가 사용 가능하면 해당 통계를 검색할 수 있습니다. 오브젝트 그리드 PMI를 사용 가능하게 하려면 오브젝트 그리드 PMI 사용 가능을 참조하십시오.

이 타스크를 참고하여 오브젝트 그리드 응용프로그램의 성능 통계를 참조하십시오.

1. 관리 콘솔을 여십시오. 예를 들어 http://localhost:9060/ibm/console과 같습니다.
2. **모니터링 및 조정 > 성능 표시기 > 현재 활동**을 클릭하십시오.
3. Tivoli Performance Viewer를 사용하여 모니터할 서버를 클릭한 후 모니터링을 사용 가능하게 하십시오.
4. 서버를 클릭하여 성능 표시기 페이지를 보십시오.
5. 구성 트리를 펼치십시오. **오브젝트 그리드 Map > clusterObjectGrid**를 클릭한 후 **employees**을 선택하십시오. **ObjectGrids > clusterObjectGrid**를 펼친 후 **DEFAULT**를 선택하십시오.
6. 오브젝트 그리드 샘플 응용프로그램에서 ObjectGridCreationServlet Servlet으로 돌아가 단추 1, **맵 채우**를 클릭하십시오. 표시기에서 통계를 볼 수 있습니다.

Tivoli Performance Viewer에서 오브젝트 그리드 통계를 볼 수 있습니다.

JMX(Java Management Extension) 또는 wsadmin 도구를 사용하여 통계를 검색하는 타스크를 자동화할 수 있습니다. 『스크립트에서 오브젝트 그리드 PMI 통계 검색』을 참조하십시오.

스크립트에서 오브젝트 그리드 PMI 통계 검색

이 타스크를 참고하여 오브젝트 그리드 응용프로그램에서 성능 통계를 검색하십시오.

Application Server 환경에서 PMI(Performance Monitoring Infrastructure)를 사용 가능하게 하십시오. 자세한 정보는 오브젝트 그리드 PMI 사용 가능 또는 스크립트에서 오브젝트 그리드 PMI 사용 가능을 참조하십시오. 또한 로그인하여 wsadmin 도구를 사용할 수 있어야 합니다. wsadmin 도구에 대한 자세한 정보는 WebSphere Extended Deployment 6.0.x Information Center에 있는 스크립트 사용(wsadmin)을 참조하십시오.

이 타스크를 참고하여 Application Server 환경에서 성능 통계를 확보하십시오. 검색할 수 있는 오브젝트 그리드 통계에 대한 자세한 정보는 318 페이지의 『오브젝트 그리드 통계』를 참조하십시오.

1. 명령행 프롬프트를 여십시오. *install_root*/bin 디렉토리를 탐색하십시오. wsadmin을 입력하여 wsadmin 명령행 도구를 시작하십시오.
2. 환경 변수를 설정하십시오. 다음 명령을 실행하십시오.

```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
wsadmin>set mySrvName [$AdminControl completeObjectName type=Server,
name=APPLICATION_SERVER_NAME,*]
```

3. mapModule 통계를 확보하도록 변수를 설정하십시오. 다음 명령을 실행하십시오.

```
wsadmin>set params [java::new {java.lang.Object[]} 3]
wsadmin>$params set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params set 1 [java::new java.lang.String mapModule]
wsadmin>$params set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs [java::new {java.lang.String[]} 3]
wsadmin>$sigs set 0 javax.management.ObjectName
wsadmin>$sigs set 1 java.lang.String
wsadmin>$sigs set 2 java.lang.Boolean
```

4. mapModule 통계를 확보하십시오. 다음 명령을 실행하십시오.

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params $sigs
```

5. objectGridModule 통계를 확보하도록 변수를 설정하십시오. 다음 명령을 실행하십시오.

```
wsadmin>set params2 [java::new {java.lang.Object[]} 3]
wsadmin>$params2 set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params2 set 1 [java::new java.lang.String objectGridModule]
wsadmin>$params2 set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs2 [java::new {java.lang.String[]} 3]
wsadmin>$sigs2 set 0 javax.management.ObjectName
wsadmin>$sigs2 set 1 java.lang.String
wsadmin>$sigs2 set 2 java.lang.Boolean
```

6. objectGridModule 통계를 확보하십시오. 다음 명령을 실행하십시오.

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params2 $sigs2
```

리턴되는 통계에 대한 자세한 정보는 318 페이지의 『오브젝트 그리드 통계』를 참조하십시오.

오브젝트 그리드 및 외부 트랜잭션 상호 작용

일반적으로 오브젝트 그리드 트랜잭션은 session.begin 메소드로 시작하여 session.commit 메소드로 종료됩니다. 그러나 오브젝트 그리드가 임베드된 경우 외부 트랜잭션 조정자에서 트랜잭션을 시작 및 종료할 수 있습니다. 이 경우 session.begin 메소드를 호출하여 session.commit 메소드로 종료하지 않아도 됩니다.

외부 트랜잭션 조정

오브젝트 그리드 TransactionCallback 플러그인은 오브젝트 그리드 세션을 외부 트랜잭션과 연관시키는 isExternalTransactionActive(Session session) 메소드로 확장되었습니다. 메소드 헤더는 다음과 같습니다.

```
public synchronized boolean isExternalTransactionActive(Session session)
```

예를 들어 오브젝트 그리드는 WebSphere Application Server 및 WebSphere Extended Deployment를 통합하도록 설정될 수 있습니다. 이 심리스(seamless) 통합의 핵심은 WebSphere Application Server 버전 5.x 및 6.x에서의 ExtendedJTATransaction API 사용입니다. 그러나 WebSphere Application Server 버전 6.0.2를 사용하는 경우 이 메소드를 지원하려면 APAR PK07848을 적용해야 합니다. 오브젝트 그리드 세션을 WebSphere Application Server 트랜잭션 ID와 연관시키는 다음 샘플 코드를 사용하십시오.

```
/**
* This methodis required to associate an objectGrid session with a WebSphere
* transaction ID.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
 // remember that this localid means this session is saved for later.
 localIdToSession.put(new Integer(jta.getLocalId()), session);
 return true;
}
```

외부 트랜잭션 검색

종종 사용할 오브젝트 그리드 TransactionCallback 플러그인에서 외부 트랜잭션 서비스 오브젝트를 검색해야 할 수도 있습니다. WebSphere Application Server 서버의 경우 다음 예에 표시된 대로 ExtendedJTATransaction 오브젝트를 해당 네임 스페이스에서 찾습니다.

```
public J2EETransactionCallback() {
 super();
 localIdToSession = new HashMap();
String lookupName="java:comp/websphere/ExtendedJTATransaction";
try
{
  InitialContext ic = new InitialContext();
  jta = (ExtendedJTATransaction)ic.lookup(lookupName);
  jta.registerSynchronizationCallback(this);
}
 catch(NotSupportedException e)
{
throw new RuntimeException("Cannot register jta callback", e);
}
 catch(NamingException e){
throw new RuntimeException("Cannot get transaction object");
}
}
```

다른 제품의 경우 트랜잭션 서비스 오브젝트를 검색할 때 유사한 접근 방식을 사용할 수 있습니다.

외부 콜백에서 확약 제어

TransactionCallback 플러그인은 오브젝트 그리드 세션을 확약 또는 롤백하도록 외부 신호를 수신해야 합니다. 이 외부 신호를 수신하려면 외부 트랜잭션 서비스에서 콜백을 사용하십시오. 외부 콜백 인터페이스를 구현하고 외부 트랜잭션 서비스에서 등록해야 합니다. 예를 들어 WebSphere Application Server의 경우 다음 예와 같이 SynchronizationCallback 인터페이스를 구현해야 합니다.

```
public class J2EETransactionCallback implements
com.ibm.websphere.objectgrid.plugins.TransactionCallback,
SynchronizationCallback
{
public J2EETransactionCallback() {
 super();
String lookupName="java:comp/websphere/ExtendedJTATransaction";
 localIdToSession = new HashMap();
try
{
  InitialContext ic = new InitialContext();
  jta = (ExtendedJTATransaction)ic.lookup(lookupName);
  jta.registerSynchronizationCallback(this);
}
 catch(NotSupportedException e)
{
throw new RuntimeException("Cannot register jta callback", e);
}
  catch(NamingException e)
{
throw new RuntimeException("Cannot get transaction object");
}
}
public synchronized void afterCompletion(int localId, byte[] arg1,
boolean didCommit)
{
 Integer lid = new Integer(localId);
 // find the Session for the localId
 Session session = (Session)localIdToSession.get(lid);
 if(session != null)
{
try
{
   // if WebSphere Application Server is committed when
   // hardening the transaction to backingMap.
   // We already did a flush in beforeCompletion
   if(didCommit)
{
session.commit();
}
else
{
    // otherwise rollback
    session.rollback();
}
```

```
}
  catch(NoActiveTransactionException e)
{
   // impossible in theory
}
  catch(TransactionException e)
{
   // given that we already did a flush, this should not fail
}
finally
{
   // always clear the session from the mapping map.
   localIdToSession.remove(lid);
}
}
}
public synchronized void beforeCompletion(int localId, byte[] arg1)
{
 Session session = (Session)localIdToSession.get(new Integer(localId));
 if(session != null)
{
try
{
   session.flush();
}
  catch(TransactionException e)
{
   // WebSphere Application Server  does not formally define
   // a way to signal the
   // transaction has failed so do this
throw new RuntimeException("Cache flush failed", e);
}
}
}
}
```

TransactionCallback 플러그인에서 오브젝트 그리드 API 사용

이 플러그인이 오브젝트 그리드의 TransactionCallback 플러그인으로 사용되는 경우 자동 확약을 사용 불가능하게 합니다. 오브젝트 그리드의 표준 사용법 패턴은 다음과 같습니다.

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

이 TransactionCallback 플러그인이 사용 중인 경우 오브젝트 그리드에서는 컨테이너 관리 트랜잭션이 표시된 경우 응용프로그램에서 오브젝트 그리드를 사용한다고 가정합니다. 이 환경에서 이전 코드 스니펫은 다음 코드로 변경됩니다.

```
public void myMethod()
{
 UserTransaction tx = ...;
   tx.begin();
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
 tx.commit();
}
```

myMethod 메소드는 웹 응용프로그램 경우와 유사합니다. 응용프로그램은 표준 UserTransaction 인터페이스를 사용하여 트랜잭션을 시작, 확약 및 롤백합니다. 오브젝트 그리드는 자동으로 컨테이너 트랜잭션을 시작 및 확약합니다. 메소드가 TX_REQUIRES 속성을 사용하는 EJB(Enterprise JavaBeans) 메소드인 경우 UserTransaction 참조 및 호출을 제거하여 동일한 방식으로 작동하는 메소드 및 트랜잭션을 시작 및 확약하십시오. 이 경우 컨테이너에서 트랜잭션을 시작 및 종료해야 합니다.

오브젝트 그리드 및 파티션 기능 통합

ObjectGridPartitionCluster 샘플 응용프로그램을 사용하여 오브젝트 그리드 및 파티션 기능(WPF)의 결합된 기능에 대해 학습하십시오.

오브젝트 그리드 및 파티션 기능을 함께 작동하는 방법에 대한 요약은 331 페이지의 『오브젝트 그리드 및 파티션 기능』을 참조하십시오.

파티션 기능과 함께 오브젝트 그리드를 사용하려면 환경에 WebSphere Extended Deployment가 설치되어 있어야 합니다.

ObjectGridPartitionCluster 샘플에서는 오브젝트 그리드 및 파티션 기능(WPF)의 결합된 기능을 보여줍니다. 오브젝트 그리드 기능에서는 트랜잭션에 따라 캐싱 키 및 값 쌍 용량을 제공하고, 파티션 기능은 오브젝트 특성에 따라 컨텍스트 기반 라우팅 용량을 제공합니다.

- ObjectGridPartitionCluster 샘플 응용프로그램을 설치 및 실행하십시오. 자세한 정보는 333 페이지의 『ObjectGridPartitionCluster 샘플 응용프로그램 설치 및 실행』을 참조하십시오.
- 샘플 응용프로그램의 소스 코드를 보거나 수정할 경우 개발 도구에 엔터프라이즈 아카이브(EAR) 파일을 로드할 수 있습니다. 자세한 정보는 336 페이지의 『통합 오브젝트 그리드 및 파티션 기능 응용프로그램 빌드』를 참조하십시오.
- 샘플 응용프로그램에 대해 학습하십시오. 샘플 응용프로그램에 있는 코드에 대한 설명은 341 페이지의 『예: 오브젝트 그리드 및 파티션 기능 프로그래밍』을 참조하십시오.

기타 WebSphere Application Server 기능과 오브젝트 그리드를 통합하는 방법에 대한 자세한 정보는 313 페이지의 제 10 장 『WebSphere Application Server에서 오브젝트 그리드 통합』을 참조하십시오. 오브젝트 그리드 프로그래밍 모델에 대한 자세한 정보는 99 페이지의 제 9 장 『오브젝트 그리드 API(Application Programming Interface) 개요』를 참조하십시오.

오브젝트 그리드 및 파티션 기능

오브젝트 그리드 및 파티션 기능(WPF) 기능을 함께 작동하여 키 및 값 쌍에 대한 캐싱과 오브젝트 특성을 기준으로 컨텍스트 기반 라우팅을 제공할 수 있습니다.

ObjectGridPartitionCluster 샘플은 오브젝트 그리드 및 파티션 기능(WPF)의 결합된 기능을 설명합니다. 오브젝트 그리드 및 파티션 기능은 WebSphere Extended Deployment 제품에 있는 두 개의 기능입니다. 오브젝트 그리드 기능은 트랜잭션 방식으로 키 및 값 쌍 캐싱 기능을 제공하고, 파티션 기능은 오브젝트 특성에 따라 컨텍스트 기반 라우팅 기능을 제공합니다.

Loader 및 TransactionCallback 플러그인 기능을 설명하는 것 이외에 이 샘플은 ObjectGridEventListener, ObjectTransformer 및 OptimisticCallback 플러그인 사용 방법도 설명합니다. 특히, 샘플은 로컬 오브젝트 그리드 트랜잭션 전달 방법과 변경이 예상되지 않는 버전 검사기를 사용하거나 사용하지 않고 한 서버에서 다른 서버로 변경된 오브젝트를 무효화하는 방법을 설명합니다.

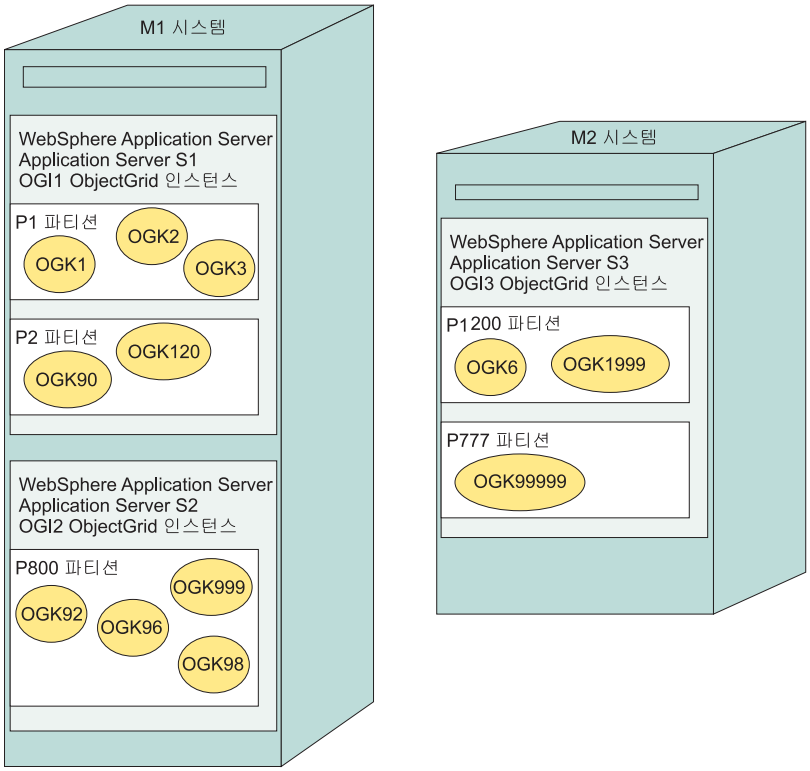
파티션 기능 컨텍스트 기반 라우팅 기능을 사용하여 동일한 키에 대한 오브젝트 갱신, 삽입 및 제거 요청이 동일한 JVM(Java Virtual Machine)으로 라우트되고 워크로드 관리를 사용하여 오브젝트 검색 요청을 모든 ObjectGrid JVM 간에 분배할 수 있도록 해야 합니다. 파티션 기능을 사용하면 서로 다른 클러스터 구성원 오브젝트 그리드 인스턴스 간에 데이터 무결성이 유지보수됩니다.

오브젝트 그리드 일관성 및 무결성을 유지보수하기 위해 파티션 기능을 사용하여 대형 오브젝트 그리드를 파티션된 다수의 오브젝트 그리드로 분산시킬 수 있으며, 파티션 기능 컨텍스트 기반 라우팅은 오브젝트 그리드 키에 따라 요청을 전달합니다. 예를 들어, JVM ObjectGrid에 공급할 수 없는 많은 수의 오브젝트를 처리하려면 오브젝트 그리드가 필요합니다. 파티션 기능을 사용하면 partitionLoadEvent 메소드를 사전 로드로 사용하여 서로 다른 서버에 데이터를 로드할 수 있으며, 파티션 기능 컨텍스트 기반 라우팅은 적합한 오브젝트 그리드를 찾습니다.

샘플은 해시 기반 파티션 세트 및 파티션 클러스터 라우팅 컨텍스트를 작성합니다.

- 다대다 계획을 사용하여 오브젝트 그리드 키를 파티션하고 WPF 파티션에 맵핑할 수 있습니다.
- 다대다 계획으로 WPF 파티션을 WebSphere Application Server 클러스터로 호스트할 수 있습니다.

다음 다이어그램은 ObjectGridPartitionCluster 샘플의 일반 설정 및 구성을 표시합니다.



위 다이어그램에서 M1 시스템과 M2 시스템은 ObjectGridPartitionCluster 샘플을 전개하는 데 사용됩니다. 각 물리적 시스템은 하나 이상의 WebSphere Application Server를 호스트할 수 있습니다. 예를 들어, M1 시스템은 두 개의 Application Server(S1 Application Server 및 S2 Application Server)를 호스트합니다. M2 시스템은 하나의 서버(S3 Application Server)를 호스트합니다. 각 서버마다 오브젝트 그리드 인스턴스가 있습니다(S1 Application Server의 경우 OGI1ObjectGrid 인스턴스, S2 Application Server의 경우 OGI2 ObjectGrid 인스턴스, S3 Application Server의 경우 OGI3 ObjectGrid 인스턴스).

각 Application Server는 다수의 파티션을 호스트할 수 있습니다. 예를 들어, S1 서버는 P1 파티션과 P2 파티션을 호스트하고 S3 서버는 P1200 파티션과 P777 파티션을 호스트합니다.

각 파티션은 다수의 오브젝트 그리드 키를 호스트할 수 있습니다. 예를 들어, P1 파티션은 OGK1, OGK2 및 OGK3 ObjectGrid 키를 호스트하고 P800 파티션은 OGK92, OGK96, OGK98 및 OGK9999 파티션을 호스트합니다.

모든 오브젝트 그리드 갱신, 삽입 및 제거 요청은 오브젝트 그리드 키에 따라 라우트됩니다. 두 개의 오브젝트 검색 옵션(워크로드 관리 계획의 임의 서버에서 또는 이 키의 특정 서버 파티션에서)이 있습니다.

ObjectGridPartitionCluster 샘플 응용프로그램 설치 및 실행

오브젝트 그리드 및 파티션 기능 사이에서 기능을 테스트하려면 이 타스크를 참고하여 ObjectGridPartitionCluster 샘플 응용프로그램을 설치 및 실행하십시오.

WebSphere Extended Deployment를 설치하십시오. 지시사항은 WebSphere Extended Deployment Library 페이지를 참조하십시오.

ObjectGridPartitionCluster 샘플을 실행하는 바람직한 환경으로는 두 개의 물리적 시스템에 WebSphere Extended Deployment를 설치하는 경우 또는 Deployment Manager로 두 개의 노드를 작성하여 함께 연합하는 경우가 있습니다.

1. 이 샘플 기능을 적절히 보려면 클러스터 구성원이 세 개 이상인 클러스터를 구성하십시오.
2. `D_ObjectGridPartitionClusterSample.ear` 파일을 설치하십시오. 파티션 기능(WPF) 전개 `D_ObjectGridPartitionClusterSample.ear` 파일을 설치 및 실행할 준비가 되었습니다. 샘플 소스 코드를 수정할 경우 빌드 및 WPF 전개 지시사항에 따라 엔터프라이즈 아카이브(EAR) 파일을 빌드 및 전개하십시오.

   응용프로그램 EAR 파일을 설치하는 일반적인 방법은 관리 콘솔을 사용하는 것입니다. 엔터프라이즈 응용프로그램 설치 프로시저에 따라 `D_ObjectGridPartitionClusterSample.ear` 파일을 설치하십시오. 관리 콘솔의 이 파트에 액세스하려면 **응용프로그램 > 새 응용프로그램 설치**를 클릭하십시오. 설치 중 EAR 파일을 전개하지 마십시오. 설치 위치를 선택하도록 요청하는 단계를 제외하고 기본 설정을 사용하십시오. 이 단계에서 기본 server1 서버 대신 사용자가 정의한 클러스터를 선택하십시오.
3. ObjectGridPartitionClusterSample 클라이언트를 실행하십시오.
   a. 클러스터를 시작하십시오. 관리 콘솔에서 **서버 > 클러스터**를 클릭하십시오. 클러스터를 선택한 후 **시작**을 클릭하십시오.
   b. **WAS_INSTALL_ROOT₩bin₩wpfadmin balance** 스크립트 명령을 실행하십시오. **WAS_INSTALL_ROOT₩bin₩wpfadmin list** 명령을 사용하여 파티션이 활성 상태인지 확인하십시오. wpfadmin 스크립트 및 해당 명령에 대한 정보는 WebSphere Extended Deployment Information Center의 Partitioning Facility Guide를 참조하십시오.
   c. ObjectGridPartitionClusterSample 클라이언트를 실행하려면 다음 명령을 실행하십시오.

      ```
      WAS_INSTALL_ROOT/bin/launchClient.bat|sh ₩
      WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear ₩
      -CCBootstrapPort=PORT
      ```

      여기서 PORT는 서버를 시작한 후 서버 `SystemOut.log` 파일에서 찾을 수 있는 서버 RMI 포트입니다. 일반적으로 이 포트 값은 `9810, 9811, 9812` 값 중 하나입니다.

예를 들어 다음 명령을 실행할 수 있습니다.

```
WAS_INSTALL_ROOT/bin/launchClient.bat|sh
WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
  -CCBootstrapPort=9811
```

이 스크립트의 고급 사용법은 『ObjectGridPartitionClusterSample 응용프로그램 클라이언트 옵션』을 참조하십시오.

4. 파티션 수를 변경하십시오. ObjectGridPartitionCluster 세션 엔터프라이즈 Bean에서 작성하는 파티션 수를 변경하십시오. PFClusterObjectGridEJB 세션 Bean에서 작성하는 파티션 수는 `META-INFWejb-jar.xml` 파일에 있는 NumberOfPartitions 환경 항목에서 결정됩니다. 기본값은 `10`입니다. 이 환경 변수 값을 변경한 후 응용프로그램을 다시 설치하면 다른 수의 파티션을 작성할 수 있습니다. 파티션 수를 `999999` 미만으로 설정하십시오.
5. 분산 리스너 옵션을 변경하십시오. 다음 오브젝트 그리드 분배 리스너 옵션을 변경할 수 있습니다.

표 17. 분배 리스너 옵션

변수 이름	설명
enableDistribution,	EJB 전개 설명자에 있는 enableDistribution 환경 항목 변수를 사용하여 오브젝트 그리드 분배 리스너를 사용 가능하게 할 수 있습니다. 기본값은 사용 가능을 의미하는 `true`입니다. 분배 리스너를 끄려면 값을 `false`로 설정하십시오.
propagationMode	EJB 전개 설명자에 있는 propagationMode 환경 항목 변수를 사용하여 전달 모드를 변경할 수 있습니다. 기본값은 `update`입니다. 기본값을 사용하지 않을 경우 값을 `invalidate`로 변경할 수 있습니다.
propagationVersionOption,	EJB 전개 설명자에 있는 propagationVersionOption 환경 항목 변수를 사용하여 전달 버전 옵션을 변경할 수 있습니다. 기본값은 `enable`입니다. 값을 `disable`로 설정할 수 있습니다.
compressionMode	EJB 전개 설명자에 있는 compressionMode 환경 항목 변수를 사용하여 압축 모드를 변경할 수 있습니다. 기본값은 `enable`입니다. 값을 `disable`로 설정할 수 있습니다.

기본값은 버전 확인을 통해 갱신을 전달하는 것입니다. 버전 확인 없이 값을 무효화 모드로 설정할 수도 있습니다.

ObjectGridPartitionCluster 샘플 응용프로그램을 설치 및 실행했습니다.

ObjectGridPartitionClusterSample 응용프로그램 클라이언트 옵션

`D_ObjectGridPartitionClusterSample.ear` 파일 실행 시 고급 사용을 적용하려면 이 옵션을 사용하십시오.

고급 샘플 사용법

D_ObjectGridPartitionClusterSample.ear 파일 설치 및 실행에 대한 자세한 정보는 333 페이지의 『ObjectGridPartitionCluster 샘플 응용프로그램 설치 및 실행』을 참조하십시오.

샘플의 고급 사용은 다음 전체 사용법 안내서를 참조하십시오.

```
WAS_INSTALL_ROOT/bin/launchClient.bat|sh
WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
-CCproviderURL=corbaloc::HOSTNAME:SERVER_RMI_PORT [-loop LOOP] [-threads
NUMBER_OF_THREADS] [-add NUMBER_OF_STOCKS_PER_PARTITION] [-waitForPropagation
SECONDS_TO_WAIT_FOR_PROPAGATION] [-getIteration
NUMBER_OF_ITERATION_PER_OGKEY]
```

다음 변수를 채우십시오.

- HOSTNAME: 실행 중인 Application Server의 호스트 이름을 지정합니다.
- SERVER_RMI_PORT: Application Server의 부트스트랩 포트를 지정합니다.
- LOOP: 클라이언트가 실행하는 루프 수를 지정합니다. 이 매개변수는 선택사항입니다. 기본값은 1입니다.
- NUMBER_OF_THREADS: 클라이언트가 실행하는 스레드 수를 지정합니다. 이 매개변수는 선택사항입니다. 기본값은 1입니다.
- NUMBER_OF_STOCKS_PER_PARTITION: 각 파티션에서 추가할 재고 수를 지정합니다. 이 매개변수는 선택사항입니다. 기본값은 3입니다.
- SECONDS_TO_WAIT_FOR_PROPAGATION: 새로 추가되거나 갱신된 오브젝트 그리드 오브젝트를 다른 서버로 전달하는 데 대기하는 초 수를 지정합니다. 기본값은 2초입니다.
- NUMBER_OF_ITERATION_PER_OGKEY: 워크로드 관리 방식으로 오브젝트 그리드에서 오브젝트를 검색하는 반복 수를 지정합니다. 기본값은 6입니다. 반복을 더 많이 지정하면 서로 다른 WebSphere Application Server 서버에서 동일한 키를 포함하는 오브젝트의 명확한 패턴이 나타납니다.

샘플 출력

이 명령의 출력은 다음 예와 유사하게 표시됩니다.

```
C:₩dev₩xd6₩bin>launchClient
D_ObjectGridPartitionClusterSample.ear -CCBootstrapPort=9812
IBM WebSphere Application Server, Release 6.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2004
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application
Client class com.ibm.websphere.samples.objectgrid.partitionclust
er.client.PartitionObjectGrid
ObjectGrid Partition Sample has 10 partitions
PARTITION: ObjectGridHashPartition000007->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000003->clusterdevNode02/s3
```

```
PARTITION: ObjectGridHashPartition000005->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000010->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000006->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000009->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000008->clusterdevNode01/s1
PARTITION: ObjectGridHashPartition000002->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000001->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000004->clusterdevNode01/s2
********** Partition=ObjectGridHashPartition000004**********
------ObjectGrid Operations: Stock Ticket=Stock000104 --------
get on partition for ticket: Stock000104->clusterdevNode02/s2
update: Stock000104->clusterdevNode02/s2
sleep 2 seconds.....
Iteration 1 : Stock000104->clusterdevNode01/s2
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 2 : Stock000104->clusterdevNode01/s1
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 3 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 4 : Stock000104->clusterdevNode01/s2
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 5 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 6 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
------ObjectGrid Operations: Stock Ticket=Stock000114 --------
get on partition for ticket: Stock000114->clusterdevNode01/s2
update: Stock000114->clusterdevNode02/s2
sleep 2 seconds.....
Iteration 1 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 2 : Stock000114->clusterdevNode01/s2
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 3 : Stock000114->clusterdevNode01/s1
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 4 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 5 : Stock000114->clusterdevNode01/s2
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 6 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
------ObjectGrid Operations: Stock Ticket=Stock000124 --------
get on partition for ticket: Stock000124->clusterdevNode02/s2
update: Stock000124->clusterdevNode01/s2
sleep 2 seconds.....
Iteration 1 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 2 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 3 : Stock000124->clusterdevNode01/s2
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 4 : Stock000124->clusterdevNode01/s1
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 5 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 6 : Stock000124->clusterdevNode01/s2
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
C:₩dev₩xd6₩bin>
```

통합 오브젝트 그리드 및 파티션 기능 응용프로그램 빌드

오브젝트 그리드 파티션 샘플 응용프로그램을 열어 수정한 후 설치하십시오.

이 단계를 사용하여 WebSphere Extended Deployment 환경에서 ObjectGridPartitionSample.ear 파일을 수정하고 내보내어 설치하십시오. 샘플 파

일을 변경하지 않을 경우 전개 및 파티션 기능(WPF) 사용 D_ObjectGridPartitionClusterSample.ear 파일을 사용할 수 있습니다. D_ObjectGridPartitionClusterSample.ear 파일을 사용하는 경우 다음 단계를 수행하지 않고 파일을 설치 및 실행할 수 있습니다. 두 엔터프라이즈 아카이브(EAR) 파일 모두 WAS_INSTALL_ROOT/installableApps 디렉토리에 있습니다.

1. 빌드 환경(예: IBM Rational Application Developer 버전 6.0.x 또는 Application Server Toolkit 버전 6.0.x)에서 ObjectGridPartitionSample.ear 파일을 설정하십시오. 자세한 정보는 『오브젝트 그리드 및 파티션 기능 응용프로그램 빌드로 시작하기』를 참조하십시오.
2. 샘플의 소스 코드를 수정하십시오.
3. 빌드 환경에서 ObjectGridPartitionClusterSample 응용프로그램을 EAR 파일로 내보내십시오. 자세한 정보는 339 페이지의 『IBM Rational Application Developer에서 ObjectGridPartitionClusterSample.ear 파일 내보내기』를 참조하십시오.
4. 파티션 기능에 대한 작업을 수행할 수 있도록 응용프로그램을 전개하십시오. 자세한 정보는 340 페이지의 『파티션 기능에 대한 작업을 수행하도록 ObjectGridPartitionClusterSample.ear 파일 전개』를 참조하십시오.
5. WebSphere Extended Deployment에 ObjectGridPartitionClusterSample.ear 파일을 설치하십시오. 응용프로그램 EAR 파일을 설치하는 일반적인 방법은 WebSphere Application Server 관리 콘솔을 사용하는 것입니다. 관리 콘솔의 엔터프라이즈 응용프로그램 설치 프로시저에 따라 D_ObjectGridPartitionClusterSample.ear 파일을 설치하십시오. 설치 중 파일을 전개하지 마십시오. 대신 기본값을 사용하십시오. 설치 위치를 선택하도록 요청하는 경우를 제외하고 각 단계에서 기본 설정을 사용하십시오. 이 단계에서 기본 server1 서버 대신 사용자가 정의한 클러스터를 선택하십시오.

WebSphere Extended Deployment 환경에 ObjectGridPartitionClusterSample.ear 파일을 설치했습니다.

오브젝트 그리드, 파티션 기능 및 샘플 응용프로그램을 사용하는 프로그래밍에 대한 자세한 정보는 341 페이지의 『예: 오브젝트 그리드 및 파티션 기능 프로그래밍』을 참조하십시오.

오브젝트 그리드 및 파티션 기능 응용프로그램 빌드로 시작하기

샘플 응용프로그램을 다시 빌드하려면 Application Server Toolkit 버전 6.0.x 또는 IBM Rational Application Developer 버전 6.0.x를 사용하십시오.

WAS_INSTALL_ROOT/installableApps 디렉토리의 ObjectGridPartitionClusterSample.ear 파일에는 모든 소스 코드가 들어 있습니다. Application Server Toolkit 버전 6.0.x 또는 IBM Rational Application Developer 버전 6.0.x를 사용하여 이 샘플 응용프로그램을 다시 빌드할 수 있습니다. 이 타스크에

서는 예로 Rational Application Developer를 사용하여 ObjectGridPartitionClusterSample.ear 파일의 빌드 환경을 설정합니다. 또한 WebSphere Application Server와 함께 별도의 CD로 제공되는 무료 어셈블리 도구인 Application Server Toolkit을 사용할 수 있습니다.

WAS_INSTALL_ROOT/installableApps 디렉토리에 있는 전개 및 WPF 사용 엔터프라이즈 아카이브(EAR) 파일인 D_ObjectGridPartitionClusterSample.ear 파일도 설치 및 실행할 준비가 되었습니다.

1. ObjectGridPartitionClusterSample.ear 파일을 Rational Application Developer로 가져오십시오.
   a. Rational Application Developer를 시작하십시오.
   b. **옵션:** J2EE 프로젝트에 대한 작업을 수행하도록 J2EE(Java 2 Platform, Enterprise Edition) Perspective를 여십시오. **창 > Perspective 열기 > 기타 > J2EE**를 클릭하십시오.
   c. **옵션:** 프로젝트 탐색기 보기를 여십시오. **창 > 보기 표시 > 프로젝트 탐색기**를 클릭하십시오. 다른 유용한 보기로 네비게이터 보기가 있습니다(**창 > 보기 표시 > 네비게이터**).
   d. ObjectGridPartitionClusterSample.ear 파일을 가져오십시오. **파일 > 가져오기 > EAR 파일**을 클릭한 후 **다음**을 클릭하십시오.
   e. WAS_INSTALL_ROOT/installableApps 디렉토리에서 ObjectGridPartitionClusterSample.ear 파일을 선택하십시오.
   f. **옵션: 새로 작성**을 클릭하여 새 서버 런타임 마법사를 연 후 지시사항에 따르십시오.
   g. 대상 서버 필드에서 서버 런타임의 **WebSphere Application Server V6.0** 유형을 선택하십시오.
   h. **완료**를 클릭하십시오.

   ObjectGridPartitionClusterSample, ObjectGridPartitionClusterSampleEJB 및 ObjectGridPartitionClusterSampleClient 프로젝트는 프로젝트 탐색기 보기에서 작성하고 표시해야 합니다.
2. ObjectGridPartitionClusterSampleEJB 프로젝트를 설정하십시오.
   a. J2EE Perspective의 프로젝트 탐색기 보기에서 EJB 프로젝트에 있는 **ObjectGridPartitionClusterSampleEJB** 프로젝트를 마우스 오른쪽 단추로 클릭한 후 **특성**을 선택하십시오. 특성 창이 표시됩니다.
   b. 왼쪽 패널에서 **Java 빌드 경로**를 클릭하고 오른쪽 패널에서 **라이브러리** 탭을 클릭한 후 **변수 추가**를 선택하십시오. 새 변수 클래스 경로 항목 창이 표시됩니다.
   c. **변수 구성**을 클릭하여 환경 설정 창을 여십시오.

d. **새로 작성**을 클릭하여 새 변수 항목 창을 여십시오.

e. **이름**에 ObjectGridPartitionCluster_JAR을 입력한 후 **파일**을 클릭하여 JAR 선택사항 창을 여십시오.

f. WAS_INSTALL_ROOT/lib 디렉토리를 찾아본 후 **wsobjectgrid.jar**을 선택하십시오. **열기**를 클릭하여 JAR 선택사항 창을 닫으십시오.

g. **확인**을 클릭하여 새 변수 항목 창을 닫으십시오. ObjectGridPartitionCluster_JAR 변수가 클래스 경로 변수 목록에 표시됩니다.

h. **확인**을 클릭하여 환경 설정 창을 닫으십시오.

i. 변수 목록에서 **ObjectGridPartitionCluster_JAR** 변수를 선택하고 **확인**을 클릭하여 새 변수 클래스 경로 항목 창을 닫으십시오. ObjectGridPartitionCluster_JAR 변수가 라이브러리 패널에 표시됩니다.

j. 이 프로시저를 반복하여 환경에 wpf.jar 파일을 추가하십시오.

k. wpf.jar 파일 및 wsobjectgrid.jar 파일이 빌드 클래스 경로에 있는지 확인하십시오.

빌드 환경을 설정하면 소스 코드를 수정하여 기타 변경사항을 적용할 수 있습니다. 자세한 정보는 336 페이지의 『통합 오브젝트 그리드 및 파티션 기능 응용프로그램 빌드』를 참조하십시오.

IBM Rational Application Developer에서 ObjectGridPartitionClusterSample.ear 파일 내보내기

샘플 파일을 변경하면 ObjectGridPartitionClusterSample 응용프로그램을 내보내어 WebSphere Extended Deployment 서버에 설치할 수 있는 엔터프라이즈 아카이브(EAR) 파일을 작성할 수 있습니다.

소스를 변경할 수 있도록 개발 도구로 ObjectGridPartitionSample.ear 파일을 가져와야 합니다. 자세한 정보는 337 페이지의 『오브젝트 그리드 및 파티션 기능 응용프로그램 빌드로 시작하기』를 참조하십시오. 내보내기 전에 먼저 샘플 응용프로그램을 변경하십시오.

IBM Rational Application Developer의 엔터프라이즈 응용프로그램에 있는 ObjectGridPartitionClusterSample 프로젝트에서 ObjectGridPartitionClusterSample.ear 파일을 내보낼 수 있습니다. 파티션 기능을 전개한 후 WebSphere Extended Deployment 버전 6.0 서버에 내보낸 ObjectGridPartitionClusterSample.ear 파일을 설치할 수 있습니다.

1. J2EE(Java 2 Platform, Enterprise Edition) Perspective의 프로젝트 탐색기 보기에서 엔터프라이즈 응용프로그램 아래에 있는 **ObjectGridPartitionClusterSample** 응용프로그램을 마우스 오른쪽 단추로 클릭하십시오. **내보내기 > EAR 파일**을 클릭하십시오. 내보내기 창이 표시됩니다.

2. **찾아보기**를 클릭하여 **다른 이름으로 저장** 창을 여십시오. 대상 출력 디렉토리를 찾아 파일 이름을 ObjectGridPartitionClusterSample로 지정한 후 **저장**을 클릭하십시오.
3. **찾아보기**를 클릭하여 다른 이름으로 저장 창을 여십시오. 대상 출력 디렉토리를 찾아 파일 이름을 ObjectGridPartitionClusterSample로 지정하십시오. **저장**을 클릭하십시오.

ObjectGridPartitionClusterSample.ear 파일이 지정된 대상 출력 디렉토리에 작성됩니다.

파티션 기능(WPF)의 ObjectGridPartitionClusterSample.ear 파일을 전개하면 WebSphere Extended Deployment에서 해당 파일을 실행할 수 있습니다. 자세한 정보는 『파티션 기능에 대한 작업을 수행하도록 ObjectGridPartitionClusterSample.ear 파일 전개』를 참조하십시오.

파티션 기능에 대한 작업을 수행하도록 ObjectGridPartitionClusterSample.ear 파일 전개

WebSphere Extended Deployment에 ObjectGridPartitionClusterSample.ear 파일을 설치할 경우 해당 파일에서 WPF 전개 조작을 수행해야 합니다.

기존 ObjectGridPartitionClusterSample.ear 파일이 있어야 합니다. 기존 파일을 수정하려면 336 페이지의 『통합 오브젝트 그리드 및 파티션 기능 응용프로그램 빌드』를 참조하십시오.

WebSphere Extended Deployment 환경에서 엔터프라이즈 아카이브(EAR) 파일을 준비하려면 WPF 전개 조작을 수행하십시오.

1. DEST_DIR 디렉토리를 작성하십시오.
2. ObjectGridPartitionClusterSample.ear 파일을 DEST_DIR 디렉토리에 복사하십시오. ObjectGridPartitionClusterSample.ear 파일 이름을 old_ObjectGridPartitionClusterSample.ear 파일로 바꾸십시오.
3. 다음 명령을 실행하십시오. 여기서 WORKING_DIR은 ejbdeloy 도구의 작업 디렉토리입니다(예: c:¥temp 디렉토리).

   ```
   WAS_HOME¥bin¥ejbdeploy.bat|ejbdeploy.sh
   DEST_DIR¥old_ObjectGridPartitionClusterSample.ear WORKING_DIR
   DEST_DIR¥ObjectGridPartitionClusterSample.ear
   ```

4. 다음 명령을 실행하십시오. 여기서 TEMP_DIR은 도구의 임시 디렉토리입니다. -keep 인수를 지정하면 wpfStubUtil 유틸리티에서 작성한 임시 디렉토리가 삭제되지 않습니다.

```
WAS_HOME¥bin¥wpfStubUtil.cmd|wpfStubUtil.sh
DEST_DIR¥ObjectGridPartitionClusterSample.ear
 ObjectGridPartitionClusterSampleEJB.jar com/ibm/websphere/samples/
objectgrid/partitioncluster/ejb/PFClusterObjectGridEJB.class
TEMP_DIR [-stubDebug|-keep]
```

WebSphere Extended Deployment 환경에서 ObjectGridPartitionClusterSample.ear 파일을 실행할 준비가 되었습니다. WebSphere Extended Deployment에서 제공하는 D_ObjectGridPartitionClusterSample.ear 샘플 파일은 이미 전개되었습니다. 소스 코드를 변경하지 않을 경우 응용프로그램을 설치하기 전에 먼저 이 파일을 전개하지 않아도 됩니다.

관리 콘솔을 사용하여 ObjectGridPartitionClusterSample.ear 파일을 WebSphere Extended Deployment 환경에 설치하십시오. 자세한 정보는 336 페이지의 『통합 오브젝트 그리드 및 파티션 기능 응용프로그램 빌드』를 참조하십시오.

예: 오브젝트 그리드 및 파티션 기능 프로그래밍

이 예에서는 WebSphere Extended Deployment의 Java 2 Platform, Enterprise Edition 환경에서 파티션 기능 및 ObjectGrid의 결합된 기능을 사용하는 방법을 보여줍니다.

목적

파티션 기능(WPF) 및 오브젝트 그리드의 결합된 기능을 보여주는 것 이외에도 이 예에서는 오브젝트 그리드 분배 리스너 전달 및 무효화를 보여줍니다.

오브젝트 갱신, 삽입 및 제거 요청은 파티션이 해당 오브젝트 그리드 키에서 호스트되는 특정 서버로 라우트됩니다. 오브젝트 그리드 get 메소드 요청은 모든 서버에서 관리되는 워크로드입니다.

이 예에서는 대형 오브젝트 그리드를 많은 소형 오브젝트 그리드로 파티션하는 방법 및 파티션된 오브젝트 그리드에서 무제한의 오브젝트를 호스트할 수 있도록 partitionLoadEvent 메소드를 사용하여 데이터를 사전 로드하는 방법을 보여줍니다.

개요

ObjectGridPartitionClusterSampler.ear 파일은 오브젝트 그리드 및 파티션 기능을 함께 사용하는 방법을 표시하는 재고 오브젝트를 작성합니다. 재고 오브젝트에는 다음 특성이 들어 있습니다.

- 티켓
- 회사
- serialNumber
- 설명
- lastTransaction

- 가격

여기서 lastTransaction 특성은 재고가 변경된 시간입니다. lastTransaction 특성을 사용하여 다른 JVM(Java Virtual Machine)의 오브젝트 그리드에서 오브젝트가 최신인지 여부를 표시하십시오.

샘플에서 오브젝트 그리드 인스턴스는 ObjectGridFactory 클래스를 사용하여 EJB(Enterprise JavaBeans) setContext 메소드에서 작성됩니다.

해시 기반 파티션 세트를 정의하십시오. 기본값은 10개의 파티션이지만 사용자가 파티션 수를 변경할 수 있습니다. SampleUtility.java 파일을 사용하여 재고 티켓을 이 파티션에 해시하십시오. 각 파티션에서 많은 오브젝트 그리드 키 및 값 쌍을 호스트할 수 있습니다.

샘플에서는 오브젝트 그리드 삽입, 갱신 및 제거 요청을 특정 파티션된 서버로 라우트하는 방법 및 오브젝트 그리드 get 메소드 요청이 해당 키의 특정 서버 또는 클러스터에 있는 임의의 서버에 라우트되는 방법을 보여줍니다. 샘플에서는 키의 update, insert 또는 remove 조작 때문에 특정 서버에서 값이 변경되면 해당 키의 오브젝트 값을 다른 서버와 비교합니다.

위치

각 서버가 많은 파티션을 호스트할 수 있고 각 파티션에서 서로 다른 키를 포함하는 많은 오브젝트를 호스트할 수 있는 클러스터 환경에서 이 샘플을 사용하십시오.

다음과 같은 두 개의 오브젝트 그리드 파티션 클러스터 샘플 파일이 <install_root>₩installableApps₩ 디렉토리에 있습니다.

- ObjectGridPartitionClusterSampler.ear 파일에는 소스 코드가 들어 있습니다. 소스를 보려면 파일 시스템에 EAR 파일을 펼치거나 소스를 개발 환경으로 가져오십시오. 자세한 정보는 336 페이지의 『통합 오브젝트 그리드 및 파티션 기능 응용프로그램 빌드』를 참조하십시오.
- D_ObjectGridPartitionClusterSample.ear 파일은 이미 파티션 기능이 전개된 상태입니다. Readme 파일 및 지시사항에 따라 이 파일을 빨리 실행하십시오.

설명

다음 섹션에는 오브젝트 그리드 파티션 클러스터 샘플 응용프로그램에 대한 설명이 들어 있습니다.

- 343 페이지의 『오브젝트 그리드 조작 EJB 인터페이스』
- 344 페이지의 『PartitionKey 클래스』
- 347 페이지의 『SampleUtility 클래스 및 파티션 맵핑』
- 349 페이지의 『엔터프라이즈 Bean setContext 메소드에서 오브젝트 그리드 작성』

- 351 페이지의『싱글톤 ObjectGridFactory 클래스』
- 353 페이지의『오브젝트 그리드 파티션 사전 로드』

오브젝트 그리드 조작 EJB 인터페이스

이 항목에서는 get, 파티션된 서버에서의 get, insert, update 및 remove 조작을 수행하는 오브젝트 그리드 조작 EJB(Enterprise JavaBeans) 인터페이스를 보여줍니다.

목적

오브젝트 그리드 조작 EJB 인터페이스에서는 get, 파티션된 서버에서의 get, insert, update 및 remove 조작을 수행합니다. 파티션된 서버에서의 get 메소드는 요청하는 키에 해당하는 파티션에 라우트됩니다. get 메소드는 워크 로드 관리 계획에서 임의의 서버로 라우트됩니다.

PFClusterObjectGridEJB 인터페이스

PFClusterObjectGridEJB 인터페이스 컨텐츠는 다음과 같습니다.

```
/**
* Remote interface for Enterprise Bean: PFClusterObjectGridEJB
*/
public interface PFClusterObjectGridEJB extends javax.ejb.EJBObject {
public String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
 * Get all Partitions
*
 * @return Array of Strings
 * @throws java.rmi.RemoteException
*/
 public String [] getAllPartitions() throws java.rmi.RemoteException;
/**
 * Get where partition is hosted
*
* @param partition
* @return String
 * @throws java.rmi.RemoteException
*/
 public String getServer(String partition)
 throws java.rmi.RemoteException;
/**
 * Get Stock object and its server information
 * (ServerIDResult) for a stock ticket
 * from any server in a cluster (that has had workload management)
*
 * @param ticket
* @return
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult getStock(String ticket)
 throws java.rmi.RemoteException;
/**
 * Get Stock object and its partitioned server
 * information for a stock ticket
```

```
 * from the partition this ticket key is hashed to
*
 * @param ticket
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult getStockOnPartitionedServer(String ticket)
 throws java.rmi.RemoteException;
/**
 * Update stock in a particular server where the partition is
 * active for this stock ticket key.
*
 * @param stock
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult updateStock(Stock stock)
 throws java.rmi.RemoteException;
/**
 * Remove stock in a particular server where the partition is
 * active for this stock ticket key.
*
 * @param ticket
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult removeStock(String ticket)
 throws java.rmi.RemoteException;
/**
 * Insert stock in a particular server where the partition is
 * active for this stock ticket key.
*
 * @param stock
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult insertStock(Stock stock)
 throws java.rmi.RemoteException;
/**
 * Retrieve data from all servers and compare values
*
 * @param server
 * @return ServerObjectGridVerification
 * @throws java.rmi.RemoteException
*/
 public ServerObjectGridVerification verifyObjectGrid(String server)
 throws java.rmi.RemoteException;
}
```

PartitionKey 클래스

PartitionKey 클래스는 파티션 기능 컨텍스트 기반 라우팅의 동작을 제어합니다.

다음 코드에서는 샘플 파티션 키 클래스를 설명합니다. 메소드에서 not null을 리턴하면 파티션 기능(WPF) 라우터를 통해 라우트됩니다. 메소드에서 null을 리턴하면 워크로드 관리(WLM) 라우터로 전달됩니다.

```
/**
* PartitionKey for Partitioned Stateless Session Bean WPFKeyBasedPartition
*
*/
public class PFClusterObjectGridEJB_PartitionKey {
/**
 * Number of Partitions
*
 * Default is 10.
*
*/
 static int numOfPartitions=10;
/**
 * Only once to getPartitionNumbers
*/
 static boolean getNumOfPartitions=true;
/**
 * Get the number of partitions
*
*/
 static void getPartitionNumbers(){
 //get only once
  if (getNumOfPartitions){
try {
    InitialContext ic = new InitialContext();
    PFClusterObjectGridEJBHome home =
    (PFClusterObjectGridEJBHome) PortableRemoteObject.narrow(
ic.lookup("java:comp/env/ejb/PFClusterObjectGridEJB"),
    PFClusterObjectGridEJBHome.class);
    final PFClusterObjectGridEJB session = home.create();
    String[] PARTITIONS = session.getAllPartitions();
numOfPartitions=PARTITIONS.length;
getNumOfPartitions=false;
}
   catch (ClassCastException e) {
    e.printStackTrace();
numOfPartitions=10;
}
   catch (RemoteException e) {
    e.printStackTrace();
numOfPartitions=10;
}
   catch (NamingException e) {
    e.printStackTrace();
numOfPartitions=10;
}
   catch (CreateException e) {
    e.printStackTrace();
numOfPartitions=10;
}
}
}
/**
* Return partition key
*
  * @param partition
* @return String
*/
```

```
public static String getStock(String key) {
return null;
}
/**
* Return partition key
*
* @param key
* @return String
*/
public static String getServer(String key) {
 return key;
}
/**
* Retrieve ObjectGrid data from a partitioned server where
* data's changes happen (the highest quality and integrity).
*
 * @param ticket
* @return hashcode of stock ticket
*/
public static String getStockOnPartitionedServer(String ticket) {
 if (ticket==null){
return null;
}
 getPartitionNumbers();
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
 * @param stock
* @return hashcode of stock ticket
*/
public static String updateStock(Stock stock) {
 getPartitionNumbers();
 String ticket=null;
 if (stock!=null){
  ticket=stock.getTicket();
}
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
 * @param stock
* @return hashcode of stock ticket
*/
public static String insertStock(Stock stock) {
 getPartitionNumbers();
 String ticket=null;
 if (stock!=null){
  ticket=stock.getTicket();
}
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
 * @param server
```

```
* @return String
*/
public static String verifyObjectGrid(String server) {
 return server;
}
/**
* Return partition key
*
 * @param stock
* @return hashcode of stock ticket
*/
public static String removeStock(String ticket) {
 if (ticket==null){
return null;
}
 getPartitionNumbers();
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
  * @param partition
* @return
*/
public static String getAllPartitions() {
return null;
}
}
```

각 원격 메소드에는 유효한 문자열 또는 널값을 리턴하는 해당 메소드가 있어야 합니다.

SampleUtility 클래스 및 파티션 맵핑

SampleUtility.java 파일을 사용하여 키, 재고 티켓, 해시 및 파티션을 조작하십시오. 이 파일을 사용하여 파티션에 오브젝트 그리드 키를 맵핑할 수도 있습니다. 유사한 유틸리티 클래스를 개발하여 비즈니스 요구를 만족시킬 수 있도록 파티션에 오브젝트 그리드 키를 맵핑할 수 있습니다. 오브젝트 그리드에서 파티션 기능을 사용하려면 서로 다른 파티션에 서로 다른 키를 맵핑해야 합니다.

SampleUtility 클래스

ObjectGridPartitionCluster 샘플의 유틸리티 클래스는 다음과 같습니다.

```
/**
* Utility class for ObjectGridPartitionCluster sample
*
*
*/
public class SampleUtility {
/**
 * Container for recording partitions.
*/
 static Map serverPartitions= new HashMap();
/**
```

```
 * Partition name prefix
*/
public static String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
 * Stock name prefix
*/
public static String STOCK_PREFIX="Stock";
/**
 * Retrieve the number part of partition name
*
  * @param partition
 * @return int
*/
 public static int getIntFromPartition(String partition){
  int result=-1;
  int pre=PARTITION_PREFIX.length();
  int p=partition.length();
  String num=partition.substring(pre, p);
  result=Integer.parseInt(num);
  return result;
}
/**
 * Retrieve the number part of stock ticket
*
 * @param ticket
* @return
*/
 public static int getIntFromStockTicket(String ticket){
  int result=-1;
  int pre=STOCK_PREFIX.length();
  int p=ticket.length();
  String num=ticket.substring(pre, p);
  result=Integer.parseInt(num);
  return result;
}
/**
 * Hash stock ticket to a given hash base.
*
 * @param ticket
 * @param base
 * @return int
*/
 public static int hashTicket(String ticket, int base){
  if (base<1){
   return 0;
}
  int hash=0;
  int num=getIntFromStockTicket(ticket);
  hash= num % base;
  return hash;
}
/**
 * Hash stock key to a partition
*
 * @param ticket
 * @param base
 * @return String - partition name
*/
```

```
 public static String hashStockKeyToPartition(String ticket, int base){
  String p=null;
  int hashcode=hashTicket(ticket, base)+1;
p=PARTITION_PREFIX+ padZeroToString(hashcode+"", 6);
  return p;
}
/**
 * Record server/partition
*
 * @param server
  * @param partition
*/
 public static void addServer(String server, String partition){
  serverPartitions.put(server, partition);
}
/**
 * Remove server/partition
*
 * @param server
*/
 public static void removeServer(String server){
  serverPartitions.remove(server);
}
/**
 * Get all servers where partitions are active.
*
 * @return Iterator - String
*/
 public static Iterator getAllServer(){
  return serverPartitions.values().iterator();
}
}
```

동일한 글로벌 해시 기본 및 구문 분석 변수를 해시 기본에 해시하도록 사용해야 합니다. 다음 예를 고려하십시오.

```
myKey.hashCode % hashBase
```

myKey를 해시 변수로 구문 분석하고 서로 다른 서버에서 동일한 해시 기본을 사용해야 합니다. 위 예의 경우 Java 환경에서 동일한 변수를 찾습니다. key1 % 100은 사용 불가능하지만 key2 % 90은 사용 가능합니다.

엔터프라이즈 Bean setContext 메소드에서 오브젝트 그리드 작성

엔터프라이즈 Bean setContext 메소드에서 오브젝트 그리드 인스턴스를 PFClusterObjectGridEJBBean.java 파일에서와 같이 작성한 후 사전 로드 데이터를 검색하십시오.

```
/**
* setSessionContext
*
* with ObjectGrid instance
*/
public void setSessionContext(javax.ejb.SessionContext ctx) {
 mySessionCtx = ctx;
try {
```

```
  InitialContext ic = new InitialContext();
//get PartitionManager
  ivManager = (PartitionManager)
ic.lookup("java:comp/websphere/wpf/PartitionManager");
  // get enableDistribution configuration
  boolean enableDistribution = ((Boolean)
ic.lookup("java:comp/env/enableDistribution")).booleanValue();
System.out.println("***** enableDistribution="+ enableDistribution);
  // get propagationMode configuration
String propagationMode = (String) ic.lookup("java:comp/env/propagationMode");
System.out.println("***** pMode="+ propagationMode);
  String pMode=null;
  if (propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_DEFAULT_KEY)||
   propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_INVALID_KEY) ){
pMode=propagationMode;
}
  // get propagationVersionOption configuration
  String propagationVersionOption = (String)
ic.lookup("java:comp/env/propagationVersionOption");
System.out.println("***** pVersionOption="+ propagationVersionOption);
  String pVersion=null;
  if (propagationVersionOption.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_VERS_KEY)||
   propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_NOVERS_KEY) ){
pVersion=propagationVersionOption;
}
  // get compressionMode configuration
String compressionMode = (String) ic.lookup("java:comp/env/compressionMode");
System.out.println("***** compressMode="+ compressionMode);
  String compressMode=null;
  if (compressionMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_COMPRESS_DISABLED)||
   propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_COMPRESS_ENABLED) ){
compressMode=compressionMode;
}
  // whethere preload is enabled
  bPreload = ((Boolean)
ic.lookup("java:comp/env/preload")).booleanValue();
System.out.println("***** enablePreload="+ bPreload);
  //whethere remove is enabled
  bRemove = ((Boolean)
ic.lookup("java:comp/env/remove")).booleanValue();
System.out.println("***** enableRemove="+ bRemove);
  // whethere Loader is enabled
  boolean bLoader = ((Boolean)
ic.lookup("java:comp/env/loader")).booleanValue();
System.out.println("***** enableLoader="+ bLoader);
  // get file path and name
  String filePathandName = (String)
ic.lookup("java:comp/env/filePathandName");
System.out.println("***** fileName="+ filePathandName);
  //get ObjectGrid instance
  og=ObjectGridFactory.getObjectGrid(ogName,
enableDistribution,  pMode, pVersion,
  compressMode, bLoader,
  filePathandName);
  if (og==null){
   throw new RuntimeException
```

```
("ObjectGrid insance is null in ObjectGridPartitionClusterSample");
}
System.out.println("Bean Context, getObjectGrid="
+ og + " for name="+ ogName);
  if (bPreload && !lock){
System.out.println("Preload data");
   PersistentStore store=PersistentStore.getStore(filePathandName);
   store.preload(10);
   store.verify(10);
lock=true;
   preloadData=store.getAllRecords();
}
}
catch (Exception e) {
  logger.logp(Level.SEVERE, CLASS_NAME,
"setSessionContext", "Exception: " + e);
  throw new EJBException(e);
}
}
```

싱글톤 ObjectGridFactory 클래스

오브젝트 그리드 인스턴스는 사용자 정의 설정을 사용하여 오브젝트 그리드 인스턴스를 캐시하는 사용자 정의 팩토리에서 작성됩니다.

프로그램 방식으로 오브젝트 그리드 인스턴스를 작성하는 방법에 대한 예에서 다음과 같이 ObjectGridTransformer 오브젝트를 설정하고 전달 이벤트 리스터를 구성하며 이 리스너를 오브젝트 그리드 인스턴스로 설정하십시오. 이 구성을 수행하도록 XML 파일을 사용할 수도 있습니다.

```
/**
*
* Create ObjectGrid instance and configure it.
*
*
*/
public class ObjectGridFactory {
/**
 * ObjectGrid name
*/
static String ogName="WPFObjectGridSample";
/**
 * ObjectGrid instance
*/
 static ObjectGrid og=null;
/**
 * ObjectGrid session
*/
 static Session ogSession=null;
/**
 * Map name
*/
static String mapName="SampleStocks";
/**
 * ObjectGrid cache
*/
 static Map ogCache= new HashMap();
```

```
/**
 * Get ObjectGrid instance
*
 * @param ogn
 * @param enableDist
 * @param pMode
 * @param pVersion
 * @param compressMode
* @return
*/
public static synchronized ObjectGrid getObjectGrid(String ogn,
 boolean enableDist,
 String pMode,
 String pVersion,
 String compressMode,
 boolean loader,
 String fileName){
  if (ogn!=null){
ogName=ogn;
}
else {
throw new IllegalArgumentException ("ObjectGrid name given is null");
}
  if (ogCache.containsKey(ogName)){
   return (ObjectGrid) ogCache.get(ogName);
}
try {
   ObjectGridManager manager= ObjectGridManagerFactory.
getObjectGridManager();
og=manager.createObjectGrid(ogName);
if (enableDist){
 TranPropListener tpl=new TranPropListener();
 if (pMode!=null){
  tpl.setPropagateMode(pMode);
}
 if (pVersion!=null){
  tpl.setPropagateVersionOption(pVersion);
}
 if (compressMode!=null) {
  tpl.setCompressionMode(compressMode);
}
 og.addEventListener(tpl);
}
   // Define BackingMap and set the Loader
  BackingMap bm = og.defineMap(mapName);
ObjectTransformer myTransformer=
  new MyStockObjectTransformer();
  bm.setObjectTransformer(myTransformer);
OptimisticCallback myOptimisticCallback=
  new MyStockOptimisticCallback();
  if (loader){
   TransactionCallback tcb=new MyTransactionCallback();
   Loader myLoader= new MyCacheLoader(fileName, mapName);
   og.setTransactionCallback(tcb);
   bm.setLoader(myLoader);
}
  og.initialize();
  ogCache.put(ogName, og);
```

```
}
catch (Exception e) {
}
 return og;
}
}
```

오브젝트 그리드 파티션 사전 로드

이 주제에서는 오브젝트 그리드 인스턴스를 사전 로드하는 방법을 논의합니다.

partitionLoadEvent 메소드를 사용하여 파티션이 활성화된 경우에만 이 파티션과 관련된 오브젝트를 로드하십시오. 파티션이 활성화된 경우 오브젝트를 로드하면 오브젝트 그리드가 많은 오브젝트를 처리할 수 있도록 오브젝트 그리드를 파티션합니다.

```
/**
* This is called when a specific partition is assigned to this server process.
* @param partitionName
* @return
*/
public boolean partitionLoadEvent(String partitionName) {
//preload data
 preloadDataForPartition(partitionName);
logger.logp(
Level.FINER,
CLASS_NAME,
"partitionLoadEvent",
"Loading "+ partitionName );
  return true;
}
/**
*
* preload data
*
* @param partition
*/
private synchronized void preloadDataForPartition(String partition){
 if (bPreload && (preloadData!=null)){
  Iterator itr=preloadData.keySet().iterator();
  while (itr.hasNext()){
   String ticket= (String) itr.next();
   String p=SampleUtility.
   hashStockKeyToPartition(ticket, numOfPartitions);
   if (partition.equals(p)){
    Stock stock= (Stock) preloadData.get(ticket);
System.out.println("preload in partition=" +
partition + " with data ticket="+ ticket);
    insertStock(stock);
}
}
}
}
```

대형 오브젝트 그리드의 파티션된 사전 로드를 사용하여 대형 오브젝트 그리드를 파티션하는 경우 분배 갱신을 사용 불가능하도록 할 수도 있습니다. 분배 갱신의 현재 버전은 파티션할 수 없습니다. 파티션 기능(WPF) 컨텍스트 기반 라우팅에서는 올바른 파티션에서 올바른 데이터를 찾습니다.

컨테이너 관리 Bean에 대해 작업하도록 오브젝트 그리드 구성

WebSphere Application Server 버전 6.0.2 이상에서는 CMP(Container-Managed Persistence) Bean을 외부 캐시 제품과 함께 사용할 수 있습니다.

이 타스크를 사용하여 CMP Bean을 사용함으로써 오브젝트 그리드를 내장 캐시 대신 외부 캐시로 사용하십시오. 이 기능은 WebSphere Application Server의 지속 엔진에서 제공합니다.

1. JVM 인수를 정의하여 CacheFactoryManager 어댑터 및 오브젝트 그리드 XML 구성 파일의 위치를 정의하십시오. CacheFactoryManager는 지속 엔진과 오브젝트 그리드 사이의 어댑터입니다.

   a. **서버 > Application Server >** *server_name* **> Java 및 프로세스 관리 > 프로세스 정의 > JVM(Java Virtual Machine) > 일반 JVM 인수**를 클릭하십시오.

   b. 다음 특성을 추가하십시오.

      - `-Dcom.ibm.ws.pmcache.manager=com.ibm.ws.objectgrid.adapter.pm.CacheFactoryManager`
      - `-Dcom.ibm.ws.pmcache.config=file:/d:/temp/objectGrid.xml`

      `-Dcom.ibm.ws.pmcache.config` 특성은 오브젝트 그리드의 구성 파일을 지정합니다. 값은 오브젝트 그리드 구성 파일에 대한 URL입니다.

2. 오브젝트 그리드 XML 구성 파일을 구성하십시오. 구성은 `objectGrid.xml` 파일에 있습니다. 다음 예를 고려하십시오. 응용프로그램 및 모듈 정보는 J2EE(Java 2 Platform, Enterprise Edition) 응용프로그램에 필요합니다. 정보는 `objectGrid.xml` 파일에 반영됩니다. 계정 응용프로그램에는 Savings, Checkin 및 MoneyMarket과 같은 세 개의 CMP Enterprise JavaBeans가 있습니다. 이 Enterprise JavaBeans는 PersonalBankingEJB 모듈에 포함되어 있습니다. 표시 이름은 *Accounts*이고 *PersonalBankingEJB*는 응용프로그램 전개 설명자의 EJB 모듈입니다. Savings, Checkin 및 MoneyMarket은 컨테이너 관리 엔티티 Beans(CMP)의 Enterprise Java Bean 전개 설명자 ejb-name 요소에 지정된 이름입니다. 이 구성의 샘플 스니펫은 다음과 같습니다.

   ```
   <ObjectGrids>
   <ObjectGrid name="Accounts">
   <BackingMap name="PersonalBankingEJB.jar#Savings" readOnly="true"
   pluginCollectionRef="default" />
   <BackingMap name="PersonalBankingEJB.jar#Checkin" readOnly="true"
   pluginCollectionRef="default" />
   <BackingMap name="PersonalBankingEJB.jar#MoneyMarket" readOnly="true"
   pluginCollectionRef="default" />
   </ObjectGrid>
   </ObjectGrids>
   ```

   `PersonalBankingEJB.jar` 파일은 다음 예와 같이 응용프로그램 전개 설명자의 EJB 태그 내에 지정됩니다.

```
<module id="module_1">
<ejb>PersonalBankingEJB.jar</ejb>
</module>
```

3. 응용프로그램 내에서 각 Bean의 지속 관리자 LifeTimeInCache 설정을 사용 가능하게 하여 외부 캐시를 사용하십시오. 오브젝트 그리드의 경우 전개 설명자에서 이 설정을 사용 가능하게 해야 하지만 LifeTimeInCache 설정을 무시합니다. 오브젝트 그리드 구성에는 우선순위가 있습니다.
4. 캐시에서 오브젝트를 축출하도록 명시적으로 backingMap을 구성하십시오. objectGrid.xml 파일의 XML 코드 스니펫은 다음과 같습니다.

```
<backingMapPluginCollection id="TotalTimeToLive">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.TTLEvictor">
<property name="pruneSize" type="int" value="2"
description="set max size for TTL Evictor" />
<property name="numberOfHeaps" type="int" value="1"
description="set number of TTL heaps" />
<property name="sleepTime" type="int" value="1"
description="evictor thread sleep time" />
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="LifeTimeInCache">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.TTLEvictor">
<property name="lifeTime" type="int" value="3"
description="lifetime of map entry is 3 seconds" />
<property name="pruneSize" type="int" value="2"
description="set max size for TTL Evictor" />
<property name="numberOfHeaps" type="int" value="1"
description="set number of TTL heaps" />
<property name="sleepTime" type="int" value="1"
description="evictor thread sleep time" />
</bean>
</backingMapPluginCollection>
```

   lifeTime 특성은 축출기를 제어합니다.

WebSphere Application Server와 오브젝트 그리드를 사용하는 방법에 대한 자세한 정보는 313 페이지의 제 10 장 『WebSphere Application Server에서 오브젝트 그리드 통합』을 참조하십시오.

제 11 장 오브젝트 그리드 성능 우수 사례

다음의 우수 사례를 통해 오브젝트 그리드 Map 성능을 향상시킬 수 있습니다. 이 우수 사례는 응용프로그램 및 해당 아키텍처의 컨텍스트에서만 구현됩니다.

모든 응용프로그램 및 환경은 성능과 관련하여 서로 다른 솔루션을 사용합니다. 오브젝트 그리드는 성능을 향상시키기 위해 내장 사용자 정의를 제공하지만, 응용프로그램 아키텍처에서도 성능을 향상시킬 수 있습니다. 다음 영역에서 성능 향상을 제공합니다.

- 『잠금 성능 우수 사례』

  응용프로그램 성능에 영향을 줄 수 있는 서로 다른 잠금 계획 사이에서 선택.

- 358 페이지의 『copyMode 메소드 우수 사례』

  오브젝트 그리드에서 항목을 유지보수 및 복사하는 방법을 변경할 때 사용 가능한 서로 다른 복사 모드 사이에서 선택.

- 363 페이지의 『ObjectTransformer 인터페이스 우수 사례』

  응용프로그램에 대한 콜백을 통해 오브젝트에서 완전 사본 및 오브젝트 직렬화와 같은 공통된 확장 조작의 사용자 정의 구현을 제공하도록 ObjectTransformer 인터페이스 사용.

- 365 페이지의 『플러그인 축출기 성능 우수 사례』

  가장 적게 사용(LFU) 및 가장 오래 전에 사용(LRU) 축출 계획 사이에서 선택.

- 367 페이지의 『기본 축출기 우수 사례』

  모든 backingMap에서 작성된 기본 축출기인 기본 TTL(Time to Live) 축출기의 특성.

잠금 성능 우수 사례

잠금 계획은 응용프로그램 성능에 영향을 줄 수 있습니다.

다음의 잠금 계획에 대한 추가 정보는 140 페이지의 『잠금』 주제를 참조하십시오.

잠금 중 변경이 예상되는 계획

키가 자주 충돌하는 경우 읽기 및 쓰기 맵 작업에서 잠금 중 변경이 예상되는 계획을 사용할 수 있습니다. 잠금 중 변경이 예상되는 계획은 성능에 큰 영향을 줍니다.

잠금 중 변경이 예상되지 않는 계획

잠금 중 변경이 예상되지 않음이 기본 구성입니다. 이 계획은 변경이 예상되는 계획보다 성능 및 확장성을 향상시킵니다. 변경이 예상되는 계획보다 성능이 더 좋지만 응용

프로그램에서 일부 갱신 중 변경이 예상되지 않는 장애를 허용할 수 있는 경우 이 계획을 사용하십시오. 이 계획은 대부분 읽는 데 사용되고 거의 갱신되지 않는 응용프로그램에 더 적합합니다.

비잠금 계획

비잠금 계획 사용은 읽기 전용 응용프로그램에 적합합니다. 비잠금 계획은 잠금을 얻지 않습니다. 따라서 동시성, 성능 및 확장성을 극대화합니다.

copyMode 메소드 우수 사례

오브젝트 그리드는 CopyMode 설정에 따라 값을 복사합니다. BackingMap API setCopyMode(CopyMode, valueInterfaceClass) 메소드를 사용하여 com.ibm.websphere.objectgrid.CopyMode에 정의된 다음의 final static 필드 중 하나로 복사 모드를 설정할 수 있습니다.

응용프로그램에서 ObjectMap 인터페이스를 사용하여 맵 값에 대한 참조를 얻은 경우 참조를 얻은 오브젝트 그리드 트랜잭션에서만 해당 참조를 사용하는 것이 좋습니다. 다른 오브젝트 그리드 트랜잭션에서 참조를 사용하면 오류가 발생할 수 있습니다. 예를 들어 BackingMap에서 잠금 중 변경이 예상되는 계획을 사용하는 경우 get 또는 getForUpdate 메소드 호출로 각각 S(공유) 또는 U(갱신) 잠금을 확보합니다. get 메소드는 값에 대한 참조를 리턴하고 이때 얻은 잠금은 트랜잭션을 완료할 때 해제됩니다. 다른 트랜잭션에서 맵 항목을 잠그려면 트랜잭션에서 get 또는 getForUpdate 메소드를 호출해야 합니다. 각 트랜잭션은 다중 트랜잭션에서 동일한 값 참조를 다시 사용하는 대신 get 또는 getForUpdate 메소드를 호출하여 값에 대한 고유한 참조를 얻어야 합니다.

다음 정보를 사용하여 다음 정보를 포함하는 복사 모드 사이에서 선택하십시오.

COPY_ON_READ_AND_COMMIT 모드

COPY_ON_READ_AND_COMMIT 모드는 기본 모드입니다. 이 모드를 사용할 때 valueInterfaceClass 인수는 무시됩니다. 이 모드를 사용하면 응용프로그램은 BackingMap에 있는 값 오브젝트에 대한 참조를 포함하지 않습니다. 대신 응용프로그램은 항상 BackingMap에 있는 값의 사본을 사용합니다. COPY_ON_READ_AND_COMMIT 모드를 사용하면 응용프로그램에서 BackingMap에 캐시된 데이터가 우연히 손상될 수 없습니다. 응용프로그램 트랜잭션은 지정된 키에서 ObjectMap.get 메소드를 호출하고 이때 해당 키의 ObjectMap 항목에 처음으로 액세스하는 경우 값의 사본을 리턴합니다. 트랜잭션을 확약하면 응용프로그램에서 확약한 모든 변경사항이 BackingMap에 복사되므로 응용프로그램에는 BackingMap에서 확약된 값에 대한 참조가 없습니다.

COPY_ON_READ 모드

COPY_ON_READ 모드는 트랜잭션을 확약할 때 복사를 수행하지 않아도 되므로 COPY_ON_READ_AND_COMMIT 모드에 비해 성능을 더 향상시킵니다. 이 모드를 사용할 때 valueInterfaceClass 인수는 무시됩니다. BackingMap 데이터의 무결성을 보존하려면 응용프로그램에서 트랜잭션을 확약한 후 응용프로그램에 있는 항목의 모든 참조를 제거해야 합니다. 이 모드를 사용하면 ObjectMap.get 메소드를 값에 대한 참조 대신 값의 사본을 리턴하므로 응용프로그램이 값에서 변경한 사항은 트랜잭션을 확약할 때까지 BackingMap 값에 영향을 주지 않습니다. 그러나 트랜잭션을 확약하면 변경사항을 복사하지 않습니다. 대신 ObjectMap.get 메소드에서 리턴하는 사본에 대한 참조가 BackingMap에 저장됩니다. 트랜잭션을 확약하면 응용프로그램은 모든 맵 항목 참조를 제거합니다. 응용프로그램에서 해당 참조를 제거하는 데 실패하면 BackingMap에 캐시된 데이터가 손상될 수 있습니다. 응용프로그램이 이 모드를 사용하는 경우 문제점이 발생하면 COPY_ON_READ_AND_COMMIT 모드로 전환하여 문제점이 지속되는지 확인하십시오. 문제점이 발생하지 않으면 응용프로그램은 트랜잭션을 확약한 후 모든 해당 참조를 제거하지 않습니다.

COPY_ON_WRITE 모드

COPY_ON_WRITE 모드는 지정된 키에서 트랜잭션이 처음으로 ObjectMap.get 메소드를 호출할 때 복사를 수행하지 않아도 되므로 COPY_ON_READ_AND_COMMIT 모드에 비해 성능을 더 향상시킵니다. ObjectMap.get 메소드는 값 오브젝트에 대한 직접 참조 대신 값에 대한 프록시를 리턴합니다. 프록시는 valueInterfaceClass 인수가 지정한 값 인터페이스에서 응용프로그램이 set 메소드를 호출하지 않는 한 값을 복사하지 않게 합니다. 프록시는 *쓰기 중 복사(copy on write)* 구현을 제공합니다. 트랜잭션을 확약하면 BackingMap은 프록시를 점검하여 set 메소드를 호출하여 복사되었는지 여부를 판별합니다. 복사된 경우 해당 사본에 대한 참조는 BackingMap에 저장됩니다. 이 모드의 큰 장점은 트랜잭션에서 값을 변경하도록 set 메소드를 호출하지 않는 경우 확약하거나 읽을 때 값을 복사하지 않는다는 점입니다.

COPY_ON_READ_AND_COMMIT 및 COPY_ON_READ 모드 모두 값을 ObjectMap에서 검색한 경우 완전 사본을 수행합니다. 응용프로그램이 트랜잭션에서 검색한 일부 값만 갱신하는 경우 이 모드는 최적의 모드가 아닙니다. COPY_ON_WRITE 모드는 효과적인 방식으로 이 동작을 지원하지만 응용프로그램에서 단순 패턴을 사용해야 합니다. 값 오브젝트는 인터페이스를 지원하는 데 필요합니다. 오브젝트 그리드 Session에서 값과 상호 작용할 때 응용프로그램은 이 인터페이스에서 메소드를 사용해야 합니다. 이 경우 오브젝트 그리드는 응용프로그램에 리턴되는 값에 대한 프록시를 작성합니다. 프록시에는 실제 값에 대한 참조가 들어 있습니다. 응용프로그램이 읽기만 하는 경우 항상 실제 복사와 충돌합니다. 응용프로그램이 오브젝트에서 속성을 수정하면 프록시는 실제 오브젝트를 복사하므로 해당 사본을 수정하게 됩니다. 그 다음 프록시는 이때부터 해당 사본을 사용합니다. 그러면 응용프로그램에서 읽기만 하는 오브젝

트에서 복사 조작을 완전히 방지할 수 있습니다. 모든 수정 조작은 set 접두부로 시작되어야 합니다. Enterprise JavaBeans는 정상적으로 오브젝트 속성을 수정하는 메소드의 이 메소드 이름 지정 양식을 사용하여 코드화됩니다. 이 규칙을 준수해야 합니다. 수정된 모든 오브젝트는 응용프로그램에서 수정될 때마다 복사됩니다. 이 방법은 오브젝트 그리드에서 지원되는 가장 효과적인 읽기 및 쓰기 시나리오입니다. 다음과 같이 COPY_ON_WRITE을 사용하여 맵을 구성할 수 있습니다. 이 예에서 응용프로그램은 Map에 이름을 사용하여 키로 지정된 Person 오브젝트를 저장하려고 합니다. Person 오브젝트는 다음 코드 스니펫과 유사합니다.

```
class Person
{
 String name;
 int age;
 public Person()
{
}
 public void setName(String n)
{
  name = n;
}
 public String getName()
{
  return name;
}
 public void setAge(int a)
{
  age = a;
}
 public int getAge()
{
  return age;
}
}
```

응용프로그램은 ObjectMap에서 검색한 값과 상호 작용하는 경우에만 IPerson 인터페이스를 사용합니다. 다음 예와 같이 인터페이스를 사용하도록 오브젝트를 수정하십시오.

```
interface IPerson
{
 void setName(String n);
 String getName();
 void setAge(int a);
 int getAge();
}
// Modify Person to implement IPerson interface
class Person implements IPerson
{
...
}
```

그러면 응용프로그램은 다음 예와 같이 COPY_ON_WRITE 모드를 사용하여 BackingMap을 구성해야 합니다.

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// use COPY_ON_WRITE for this Map with
// IPerson as the valueProxyInfo Class
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// The application should then use the following
// pattern when using the PERSON Map.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// the application casts the returned value to IPerson and not Person
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// make a new Person and add to Map
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// the following snippet WON'T WORK. Will result in ClassCastException
sess.begin();
// the mistake here is that Person is used rather than
// IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

첫 번째 섹션은 맵에서 이름이 Billy인 값을 검색하는 응용프로그램을 표시합니다. 응용프로그램은 리턴된 값을 Person 오브젝트가 아닌 IPerson 오브젝트에 캐스트합니다. 리턴된 프록시가 다음과 같은 두 개의 인터페이스를 구현하기 때문입니다.

- BackingMap.setCopyMode 메소드 호출에 지정된 인터페이스
- com.ibm.websphere.objectgrid.ValueProxyInfo 인터페이스

두 개 유형으로 프록시를 캐스트할 수 있습니다. 이전 코드 스니펫의 마지막 파트에서는 COPY_ON_WRITE 모드에서 허용되지 않는 사항을 보여줍니다. 응용프로그램은 Bobby 레코드를 검색하여 Person 오브젝트에 캐스트하려고 합니다. 리턴된 프록시는 Person 오브젝트가 아니므로 클래스 캐스트 예외로 이 조치에 실패합니다. 리턴된 프록시는 IPerson 오브젝트 및 ValueProxyInfo를 구현합니다.

ValueProxyInfo 인터페이스 및 부분 갱신 지원

이 인터페이스를 사용하면 응용프로그램에서 프록시에서 참조하는 확약된 읽기 전용 값 또는 이 트랜잭션 중 수정된 속성 세트를 검색할 수 있습니다.

```
public interface ValueProxyInfo
{
List /**/ ibmGetDirtyAttributes();
Object ibmGetRealValue();
}
```

ibmGetRealValue 메소드는 오브젝트의 읽기 전용 사본을 리턴합니다. 응용프로그램에서는 이 값을 수정할 수 없습니다. ibmGetDirtyAttributes 메소드는 이 트랜잭션 중 응용프로그램에서 수정된 속성을 표시하는 문자열 목록을 리턴합니다. ibmGetDirtyAttributes는 JDBC(Java database connectivity) 또는 CMP 기반 로더에서 주로 사용됩니다. 목록에 이름 지정된 속성만 테이블에 맵핑된 오브젝트 또는 SQL 문에서 갱신되어야 합니다. 그러면 로더에서 보다 효과적으로 SQL을 생성할 수 있습니다. 쓰기 트랜잭션에서 복사를 확약하고 Loader가 플러그인된 경우 로더가 수정된 오브젝트의 값을 ValueProxyInfo 인터페이스에 캐스트하여 이 정보를 얻을 수 있습니다.

COPY_ON_WRITE 또는 프록시 사용 시 equals 메소드 처리

예를 들어 다음 코드에서는 Person 오브젝트를 구성한 후 ObjectMap에 삽입합니다. 그 다음 ObjectMap.get 메소드를 사용하여 동일한 오브젝트를 검색합니다. 값은 인터페이스에 캐스트됩니다. 값이 Person 인터페이스에 캐스트되면 리턴된 값이 Person 오브젝트가 아닌 IPerson 인터페이스를 구현하는 프록시이므로 ClassCastException 예외가 발생합니다. == 조작을 사용하는 경우 동일한 오브젝트가 아니므로 동일성 확인에 실패합니다.

```
session.begin();
// new the Person object
Person p = new Person(...);
personMap.insert(p.getName, p);
// retrieve it again, remember to use the interface for the cast
IPerson p2 = personMap.get(p.getName());
if(p2 == p)
{
// they are the same
}
else
{
// they are not
}
```

equals 메소드를 대체해야 하는 경우 또다른 고려사항이 있습니다. 다음 코드 스니펫에서 표시한 대로 equals 메소드는 인수가 IPerson 인터페이스를 구현하는 오브젝트인지 확인한 후 해당 인수를 IPerson에 캐스트해야 합니다. 인수가 IPerson 인터페이스를 구현하는 프록시일 수 있으므로 인스턴스 변수가 동일한지 비교할 때 getAge 및 getName 메소드를 사용해야 합니다.

```
public boolean equals(Object obj)
{
 if ( obj == null ) return false;
 if ( obj instanceof IPerson )
{
  IPerson x = (IPerson) obj;
  return ( age.equals( x.getAge() ) && name.equals( x.getName() ) )
}
 return false;
}
```

NO_COPY 모드

NO_COPY 모드를 사용하면 응용프로그램에서 성능 향상을 위해 ObjectMap.get 메소드를 사용하여 얻은 값 오브젝트를 수정하지 않게 할 수 있습니다. 이 모드를 사용할 때 valueInterfaceClass 인수는 무시됩니다. 이 모드를 사용하면 값은 복사되지 않습니다. 응용프로그램이 값을 수정하면 BackingMap의 데이터가 손상됩니다. NO_COPY 모드는 기본적으로 응용프로그램에서 데이터를 수정하지 않는 읽기 전용 맵에서 유용합니다. 응용프로그램이 이 모드를 사용하는 경우 문제점이 발생하면 COPY_ON_READ_AND_COMMIT 모드로 전환하여 문제점이 지속되는지 확인하십시오. 문제점이 발생하지 않으면 응용프로그램은 트랜잭션 중 또는 트랜잭션을 확약한 후 ObjectMap.get 메소드가 리턴하는 값을 수정합니다.

ObjectTransformer 인터페이스 우수 사례

ObjectTransformer는 응용프로그램에 대한 콜백을 사용하여 오브젝트에서 완전 사본 및 오브젝트 직렬화와 같은 공통된 확장 조작의 사용자 정의 구현을 제공합니다.

ObjectTransformer 인터페이스에 대한 특정 세부사항은 229 페이지의 『ObjectTransformer 플러그인』 주제를 참조하십시오. 358 페이지의 『copyMode 메소드 우수 사례』 주제에 있는 CopyMode 메소드에 대한 정보 및 성능 관점에 따라 NO_COPY 모드인 경우를 제외한 모든 경우에 오브젝트 그리드가 값을 복사하는 것이 확실합니다. 오브젝트 그리드에서 사용하는 기본 복사 메커니즘은 직렬화(확장 조작이라고도 함)입니다. ObjectTransformer 인터페이스는 이 상황에서 사용될 수 있습니다. ObjectTransformer 인터페이스는 응용프로그램에 대한 콜백을 사용하여 오브젝트에서 완전 사본 및 오브젝트 직렬화와 같은 공통된 확장 조작의 사용자 정의 구현을 제공합니다.

응용프로그램은 맵에서 ObjectTransformer 인터페이스 구현을 제공할 수 있습니다. 그러면 오브젝트 그리드는 이 오브젝트에서 메소드를 위임하고 응용프로그램에 의존하여 인터페이스에서 각 메소드의 최적화된 버전을 제공합니다. ObjectTransformer 인터페이스는 다음과 같습니다.

```
public interface ObjectTransformer
{
 void serializeKey(Object key, ObjectOutputStream stream)
 throws IOException;
 void serializeValue(Object value, ObjectOutputStream stream)
 throws IOException;
 Object inflateKey(ObjectInputStream stream)
 throws IOException, ClassNotFoundException;
 Object inflateValue(ObjectInputStream stream)
 throws IOException, ClassNotFoundException;
 Object copyValue(Object value);
 Object copyKey(Object key);
}
```

다음 코드 예를 사용하여 ObjectTransformer 인터페이스를 BackingMap과 연관시킬 수 있습니다.

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

오브젝트 직렬화(serialization) 및 확대(inflation) 조정

일반적으로 오브젝트 직렬화는 오브젝트 그리드에서 성능을 가장 많이 향상시키는 항목입니다. 오브젝트 그리드는 응용프로그램에서 ObjectTransformer 플러그인을 제공하지 않는 경우 기본 직렬화 가능 메커니즘을 사용합니다. 응용프로그램은 직렬화 가능한 readObject 및 writeObject의 구현을 제공할 수 있습니다. 또는 오브젝트가 Externalizable 인터페이스를 구현하게 할 수 있습니다. 이렇게 하면 약 10배가 더 빠릅니다. Map의 오브젝트를 수정할 수 없으면 응용프로그램은 ObjectTransformer를 ObjectMap과 연관시킬 수 있습니다. serialize 및 inflate 메소드를 제공하면 시스템에 큰 영향을 주는 이와 같은 조작을 최적화하는 사용자 정의 코드를 응용프로그램에서 제공할 수 있습니다. serialize 메소드는 오브젝트를 직렬화하고 스트림을 제공합니다. 이 메소드는 제공된 스트림으로 메소드를 직렬화합니다. inflate 메소드는 입력 스트림을 제공하고 응용프로그램의 오브젝트 작성을 예상하며 스트림에서 데이터를 사용하여 오브젝트를 확대한 후 오브젝트를 리턴합니다. serialize 및 inflate 메소드를 구현하려면 서로를 미러링해야 합니다.

완전 사본 조작 조정

응용프로그램이 ObjectMap에서 오브젝트를 수신하면 오브젝트 그리드는 오브젝트 값에서 완전 사본을 수행하여 BaseMap 맵에 사본이 안전하게 남아 있도록 합니다. 그러면 응용프로그램은 오브젝트 값을 안전하게 수정할 수 있습니다. 트랜잭션을 확약하면 BaseMap 맵에 있는 오브젝트 값의 사본은 새로 수정된 값으로 갱신되고 응용프로그램은 이 시점부터 값 사용을 중지합니다. 확약 단계에서 다시 오브젝트를 복사하여 개인용 사본으로 작성할 수 있지만 이 경우 트랜잭션을 확약한 후 응용프로그램 프로그래머에게 값을 사용하지 않음을 알리므로 이 조치로 성능이 저하됩니다. 기본 오브젝트 복사 메커니즘에서는 사본을 생성하도록 복제본 또는 serialize 및 inflate 쌍을 사용하려고 합니다. serialize 및 inflate 쌍은 최악의 성능 시나리오입니다. 프로파일링에서 serialize 및 inflate가 응용프로그램에서 문제점임이 밝혀지면 ObjectTransformer 플러그인을 제공하고 더 효과적인 오브젝트 복사를 사용하여 `copyValue` 및 `copyKey` 메소드를 구현하십시오.

플러그인 축출기 성능 우수 사례

플러그인 축출기를 사용하는 경우 해당 축출기를 작성하여 BackingMap에 축출기 사용을 알릴 때까지 축출기는 활성화되지 않습니다. 가장 적게 사용(LFU) 및 가장 오래 전에 사용(LRU) 축출기에 대한 우수 사례 및 성능 팁을 사용하십시오.

가장 적게 사용(LFU) 축출기

LFU 축출기의 개념은 자주 사용하지 않는 항목을 맵에서 제거하는 것입니다. 맵의 항목은 설정된 2진 힙에 분산되어 있습니다. 특정 캐시 항목의 사용이 증가하면 힙에서 더 높은 순서로 정렬됩니다. 축출기가 일련의 축출을 시도할 때 2진 힙의 특정 지점보다 아래에 위치한 캐시 항목만 제거됩니다. 결과적으로 가장 적게 사용한 항목이 축출됩니다.

가장 오래 전에 사용(LRU) 축출기

LRU 축출기는 약간의 차이는 있지만 LFU 축출기과 동일한 개념을 준수합니다. 주된 차이는 LRU가 2진 힙 세트 대신 선입선출(FIFO) 대기열을 사용한다는 점입니다. 캐시 항목에 액세스할 때마다 대기열 헤드로 이동합니다. 결과적으로 대기열 맨 앞에 가장 최근에 사용한 맵 항목이 오고 맨 끝은 가장 오래 전에 사용한 맵 항목입니다. 예를 들어, A 캐시 항목을 50회 사용하고 A 캐시 항목 이후 바로 B 캐시 항목을 한 번만 사용했습니다. 이 상황에서, B 캐시 항목 가장 최근에 사용했으므로 B 캐시 항목은 대기열 맨 앞에, A 캐시 항목은 대기열 맨 끝에 옵니다. LRU 축출기는 대기열 맨 끝에 있는 캐시 항목(가장 오래 전에 사용한 맵 항목)을 축출합니다.

성능을 향상시키는 LFU 및 LRU 특성과 우수 사례

힙 수 LFU 축출기를 사용하는 경우 특정 맵의 모든 캐시 항목은 사용자가 지정한 힙 수대로 정렬됩니다. 그러면 성능을 크게 향상시키고 맵의 모든 순서를 포함하는 하나의 2진 힙에서 모든 축출이 동기화되지 않도록 방지할 수 있습니다. 또한 힙이 많을수록 각 힙에서 항목이 적어지므로 힙을 다시 정렬하는 데 필요한 시간을 가속화합니다. 힙 수를 BaseMap에 있는 항목 수의 10%로 설정하십시오.

대기열 수

LRU 축출기를 사용하는 경우 특정 맵의 모든 캐시 항목은 사용자가 지정한 LRU 대기열 수대로 정렬됩니다. 그러면 성능을 크게 향상시키고 맵의 모든 순서를 포함하는 하나의 대기열에서 모든 축출이 동기화되지 않도록 방지할 수 있습니다. 대기열 수를 BaseMap에 있는 항목 수의 10%로 설정하십시오.

MaxSize 특성

LFU 또는 LRU 축출기에서 항목 축출을 시작하면 MaxSize 축출기 특성을 사용하여 축출할 2진 힙 또는 LRU 대기열 요소 수를 판별합니다. 예를 들어, 힙 또는 대기열 수를 각 맵 대기열에서 약 10개의 맵 항목을 포함하도록 설정한

다고 가정하십시오. MaxSize 특성을 7로 설정하면 축출기는 각 힙 또는 대기열 오브젝트에서 세 개의 항목을 축출하여 각 힙 또는 대기열 *크기*를 7로 다시 줄입니다. 축출기는 힙 또는 대기열에 MaxSize 특성 값보다 많은 요소가 있는 경우에만 맵 항목을 축출합니다. MaxSize를 힙 또는 대기열 *크기*의 70%로 설정하십시오. 이 예에서는 값을 7로 설정합니다. BaseMap 항목 수를 사용하는 힙 또는 대기열 수로 나누어 각 힙 또는 대기열의 적절한 *크기*를 확보할 수 있습니다.

SleepTime 특성

축출기는 항상 맵에서 항목을 제거하지는 않습니다. 대신 설정된 시간 동안 일시 정지(sleep)하고 오직 *n*초마다 가동합니다. 여기서 *n*은 SleepTime 특성을 가리킵니다. 또한 이 특성은 성능을 향상시킵니다. 축출 수행을 너무 자주 실행하면 처리에 필요한 자원 때문에 성능이 떨어집니다. 그러나 축출기를 충분히 사용하지 않으면 불필요한 항목의 맵이 남아 있을 수 있습니다. 불필요한 항목이 맵에 가득 차면 맵에 필요한 처리 자원 및 메모리 요구사항 모두에 악영향을 줄 수 있습니다. 대부분의 맵에서 축출 수행을 15초로 설정하는 것이 바람직합니다. 맵을 자주 작성하고 높은 트랜잭션 비율로 사용하는 경우 값을 더 낮은 시간으로 설정하는 것이 바람직할 수 있습니다. 그러나 맵에 거의 액세스하지 않는 경우 시간을 더 높은 값으로 설정할 수 있습니다.

예

다음 예에서는 맵을 정의하고 새 LFU 축출기를 작성하며 축출기 특성을 설정하고 맵을 설정하여 축출기를 사용합니다.

```
//Use ObjectGridManager to create/get the ObjectGrid. Refer to
// the ObjectGridManger section
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");
//Set properties assuming 50,000 map entries
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

LRU 축출기 사용은 LFU 축출기 사용과 유사합니다. 예를 들면 다음과 같습니다.

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
//Set properties assuming 50,000 map entries
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

두 개 행만 LFUEvictor 예와 다르다는 점에 주의하십시오.

기본 축출기 우수 사례

기본 *TTL(Time to Live)* 축출기 우수 사례.

365 페이지의 『플러그인 축출기 성능 우수 사례』 주제에서 설명한 플러그인 축출기 이외에, 기본 TTL 축출기는 모든 BackingMap에서 작성됩니다. 기본 축출기는 *TTL(Time to Live)* 개념에 기반하여 항목을 제거합니다. 이 동작은 ttlType 속성에 정의됩니다. 다음과 같이 세 개의 ttlTypes 속성이 있습니다.

- **없음:** 항목이 만기되지 않으므로 맵에서 제거되지 않도록 지정합니다.
- **작성 시간:** 작성 시간에 따라 항목을 축출하도록 지정합니다.
- **마지막 액세스 시간:** 마지막 액세스 시간에 따라 항목을 축출하도록 지정합니다.

성능 관련 기본 축출기 특성 및 우수 사례

TimeToLive 특성

ttlType 특성과 함께 이 특성은 성능 측면에서 가장 중요합니다. CREATION_TIME ttlType을 사용하는 경우 축출기는 작성 후 경과된 시간이 해당 TimeToLive 속성 값과 같은 경우 항목을 축출합니다. TimeToLive 속성 값을 10초로 설정하면 전체 맵의 모든 항목이 10초 후 축출됩니다. CREATION_TIME ttlType에서 이 값을 설정할 때 주의해야 합니다. 이 축출기 사용은 캐시에 추가된 크기(설정된 시간 동안만 사용됨)가 충분히 큰 경우 가장 적합합니다. 이 계획을 사용하면 작성된 모든 항목은 설정된 시간이 지나면 제거됩니다.

다음은 CREATION_TIME의 TTL 유형이 유용한 경우에 대한 예입니다. 주식 시세를 얻는 웹 응용프로그램을 사용하고 있으며, 최근 시세를 확보하는 일은 중요하지 않습니다. 이 경우 주식 시세는 20분 동안 오브젝트 그리드에 캐시됩니다. 20분이 지나면 오브젝트 그리드 맵 항목이 만기되어 축출됩니다. 20분마다 또는 오브젝트 그리드 맵이 Loader 플러그인을 사용하여 데이터베이스에서 맵 데이터를 최신 데이터로 새로 고칩니다. 데이터베이스는 최근 주식 시세로 20분마다 갱신됩니다. 이 응용프로그램에서 이와 같이 수행하려면 TimeToLive 값을 20분으로 사용하는 것이 바람직합니다.

LAST_ACCESSED_TIME ttlType 속성을 사용하는 경우 CREATION_TIME ttlType을 사용할 때보다 TimeToLive를 더 낮은 수로 설정하십시오. 항목의 TimeToLive 속성은 항목에 액세스할 때마다 재설정되기 때문입니다. 즉, TimeToLive 속성이 15이고 항목이 14초 동안 그대로 있다가 14초 후 액세스된 경우 다음 15초 동안은 다시 만기되지 않습니다. TimeToLive를 비교적 높은 수로 설정한 경우 많은 항목을 축출할 수 없습니다. 그러나 항목에 자주 액세스하지 않는 경우 값을 15초 정도로 설정하면 항목을 제거할 수 있습니다.

다음은 LAST_ACCESSED_TIME의 TTL 유형이 유용한 경우에 대한 예입니다. 오브젝트 그리드 맵은 클라이언트에서 세션 데이터를 보유할 때 사용됨

니다. 세션 데이터는 일정 시간 동안 클라이언트가 세션 데이터를 사용하지 않으면 제거됩니다. 예를 들어 30분 동안 클라이언트에서 활동이 없으면 세션 데이터는 제한시간을 초과합니다. 이 경우 TimeToLive 속성을 30분으로 설정하여 LAST_ACCESSED_TIME의 TTL 유형을 사용하는 것이 바로 이 응용프로그램에 필요한 방법입니다.

예

다음 예에서는 BackingMap을 작성하고 해당 기본 축출기 ttlType 속성을 설정하며 해당 TimeToLive 특성을 설정합니다.

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESSED_TIME);
bMap.setTimeToLive(15);
```

대부분의 축출기 설정은 오브젝트 그리드 초기화보다 먼저 설정되어야 합니다. 축출기에 대한 자세한 내용은 206 페이지의 『축출기』를 참조하십시오.

제 12 장 피어 JVM(Java Virtual Machine) 사이에서 변경사항 분배

LogSequence 및 LogElement 오브젝트는 플러그인을 사용하여 오브젝트 그리드 트랜잭션에서 발생하는 변경사항을 전달합니다.

트랜잭션 방식의 변경사항을 분배할 때 JMS(Java Message Service)를 사용하는 방법에 대한 자세한 정보는 373 페이지의 『트랜잭션 변경사항을 분배하기 위한 Java Message Service』를 참조하십시오.

오브젝트 그리드 인스턴스를 ObjectGridManager에서 캐시해야 한다는 전제조건이 있습니다. 자세한 정보는 99 페이지의 『createObjectGrid 메소드』를 참조하십시오. 캐시 인스턴스 부울 값은 true로 설정되어야 합니다.

오브젝트에서는 원격 JVM(Java Virtual Machine)의 피어 오브젝트 그리드에 대한 메시지 전송을 사용하여 오브젝트 그리드에서 발생하는 변경사항을 쉽게 공개한 후 해당 JVM에 변경사항을 적용하는 방법을 응용프로그램에 제공합니다. LogSequenceTransformer 클래스는 해당 지원을 사용 가능하게 할 때 매우 중요합니다. 이 항목은 메시지를 전달하는 JMS(Java Message Service) 메시징 시스템을 사용하여 리스너를 작성하는 방법을 점검합니다.

오브젝트 그리드는 IBM 제공 플러그인과 함께 WebSphere Application Server 클러스터 구성원에서 오브젝트 그리드 트랜잭션 확약을 통해 나타나는 LogSequences 전송을 지원합니다. 이 기능은 기본적으로 사용 불가능하지만 조작 가능하게 구성될 수 있습니다. 그러나 처리자 또는 생성자가 WebSphere Application Server가 아닌 경우 외부 JMS 메시징 시스템을 사용해야 할 수도 있습니다.

1. 플러그인을 초기화하십시오. 오브젝트 그리드를 시작할 때 오브젝트 그리드는 ObjectGridEventListener 인터페이스 규정의 파트인 플러그인의 `initialize` 메소드를 호출합니다. `initialize` 메소드는 연결, 세션 및 공개자를 포함하여 해당 JMS 자원을 확보한 후 JMS 리스너인 스레드를 시작해야 합니다. initialize 메소드는 다음 예와 유사하게 표시됩니다.

```
public void initialize(Session session)
{
 mySession = session;
 myGrid = session.getObjectGrid();
try
{
  if(mode == null)
{
throw new ObjectGridRuntimeException("No mode specified");
}
  if(userid != null)
{
   connection = topicConnectionFactory.createTopicConnection(
```

```
   userid, password);
}
else
   connection = topicConnectionFactory.createTopicConnection();
  // need to start the connection to receive messages.
  connection.start();
  // the jms session is not transactional (false).
  jmsSession = connection.createTopicSession(false,
  javax.jms.Session.AUTO_ACKNOWLEDGE);
  if(topic == null)
   if(topicName == null)
{
throw new ObjectGridRuntimeException("Topic not specified");
}
else
{
    topic = jmsSession.createTopic(topicName);
}
  publisher = jmsSession.createPublisher(topic);
// start the listener thread.
listenerRunning = true;
  listenerThread = new Thread(this);
  listenerThread.start();
}
 catch(Throwable e)
{
throw new ObjectGridRuntimeException("Cannot initialize", e);
}
}
```

스레드를 시작하는 코드에서는 J2EE(Java 2 Platform, Standard Edition) 스레드를 사용합니다. WebSphere Application Server 버전 6.x 또는 WebSphere Application Server 버전 5.x 엔터프라이즈 서버를 실행하는 경우 비동기 Bean API(Application Programming Interface)를 사용하여 이 디먼 스레드를 시작하십시오. 또한 공통 API를 사용할 수 있습니다. 다음은 작업 관리자를 사용하여 동일한 조치를 표시하는 대체 스니펫 예입니다.

```
// start the listener thread.
listenerRunning = true;
workManager.startWork(this, true);
```

또한 플러그인은 Runnable 인터페이스 대신 Work 인터페이스를 구현해야 합니다. 그리고 listenerRunning 변수를 false로 설정하여 release 메소드를 추가해야 합니다. 플러그인은 해당 생성자의 WorkManager 인스턴스 또는 IoC(Inversion of Control) 컨테이너를 사용하는 경우 삽입을 통해 제공됩니다.

2. 변경사항을 전송하십시오. 다음은 오브젝트 그리드에서 변경된 로컬 변경사항을 공개하는 transactionEnd 메소드 샘플입니다. 여기에서는 JMS를 사용합니다. 그러나 신뢰 가능한 공개 및 등록 메시징이 가능한 다른 메시지 전송을 사용할 수도 있습니다.

```
// This method is synchronized to make sure the
// messages are published in the order the transaction
// were committed. If we started publishing the messages
// in parallel then the receivers could corrupt the Map
// as deletes may arrive before inserts etc.
public synchronized void transactionEnd(String txid,
boolean isWriteThroughEnabled,
boolean committed, Collection changes)
{
try
{
  // must be write through and commited.
  if(isWriteThroughEnabled && committed)
{
   // write the sequences to a byte []
   ByteArrayOutputStream bos = new ByteArrayOutputStream();
   ObjectOutputStream oos = new ObjectOutputStream(bos);
   if (publishMaps.isEmpty()) {
    // serialize the whole collection
    LogSequenceTransformer.serialize(changes, oos, this, mode);
}
else {
    // filter LogSequences based on publishMaps contents
    Collection publishChanges = new ArrayList();
    Iterator iter = changes.iterator();
    while (iter.hasNext()) {
     LogSequence ls = (LogSequence) iter.next();
     if (publishMaps.contains(ls.getMapName())) {
     publishChanges.add(ls);
}
}
   LogSequenceTransformer.serialize(publishChanges, oos, this,
mode);
}
  // make an object message for the changes
  oos.flush();
  ObjectMessage om = jmsSession.createObjectMessage(
  bos.toByteArray());
  // set properties
  om.setStringProperty(PROP_TX, txid);
  om.setStringProperty(PROP_GRIDNAME, myGrid.getName());
  // transmit it.
  publisher.publish(om);
}
}
 catch(Throwable e)
{
throw new ObjectGridRuntimeException("Cannot push changes", e);
}
}
```

이 메소드에서는 여러 인스턴스 변수를 사용합니다.

- **jmsSession** 변수: 메시지를 공개할 때 사용하는 JMS 세션. 플러그인을 초기화할 때 작성됩니다.
- **mode** 변수: 분배 모드.

- **publishMaps** 변수: 공개할 변경사항과 함께 각 Map의 이름을 포함하는 세트. 변수가 비어 있는 경우 모든 Map이 공개됩니다.
- **publisher** 변수: 플러그인 초기화 메소드에서 작성된 TopicPublisher 오브젝트.

3. 갱신 메시지를 수신 및 적용하십시오. 다음은 run 메소드입니다. 이 메소드는 응용프로그램 루프가 중지될 때까지 루프에서 실행됩니다. 각 루프 반복에서는 JMS 메시지를 수신한 후 오브젝트 그리드에 적용합니다.

```
private synchronized boolean isListenerRunning()
{
 return listenerRunning;
}
public void run()
{
try
{
System.out.println("Listener starting");
  // get a jms session for receiving the messages.
  // Non transactional.
  TopicSession myTopicSession;
  myTopicSession = connection.createTopicSession(false,
  javax.jms.Session.AUTO_ACKNOWLEDGE);
  // get a subscriber for the topic, true indicates don't receive
  // messages transmitted using publishers
  // on this connection. Otherwise, we'd receive our own updates.
  TopicSubscriber subscriber = myTopicSession.createSubscriber(topic,
  null, true);
System.out.println("Listener started");
  while(isListenerRunning())
{
   ObjectMessage om = (ObjectMessage)subscriber.receive(2000);
   if(om != null)
{
    // Use Session that was passed in on the initialize...
    // very important to use no write through here
    mySession.beginNoWriteThrough();
    byte[] raw = (byte[])om.getObject();
    ByteArrayInputStream bis = new ByteArrayInputStream(raw);
    ObjectInputStream ois = new ObjectInputStream(bis);
    // inflate the LogSequences
    Collection collection = LogSequenceTransformer.inflate(ois,
     myGrid);
    Iterator iter = collection.iterator();
    while (iter.hasNext()) {
     // process each Maps changes according to the mode when
     // the LogSequence was serialized
     LogSequence seq = (LogSequence)iter.next();
     mySession.processLogSequence(seq);
}
    mySession.commit();
   } // if there was a message
  } // while loop
  // stop the connection
  connection.close();
}
 catch(IOException e)
{
```

```
System.out.println("IO Exception: " + e);
}
 catch(JMSException e)
{
System.out.println("JMS Exception: " + e);
}
 catch(ObjectGridException e)
{
System.out.println("ObjectGrid exception: " + e);
System.out.println("Caused by: " + e.getCause());
}
 catch(Throwable e)
{
System.out.println("Exception : " + e);
}
System.out.println("Listener stopped");
}
```

LogSequenceTransformer 클래스와 ObjectGridEventListener, LogSequence 및 LogElement API를 통해 신뢰 가능한 공개 및 등록을 변경사항을 분배하고 분배시킬 Map을 필터하는 데 사용할 수 있습니다. 이 타스크의 스니펫에서는 JMS를 통해 이 API를 사용하여 공통 메시지 전송을 공유하는 다양한 플랫폼 세트에 호스트된 응용프로그램이 공유하는 피어 투 피어 오브젝트 그리드를 빌드하는 방법을 표시합니다.

트랜잭션 변경사항을 분배하기 위한 Java Message Service

서로 다른 계층 간에 또는 혼합 플랫폼의 환경에 분배되는 변경사항에 JMS(Java Message Service)를 사용하십시오.

JMS는 서로 다른 계층 간에 또는 혼합 플랫폼의 환경에서 분배되는 변경사항에 이상적인 프로토콜입니다. 예를 들어, 오브젝트 그리드를 사용하는 일부 응용프로그램을 Gluecode 또는 Tomcat에 전개하고 반면에 기타 응용프로그램을 WebSphere Application Server 버전 6.0에서 실행할 수 있습니다. JMS는 이 서로 다른 환경의 오브젝트 그리드 피어 간에 분배되는 변경사항에 이상적입니다. 고가용성 관리자 메시지 전송이 아주 빠르긴 하지만 단일 코어 그룹에 있는 JVM으로만 변경사항을 분배할 수 있습니다. JMS가 속도는 느리지만 대형의 보다 다양한 응용프로그램 클라이언트 세트가 오브젝트 그리드를 공유할 수 있게 합니다. 예를 들어, JMS는 WebSphere Extended Deployment에 전개된 응용프로그램과 Fat Swing 클라이언트 간에 오브젝트 그리드에 데이터를 공유하는 경우에 이상적입니다.

개요

JMS는 ObjectGridEventListener 리스너로 작동하는 Java 오브젝트를 사용하여 트랜잭션 변경사항을 분배할 수 있도록 구현됩니다. 이 오브젝트는 다음 네 가지 방법으로 상태를 전달할 수 있습니다.

무효화 메시지를 수신할 때 축출, 갱신 또는 삭제된 항목이 모든 피어 JMS(Java Virtual Machine)에서 제거됩니다.

조건부 무효화
: 로컬 버전이 공개자의 버전과 동일하거나 이전인 경우에만 항목이 축출됩니다.

푸시
: JMS 메시지를 수신할 때 축출, 갱신, 삭제 또는 삽입된 항목이 모든 피어 JVM에 추가되거나 겹쳐쓰기됩니다.

조건부 푸시
: 로컬 항목이 공개 중인 버전보다 최신 버전이 아닌 경우, 수신 측에서만 항목이 갱신 또는 추가됩니다.

공개를 위한 변경사항 청취

플러그인은 ObjectGridEventListener 인터페이스를 구현하여 transactionEnd 이벤트를 인터셉트합니다. 오브젝트 그리드가 이 메소드를 호출할 때 플러그인은 트랜잭션이 접촉하는 각 Map의 LogSequence 목록을 JMS 메시지로 변환한 후 이를 공개합니다. 모든 Map 또는 Map 서브세트에 대해 변경사항을 공개하도록 플러그인을 구성할 수 있습니다. LogSequence 오브젝트는 공개 사용이 가능한 Map에 대해 처리됩니다. LogSequenceTransformer 오브젝트 그리드 클래스는 각 Map의 필터된 LogSequence를 스트림으로 직렬화합니다. LogSequence가 모두 스트림으로 직렬화된 후에 JMS ObjectMessage가 작성되고 잘 알려진 주제로 공개됩니다.

JMS 메시지 청취 및 로컬 오브젝트 그리드에 적용

또한 동일한 플러그인은 루프로 스핀되는 스레드를 시작하고 잘 알려진 주제에 공개된 메시지를 모두 수신합니다. 메시지가 도착하면 메시지 컨텐츠를 LogSequenceTransformer 클래스로 전달하여 LogSequence 오브젝트 세트로 변환합니다. 그러면 비동시 기록 트랜잭션이 시작됩니다. 각 LogSequence 오브젝트가 Session.processLogSequence 메소드에 제공되며, 이 메소드는 로컬 Map을 변경사항으로 갱신합니다. processLogSequence 메소드는 분배 모드를 이해하고 있습니다. 트랜잭션이 확약되고 로컬 캐시가 이제 변경사항을 반영합니다.

JMS를 사용하여 트랜잭션 변경사항을 분배하는 방법에 대한 자세한 정보는 369 페이지의 제 12 장『피어 JVM(Java Virtual Machine) 사이에서 변경사항 분배』를 참조하십시오.

제 13 장 주입 기반 컨테이너 통합

IoC(Inversion of Control) 프레임워크를 사용하여 오브젝트 그리드를 완벽히 구성할 수 있습니다. 따라서 오브젝트 그리드 XML 구성 프레임워크를 사용하지 않아도 됩니다.

주입 기반 컨테이너

IoC(Inversion of Control)라고도 하는 주입 기반 컨테이너는 클라이언트 측과 서버 측에서 응용프로그램이 사용하는 공통 패턴입니다. 이 컨테이너의 개방형 소스 구현이 여러 개 존재합니다. 새 EJB(Enterprise JavaBeans) 버전 3.0 스펙도 이 개념을 일부 차용하고 있습니다. 이 프레임워크의 대부분은 Enterprise JavaBean 컨테이너이며, 특정 Bean의 인스턴스를 작성하는 책임을 지고 있습니다. 이 프레임워크는 특성 세트를 사용하여 Bean을 초기화하고 Enterprise JavaBean 속성에 getter 및 setter 쌍을 사용하여 필요한 기타 Enterprise JavaBeans를 연결할 수도 있습니다. 오브젝트 그리드 API(Application Programming Interface)는 이 컨테이너와 잘 작동하도록 설계되었습니다. ObjectGridManager.createObjectGrid 메소드를 시작하면 응용프로그램이 작업 중인 오브젝트 그리드에 대한 참조를 갖거나 필요할 때 오브젝트 그리드 세션을 제공하도록 컨테이너에 요청할 수 있게 오브젝트 그리드를 부트스트랩하도록 이 컨테이너를 구성할 수 있습니다.

지원되는 패턴

다음 섹션에서는 IoC 프레임워크를 사용하는 응용프로그램이 오브젝트 그리드 API를 깔끔하게 사용할 수 있는지 확인하기 위해 수행해야 할 작업을 설명합니다.

ObjectGridManager를 사용하여 오브젝트 그리드 작성

오브젝트 그리드 API는 IoC 프레임워크와 잘 작동하도록 설계되었습니다. 이런 프레임워크에서 사용되는 루트 싱글톤은 오브젝트 그리드Manager 인터페이스인데, 이 인터페이스에는 이름 지정된 오브젝트 그리드에 대한 참조를 리턴하는 여러 개의 `createObjectGrid` 팩토리 메소드가 있습니다. IoC 프레임워크에서 이 오브젝트 그리드 참조를 싱글톤으로 설정하여 Bean에 대한 후속 요청이 동일한 오브젝트 그리드 인스턴스를 리턴하도록 할 수 있습니다.

오브젝트 그리드 플러그인

오브젝트 그리드에는 다음 플러그인이 포함되어 있습니다.

- TransactionCallback
- ObjectGridEventListener
- SubjectSource

- MapAuthorization
- SubjectValidation

각 플러그인은 인터페이스를 구현하는 단순 JavaBean입니다. IoC 프레임워크를 사용하여 이런 플러그인을 작성하고 오브젝트 그리드 인스턴스의 해당 속성에 이를 연결할 수 있습니다. 이 플러그인 각각에는 IOC 프레임워크와의 완전한 통합을 위해 오브젝트 그리드 인터페이스에 대응하는 set 메소드가 있습니다.

맵 작성

오브젝트 그리드 인터페이스의 createMap 팩토리 메소드를 사용하여 새로 이름 지정된 맵을 작성할 수 있습니다. BackingMap(createMap이 리턴한 오브젝트)이 필요로 하는 플러그인은 IoC 프레임워크를 사용하여 구성된 후 해당 속성 이름을 사용하여 BackingMap에 연결됩니다. 다른 오브젝트가 BackingMap을 참조하지 않기 때문에 IoC 프레임워크는 이를 자동으로 구성하지 않습니다. 각 BackingMap을 일시적인 해결책으로 기본 응용프로그램 Bean의 더미(dummy) 속성에 연결할 수 있습니다. setMaps() 메소드를 사용하여 이전에 이 오브젝트 그리드에 정의된 BackingMaps를 지우고 제공된 BackingMaps 목록으로 바꾸십시오.

BackingMap 플러그인

BackingMap의 플러그인은 오브젝트 그리드의 플러그인과 동일한 방식으로 작동합니다. 각 플러그인마다 BackingMap 인터페이스에 대응하는 set 메소드가 있습니다. BackingMap 플러그인은 다음과 같습니다.

- Loader
- ObjectTransformer
- OptimisticCallback
- Evictor
- MapEventListener

사용법 패턴

IoC 프레임워크가 오브젝트 그리드를 생성하도록 설정된 구성 파일을 갖게 된 후 gridName_Session이라는 또는 유사한 엔터프라이즈 Bean을 작성하십시오. 오브젝트 그리드 싱글톤 Enterprise JavaBean에서 getSession 메소드를 호출하여 얻은 Enterprise JavaBean으로 이를 정의하십시오. 그러면 응용프로그램이 IoC 프레임워크를 사용하여 오브젝트가 새 세션을 필요로 할 때마다 gridName_Session 오브젝트에 대한 참조를 얻습니다.

요약

Bean 인스턴스화 및 구성에 이미 IoC 프레임워크를 사용하는 환경에서 오브젝트 그리드를 사용하는 것은 간단합니다. IoC 프레임워크를 사용하여 오브젝트 그리드를 완벽

히 구성할 수 있으며, 따라서 XML 구성 프레임워크를 사용하지 않아도 됩니다. 오브젝트 그리드는 기존 IoC 프레임워크와 심리스(seamlessly)하게 작동합니다.

제 14 장 문제점 해결

이 섹션에서는 응용프로그램 오류 또는 응용프로그램 디자인 문제로 인해 발생하는 문제점을 해결하기 위한 시나리오를 설명합니다. 오브젝트 그리드에 결함이 있다고 의심되면 ObjectGridManager 추적 섹션에 설명된 대로 오브젝트 그리드 추적을 사용 가능하게 해야 할 수도 있습니다.

간헐적이고 명확하지 않은 오류

응용프로그램에서는 CopyMode 섹션에서 설명한 대로 COPY_ON_READ, COPY_ON_WRITE 또는 NO_COPY 복사 모드를 사용하여 성능을 향상시키려고 합니다. 문제점 증상이 변경되고 명확하지 않거나 예상치 않은 문제점인 경우 응용프로그램에서 간헐적인 문제점이 발생합니다. 간헐적인 문제점은 복사 모드가 COPY_ON_READ_AND_COMMIT 모드로 변경되면 발생하지 않습니다.

문제점

사용 중인 복사 모드의 프로그래밍 규정을 응용프로그램에서 위반한 결과, 오브젝트 그리드 맵에서 데이터가 손상되어 문제점이 발생했을 수 있습니다. 데이터가 손상되면 예상할 수 없는 오류가 간헐적으로 또는 명확하지 않거나 예상치 않은 방식으로 발생할 수 있습니다.

솔루션

솔루션은 응용프로그램이 사용 중인 복사 모드에서 언급한 프로그래밍 규정을 준수하는 것입니다. COPY_ON_READ 및 COPY_ON_WRITE 복사 모드의 경우 응용프로그램이 값 참조가 확보된 트랜잭션 범위를 벗어난 값 오브젝트에 대한 참조를 사용함을 의미합니다. 이 모드를 사용하려면 응용프로그램은 트랜잭션을 완료한 후 값 오브젝트에 대한 참조를 제거하고 값 오브젝트에 액세스해야 하는 각 트랜잭션에서 값 오브젝트에 대한 새 참조를 확보한다는 데 동의해야 합니다. NO_COPY 복사 모드의 경우 응용프로그램은 값 오브젝트를 변경하지 않는다는 데 동의해야 합니다. 이 경우 값 오브젝트를 변경하지 않도록 응용프로그램을 변경하거나 응용프로그램에서 다른 복사 모드를 사용해야 합니다. 복사 모드 설정에 대한 추가 정보는 CopyMode 섹션을 참조하십시오.

일반 예외 처리 기술

Throwable 오브젝트의 근본 원인을 알면 문제점의 소스를 분리하는 데 도움이 됩니다. 다음은 발생한 Throwable의 근본 원인을 찾기 위해 예외 핸들러에서 사용할 수 있는 유틸리티 메소드에 대한 예입니다.

예

```
static public Throwable findRootCause( Throwable t )
{
// Start with Throwable that occurred as the root.
Throwable root = t;
// Follow cause chain until last Throwable in chain is found.
Throwable cause = root.getCause();
while ( cause != null )
{
root = cause;
cause = root.getCause();
}
// Return last Throwable in the chain as the root cause.
return root;
}
```

특정 예외 처리 기술

중복 삽입

일반적으로 이 문제점은 분산 트랜잭션 전달 환경에서만 발생합니다. 자주 발생하지는 않습니다.

메시지

```
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl < processLogSequence Exit
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl > commit for:
TX:08FE0C67-0105-4000-E000-1540090A5759 Entry
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl > rollbackPMapChanges for:
TX:08FE0C67-0105-4000-E000-1540090A5759
as result of Throwable: com.ibm.websphere.objectgrid.plugins.
CacheEntryException:
Duplicate key on an insert!
Entry com.ibm.websphere.objectgrid.plugins.CacheEntryException:
Duplicate key on an insert!
at com.ibm.ws.objectgrid.map.BaseMap.applyPMap(BaseMap.java:528)
at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:405)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.commitPropagatedLogSequence
(TranPropWorkerThread.java:553)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.processCommitRequest
(TranPropWorkerThread.java:449)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.run
(TranPropWorkerThread.java:200)
at java.lang.Thread.run(Thread.java:568)
```

문제점

필터된 로그 순서를 하나의 JVM에서 다른 JVM으로 전달하는 경우 외부 로그 순서가 두 번째 JVM에서 처리됩니다. 이 키의 항목이 있을 수 있거나 두 개의 로그 순서 조작 코드가 서로 다릅니다. 이 문제점은 가끔 발생합니다.

영향 및 솔루션

이 문제점이 발생하면 항목이 다른 JVM에서 갱신되지 않으므로 오브젝트 그리드에서 불일치가 발생할 수 있습니다. 그러나 이 문제점을 방지하는 해결 방법이 있습니다. 오

브젝트 갱신/삽입/제거 이외에도 오브젝트 검색 시 오브젝트에서 파티션 기능(WPF)을 사용할 수 있습니다. 이 기술에 대한 자세한 정보는 WPF 및 오브젝트 그리드 통합 섹션을 참조하십시오.

변경이 예상되지 않는 충돌 예외

OptimisticCollisionException 예외를 직접 수신하거나 ObjectGridException 예외 수신 중 해당 예외를 수신할 수 있습니다. 다음 코드는 예외를 발견한 후 해당 메시지를 표시하는 방법에 대한 예입니다.

```
try {
...
} catch (ObjectGridException oe) {
System.out.println(oe);
}
```

예외 원인

OptimisticCollisionException 예외는 두 개의 서로 다른 클라이언트가 거의 동시에 동일한 맵 항목을 갱신하는 상황에서 작성됩니다. 한 클라이언트의 세션이 확약되어 맵 항목을 갱신합니다. 그러나 다른 클라이언트는 확약 이전에 이미 데이터를 읽었으므로 이전 또는 잘못된 데이터를 포함합니다. 그러면 다른 클라이언트에서 잘못된 데이터를 확약하려고 합니다. 이때 예외가 작성됩니다.

예외를 트리거하는 키 검색

예외와 같은 문제점을 해결할 때 예외를 트리거하는 항목에 해당하는 키를 검색하는 것이 유용할 수 있습니다. OptimisticCollisionException의 이점은 해당 키를 표시하는 오브젝트를 리턴하는 getKey 메소드에서 빌드되었다는 점입니다. 다음은 OptimisticCollisionException를 발견할 때 키를 검색 및 인쇄하는 방법에 대한 예입니다.

```
try {
...
} catch (OptimisticCollisionException oce) {
 System.out.println(oce.getKey());
}
```

OptimisticCollisionException은 ObjectGridException의 원인일 수 있습니다. 이 경우 다음 코드를 사용하여 예외 유형을 판별하고 키를 인쇄할 수 있습니다. 아래 코드는 일반 예외 처리 기술 섹션에서 설명한 대로 findRootCause utility 메소드를 사용합니다.

```
try {
...
}
catch (ObjectGridException oe) {
 Throwable Root = findRootCause( oe );
 if (Root instanceof OptimisticCollisionException) {
```

```
  OptimisticCollisionException oce = (OptimisticCollisionException)Root;
 System.out.println(oce.getKey());
}
}
```

LockTimeoutException 예외

메시지

LockTimeoutException 예외를 직접 발견하거나 ObjectGridException 발견 중 해당 예외를 발견할 수 있습니다. 다음 코드 스니펫에서는 예외를 발견하고 메시지를 표시하는 방법을 표시합니다.

```
try {
...
}
catch (ObjectGridException oe) {
System.out.println(oe);
}
```

결과는 다음과 같습니다.

```
com.ibm.websphere.objectgrid.plugins.LockTimeoutException: %Message
```

%Message는 예외가 작성되고 정확한 오류 메시지를 표시하도록 예외 특성보 및 메소드를 사용할 때 매개변수로 전달되는 문자열입니다. 대부분의 경우 요청된 잠금 유형 및 트랜잭션이 조치를 취하는 맵 항목을 설명합니다.

예외 원인

트랜잭션 또는 클라이언트가 특정 맵 항목에서 잠금을 부여할지 묻는 경우 현재 클라이언트에서 잠금을 해제하기를 기다리는 경우가 자주 있습니다. 잠금 요청이 장기간 대기 상태로 남아 있어서 잠금을 부여하지 못한 경우 LockTimeoutException이 작성됩니다. 이를 통해 교착 상태를 방지합니다. 추가 정보는 다음 섹션에서 설명합니다. 잠금 중 변경이 예상되는 계획을 사용하는 경우 트랜잭션을 확약해야만 잠금이 해제되므로 이 예외가 나타날 수 있습니다.

잠금 요청 및 예외에 대한 추가 정보 확보

LockTimeoutException은 예외를 트리거하는 상황에 대한 심도 깊은 설명이 들어 있는 문자열을 리턴하는 getLockRequestQueueDetails 메소드에 빌드되어 있습니다. 다음은 예외를 발견하고 오류 메시지를 표시하는 일부 코드에 대한 예입니다.

```
try {
...
}
catch (LockTimeoutException lte) {
 System.out.println(lte.getLockRequestQueueDetails());
}
```

출력 결과는 다음과 같습니다.

```
lock request queue
->[TX:163C269E-0105-4000-E0D7-5B3B090A571D, state =
 Granted 5348 milli-seconds ago, mode = U]
->[TX:163C2734-0105-4000-E024-5B3B090A571D, state =
 Waiting for 5348 milli-seconds, mode = U]
->[TX:163C328C-0105-4000-E114-5B3B090A571D, state =
 Waiting for 1402 milli-seconds, mode = U]
```

ObjectGridException catch 블록에서 예외가 발생한 경우 다음 코드에서 예외를 판별하고 대기열 세부사항을 인쇄합니다. 일반 예외 처리 기술 섹션에서 설명한 findRootCause 유틸리티 메소드를 사용합니다.

```
try {
...
}
catch (ObjectGridException oe) {
 Throwable Root = findRootCause( oe );
 if (Root instanceof LockTimeoutException) {
  LockTimeoutException lte = (LockTimeoutException)Root;
 System.out.println(lte.getLockRequestQueueDetails());
}
}
```

가능한 솔루션

LockTimeoutException으로 응용프로그램에서 가능한 교착 상태를 방지합니다. 이 유형의 예외는 설정된 일정 시간을 대기한 후 처리됩니다. 그러나 대기 시간은 BackingMap에서 사용 가능한 setLockTimeout(int) 메소드를 사용하여 설정할 수 있습니다. setLockTimeout 메소드를 사용하면 LockTimeoutException은 발생하지 않습니다. 실제로 응용프로그램에 교착 상태가 없는 경우 잠금 제한시간을 조정하면 LockTimeoutException을 방지하는 데 도움이 될 수 있습니다.

다음 코드에서는 오브젝트 그리드를 작성하고 맵을 정의하며 해당 LockTimeout 값을 30초로 설정하는 방법을 표시합니다.

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("MapName");
//This will set the amount of time that a
// lock request will wait before an exception is thrown
bMap.setLockTimeout(30);
```

이전 코드는 하드 코드화된 오브젝트 그리드 및 맵 특성에서 사용할 수 있습니다. XML 파일에서 오브젝트 그리드를 작성하는 경우 LockTimeout 특성은 backingMap 태그에서 설정할 수 있습니다. 다음은 LockTimeout 값을 30초로 설정하는 backingMap 태그의 예입니다.

```
<backingMap name="MapName" lockStrategy="PESSIMISTIC" lockTimeout="30">
```

LockDeadlockException

메시지

LockDeadLockException을 직접 발견하거나 ObjectGridException을 발견하는 중 해당 예외를 확보할 수 있습니다. 다음은 예외를 발견한 후 표시된 메시지를 표시하는 코드 예입니다.

```
try {
...
} catch (ObjectGridException oe) {
System.out.println(oe);
}
```

결과는 다음과 같습니다.

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: %Message
```

%Message는 예외가 작성되고 처리될 때 매개변수로 전달되는 문자열을 표시합니다.

예외 원인

교착 상태의 가장 일반적인 유형은 잠금 중 변경이 예상되는 계획을 사용할 때 발생하며 두 개의 독립된 클라이언트 각각이 특정 오브젝트에서 공유된 잠금을 소유합니다. 그 다음 해당 오브젝트에서 독점 잠금으로 승격하려고 합니다. 다음은 예외가 처리되는 트랜잭션 블록에서 발생한 해당 상황을 표시하는 다이어그램입니다.

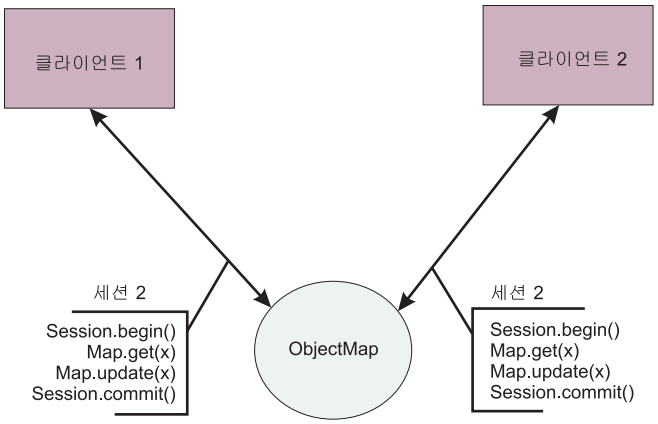


그림 23. 가능한 교착 상태 상황에 대한 예

이 그림은 예외가 발생한 경우 프로그램에서 발생하는 상황에 대한 요약 보기입니다. 많은 스레드에서 동일한 ObjectMap을 갱신하는 응용프로그램에서 이 상황이 발생할 수 있습니다. 다음은 두 개의 클라이언트가 이전 그림에서 설명한 트랜잭션 코드 블록을 실행하는 경우에 대한 단계별 예입니다.

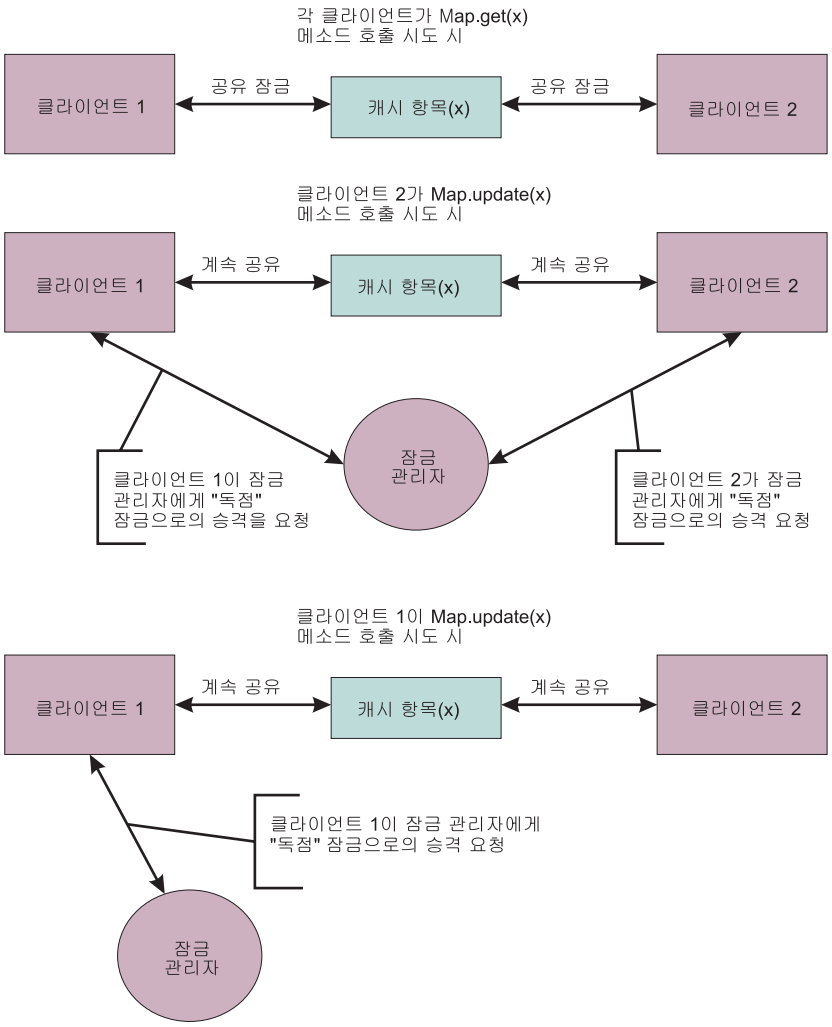


그림 24. 교착 상태 상황

그림과 같이 클라이언트 모두 독점 잠금으로 승격하려고 하며 여전히 공유된 잠금을 소유하는 경우 두 클라이언트 모두 실제로 성공할 수 없습니다. 클라이언트는 항상 다른 클라이언트에서 공유된 잠금을 해제하기를 기다리므로 `LockDeadlockException`이 발생합니다.

가능한 솔루션

이 예외를 수신하는 것이 바람직할 수도 있습니다. 스레드가 많은 경우(모든 스레드가 특정 맵에서 트랜잭션을 실행함) 이전에 설명한 상황(그림 1)이 발생할 수 있습니다. 프로그램이 정지되지 않도록 이 예외가 발생합니다. 이 예외를 발견하면 사용자에게 상황을 알릴 수 있으며 사용자가 원하는 경우 원인의 추가 정보를 확보할 수 있도록 catch 블록에 코드를 추가할 수 있습니다. 잠금 중 변경이 예상되는 계획에서만 이 예외가 나타나므로 한 가지 단순한 솔루션으로 단순히 잠금 중 변경이 예상되지 않는 계획을 사용하는 방법이 있습니다. 그러나 변경이 예상되어야 하는 경우 get 메소드 대신

getForUpdate 메소드를 사용할 수 있습니다. 그러면 이전에 설명한 상황에서 예외가 발생하지 않습니다.

XML 구성 문제점 진단

존재하지 않는 플러그인 콜렉션 참조

XML을 사용하여 BackingMap 플러그인을 정의하는 경우 backingMap 요소의 pluginCollectionRef 속성은 backingMapPluginCollection을 참조해야 합니다. pluginCollectionRef 속성은 backingMapPluginCollection 요소 중 하나의 ID와 일치해야 합니다.

메시지

pluginCollectionRef 속성이 backingMapPluginConfiguration 요소의 ID 속성과 일치하지 않는 경우 다음 메시지와 유사한 메시지가 로그에 표시됩니다.

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E:
This is an English only Error message:
Invalid XML file. Line: 14; URI: null;
Message: Key 'pluginCollectionRef' with
value 'bookPlugins' not found for identity
constraint of element 'objectGridConfig'..
```

다음 메시지는 추적이 사용 가능한 로그에서 나타난 예외입니다.

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E: This is an
English only Error message:
Invalid XML file. Line: 14; URI: null; Message: Key
'pluginCollectionRef' with
value 'bookPlugins' not found for identity constraint
of element 'objectGridConfig'..
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException:
Invalid XML file: etc/test/document/bookstore.xml
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R at
com.ibm.ws.objectgrid.config.XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R at
com.ibm.websphere.objectgrid.ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R Caused by: org.xml.sax.
SAXParseException: Key 'pluginCollectionRef' with value 'bookPlugins'
not found for identity
constraint of element 'objectGridConfig'.
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.error(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:011 CDT] 686c060e SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:011 CDT] 686c060e SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

문제점

이 오류를 생성하는 데 사용되는 XML 파일은 아래 표시됩니다. Book BackingMap의 pluginCollectionRef 속성은 bookPlugin으로 설정되었으며 단일 backingMapPluginCollection의 ID는 collection1입니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config
../objectGrid.xsd" xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" pluginCollectionRef="bookPlugin" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

솔루션

문제점을 수정하려면 각 pluginCollectionRef 값이 backingMapPluginCollection 요소 중 하나의 ID와 일치해야 합니다. 이 예에서는 단순히 pluginCollectionRef 이름을 collection1로 변경하여 이 오류를 방지합니다. 문제점을 수정하는 다른 방법으로 기존 backingMapPluginCollection ID를 pluginCollectionRef와 일치하도록 변경하거나 pluginCollectionRef와 일치하는 ID의 추가 backingMapPluginCollection을 추가하는 방법이 있습니다.

필수 속성 누락

XML 파일의 많은 요소에는 여러 가지 선택적 속성이 들어 있습니다. 파일에서 선택적 속성을 포함하거나 제외할 수 있습니다. XML은 이 두 가지 방식으로 유효성 검증에 성공합니다. 그러나 일부 필수 속성이 있습니다. 이 필수 속성과 연관된 요소를 사용할 때 해당 속성이 표시되지 않으면 XML 유효성 검증에 실패합니다.

메시지

필수 속성이 누락된 경우 다음 중 하나와 유사한 메시지가 로그에 표시됩니다. 이 예에서는 type 속성이 property 요소에서 누락되었습니다.

```
[7/15/05 13:41:41:267 CDT] 6873dcac XmlErrorHandl E CWOBJ9002E:
This is an English only
Error message: Invalid XML file.
Line: 12; URI: null; Message: cvc-complex-type.4:
Attribute 'type' must appear on element 'property'..
```

다음 메시지는 추적이 사용 가능한 로그에서 나타난 예외입니다.

```
[7/15/05 14:08:48:506 CDT] 6873dff9 XmlErrorHandl E CWOBJ9002E: This is an English
only Error message: Invalid XML file.
Line: 12; URI: null; Message: cvc-complex-type.4: Attribute 'type'
must appear on element 'property'..
[7/15/05 14:08:48:526 CDT] 6873dff9 SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException: Invalid XML file: etc/test/document/bookstore.xml
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R at com.ibm.ws.objectgrid.config.
XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R at com.ibm.websphere.objectgrid.
ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R Caused by: org.xml.sax.
SAXParseException: cvc-complex-type.4:
Attribute 'type' must appear on element 'property'.
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.error(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

문제점

다음 예는 위 오류를 생성하는 데 사용되는 XML 파일입니다. Evictor의 특성에는 세 개의 필수 속성 중 두 개만 들어 있다는 점에 주의하십시오. name 및 value 속성은 모두 있지만 type 속성이 누락되었습니다. 이 속성이 누락되면 XML 유효성 검증에 실패합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" pluginCollectionRef="collection1" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
<property name="maxSize" value="89" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

솔루션

이 문제점을 해결하려면 XML 파일에 필수 속성을 추가하십시오. 위에 표시된 XML 파일 예에서 type 유형을 추가한 후 정수 값을 지정해야 합니다.

필수 요소 누락

소수의 XML 요소가 스키마에 필요합니다. 해당 요소가 없으면 XML은 유효성 검증에 실패합니다.

메시지

필수 요소가 누락된 경우 다음 중 하나와 유사한 메시지가 로그에 표시됩니다. 이 경우 objectGrid 요소가 누락되었습니다.

```
[7/15/05 14:54:23:729 CDT] 6874d511 XmlErrorHandl E CWOBJ9002E:
This is an English only Error message: Invalid XML file.
Line: 5; URI: null; Message: cvc-complex-type.2.4.b: The content of
element 'objectGrids' is not complete.
One of '{"http://ibm.com/ws/objectgrid/config":objectGrid}' is expected..
```

이 오류에 대한 자세한 정보를 보려면 추적을 사용 가능하게 하십시오. ObjectGridManager 섹션에서는 추적을 켜는 방법에 대한 정보를 다룹니다.

문제점

다음 예는 이 문제점을 생성하는 데 사용된 XML 파일입니다. objectGrids 요소에는 objectGrid 하위 요소가 없다는 점에 주의하십시오. XML 스키마에 따라 objectGrid 요소는 objectGrids 태그에서 한 번 이상 나타나야 합니다. 이 요소가 누락되면 XML 유효성 검증에 실패합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
</objectGrids>
</objectGridConfig>
```

솔루션

이 문제점을 수정하려면 필수 요소가 XML 파일에 있어야 합니다. 이전 예에서 objectGrid 요소가 objectGrids 태그에 한 번 이상 나타나야 합니다. 필수 요소가 표시된 경우 XML 파일 유효성을 검증할 수 있습니다.

다음의 유효한 XML 파일에는 표시된 필수 요소가 들어 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore" />
</objectGrids>
</objectGridConfig>
```

유효하지 않은 속성의 XML 값

메시지

XML 파일의 일부 속성에는 특정 값만 지정할 수 있습니다. 이 속성에는 스키마에서 열거된 승인 가능한 값이 들어 있습니다. 이 속성은 다음과 같습니다.

- objectGrid 요소의 authorizationMechanism 속성
- backingMap 요소의 copyMode 속성
- backingMap 요소의 lockStrategy 속성
- backingMap 요소의 ttlEvictorType 속성
- property 요소의 type 속성

이 속성 중 하나에서 유효하지 않은 값이 지정되면 XML 유효성 검증에 실패합니다.

속성을 열거된 값에 속하지 않는 값으로 설정한 경우 다음 메시지가 로그에 표시됩니다.

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E CWOBJ9002E: This is an English
only Error message: Invalid XML file. Line: 6; URI: null; Message:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid with
respect to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
COPY_ON_WRITE, NO_COPY]'. It must be a value from the enumeration..
```

다음은 추적이 사용 가능한 로그에서 나타난 예외입니다.

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E CWOBJ9002E: This is an
  English only Error message: Invalid XML file. Line: 6; URI: null; Message:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid with
respect to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
COPY_ON_WRITE, NO_COPY]'. It must be a value from the enumeration..
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R com.ibm.websphere.objectgrid
.ObjectGridException: Invalid XML file: etc/test/document/backingMapAttrBad.xml
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R at com.ibm.ws.objectgrid.config.
XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R at com.ibm.websphere.objectgrid.
ProcessConfigXML$2.run(ProcessConfigXML.java:99)...
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R Caused by:
org.xml.sax.SAXParseException:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid
with respect
to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE,
NO_COPY]'.
It must be a value from the enumeration.
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
.ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
.ErrorHandlerWrapper.error(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.xs.XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.xs.XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

문제점

특정 세트 값에서 값을 지정한 속성이 잘못 지정되었습니다. 이 경우 copyMode 속성이 열거된 값 중 하나로 설정되지 않았습니다. INVALID_COPY_MODE로 설정되었습니다. 다음은 이 오류를 생성하는 데 사용되는 XML 파일입니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore" />
<backingMap name="book" copyMode="INVALID_COPY_MODE"/>
</objectGrid>
</objectGrids>
</objectGridConfig>
```

솔루션

이 예에서 copyMode의 값이 유효하지 않습니다. 이 속성을 COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE 또는 NO_COPY와 같이 유효값 중 하나로 설정하십시오.

구현 지원 없이 XML 유효성 검증

IBM SDK(Software Development Kit) 버전 1.4.2에는 스키마에서 XML 유효성 검증 시 사용할 일부 JAXP 기능 구현이 들어 있습니다.

메시지

이 구현이 들어 있지 않은 SDK를 사용하는 경우 유효성 검증에 실패할 수 있습니다. 이 구현이 들어 있지 않은 SDK를 사용하여 XML 유효성을 검증할 경우 Xerces를 다운로드하여 클래스 경로에 Java 아카이브(JAR) 파일을 포함시키십시오.

필수 구현이 들어 있지 않은 SDK를 사용하여 XML 유효성을 검증하면 다음 오류가 로그에 표시됩니다.

```
[7/19/05 10:50:45:066 CDT] 15c7850 XmlConfigBuil d XML validation is enabled
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R com.ibm.websphere
.objectgrid[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at
   com.ibm.ws.objectgrid
.ObjectGridManagerImpl.getObjectGridConfigurations
   (ObjectGridManagerImpl.java:182)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid
.ObjectGridManagerImpl.createObjectGrid(ObjectGridManagerImpl.java:309)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid.test.
config.DocTest.main(DocTest.java:128)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R Caused by: java.lang
.IllegalArgumentException: No attributes are implemented
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at org.apache.crimson.jaxp.
DocumentBuilderFactoryImpl.setAttribute(DocumentBuilderFactoryImpl.java:93)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid.config
```

```
.XmlConfigBuilder.<init>(XmlConfigBuilder.java:133)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.websphere.objectgrid
.ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
```

문제점

사용하는 SDK에 스키마에서 XML 파일 유효성을 검증하는 데 필요한 JAXP 기능 구현이 들어 있지 않습니다.

솔루션

Apache Xerces를 다운로드하여 클래스 경로에 JAR을 포함시키면 XML 파일 유효성을 검증할 수 있습니다.

오브젝트 그리드 메시지

이 참조 정보에서는 오브젝트 그리드 사용 중 발생할 수 있는 메시지에 대한 추가 정보를 제공합니다. 메시지는 메시지 키로 식별되고 설명 및 사용자 응답을 포함합니다. 메시지는 정보, 경고 또는 오류일 수 있으며 메시지 키의 마지막 문자(I, W 또는 E)로 표시됩니다. 메시지의 설명 파트에서는 메시지의 발생 원인을 설명합니다. 메시지의 사용자 응답 파트에서는 경고 또는 오류 메시지인 경우 취해야 하는 조치를 설명합니다.

```
 CWOBJ0001E: Method, {0}, was called after initialization completed.
Explanation: After initialization completes, certain method
invocations are no longer accepted.
User response: Restructure your code so that the configuration
completes before use of the runtime is initiated.

 ICWOBJ0002W: ObjectGrid component is ignoring an unexpected
exception: {0}.
Explanation: CMSG0001
User response: CMSG0002

 CWOBJ0005W: The thread created an InterruptedException: {0}
Explanation: An InterruptedException occurred.
User response:Check the exception message to see whether this
interruption is expected.

 CWOBJ0006W: An exception occurred: {0}
Explanation: An exception occurred during the runtime.
User response: Check the exception message to see whether this is an
expected exception.

 CWOBJ0007W: The value null was specified for {0}, a default
value of {1} is used.
Explanation: A null value was specified for the variable. A
default value is used.
User response: Set the appropriate value. Refer to ObjectGrid
documentation to know the valid values for the variables or properties.

 CWOBJ0008E: The value {0} provided for the
property {1} is invalid.
Explanation: An invalid value was specified
for the variable.
User response: Set the appropriate value. Refer to ObjectGrid
document to know the valid values for the variables or properties.
```

CWOBJ0010E: Message key {0} is missing.
Explanation: A message key is missing in the message resource bundle
User response: CMSG0002

CWOBJ0011W: Cannot deserialize field {0} in class {1};
using the default value.
Explanation: During the deserialization of an object,
an expected field was not found. This field was probably not found
because the object was deserialized by a different version of the class
than the one that serialized it.
User response: This warning indicates a potential problem.
No user action is required unless further errors arise.

CWOBJ0012E: The LogElement type code, {0} ({1}), is not
recognized for this operation.
Explanation: An internal error occurred in the ObjectGrid runtime.
User response: CMSG0002

CWOBJ0013E: An exception occurred while attempting to evict
entries from the cache: {0}
Explanation: A problem occurred while attempting to apply
the eviction entries to the cache.
User response: Check the exception message to see whether this is
an expected exception.

CWOBJ0014E: The ObjectGrid runtime detected an attempt to
nest transactions.
Explanation: The nesting of transactions is not permitted.
User response: Modify the code to avoid the nesting of transactions.

CWOBJ0015E: An exception occurred while attempting
to process a transaction: {0}
Explanation: A problem occurred during transaction processing.
User response: Check the exception message to see
whether this exception is expected.

CWOBJ0016E: No active transaction is detected for the
current operation.
Explanation: An active transaction is necessary to
perform this operation.
User response: Modify the code to start a transaction
before performing this operation.

CWOBJ0017E: A duplicate key exception was detected
during the processing of the ObjectMap operation: {0}
Explanation: The key for the entry already exists in the cache.
User response: Modify the code to avoid inserting the same
key more than once.

CWOBJ0018E: The key was not found during the processing of the
ObjectMap operation: {0}
Explanation: The key for the entry does not exist in the cache.
User response: Modify the code to ensure that the entry exists before
attempting the operation.

CWOBJ0019W: Did not find data in the cache entry slot reserved
for {0} to use for ObjectMap name {1}.
Explanation: An internal error occurred in the
ObjectGrid runtime.
User response: CMSG0002

CWOBJ0020E: Cache entry is not in BackingMap {0}.
Explanation: Internal error in ObjectGrid runtime.
User response: CMSG0002

CWOBJ0021E: A usable ObjectTransformer instance was not found
during the deserialization of the LogSequence object for {0} ObjectGrid and
{1} ObjectMap.
Explanation: The receiving side of a LogSequence object
does not have the proper configuration to support the required

ObjectTransformer instance.
User response: Verify the configuration of the ObjectGrid
instances for both the sending and receiving sides of the LogSequence object.

CWOBJ0022E: The caller does not own mutex: {0}.
Explanation: An internal error occurred in the ObjectGrid runtime.
User response: CMSG0002

CWOBJ0023E: The CopyMode ({0}) is not recognized for this operation.
Explanation: An internal error occurred in the ObjectGrid runtime.
User response:CMSG0002

CWOBJ0024E: Cannot deserialize field {0} in class {1}.
Deserialization failed.
Explanation: During deserialization of an object, a required
field was not found. This problem is likely an ObjectGrid runtime error.
User response:CMSG0002

CWOBJ0025E: The serialization of the LogSequence object failed.
The number of serialized LogElement objects ({0}) does not match the number of
read LogElement objects ({1}).
Explanation: An internal error occurred in the ObjectGrid runtime.
User response:CMSG0002

CWOBJ0026E: The JMX credential type is not right. It should be of type {0}.
Explanation: The JMX credential type is not right. If basic
authentication is used, the expected type is String[] with the first element
being user name and the second being password. If the client certificate is
used, the expected type is Certificate[].
User response: Use the right credentials.

CWOBJ0027E: Internal runtime error. Clone method not supported: {0}
Explanation: An internal error occurred in the ObjectGrid runtime.
CLONE_METHOD_NOT_SUPPORTED_CWOBJ0027.useraction=CMSG0002

CWOBJ0028E: An error occurred in {0} for the map {1}. The key,
{2}, was not found in the map. LogElement type is {3}.
Explanation: An internal error occurred
when trying to evict an entry.
User response:CMSG0002

CWOBJ0029E: An error occurred in {0} for the map {1}. CacheEntry is
missing a {2} object for key {3}. LogElement type is {4}.
Explanation: An internal error occurred when trying to evict an entry.
User response:CMSG0002

CWOBJ0900I: The ObjectGrid runtime component is started for server {0}.
Explanation: The ObjectGrid component is started.
User response: None. Informational entry.

CWOBJ0901E: "{0}" system property is required to start ObjectGrid
component for server {1}.
Explanation: ObjectGrid runtime component requires "{0}" to be
specified as a Java Virtual Machine system property.
User response: See Information Center for using WebSphere Administrator
Console for providing ObjectGrid required custom properties.

CWOBJ0902W: Error prevented the ObjectGrid runtime component from starting
for server {0}.
Explanation:A prior error prevented the ObjectGrid component from starting.
User response: See prior error messages to determine what prevented
ObjectGrid component from starting.

CWOBJ0910I: The ObjectGrid runtime component is stopped for server {0}.
Explanation: The ObjectGrid component is stopped.
User response: None. Informational entry.

CWOBJ0911I: Starting the ObjectGrid runtime component for server {0}.
Explanation: The ObjectGrid component is starting.
User response: None. Informational entry.

CWOBJ1001I: ObjectGrid Server {0} is ready to process requests.
Explanation: ObjectGrid Server is ready to process requests.
User response: The services for this ObjectGrid Server are available.

CWOBJ1002E: Server port {0} is already in use.
Explanation: ObjectGrid server cannot be started due to port conflict.
User response: Users need to choose another port.

CWOBJ1003I: DCS Adapter service is disabled by configuration,
to enable it, please change your configuration with an endpoint defined.
Explanation:DCS adapter is turned off.
User response: Users can turn on DCS adapter by changing the configuration.

CWOBJ1004E: Server topic is null
Explanation: Server topic is null
User response:CMSG0002

CWOBJ1005E: The incoming request queue is null.
Explanation: Client request handler cannot retrieve requests.
User response:CMSG0002

CWOBJ1006E: The outgoing result queue is null.
Explanation: Client request handler cannot give result to client.
User response:CMSG0002

CWOBJ1007E: ObjectGrid client request is null.
Explanation: Client request handler cannot handle request that does
not contain any information about the request.
User response: Check your request

CWOBJ1008E: ObjectGrid client request TxID is null.
Explanation: We use TXID to match connections and do pooling, TXID cannot be null.
User response:CMSG0002

CWOBJ1009E: ObjectGrid client received a null response from the server.
Explanation: Encountered a null response from server.
User response:CMSG0002

CWOBJ1010I: Shutdown request is processing.
Explanation: Cluster servers are processing the shutdown request.
User response:none

CWOBJ1011I: Shutdown request is sending.
Explanation: Cluster servers are processing the shutdown request
User response:none

CWOBJ1012I: Shutdown request is performed.
Explanation: Cluster servers are processing the shutdown request.
User response:none

CWOBJ1110I: Starting the transport for ObjectGrid cluster {0} using
IP Address {1}, port {2}, transport type {3}.
Explanation: The ObjectGrid cluster member transport is starting.
User response: None. Informational entry.

CWOBJ1111W: Resolution of IP Addresses for host name {0} found only the
loopback address. The loopback address will be used.
Explanation: There may be a problem with the host name or DNS resolution.
For production related implementation, a non-loopback address is normally
expected.
User response: Modify the host name or determine if a DNS problem exits.

CWOBJ1112E: An error was encountered while looking up the IP address
for the host name of an ObjectGrid cluster member. The host name is {0}
and the server name is {1}. The member will be excluded from the cluster.
Explanation: Unable to resolve the IP address for the indicated host. The
ObjectGrid cluster member for the specified host will be excluded.
User response: Correct the host name lookup problem and retry.

CWOBJ1113E: ObjectGrid cluster transport service on this process
is not started. This cluster member is not defined in the configuration.
Explanation: This ObjectGrid cluster member is not a configured member

of the cluster. If this cluster member should be a member of a
ObjectGrid cluster, repair the configuration.
User response: Review the current configuration.

CWOBJ1114E: ObjectGrid cluster transport service on this process
could not process the incoming message. The message is {0} and the
exception is {1}.
Explanation: An unexpected internal error has been detected.
User response: Review the IBM ObjectGrid internet support web site for
a similar problem or contact IBM service.

CWOBJ1115E: An unrecognized view change event was received from the
ObjectGrid cluster transport. The view identifier is {0} and the event is {1}.
Explanation: The type of the event is not recognized. The HA Manager does
not know how to respond to the event.
User response: Review the IBM ObjectGrid internet support web site for a
similar problem or contact IBM service.

CWOBJ1116E: An attempt by another process to connect to this process
via the ObjectGrid cluster transport has been rejected. The connecting
process provided a name of {0}, a target of {1},
a member name of {2} and an IP address of {3}. The error message is {4}.
Explanation: The ObjectGrid cluster transport has rejected the
connection attempt.
User response: This may be a connection attempt from an unauthorized party.

CWOBJ1117E: An attempt to authenticate a connection has failed. The
exception is {0}.
Explanation: The ObjectGrid cluster transport has rejected the connection
attempt.
User response: This may be a connection attempt from an unauthorized party.

CWOBJ1118I: ObjectGrid Server Initializing [Cluster: {0} Server: {1}].
Explanation: The ObjectGrid cluster member is initializing.
User response: None. Informational entry.

CWOBJ1119I: ObjectGrid client failed to connect to host: {0} port: {1}.
Explanation: ObjectGrid client failed to connect.
User response: None. Informational entry.

CWOBJ1120I: ObjectGrid Client connected successfully to host: {0} port: {1}.
Explanation: ObjectGrid Client connected successfully.
User response: None. Informational entry.

CWOBJ1201E: No valid client access end points are defined.
Explanation:=No valid client access end points are defined.
User response: Define a valid client access end point.

CWOBJ1202E: SSL Server Socket failed to initialize. The exception
message is {0}
Explanation:SSL Server Socket fails to initialize. The SSL settings
might be wrong or the port number is already in use.
User response: Examine the exception to see what went wrong.

CWOBJ1203W: Received a timeout event from the server for
transaction: {0}
Explanation: Client did not receive expected response message
from the server within a configured timeout limit.
User response: Look for prior messages that may explain the timeout.
If none found, try increasing the timeout limit.

CWOBJ1204W: Received a message of unknown message type.
The message is: {0}
Explanation: An unexpected internal error has been detected.
User response: Review the IBM ObjectGrid internet
support web site for a similar problem or contact IBM service.

CWOBJ1205E: SSL Initialization failed. The exception
message is {0}
Explanation:SSL Initialization failed. The SSL settings
might be wrong.
User response: Examine the exception to see what
went wrong.

CWOBJ1206W: SSL Initialization failed. The exception message is {0}
Explanation:SSL Initialization failed. The SSL settings might be wrong.
User response: Examine the exception to see what went wrong.

CWOBJ1207W: The property {0} on plug-in {1} is using an
unsupported property type.
Explanation: Java primitives and their java.lang counterparts
are the only supported property types. java.lang.String is also supported.
User response: Check the property type and change it to
one of the supported types.

CWOBJ1208W: The specified plug-in type, {0}, is not one of
the supported plug-in types.
Explanation: This type of plug-in is unsupported.
User response: Add one of the supported plug-in
types.

CWOBJ1211E: The Performance Monitoring Infrastructure
(PMI) creation of {0} failed. The exception is {1}.
Explanation: An attempt to create ObjectGrid PMI failed.
User response: Examine the exception message
and the first failure data capture (FFDC) log.

The following messages are used to gather the user id and password on the panel
or standard input.
LOGIN_PANEL_TITLE=Login at the target server
GENERIC_LOGIN_PROMPT=Enter login information
USER_ID=User identity
PASSWORD=Password
OK=OK
CANCEL=Cancel

CWOBJ1215I: ObjectGrid Transaction Propagation Event Listener is
initializing [ObjectGrid {0}].
Explanation: This informational message indicates that the ObjectGrid Transaction
Propagation Event Listener is initializing.
User response: None. Informational entry.

CWOBJ1216I: ObjectGrid Transaction Propagation Event Listener
is initialized [ObjectGrid {0}].
Explanation: ObjectGrid Transaction Propagation Event Listener Initialized.
User response:=None. Informational entry.

CWOBJ1217I: ObjectGrid Transaction Propagation Service Point Initialized
[ObjectGrid {0}].
Explanation: This informational message indicates that the ObjectGrid
Transaction Propagation Event Listener is initialized.
User response: None. Informational entry.

CWOBJ1218E: ObjectGrid Transaction Propagation Event Listener failure
occurred [ObjectGrid {0} Exception message {1}].
Explanation: ObjectGrid runtime encountered an ObjectGrid Transaction
Propagation failure.
User response: Examine the exception to determine the failure.

CWOBJ1219E: ObjectGrid Transaction Propagation Service End Point
failure occurred [ObjectGrid {0} Exception message {1}].
Explanation: ObjectGrid runtime encountered an ObjectGrid Transaction Propagation
Service End Point failure.
User response: Examine the exception to determine the failure.

CWOBJ1220E: ObjectGrid Transaction Propagation Service is
not supported in this environment.
Explanation:=ObjectGrid Transaction Propagation Service is

not supported on z/OS or the standalone ObjectGrid server environment.
User response: Do not use ObjectGrid Transaction Propagation
Service on z/OS or in the standalone ObjectGrid server environment

CWOBJ1300I: Adapter successfully initialized ObjectGrid.
Explanation: Adapter successfully initialized ObjectGrid.
User response: None. Informational entry.

CWOBJ1301E: Adapter failed to initialize ObjectGrid. Exception occurred
[Exception message {0}].
Explanation: Adapters attempt to initialize ObjectGrid failed.
User response: Examine the exception to determine the failure.

CWOBJ1302I: Adapter stopped.
Explanation: Adapter stopped.
User response: None. Informational Only.

CWOBJ1303I: Adapter started.
Explanation:PMA_CWOBJ1303.explanation=Adapter started.
User response: None. Informational Only.

CWOBJ1304I: ObjectGrid security is enabled.
Explanation: ObjectGrid security is enabled.
User response:none

CWOBJ1305I: ObjectGrid security is disabled.
Explanation: ObjectGrid security is disabled.
User response:none

CWOBJ1306W: Cannot retrieve the client certificates from the SSL socket.
Explanation: For some reason, the runtime cannot retrieve the client
certificates from the SSL socket.
User response: Check your SSL configurations.

CWOBJ1307I: Security of the ObjectGrid instance {0} is enabled.
Explanation: Security of the ObjectGrid instance {0} is enabled.
User response:none

CWOBJ1308I: Security of the ObjectGrid instance {0} is disabled.
Explanation: Security of the ObjectGrid instance {0} is disabled.
User response:none

CWOBJ1309E: Unexpected error occured in the
connect token creation: {0}.
Explanation: An unexpected error occurs in the
connection token creation.
User response: Check the security configuration

CWOBJ1310E: An attempt by another process to
connect to this process via the core group transport has been rejected. The
connecting process provided a source core group name of {0}, a target of {1},
a member name of {2} and an IP address of {3}. The error message is {4}.
Explanation: The High Availability Manager has
rejected a connection attempt.
User response: This may be a connection attempt
from an unauthorized party.

CWOBJ1400W: Detected multiple ObjectGrid runtime JARS files in the JVM.
Using multiple ObjectGrid runtime JAR files may cause problems.
Explanation: Usually only one ObjectGrid runtime JAR should be
found in a JVM.
User response: Use the appropriate ObjectGrid runtime JAR for
your configuration.

CWOBJ1401E: Detected a wrong ObjectGrid runtime JAR file for this
configuration. Detected configuration is {0}. Expected Jar file is {1}.
Explanation: Each ObjectGrid runtime JAR file corresponds to a particular
supported configuration.
User response: Use the appropriate ObjectGrid runtime JAR for your configuration.

CWOBJ1402E: ObjectGrid connection link callback not found for id: {0}.
Explanation: Internal error in ObjectGrid runtime.
User response:CMSG0002

CWOBJ1500E: An exception occurred when attempting to create a GroupName for HA Group ({0}): {1}.
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1501E: An exception occurred when member ({0}) attempted to join HA Group ({1}): {2}.
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1503E: Cannot access ObjectGrid ({0}) for applying updates to replica member ({1}).
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1504E: An exception occurred when attempting to process the LogSequences for replica ({0}): {1}.
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1505E: More than one replication group member reported back as the primary. Only one primary can be active. ({0}).
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1506E: More than one primary replication group member exists in this group ({1}). Only one primary can be active. ({0}).
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1507W: An exception occurred when attempting to end the replication process for BackingMap ({0}): {1}.
Explanation: While attempting to shut down a primary replication group member, an exception occurred during the clean up processing.
User response:CMSG0002

CWOBJ1508E: An exception occurred when attempting to send message ({0}) from sender ({1}) to receiver ({2}): {3}.
Explanation:A problem occurred while attempting to send a message between replication group members.
User response:CMSG0002

CWOBJ1509E: An exception occurred when attempting to serialize message ({0}): {1}.
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1510E: An exception occurred when attempting to inflate message ({0}): {1}.
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1511I: {0} ({1}) is open for business.
Explanation: Specified replication group member is now ready to accept requests.
User response: None.

CWOBJ1512W: {0} already exists in replication group {1}.
Explanation: The specified replication group member is already active in this replication group.
User response: None.

CWOBJ1513E: Synchronous replication failed on {0} ({1}). This member is no longer active.
Explanation:A problem was encountered that prevented

synchronous replication from successfully completing.
User response: Review previous messages in the log to help
diagnose the problem. Stopping and restarting the specified server may be required.

CWOBJ1514I: Primary ({0}) is being downgraded to either a
replica or standby.
Explanation: This is not a normal operation, but ObjectGrid
processing can continue.
User response:CMSG0002

CWOBJ1515I: Minimum configuration requirements not satisfied
for replication group ({0}).
Explanation: The necessary primary and replica configuration
requirements were not met with the recent replication group member change.
User response: Wait for additional resources to be started and
recognized for this configuration.

CWOBJ1516E: An exception occurred when attempting
to activate the replication process for ObjectGrid ({0}): {1}.
Explanation: While attempting to start a primary
replication group member, an exception occurred during the activation processing.
User response:CMSG0002

CWOBJ1517E: Synchronous replication failed for
transaction {2} on {0} ({1}). This member is no longer active.
Explanation:A problem was encountered that prevented
synchronous replication from successfully completing.
User response: Review previous messages in the log to help
diagnose the problem. Stopping and restarting the specified server may
be required.

CWOBJ1518E: An exception occurred when attempting to
commit replica transaction ({0}) for primary transaction ({1}) on
Replica ({2}): {3}.
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1519E: An exception occurred when attempting
to rollback the LogSequences for replica ({0}): {1}.
Explanation:CMSG0001
User response:CMSG0002

CWOBJ1610W: Try to reset a null cluster for {0}.
Explanation: Replication group cluster data are not available.
User response:none

CWOBJ1611I: Replication group cluster {0} is open for business.
Explanation: Now replication group cluster can accept requests.
User response:none

CWOBJ1612I: Replication group cluster {0} is closed for business.
Explanation: Now replication group cluster cannot accept requests.
User response:=none

CWOBJ1620I: Replacing target for wrongly routed request due to
changes in the server. The new target is {0}.
Explanation: Old routing target replaced with new target.
User response: If the intended replication group is out of
service, you need to bring it back.

CWOBJ1630I: Replication group cannot serve this request {0}.
Explanation: Routing is refused due to the unavailable service
such as the Domino effect
User response: Information only.

CWOBJ1632E: Original request does not have a valid ID; no way to forward
this request.
Explanation: No way to forward this request because the original request
does not have a valid ID.
User response: Report to IBM support

CWOBJ1634I: Router cannot find the forwarding target; using blind forwarding.
Explanation: Router cannot find the forwarding target.
User response: None

CWOBJ1660I: Replication group member has changed. This server
does not host what is requested anymore. The original request is {0}.
Explanation: Replication group member has changed.
User response: If the intended replication group is out of service,
you need to bring it back.

CWOBJ1661I: Cluster data are updated for replication group:
{0}
Explanation: Cluster data are updated
User response:none

CWOBJ1663E: Server router cannot verify server routing for {0},
because cluster data for this replication group are null in the server.
Explanation: No replication group cluster data are
available to verify.
User response: Report to IBM support

CWOBJ1668W: Request is coming to the server that has not completely started.
N**Explanation:**Server startup takes 60-120 seconds. Request will
be automatically retried if you have configured so (by default the request will be
automatically retried).
User response: Adjust your configuration or start your clients 60-120
seconds after you start your servers.

CWOBJ1680W: The configured TCP connection timeout is smaller than
retryInterval * max(startupRetries, maxRetries), so there is possibility that
connection will time out.
Explanation: The configured TCP connection timeout should be larger
than retryInterval * max(startupRetries, maxRetries).
User response: Adjust your configuration.

CWOBJ1682W: The configured transaction timeout is smaller than maxForwards *
retryInterval * max(startupRetries, maxRetries), so there is possibility that
transaction will time out.
Explanation: The configured transaction timeout should be larger than
maxForwards * retryInterval * max(startupRetries, maxRetries).
User response: Adjust your configuration.

CWOBJ1700I: Standalone HAManager
is initialized with coregroup {0}.
Explanation:standalone HAManager
is initialized successfully.
User response:none

CWOBJ1701I: Standalone HAManager is already initialized.
Explanation: Standalone HAManager is
already initialized successfully.
User response:none

CWOBJ1702E: Standalone HAManager is
not initialized, so it cannot be started.
Explanation: Standalone HAManager is not initialized.
User response: Initialize it
before starting it.

CWOBJ1710I: Standalone HAManager is
started successfully.
Explanation: Standalone HAManager is
started successfully.
User response:none

CWOBJ1711I: Standalone HAManager is already started successfully.
Explanation: Standalone HAManager is
already started successfully.
User response:none

CWOBJ1712E: Standalone HAManager is not started.
Explanation: Standalone HAManager is not started.
User response: Initialize and start it before using it.

CWOBJ1713E: Standalone HAManager failed to start.
Explanation: Standalone
HAManager failed to start.
User response: Check if ports
are used already.

CWOBJ1720I: HAManager Controller detected that
ObjectGrid server is in the WebSphere environment, using WebSphere HAManager
instead of initializing and starting standalone HAManager.
Explanation: ObjectGrid server is running in the WebSphere environment.
User response: None

CWOBJ1730I: HAManager Controller detected
that the WebSphere external HAManager is null.
Explanation: Cannot get the external HAManager from WebSphere.
User response: None

CWOBJ1790I: Need to initialize and
start the standalone HAManager.
Explanation: Cannot get the external HAManager from WebSphere.
Need to initialize and start the standalone HAManager.
User response: None

CWOBJ1792I: The maximum number of threads is {0} and the minimum number
of threads is {1}.
Explanation: Configure thread pool.
User response: Information only.

CWOBJ1800I: Forwarding is required for request {0} with response of {1}.
Explanation: Forward
routing is required.
User response: None. Handled
automatically

CWOBJ1810I: Forwarding is required for response {0}.
Explanation: Forwarding is required for response.
User response: None

CWOBJ1811E: Forwarding is required, but the
original request cannot be found.
Explanation: Forwarding is required, but the
original request cannot be found.
User response: None

CWOBJ1820E: Forwarding request does not have a
replication group identifier.
Explanation: There is not any replication group
identifier in this forwarding request.
User response: Contact IBM Support

CWOBJ1870I: Server service is not available for response {0}.
Explanation: Server service is not available due to the Domino
effect or other events.
User response: Bring at least the minimum number of servers up.

CWOBJ1871E: Detected unavailable service, received null
response, no way to retry.
Explanation: Null response from the unavailable service
User response: Contact IBM support

CWOBJ1872I: Service is unavailable with response of {0}.
Explanation: Service is not available
User response: Bring at least the minimum number of
servers up or check if server startup is successful.

CWOBJ1890I: Re-routing request {0} due to
an un-responsive server.
Explanation: The request for intended server failed
to complete. Request was re-routed to another server.
User response: None. Handled automatically. If the intended
replication group is out of service, you need to bring it back.

CWOBJ1891E: All servers are not available in replication group {0}.
Explanation: All servers were either not started or have
failed. They are not available
User response: If the intended replication group is out of service,
you need to bring it back.

CWOBJ1898W: Forwarding is required, but router cannot find new
available target for response {0}
Explanation: Service is not available.
User response: Make service available.

CWOBJ1899W: Forwarding is required, but router cannot find right
replication group for response {0}
Explanation: Replication group ID is lost.
User response: Contact IBM Support

CWOBJ1900I: Client server remote procedure call service is initialized.
Explanation: Client server remote procedure call service is initialized.
User response: None

CWOBJ1901I: Client server remote procedure call service is started.
Explanation: Client server remote procedure call service is started.
User response: None

CWOBJ1902I: Client server remote procedure call handler
threads are started.
Explanation: Client server remote procedure call handler
threads are started.
User response: None

CWOBJ1903I: Configuration network service is initialized.
Explanation: Configuration network service is initialized.
User response: None

CWOBJ1904I: Configuration network service is started.
Explanation: Configuration network service is started.
User response: None

CWOBJ1905I: Configuration handler is started.
Explanation: Configuration handler is started.
User response: None

CWOBJ1913I: System administration network service is initialized.
Explanation: System administration network service is initialized.
User response: None

CWOBJ1914I: System administration network service is started.
Explanation: System administration network service is started.
User response: None

CWOBJ1915I: System administration handler is started.
Explanation: System administration handler is started.
User response: None

CWOBJ2000E: No member in this replication group {0}.
Explanation: No member can be found in this replication group.
User response: Check if servers are started or data are available

CWOBJ2001W: No available member in this replication group {0}.
Explanation: No available member can be found in this replication group.
User response: Check if server service is available

CWOBJ2002W: No available routing table for this replication group {0}.
Explanation: No available routing table for this replication group.
User response: Check if clients have brought in routing table

CWOBJ2003I: Cannot find routing cache for cache key {0}, creating new routing cache.
Explanation: First time routing or cluster changes.
R**User response:**none

CWOBJ2010E: Target for this request is null.
Explanation: Request did not come with target information.
User response:contact IBM support.

WOBJ2060I: Client received new version of replication group cluster {0}.
Explanation: Client received new version of replication group cluster
User response:none

CWOBJ2068I: Reachability control detected problem in replication group member {0}.
Explanation: Some server cannot be reached, reachability mechanism will handle it.
User response: None.

CWOBJ2069I: Reachability control timer releases replication group member {0}.
Explanation: This member is available for routing.
User response:none

CWOBJ2086I: Routing thread control is activated due to overload for replication group {0}.
Explanation: Thread control is in action.
User response:none

CWOBJ2088I: Reachability control is activated to regulate the server availability for replication group {0}.
Explanation: Reachability is in action.
User response:none

CWOBJ2090W: Cannot find routing table for replication group {0}.
Explanation: Replication group cluster is null.
User response:none

CWOBJ2091W: Routing table is not null, but it does not contain any servers for replication group {0}.
Explanation: Replication group cluster is empty.
User response:none

CWOBJ2092I: Routing table is null in runtime for replication group {0}.
Explanation: Getting routing table from runtime.
User response:none

CWOBJ2093I: Routing table is not null in replication group cluster store for replication group {0}
Explanation: Getting routing table from cluster store.
User response:none

CWOBJ2096I: Routing table was obtained from replication group cluster store for replication group {0}.
Explanation: Obtained replication group cluster from replication group cluster store.
User response:none

CWOBJ2097I: Routing is based on round robin algorithm for replication group {0}.
Explanation: Routing is based on round robin algorithm.
User response:none

CWOBJ2098I: Routing is based on random selection for replication group {0}.
Explanation: Routing is based on random selection.
User response:none

CWOBJ2100I: Connection ({0}) is stale, it cannot be reused.
Explanation: Connection is stale.
User response:none

CWOBJ2101W: Connection cannot be acquired after the maximum wait time.
Explanation: There are not any connections left in the pool.
User response: Increase the maximum number of connections in the configuration.

CWOBJ1600I: ManagementGateway service started on port ({0}).
Explanation: ManagementGateway service is ready to process requests.
User response: ManagementGateway service is available.

CWOBJ1601E: ManagementGateway service failed to start on port ({0}).
Explanation: ManagementGateway service failed to start.
User response: Ensure specified port is not already in use.

CWOBJ1602E: ManagementGateway service failed to connect to server at ({0}):({1}).
Explanation: ManagementGateway service failed to connect to server.
User response: Ensure server is running.

CWOBJ1603E: Management service failed to respond to ({0}) remote request.
Explanation:CMSG0001
User response:CMSG0002

CWOBJ2400E: Invalid Configuration: backing map {0} is a member of more than one map-set.
Explanation:A backingMap can belong to only one map-set.
User response: Edit the cluster XML file so that each backing map belongs to only one map-set.

CWOBJ2401E: Invalid Configuration: backing map {0} in distributed ObjectGrid {1} is not in a map-set.
Explanation: Each backing map of a distributed ObjectGrid must be placed in a map-set.
User response: Edit the cluster XML file so that each backing map in a distributed ObjectGrid belongs to a map-set.

CWOBJ2402E: Invalid Configuration: map-set has a reference to a {0} map. This backing map does not exist in the ObjectGrid XML file.
Explanation: Each map within a map-set must reference a backing map from the ObjectGrid XML file.
User response: Edit the XML file(s) so that each map within the map-set references a backing map from the ObjectGrid XML file.

CWOBJ2403E: The XML file is invalid. A problem has been detected with {0} at line {1}. The error message is {2}.
Explanation: The XML file does not conform to the schema.
User response: Edit the XML file so that it is conforms to the schema.

CWOBJ2404W: The value specified for {0} is {1}. This is an invalid value. {0} will not be set.
Explanation: The value for this configuration attribute is not valid.
User response: Set the configuration attribute to a proper value in the XML file.

CWOBJ2405E: The objectgrid-binding ref {0} in the Cluster XML file does not reference a valid ObjectGrid from the ObjectGrid XML file.
Explanation: Each of the objectgrid-bindings must reference an ObjectGrid from the ObjectGrid XML file.
User response: Edit the XML files so that the objectgrid-binding in the Cluster XML references a valid ObjectGrid in the ObjectGrid XML.

CWOBJ2500E: Failed to start ObjectGrid server {0}.
Explanation: The ObjectGrid server failed to start properly.
User response: Check the log for exceptions.

CWOBJ2501I: Launching ObjectGrid server {0}.
Explanation: An ObjectGrid server is starting up.
User response:none

CWOBJ2502I: Starting ObjectGrid server using ObjectGrid XML file URL "{0}" and Cluster XML file URL "{1}".
Explanation: An ObjectGrid server is starting using a cluster XML file and an ObjectGrid xml file.
User response:none

CWOBJ2503I: Bootstrapping to a peer Objectgrid server on host {0}
and port {1}.
Explanation: This ObjectGrid server will bootstrap to a peer server
to retrieve information required to start.
User response:none

COBJ2504I: Attempting to bootstrap to a peer ObjectGrid server using
the following host(s) and port(s) "{0}".
Explanation: This ObjectGrid server will use the list of hosts and ports
provided in an attempt to connect to a peer ObjectGrid server.
User response:none

CWOBJ2505I: Attempting to bootstrap to a peer ObjectGrid server using
the Cluster XML file URL "{0}".
Explanation: This ObjectGrid server will use the list of servers in the
Cluster XML file in an attempt to connect to a peer ObjectGrid server.
User response:none

CWOBJ2506I: Trace is being logged to {0}.
Explanation: The trace file has been set on the command line.
User response: See the specified trace file for ObjectGrid server
start-up trace.

CWOBJ2507I: Trace specification is set to {0}.
C**Explanation:**The trace specification has been set on the command line.
User response:none

CWOBJ2508I: A security properties file "{0}" has been specified and
will be used to start the server.
Explanation:A security properties file has been provided to start a
secure server.
User response:none

CWOBJ2509E: Timed out after waiting {0} seconds for the server to start.
Explanation: The ObjectGrid server did not start within the timeout interval.
User response: Check the log for exceptions.

CWOBJ2510I: Stopping ObjectGrid server {0}.
Explanation: Stopping ObjectGrid server.
User response:

CWOBJ2511I: Waiting for the server to stop.
Explanation: Waiting for the ObjectGrid server to stop.
User response:none

CWOBJ2512I: ObjectGrid server {0} stopped.
Explanation: ObjectGrid server stopped.
User response:none

CWOBJ2513E: Timed out after waiting {0} seconds for the server to stop.
Explanation: The ObjectGrid server did not stop within the timeout interval.
User response: Check the log for exceptions.

CWOBJ2514I: Waiting for ObjectGrid server activation to complete.
Explanation: The ObjectGrid server has launched. Waiting for the server to
complete activation.
User response:none.

CWOBJ2515E: The arguments provided are invalid. Here are the valid
arguments.{0}{1}
Explanation: The arguments provided to this script are invalid.
User response: Enter valid arguments.

CWOBJ2516I: ObjectGrid server has completed activation.
Explanation: The ObjectGrid server is active and ready to process requests.
User response:none.

CWOBJ2517I: Successfully bootstrapped to peer Objectgrid server
on host {0} and port {1}.
Explanation: This ObjectGrid server has successfully bootstrapped
to a peer server to retrieve information required to start this server.
User response:none

CWOBJ2407W: The {0} property on the {1} plug-in class could not be set. The exception is {2}.
Explanation: The property for this plug-in could not be set.
User response: See the exception for more information.

주의사항

이 책에서 IBM 제품, 프로그램 또는 서비스를 언급하는 것이 IBM이 영업하는 모든 나라에서 이들 제품, 프로그램 또는 서비스를 사용할 수 있다는 것을 의미하지는 않습니다. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다는 것이 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 타사의 기능상 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수 있습니다. IBM이 명시적으로 지정한 경우를 제외하고, 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 확인은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

(i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 본 프로그램에 관한 정보를 얻고자 하는 라이센스 사용자는 다음 주소로 문의하십시오.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터

이러한 정보는 해당 조건(예를 들어, 사용료 지불 등)하에서 사용될 수 있습니다.

상표 및 서비스표

다음 용어는 미국 또는 기타 국가에서 사용되는 IBM Corporation의 상표입니다.

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java 및 모든 Java 기반 상표는 미국 또는 기타 국가에서 사용되는 Sun Microsystems, Inc.의 상표입니다.

LINUX는 미국 또는 기타 국가에서 사용되는 Linus Torvalds의 상표입니다.

Microsoft, Windows, Windows NT 및 Windows 로고는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 The Open Group의 등록상표입니다.

기타 회사, 제품 및 서비스 이름은 타사의 상표 또는 서비스표입니다.