WebSphere® Application Server V5.0:

HTTP Session Management

WebSphere. software

e business software

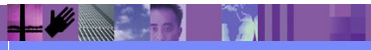
Updated 5/16/2003

Objectives

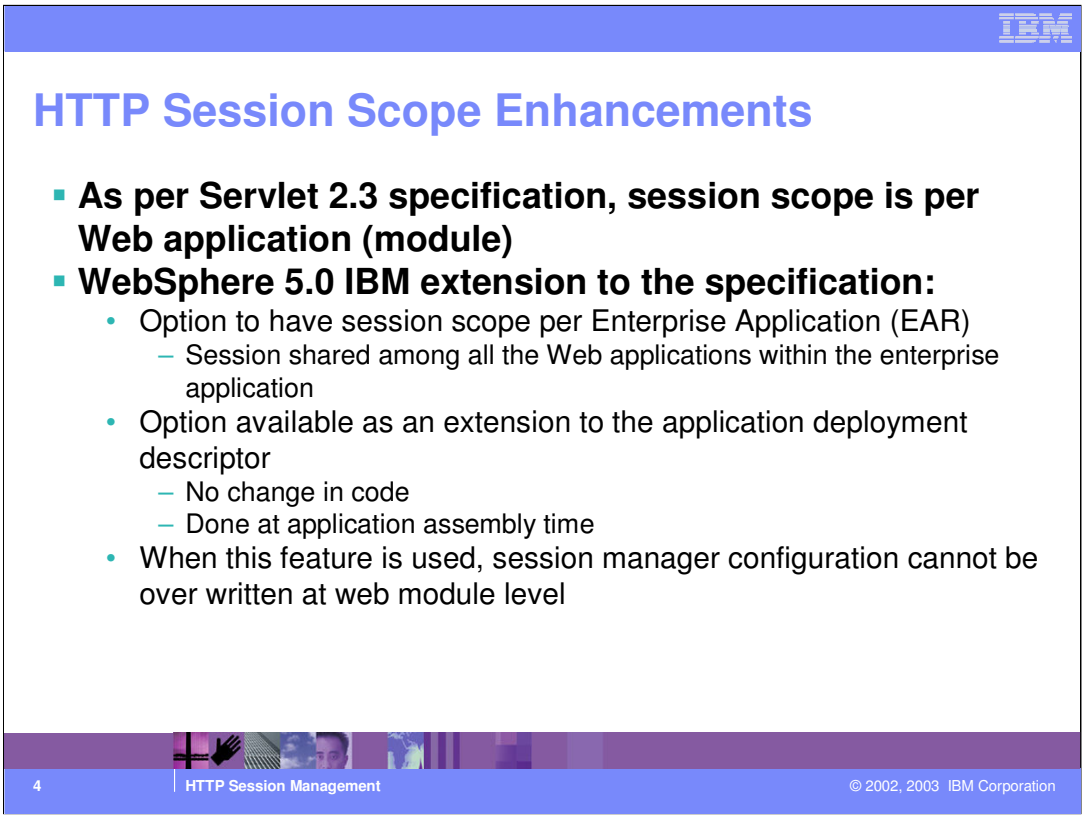
- **Discuss the enhancements to the standard HTTP session support**
- **Explain session persistence in WebSphere 5.0**
  - Including WebSphere Internal Messaging
- **Communicate Troubleshooting Tips**

Session Manager Configuration Enhancement

- **In WebSphere 4.0:**
  - By default, SessionManager configuration is at the Application Server level (Web Container)
  - Configuration applied to all web modules within that server
- **New in WebSphere 5.0:**
  - SessionManager configuration can be defined for:
    - Application Servers
    - Enterprise Applications
    - Web Modules
  - Configurations can be set in the Admin Console



In 3.x/4.0 the Session manager is configurable at the Application server level. So all the applications share the same SessionManager settings. This doesn't allow isolation at the application level. Different applications might have different level of requirement on session manager, like persistence, cookies etc. This will be useful for the ASPs who run more than one application on a server. This also reduces the load the database when used with session persistence because applications do not have to share the same database for session persistence. By Default, SessionManager settings defined at the application server level are inherited by the Enterprise applications and by the web modules within them.

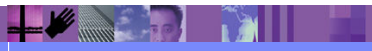


The Servlet 2.2 Specification limited the scope of sessions to the web module, and Servlet 2.3 does not remove this restriction. However, WebSphere 5.0 offers an extension to the specification, such that you can configure session scoping on the Enterprise application. This feature is not in the Admin Console, but available in the Application Assembly tool and in Application Developer.

There is also a configuration option to handle session persistence at the Enterprise application level and at the Web Module level. If you chose to scope sessions at the Enterprise application level, you will not be able to alter the session persistence at the module level.

HTTP Session Enhancements

- **Session Affinity Improvement: server order based session affinity**
  - To ensure that subsequent concurrent request for session when a cluster member is dead does not end up in 2 different random clones.
  - Plug-in maintains a list of cluster member order to go to for session failover
- 5.0.1+ **Session key length for wireless devices**
  - Wireless devices do not typically support cookies
  - Web Container>Custom Properties:
    - SessionIdentifierMaxLength
    - HttpSessionCloneId



Cluster Member order based session affinity: When a clone member is marked dead, the Version 4.0 plug-in directs the request to one of the available clones randomly. In this case, there is a chance that two concurrent requests for a session after clone went down might end up in different clones. To avoid this, the Version 5.0 plug-in will maintain the cluster member list in order and rather than picking randomly. The plug-in will pick the cluster member next in its list to avoid the breaking of session affinity. This is also useful in case of URL-rewriting where cluster member information doesn't get updated if rewritten URLs aren't updated.

Speaking of URL Rewriting, most Wireless Application Protocol (WAP) devices do not support cookies. The preferred way to track sessions for WAP devices is to use URL rewriting. However on most WAP devices, the maximum allowed URL length is 128 characters. With URL rewriting, a session identifier is added to the URL itself, effectively decreasing the space available for the actual URL and the number of parameters that can be sent on a request.

As of Fix Pack 1, you can reduce the length of session identifier, by configuring the session ID length and clone ID on an application server's web container.

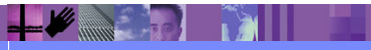
HTTP Session Enhancements

- **Configure access to the Session - Concurrent or Serial**
  - Session can be accessed concurrently
    - It is the developer's responsibility to serialize the access to the session
  - Option to provide a serialized access to the session in a given JVM so that access to the thread safe attributes in session can be avoided
    - No special code required by the developer
  - When not to use:
    - Not recommended when framesets are used heavily

Serialize the access to the Session: As per servlet API, session can be accessed concurrently and it would be responsibility of the developer to serialize the access to the session. As a quality of service, we provide a serialized access to the session in a given JVM so that access to the thread safe attributes in session can be avoided without having to write any special code by the developer. The wait time is configurable.

HTTP Session Replication

- **Two ways to have HTTP session state persistence (needed for failover):**
  - In database - similar to WebSphere 4.0
  - Replication of HTTP session state in memory
    - New in WebSphere 5.0
    - Multiple topology models possible:
      - Single Replica ('Buddy')
      - Dedicated replication server
      - N-way peer-peer
    - All features available in database persistence is available here, except DB2® variable row size and Multi-row features



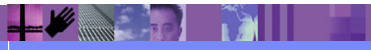
We still have the capability of persisting sessions to a data table, as we did in 4.0. We also have the capability of persisting sessions to a memory location.

Memory to Memory replication can use one of three basic topologies - Single replica, client/server and peer-to-peer. Each has advantages which we examine in the slides to come.

Memory to memory session replication has all the features that database persistence has (except the database-specific options, like row size).

HTTP In Memory Session Replication

- **Configure replication topology; system handles the rest**
- **Replications driven by HTTP session manager**
  - at end of HTTP request, or
  - in synch(), or
  - at timer interval (default)
- **Data transfer between JVMs can be encrypted**
- **Features available in WebSphere ND and above**
- **Performance advantage over DB persistence**



Configuration is very straightforward; you configure the topology, the system handles the rest.

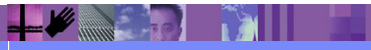
When to persist is also configurable - at the end of each request, at a timer interval, or when directed by the application code - which is to say, when the synch() method is called.

Memory to Memory Replication traffic can be encrypted. It is available in WebSphere Network Deployment and above.

Session persistence to memory has some performance advantages over database persistence, and scales well.

WebSphere Internal Messaging

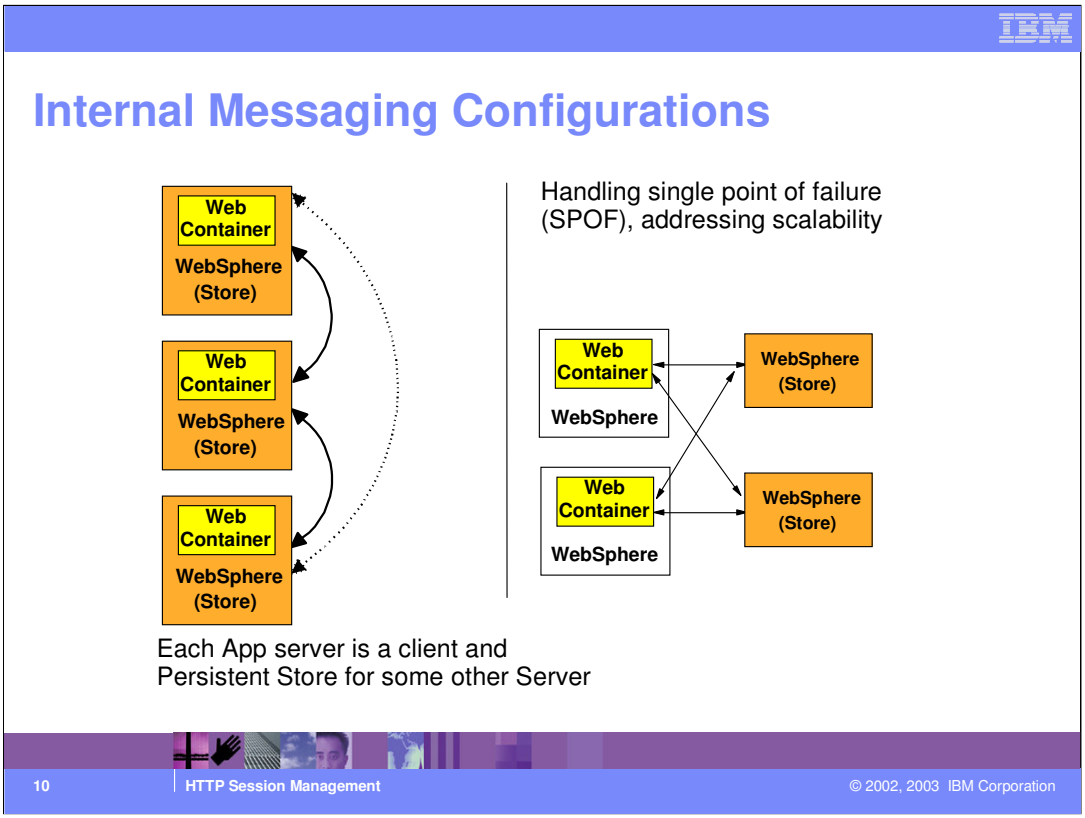
- **Build on small, fast 'pub/sub' engine**
- **Runs in an existing server process**
  - separate thread per queue
- **Can be set up peer-to-peer or client-server**
  - Default is peer-to-peer
- **Multiple 'channels' available for partitioning**
- **Also used for replication of Dynamic Cache data**



WebSphere Internal Messaging is built on a small, fast publish/subscribe engine which uses a JMS-like function to communicate session information from place to place. There is a separate thread spawned for each queue. By default, the topology is peer-to-peer, but that particular topology, while requiring the least configuration, does not scale as well as the others.

One feature to permit better scaling is Partitioning, which we'll detail in a few slides.

Session data is not all that can be replicated - Dynamic Data caches can also be moved from machine to machine using WebSphere Internal Messaging.



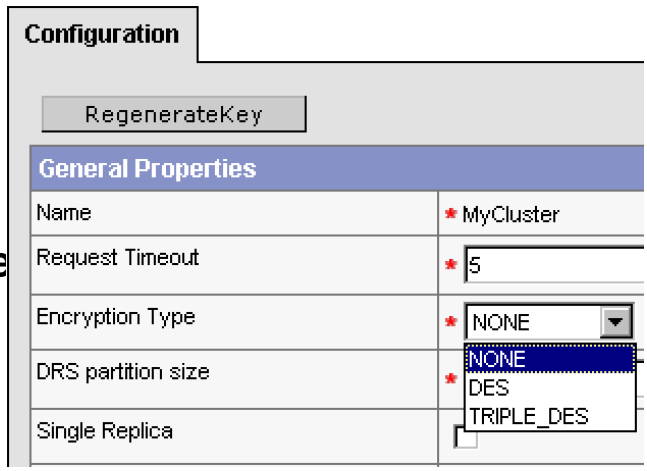
WebSphere Internal Messaging can be configured as N-way Peer-to-Peer, as on the left. While this is the simplest to configure, there are performance implications when scaling up to large environments - because every server has to persist the sessions of every other server.

A special case of Peer-to-peer is the Single Replica option, where session info is persisted to only one other application server. This scales well, but introduces something like a single point of failure - i.e., if BOTH application servers are taken out at once (for instance, the whole node goes down), then persistence breaks.

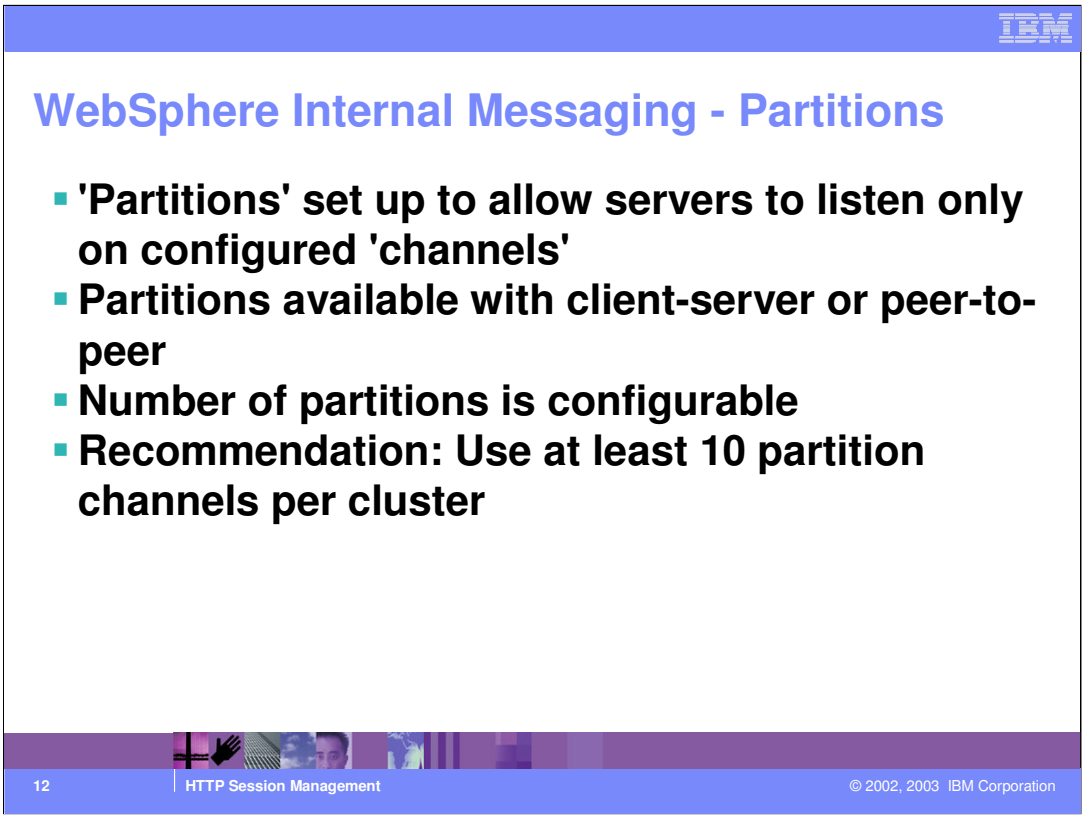
So there is another topology option - configuring Client/Server setup, where one app server is dedicated to persist all the sessions for all the app servers. This provides failover, but then the server becomes a single point of failure. So we can set up to have two app servers at different locations both persisting the sessions for all the other app servers, as illustrated on the right.

WebSphere Internal Messaging - Security

- **Environment-> Internal Replication Domains-> <name>**
- **Messages can be encrypted - DES or Triple DES**



The messaging traffic that carries session information from replicator to replicator can be encrypted for added protection in the runtime environment. A button is provided to regenerate the encryption key that is used.



Partitioning is for a large environment, so that members of one cluster need not deal with the 'chatter' from other clusters. Each cluster member persists its session data to the configured listeners - that is, all the other app servers listening to that partition ID.

This can be configured to match the environment. If it is needful that an application server be dedicated to the task of session persistence, and serve no end-user requests, it can still be set up to handle only certain partition IDs. Or each application server can listen to all the available partition IDs in a peer-to-peer scenario.

If a cluster is constructed such that not all members listen to the same partition IDs, internal logic will add the partition ID necessary to handle sessions should an application server fail.

It is suggested that each segment of a replication environment consist of at least ten partition IDs, so that cluster members have multiple paths to communicate with the application server that is persisting its session data.

These partitions/groups correspond to Topics. There is one and only one thread associated with a given subscriber for a Topic. Through testing, we selected 10 subscriber threads to handle the traffic.

Performance Viewer - New Session Counters

Name	Description	Value	Select
Created Sessions	Number of sessions that were created	1	☑
Invalidated Sessions	Number of sessions that were invalidated	1	☑
Session Life Time (ms)	Average session life time in milliseconds (tim...	2,122,512	☑
Active Sessions	Total number of sessions that are currently be...	0.0 (Avg: 0.0...	☐
Live Sessions	Total number of sessions that are currently live	0.0 (Avg: 0.2...	☐
No Room For New Se...	Applies only to session in memory with Allow...	0	☐
Cache Discards	Number of session objects that have been for...	0	☐
External Read Time	Time (milliseconds) taken in reading the sess...	0	☐
External Read Size	Size of session data read from persistent stor...	0	☐
External Write Time	Time (milliseconds) taken in writing the sessi...	0	☐
External Write Size	Size of session data written to persistent store...	0	☐
Affinity Breaks	Number of http session affinities are broken, n...	0	☐
Time Since Last Activ...	The time difference in milliseconds of the prev...	10,789.762	☐
Invalidated Via Timeout	The number of sessions that are invlidated via...	1	☐
Activate Non-exist Ses...	Number of requests for a session that no long...	0	☐

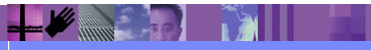
Tivoli Performance Viewer is the updated and re-branded Resource Analyzer that we used in version 4.0.

The first four items in the list were available in WebSphere 3.5.5; the remaining items are new in 5.0.

Troubleshooting Tips

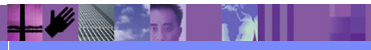
Tracing

- **Improved tracing and error messages**
- **Trace specification**
  - com.ibm.ws.webcontainer.httpsession.*=all=enabled
  - Also available as a trace group:
    IBM HttpSession=all=enabled



Unable to get at the Session

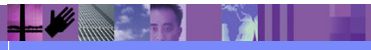
- **Is session cookie getting set on the browser when session is created ?**
  - Make sure cookies are enabled on the browser.
  - Check Cookie settings on the SessionManager.
    - Check Cookie domain specified on the SessionManager - should match servlet URI mapping.
    - Check whether the URL is hierarchically below the Cookie path specified on the Session Manager.
    - Look for getIHttpsession(...) which is start of session request.
    - Look for releaseSesson(..) which is end of servlet request.
    - If problem still persists, enable Web Container trace also.



The cookie domain should match servlet URI mapping. For example, if the cookie domain is set as myCompany.com, the servlet should be accessed with that domain name. (http://mySystem.myCompany.com/myapp/servlet/sessionservlet)

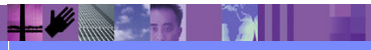
Unable to get at the Session

- **Check the cookie flow between browser and WebSphere**
  - On the browser, enable "cookie warning" - should prompt you next time you hit the servlet
  - On WebSphere, enable SessionManager trace
    - Access the session servlet from the browser
      - Browser will prompt for the cookie, note down the session ID
      - Reload the servlet, note down the cookie if new cookie is sent
  - In the session trace, look for the session id and trace the request by thread
    - Make sure session is id is flowing
  - If problem persists, trace the Web Container too



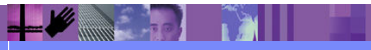
Database Session Persistence

- **If you use database persistence, check if datasource is specified correctly**
  - Specify JNDI name of the datasource correctly
  - Datasource specified should be non-JTA, that is, not XA enabled
  - Specify correct user ID and password for accessing the database
- **On UNIX, with DB2 on local machine, configure database with loopback in TCP/IP communication mode.**



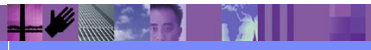
Database Session Persistence

- **Check the stderr and stdout for appropriate database error messages**
- **Some versions of DB2 (defect 108511 workaround)**
  - Specify cursorhold=1 property on the datasource used for SessionManager. Otherwise threads will hang in the database and eventually system hangs
- **For row size other than 4 KB**
  - Make sure specified row size matches the DB2 page size
  - Make sure tablespace name is specified correctly



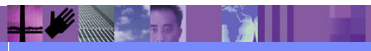
Session Persistence not working

- **WebSphere Internal Messaging**
  - Check log files for errors starting internal messaging
  - Trace com.ibm.ws.runtime.component.SystemMessageServerImpl=all=enabled
  - Group name WebSphere Internal Replicaiton=all=enabled
    - Yes, it is spelled that way in the console.
  - Trace com.ibm.ws.webcontainer.httpsession.*=all=enabled
- **Database persistence**
  - check log files for errors w/ connection



FAQ - Multiple browser instances

- **Session Sharing**
  - When a simple session counter servlet is accessed from multiple instances of browser on same machine (for example two instances of IE), session is getting shared between them
    - Depends on how second instance of browser is started
      - If it is started using CTRL+N, then new instance is created as sub process of original instance
        - Both share same in-memory cookies
  - The Cookie Max Age determines the behavior
    - If Max Age is set to some positive value, all the browser instances share the cookies
    - Cookie is persisted to file for the specified max age time
    - Doesn't matter how browser instances start in this case



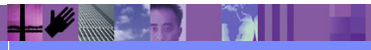
Multiple instances of browsers started as sub process share the cookies, that's why session gets shared between. It is the browser behavior that is defining this.

JSPs and HTTP Sessions

- **When hitting a JSP, the Web Container creates a session**
- **JSPs by default request.getSession(true),**
  - Session is created if none exists for the client
- **To prevent a JSP from creating a new session, set the session scope to false in the JSP**

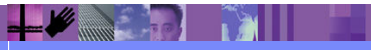
Problem Determination - Maxinactive Interval

- **Problem: Session is not getting invalidated after specified MaxInactive interval**
  - SessionManager invalidation process runs periodically to invalidate inactive sessions
  - The frequency is tied to the MaxInactive interval specified
    - For the default of 30 min , the process runs about every 5 minutes (310 sec)
    - Sessions get invalidated within maxinactiveInterval*60+x seconds = about 35 minutes
      - Custom value of the frequency can be specified using system property ReaperInterval=value



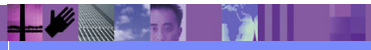
Problem Determination - Session Affinity

- **Make sure that the plug-in configuration has been regenerated after creating new clones or installing new applications**
- **For remote Web Server configurations**
  - After regenerating the plug-in, copy the plugin-cfg.xml from WebSphere machine to WebServer host
  - Manually edit the paths for stashfile and keyring on the secured HTTP transports



Summary

- **WebSphere Application Server 5.0 provides variety of options for HTTP Session support**
- **Extensions beyond the J2EE specifications**
- **New option for Session Failover - Memory to Memory session replication**
  - Removes Relational DB requirements
  - Performance advantages
- **Troubleshooting aids and tools**



Trademarks and Disclaimers