**What is current role & responsibilities?**

Onsite Coordinator, Tech Lead.

* Providing project estimates.
* Development using the applicable technologies and following **Agile methodology**.
* Preparing Unit Test Plans and Test Automation using **JUnit** test framework.
* Configuration Management using configuration management with SVN and using SmartSVN client tool.
* Implementing **DevOps** related automations using tools like JUnit, Jenkins etc.
* **Doing POCs** (Proof of concepts) for different proposals using client recommended Open-Source technologies.
* Successful initiation, planning, execution, control and completion of the project by guiding team members on technical aspects.
* Conducting reviews of code, and technical documents/artifacts.
* Lead the customer interface for the project on an everyday basis, proactively addressing any issues before they are escalated.
* **Leading** project development and maintenance activities.

**Achievements:**

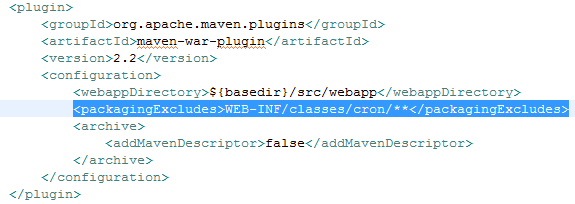
* **Automations for PDTS -** Build process, Test Automation, Continuous Integration and Automated Deployment using Jenkins, LoginAs Function, Log Rotation tool.

**What do you do on day-to-day activity?**

1. **Development:**
   * Coding in core java, using JDBC API, struts1 - struts2 based code update, JSPs update, Servlet updates etc.
   * Test cases writing.
   * Build code using Maven, Deploy code to WebLogic Server using Jenkins, DevOps.
   * Spring Boot, Hibernate, MicroServices based code update.
   * AngularJS for UI code. HTML5, CSS & Bootstrap.
   * Writing SQL queries, updating PL-SQL procedures, packages & functions etc.
   * Shell scripts - cron job script updates on App Script Server.
   * Delivery in Agile sprints.
2. **Unit Test:**
   * Running test cases with Maven, Jenkins.
   * Increase test coverage for existing code - struts 1 (MockStrutsTestCase API); struts 2 (XworkTestCase API).
   * Manual testing and test results generation.
3. **Others:**
   * Estimation
   * CFM using SVN (with tool SmartSVN).
   * Doing POCs (Proof of concepts) for different proposals using client recommended Open-Source technologies.
   * Reporting activities - report progress of team on different assignments (POCs and Project work).

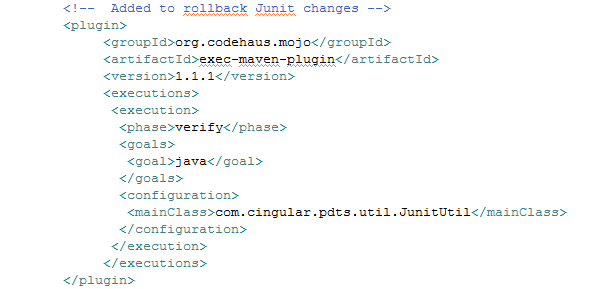
**Maven WAR plug-in for excluding classes:**

To exclude some set of classes from the final WAR file. Do it like this:



**Run a class as a part of Maven Build process:**

Do it like this:



**PDTS Load Balancing request details:**

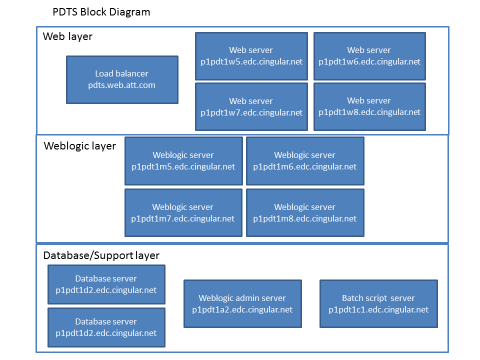
* Provide the load balancer requirements as a part of **LB Decision Process Questionnaire template**.
  + Any external access requirements by a third party? (Internet, FriendsNet, Business Partner Link, Remote Call Center, etc.)
  + Are global server load balancing services required? (for e.g. hot-hot, hot-warm, hot-cold, etc.)
  + App info
  + Content rules:
    - Web server info like type (Ex. Apache)
    - URL
    - Load Distribution Algorithm [we had used **Round Robin** (AKA Cyclic)]
    - Sticky Client Method? AKA "Persistence" [we had used **Cookie** (HTTP)]
      * Other options were: IP Address, SSL (HTTPS).

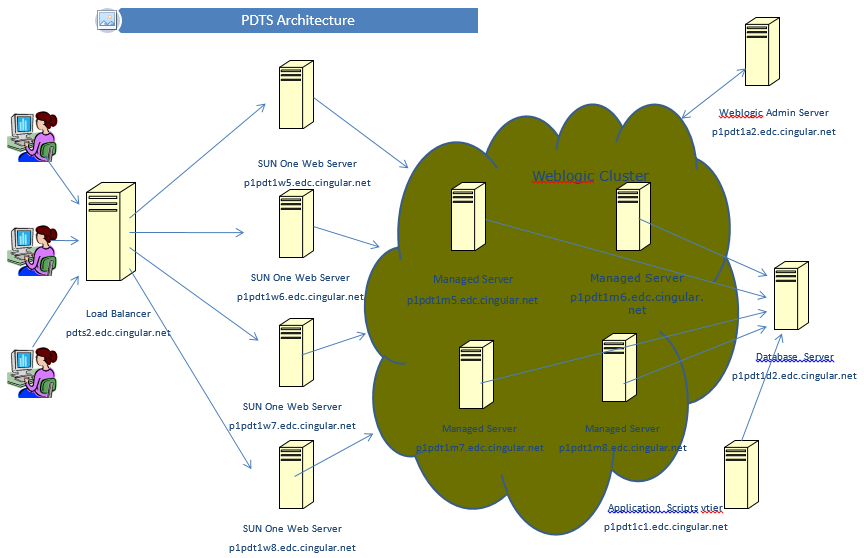
Once connected, if a client should remain associated with the same server for a period of time, what will be the basis for that persistence? Available strategies for persistence will vary by data center and load balancer type.

* + - Keep-Alive Path (If keep alive type is one of the HTTP types, enter the path the LB can pull to make sure the service is up) ex. /pdts/index.jsp
    - Server details (IPs & Hostnames)
    - Capacity metrics
      * Estimated connections / sec during busy-hour Ex. 9
      * Estimated simultaneous active sessions Ex. 2000
      * Estimated bandwidth requirements per connection Ex. 50000 (In bytes / connection. For an HTTP app, the average page/element size. Again, content-rule specific.)
      * Forecasted growth of traffic metrics over next 6 months Ex.10%

**PDTS Overview:**

Personnel Development Tracking System (PDTS) was designed to standardize Discussion and Discipline coaching/interactions. It also provides a better experience for managers and employees by improving back office communications. Over 10k AT&T Mobility managers use PDTS (with a standardized and HR/Legal approved way) to capture coaching discussions and performance topics. Identifying, sharing, and resolving coaching issues across groups using a large notification and approval structure is now a critical process within many of the Business Units using PDTS. With extensive reporting options, PDTS also serves as an effective tool to insulate AT&T resources against legal claims made in regards to past performance and specific individual/supervisor responsibility.

****



**SRS Overview:**

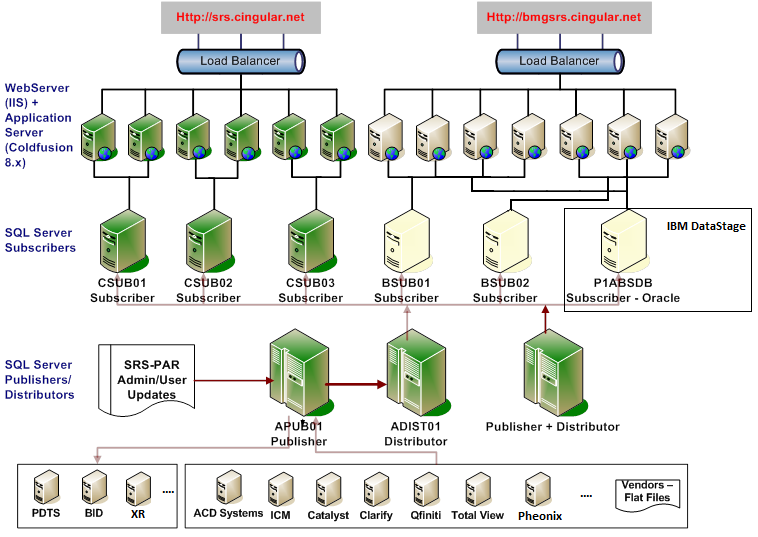
SRS-PAR is a **customer service reporting tool**. This application is developed in response to the Personal Accountability Reporting initiative, and is intended to provide individual customer service agents daily feedback on their performance meeting standard call center metrics.

It helps call center personnel to monitor call handling performance and quality statistics. It also provides call center managers, directors, and executive managers with a range of team-level to enterprise-level call handling metrics reports.

It generates various reports based on the data gathered from different ACD and ICM platforms. Some of the reports include Application stats, Performance stats, Login events, Agent audits (repeat call stats, CTN stats, Clarify interactions) and PAR reports for reps and management.

SRS-PAR Hierarchy is determined

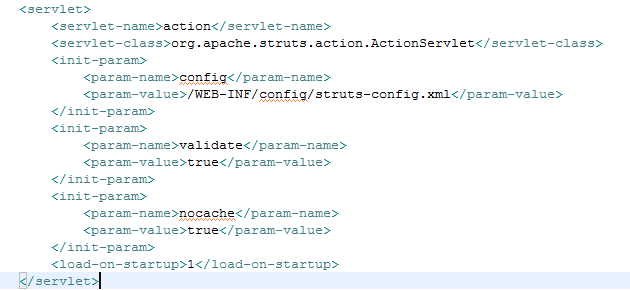
* Payroll system, eLink for internal/company owned call centers
* Via file feeds for vendor call centers.



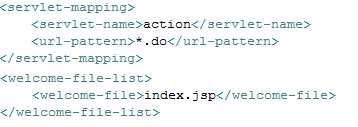
**Struts 1.x Flow:**

1. When a client request is given, a web container will receive request. Web container loads web.xml and verifies whether the url-patterns are verified or not, if matches web-container transfers request to ActionServlet.

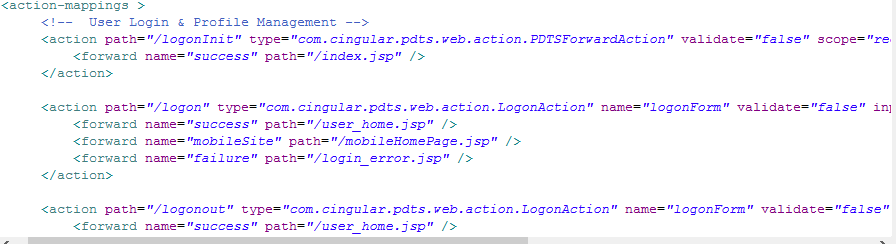
WEB-INF/web.xml has servlet configured. It uses struts-config.xml as a param to servlet “org.apache.struts.action.ActionServlet”.



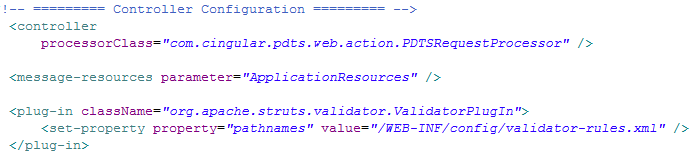
1. This servlet is mapped to url pattern \*.do. welcome-file-list has welcome-file configured as index.jsp.



1. Action mappings are mentioned in struts-config file.

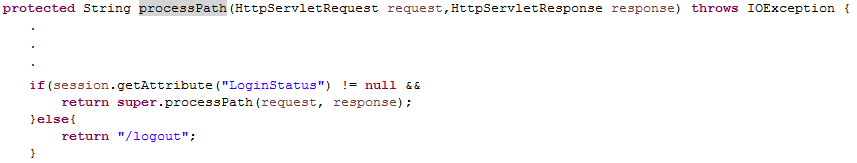


1. Controller configurations are also part of this file.



PDTSRequestProcessor extends org.apache.struts.action.RequestProcessor (RequestProcessor contains the processing logic that the Struts controller servlet performs as it receives each servlet request from the container. You can customize the request processing behaviour by subclassing this class and overriding the method(s) whose behaviour you are interested in changing.).

1. PDTS has overridden processPath method which identifies and returns the path component (from the request URI) that we will use to select an ActionMapping to dispatch with.



1. struts-config file also has the form-beans (usually getter setters with validate and other methods extended from org.apache.struts.action.ActionForm) configured.

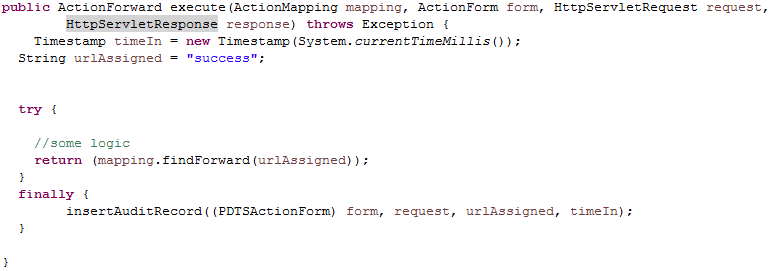
An **ActionForm** is a JavaBean optionally associated with one or more ActionMappings. Such a bean will have had its properties initialized from the corresponding request parameters before the execute method is called.

When the properties of this bean have been populated, but before the **execute** method of the action is called, this bean's **validate** method will be called, which gives the bean a chance to verify that the properties submitted by the user are correct and valid.

1. struts-config file action mapping type should be an Action class (should extend org.apache.struts.action.Action if defined as any custom class).

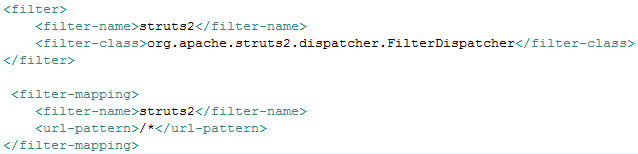
An **Action** is an adapter between the contents of an incoming HTTP request and the corresponding business logic that should be executed to process this request. The controller (ActionServlet) will select an appropriate Action for each request, create an instance (if necessary), and call the **execute** method.

**execute** method: Process the specified HTTP request, and create the corresponding HTTP response (or forward to another web component that will create it), with provision for handling exceptions thrown by the business logic. Return an **ActionForward** instance describing where and how control should be forwarded, or null if the response has already been completed.



**Struts 2.x flow:**

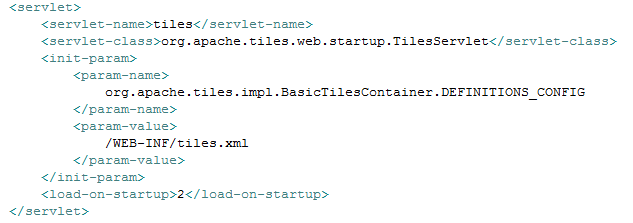
In WEB-INF/web.xml, the struts2 filter for handling requests is mentioned. It handles both the preparation and execution phases of the Struts dispatching process.



The **FilterDispatcher** (org.apache.struts2.dispatcher.FilterDispatcher) is used in the early Struts2 development, and **it’s deprecated since Struts 2.1.3**.

If you are using Struts version >= 2.1.3, it’s always recommended to upgrade the new filter class –**StrutsPrepareAndExecuteFilter** (org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter).

There is a listener configured for Struts Tiles framework org.apache.struts2.tiles.StrutsTilesListener & respective TilesServlet is also configured with param file tiles.xml.



Tag Libraries & MIME types are also configured in web.xml.



* When a **client request** is given, a **web container** will receive request. Web container **loads web.xml** and verifies whether the url-patterns are verified or not, if matches web-container transfer the request to **FilterDispatcher**.
* FilterDispatcher hand overs the request to **ActionProxy**. ActionProxy contacts **ConfigurationManager** class (which loads struts.xml), to know the suitable Action for the request.
* ActionProxy delegates the request along with its information to **ActionInvocation**.
* ActionInvocation executes the interceptors (the before clause) in advance of invoking the Action itself.
* Once the **Action** returns, the ActionInvocation is responsible for looking up the proper result associated with the Action result code mapped in struts.xml. The result is then executed (It may use the Struts Tags provided by the framework).
* Interceptors are executed again (in reverse order, calling the after clause). The result is routed back to the **FilterDispatcher**.
* And Finally, FilterDispatcher uses RequestDispatcher’s forwarding mechanism and forward a view as a response back to the client.

**Web Server vs App Server:**

**Server** is a device or a computer program that accepts and responds to the request made by other program, known as **client**.

**Two types of servers:**

1. **Web Server:** Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

Ex. Apache Tomcat, Resin Jetty and Sun Java System Web Server.

1. **Application Server:** Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc. It provides the middleware services for state maintenance and security, along with persistence and data access. Designed to install, operate and host associated services and applications for the IT services, end users and organizations.

Ex. JBoss, IBM WebSphere, Oracle/BEA WebLogic, TomEE, Glassfish

A **Web server** serves content to the web using **http** protocol whereas **Application Server** hosts and **exposes** **business logic and processes**. Web server exposes everything through the http protocol, while the application server is not restricted to it. In many scenarios, web server is being used to create the front-end of the application server, that is, it exposes a set of web pages that allow the user to interact with the business rules found into the application server.

If you are deploying your enterprise applications (means you have JPA, EJB or some technology which is part of Java EE) to a server which comply with JavaEE, then the lib need not contain the API implementation jars.

Apache Tomcat, Jetty and Sun Java System Web Server are only Java Web (Servlet) containers, meaning they can only execute Servlets/JSP - they don't provide the full JEE API stack.

As such, they can only deploy .war files, not .ear (that would also include .jar modules with EJBs), and do not support out of the box some JEE APIs like JSF or CDI or other functionalities/APIs. It is important to note that, since JEE6, .war files may contain EJBs.

While **a Web server** may not itself support transactions or database connection pooling, it may employ various strategies for fault tolerance and scalability such as load balancing, caching, and clustering - features oftentimes erroneously assigned as features reserved only for application servers.

Every Java EE server has a **Web Container + EJB Container**. Ex. JBoss Application Server uses JbossWeb (an Apache Tomcat fork) as its web container + JBoss EJB container. The others (IBM WebSphere, Oracle/BEA WebLogic, TomEE, Glassfish) also have their web container + EJB container.

Moreover, the **application server** manages its own resources. Such gate-keeping duties include security, transaction processing, resource pooling, and messaging. Like a Web server, an application server may also employ various scalability and fault-tolerance techniques.

What is very common is placing a web server (such as Apache HTTPD or IIS) before the Java Web Container. There are two main motivations for this:

1. Make the HTTPD serve the static content (such as images) and forward the rest to the Java web container. This is done because web servers are usually **better optimized at the task of delivering static content**.
2. **Security**: Expose only the HTTPD in the **DMZ (Demilitarized Zone).** One can set up an Apache HTTPD at the DMZ, and make it simply forward everything to Web Containers (Tomcats, etc.) and JavaEE Servers (JBosses, etc.).

**NOTE:** DMZ - a physical or logical subnetwork that contains and exposes an organization's external-facing services to an untrusted network such as the Internet. The purpose of a DMZ is to add an additional layer of security to an organization's local area network (LAN); an external network node can access only what is exposed in the DMZ, while the rest of the organization's network is firewalled.

**war - web archive.** It is used to deploy web applications according to the servlet standard. It is a **jar file** **containing** a special directory called **WEB-INF** and several files and directories inside it (web.xml, lib, classes) as well as all the HTML, JSP, images, CSS, JavaScript and other resources of the web application.

**ear - enterprise archive**. It is used to deploy enterprise application containing EJBs, web applications, and 3rd party libraries. It is also a **jar file**, it has a special directory called **APP-INF** that contains the application.xml file, and it contains jar and war files.

**jar (java archive)** - A JAR file encapsulates one or more Java classes, a manifest, and a descriptor. JAR files are the lowest level of archive. JAR files are used in J2EE for packaging EJBs and client-side Java Applications.

**tar (tape archives)** - Format used is file written in serial units of fileName, fileSize, fileData - **no compression**. Can be huge.

**App Server hanging - how to handle it?**

If an application server process spontaneously closes, or web modules running on the application server stop responding to new requests:

1. Look at the app server documentation or use performance/debugging tools on how to handle server hanging issues; or on how to detect memory leak problems.

2. Analyze **Vendor's performance/debugging tools** (Ex. Tivoli performance viewer for IBM WebSphere) to determine if any of the application server resources, such as the Java heap, or database connections, have reached their maximum capacity. If there is a resource problem, review the application code for a possible cause:

* If **database connections** are being assigned to a request but are not being released when the requests finish processing, **ensure** that the application code performs a **close()** on any opened Connection object within a finally{} block.
* If there is a **steady increase in servlet engine threads** in use, review application synchronized code blocks for possible deadlock conditions.
* If there is a **steady increase in a JVM heap size**, review application code for memory leak opportunities, such as static (class-level) collections, that can cause objects to never get garbage-collected.

You can **increase the log level to capture more info** as required.

3. Determine if the application **server is running out of memory**. If you determine that the application server is running out of memory, one of the following situations might be occurring:

* There is a memory leak in application code that you must address.
* The JVM is using the maximum heap size that it is allowed to use. In this situation, you should increase the maximum heap size setting for application server if you have enough storage available to do so.
* The server runtime is experiencing a problem. If you determine that there is a problem with the server runtime, make sure that you have applied all of the service updates for the product.

4. **Browse the thread dump for clues:** The JVM creates a thread dump whenever an application server process spontaneously closes. You can also force an application to create a thread dump. After a dump is created, you can check the dump for clues as to why new requests are not being processed.

* Recent JDKs ship with **VisualVM**. You can attach it to a running JVM instance to monitor thread and memory usage.
* Can also use **JStack** utility to capture thread dump.
* If you are on Linux you could use **kill -3** to get a thread dump and look for deadlocks using Thread Dump Analyzer.

**How to handle OutOfMemory exception?**

One **common indication of a memory leak** is the java.lang.OutOfMemoryError exception.

It is thrown by the Java Virtual Machine (JVM) when it cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

The **OutOfMemoryError** extends the **VirtualMachineError** class, which indicates that the JVM is broken, or it has run out of resources and cannot operate. Furthermore, the the VirtualMachineError extends the **Error** class, which is used to indicate those serious problems that an application should not catch.

The OutOfMemoryError exists since the 1.0 version of Java.

There are 3 possible causes for an OutOfMemoryError.

1. JVM has a real memory leak, caused by a bug in the internal implementation of the JVM heap management. This is **extremely unlikely**. All JVMs are tested very strenuously for this.
2. You simply haven't got enough memory available for your application. There are **two** **possible solutions** to this situation, either increase the available JVM heap size, or decrease the amount of memory your application needs. Increasing the available JVM heap size is simply done with the -Xmx parameter to the JVM. If you still have the memory problem, then you need to reduce the amount of memory being used by your application. Reducing application memory may be simple, you may just be allowing some collections/caches/tables/objects to be too big, for example using many large buffers. Or it can be complex, requiring you to reimplement some classes, or even redesign the application or tune object sizes.

With some JVMs (e.g. the Sun JVMs), there is also a "**Perm**" space which holds **JVM structures** and **class** objects. If you are using a very large number of classes, it is possible to run out of space in "Perm" space, and you may need to increase the size of that space, e.g. with the Sun JVM using the -XX:PermSize and -XX:MaxPermSize options.

1. The most common, **unintentional object retention**. You are hanging on to objects without realizing it and this is causing your heap to grow and grow until you have nothing space. Find the root object holding on to the unintentionally retained objects, and change it to release those objects.

A summary of the procedure of finding such root object is

- Wait until the application has reached the steady state, where you would expect most new objects are temporary objects that can be garbage collected; typically this is after all the application initializations have finished.

- Force a garbage collection, and take an object snapshot of the heap.

- Do whatever work it is that is causing unintentionally retained objects.

- Force another garbage collection and then take a second object snapshot of the heap.

- Compare the two snapshots to see which objects have increased in number from the first snapshot to the next. Because you forced garbage collections before the snapshots, the objects left should all be objects referenced by the application, and comparing the two snapshots should identify exactly those newly created objects that are being retained by the application.

- Using your knowledge of the application, determine from the snapshot comparison which of the objects are being unintentionally retained.

- Track back-references to find which objects are referencing the unintentionally retained objects, until you reach the root object that is causing the problem.

Each Java application can use only a limited amount of memory. This amount is specified at the startup of the JVM, by the following parameters:

* -Xms <size>: specifies the initial memory allocation pool (heap size).
* -Xmx <size>: specifies the maximum memory allocation pool for a Java Virtual Machine (JVM).

Ex.:

Random random = new Random();

Map sampleMap = new HashMap();

// Loop forever...

while(true) {

// Create and store a random pair.

int randomValue = random.nextInt();

sampleMap.put(randomValue, String.valueOf(randomValue));

}

Application only stores values to the HashMap and never retrieves values from it. Thus, memory space is occupied for no reason and the garbage collector cannot reclaim that space, because our application holds a reference to that HashMap.

There are many different causes that can throw an OutOfMemoryError in Java:

* java.lang.OutOfMemoryError: Java heap space.
* java.lang.OutOfMemoryError: PermGen space.

In Java, the permanent generation is the memory space where class and method objects are stored.

* java.lang.OutOfMemoryError: Requested array size exceeds VM limit.

This error indicates that a Java application attempts to allocate an array, whose size is larger than the heap size.

Ex.:

int[] matrix = new int[Integer.MAX\_VALUE];

for(int i = 0; i < matrix.length; ++i) {

matrix[i] = i+1;

}

**To summarize -** how to deal with the OutOfMemoryError:

1. The most obvious solution to this error is to increase the available memory size for the Java Virtual Machine. If your application requires more memory then, you shall grant it to your application.
2. Reduce the amount of memory being used by your application - using smaller collections/caches/tables/objects/..., i.e. by tuning object sizes, by redesign, and by reimplementation.
3. Verify that your application does not store unnecessary information (unintentional object retention). Store and maintain only those pieces of information required for the proper execution of your Java application.
4. You can use the available memory analyzer tools, in order to carefully observe the portions of memory occupied by your application. Examples of such tools are the Eclipse Memory Analyzer and Java Heap Analysis Tool (JHAT).

**Why didn't you use mockito for TestCase writing in PDTS?:**

**JUnit** is the Java library used to write tests (offers support for running tests and different extra helpers - like setup and teardown methods, test sets etc.). **Mockito** is a library that enables writing tests using the mocking approach.

We have **Struts1** & **Struts2** framework based code base. And we found, we could use APIs like **XWorkTestCase** (class com.opensymphony.xwork2.XWorkTestCase extends junit.framework.TestCase) for Struts 2 & **MockStrutsTestCase** (class servletunit.struts.MockStrutsTestCase extends junit.framework.TestCase) for Struts 1. Both of them support mock servlet objects that we can use while writing test cases. Hence, we didn't use for Mockito library.

**How have you written test case with:**

**Struts 1:**

1. @Before annotated **setUp** method is overridden to load properties (like DB conn params), setup data source (OracleConnectionPoolDataSource) in JNDI context, mock request & session attributes (like set LoginStatus to active, userId, GL Cookie etc.).
2. **setContextDirectory**(new File(System.getProperty("user.dir") + "/src/webapp")); //Set up context dir.
3. **setConfigFile**("./WEB-INF/config/struts-config.xml"); //Set up config files relative to above context dir.
4. **setRequestPathInfo**(JunitConstant.MBL\_EDIT\_DISCUSSION\_ACTION.replace(".do", ""));
5. **request.setRequestURL**(JunitConstant.DEFAULT\_REQ\_URL + JunitConstant.MBL\_EDIT\_DISCUSSION\_ACTION);
6. **request.setServletPath**(JunitConstant.MBL\_EDIT\_DISCUSSION\_ACTION);
7. Instantiate & provide any other test data to the request or session params or to any other user bean objects used in the method being tested.
8. Define test with **@Test** annotation.
9. Perform test with **actionPerform()** method.
10. Verify using: verifyForward(forwardName); verifyNoActionErrors(); OR verifyActionErrors(new String[] {actionEMsgKey}); verifyForwardPath(forwardPath);
11. Revert the changes for the next test case run from scratch.

**Struts 2:**

1. @Before annotated **setUp** Method is overriden to load properties (like DB conn params), setup data source (OracleConnectionPoolDataSource) in JNDI context, init Servelet objects & dispatcher.

servletContext = new MockServletContext(new DefaultResourceLoader());

response = new MockHttpServletResponse();

request = new MockHttpServletRequest();

pageContext = new MockPageContext(servletContext, request, response);

1. Get Action Proxy(com.opensymphony.xwork2.ActionProxy) for the action to be tested (ex. /editDiscussion.action).
2. Get the Action class instance from the proxy object & use it to ActionContext (this is required to set ServletActionContext with request session).
3. Instantiate & provide any other test data to the request or session params or to any other user bean objects used in the method being tested.
4. Define test with @Test annotation.
5. Perform test by calling directly the method with Action class instance. It will return result as String.
6. Verify result string with the expected one using assertEquals method.
7. Revert the changes for the next test case run from scratch.

**Which Java Features you have used in your Projects?**

* **Core Java coding** - interfaces, beans, utils, dao layers; String functions, StringBuffer/Builder, POJOs, Date functions, Exception Handling, Static Functions like isNull, isNumeric etc., Generics, Collection APIs (like HashMap, ArrayList etc.)
* **File I/O** - for properties or config, writing contents to excel or pdf other files.
* **Log4J Logging**
* **Apache APIs** - for String functions (StringUtils), Validating email address (EmailValidator),POI API (XSSFWorkbook) for excel file generation.
* **JDBC APIs** - Read Config from DB, Run procs through callable statements & get results, run report queries & process them etc.
* **javax.crypto.Cipher** API for password encryption
* **Servlets -** for realizing app functions (ex. generating PDF file; loading meta data)
* **JSPs, JSPFs etc.** (& related tag libraries, display tag lib & decorator classes) for view logic for Struts 1 & Struts 2 framework
* **JNDI lookup for App Server resources** - data sources, jms queues etc.
* **JMS for loading meta data** -> Uses Topic destination object for JMS communication.
  + Lookup TopicConnectionFactory & use it to lookup further the javax.jms.Topic.
  + Send Message every time meta data is updated i.e. **publish message** [**TopicConnection > TopicSession > TopicPublisher** (using createTextMessage method of TopicSession and publish method of TopicPublisher)]
  + Use TopicConnectionFactory to get **TopicConnection** and use it to set up **Message Listener** [TopicConnection > TopicSession > TopicSubscriber (setMessageListener method passing class name which is implementing javax.jms.MessageListener)].

**What part of java you like most**

**Spring Boot based app development** has been the best thing that I had done till date.

I like to work on value-adds that can be useful to others. Not specific to any technologies, I had done few VAs related to POCs, Build process, Test Automation, Continuous Integration and Automated Deployment using Jenkins, LoginAs Function, Log Rotation tool, and Email framework.

**Technologies used were** - Spring Boot, AngularJS, Unix Shell Scripts, Jenkins, Maven, Struts 2 framework, Java/J2EE.

**JSPF means**

**JSP Fragments** can be compared to server side includes. These fragments are not compiled on their own, however there are compiled alongside the page in which its included. JSPF files can be thought of as a **first step in refactoring large JSP pages**.

**Any perf tuning work done**

In the current project - none as the applications were not expecting any performance issues except for the query tuning at some places. That were taken care by the DBA team. I was just involved to take care of a memory leak when DBA highlighted some long open app user sessions.

There was a performance test team in my earlier assignment. They used to stress test the app for any performance issues. And we were handling them if any. As a part of that only, I migrated one soap based web app to axis2 web service based web app.

Otherwise, I was just following **some of the standards** that were set to avoid performance issues i.e. simple Guidelines:

* Set a goal: like maximum concurrent users the application will support for a given limit on response times; the response times should not increase more than 10 percent during the peak hour of user load.
* Identify problem areas: A little investigation into problems might reveal the specific component that causes poor performance. For example, if the CPU usage on an application server is high, you will want to focus on tuning the application server first.
* Follow a methodical and focused path: try to make changes that are expected to have the biggest impact on performance. Your time is better spent tuning a method that takes 10 seconds but gets called 100 times than tuning a method that takes one minute but gets called only once.

Use **Stress tools** - Many different stress tools are available in the market today.

And **Performance monitors** - Using a monitoring tool, you collect data for various system performance indicators for all the appropriate nodes in your network topology. Many stress tools also provide monitoring tools.

**J2EE Environment tuning:**

A J2EE application environment usually consists of an application server, Web server, and a backend database.

1. **Web/App Server:**
   * Most important setting for your Windows Apache HTTP Server is the option for number of threads. This value should be high enough to handle the maximum **number of concurrent users**, but not so high that it starts adding its own **overhead of too many context switches.** The optimum value can be determined by monitoring the number of threads in use during peak hours.

The JVM performs best with the fewest busy threads. A good starting point for **thread count can be found** with the following equations.

(Thread count) = Number of Transactions / Time (in seconds)

OR

(Thread count)=Throughput (transactions/sec)

* **Other tuning options:** Do not load unnecessary modules; Try to minimize logging as much as possible; Use the latest Java version.
* **Heap Size:** Keep its value **optimum**. If you still run out of memory, look into your application design to reduce memory usage, identify any memory leaks, or try various garbage collector options in the JVM.

1. **Database Tuning:**

* Run DB Server on dedicated instead of shared machine.
* Keep your application database and your temporary database on different hard disks.
* Consider taking local backups and moving them to a different machine. The backups should complete much faster.
* Normalize your database to the **third normal form**. This is usually the best compromise, as the fourth and fifth forms of normalization can result in performance degradation.

1. **Application Tuning:**

* Explicitly assigning a null value to variables that are no longer needed helps the garbage collector to easily identify the parts of memory that can be safely reclaimed.
* Do not synchronize code blocks or methods unless synchronization is required. Keep synchronized blocks or methods as short as possible to avoid scalability bottlenecks.
* Only declare methods as final when absolutely necessary: Modern optimizing dynamic compilers can perform inlining and other inter-procedural optimizations, even if Java methods are not final.
* Declare constants as static final: The dynamic compiler can perform some constant folding optimizations easily, when the hint is provided.
* Avoid finalizers: Adding finalizers to your code makes the garbage collector more expensive and unpredictable. Finalizers may not always be executed, before the program exits. Releasing critical resources in finalize() methods may lead to unpredictable application behavior.
* Declare method arguments final if they are not modified in the method.
* Use StringBuffer rather than using + operator when you concatenate multiple strings.
* Use a logging mechanism (like log4j) that lets you **switch off logging** in the production environment to reduce logging overhead.
* Instead of creating and destroying resources every time you need them, use a resource pool for every resource (ex. JDBC Connection, Thread) that is costly to create.

A rule of thumb is to match the number of database connections to the number of execute threads. Over-configuring this value could cause unnecessary overhead to the database, while under-configuring could tie up all execution threads waiting on database I/O.

* Try to minimize the objects you store in HttpSession. Extra objects in HttpSession not only lead to more memory usage, they also add additional overhead for serialization/deserialization in case of persistent sessions.
* Where ever possible, use RequestDispatcher.forward() instead of HttpServletResponse.sendRedirect(), as the latter involves a trip to the browser.
* Minimize the use of SingleThreadModel in servlets so that the servlet container does not have to create many instances of your servlet.
* Java stream objects perform better than reader/writer objects because they do not have to deal with string conversion to bytes. Use OutputStream in place of PrintWriter.
* Reduce the default session timeout either by changing your servlet container configuration or by calling HttpSession.setMaxInactiveInterval() in your code.
* Always add directive <%@ page session="false"%> to JSP pages where you do not need a session.
* Use the jspInit() method to cache static data, and release them in the jspDestroy() method.
* Excessive use of custom tags also may result in poor performance. Keep performance in mind while designing your custom tag library.
* In Java, serialization and deserialization of objects is a CPU-intensive procedure and is likely to slow down your application. Use the **transient** keyword to reduce the amount of data serialized. Customized readObject() and writeObject() methods may be beneficial, in some cases.
* Serialization uses reflection extensively, and this also makes it slow.
* The **Externalizable** interface is provided to solve Serialization's performance problems. Make classes Externalizable on a case-by-case basis to improve performance.
* Start the consumer before you start the producer so that the initial messages do not need to queue.
* Close resources (e.g. connections, session objects, producers, consumers) when finished with them.
* Choose non-durable (NON\_PERSISTENT) messages wherever appropriate to avoid the persistency overhead.
* Tune the Destination parameters: a smaller capacity increases message throughput; a higher redelivery delay and lower redelivery limit reduces the overhead.
* Set the TimeToLive value as low as feasible (default is for messages to never expire).
* Receive messages asynchronously with a MessageListener implementation.
* Use the print() method rather than the println() method.
* Use a ServletOutputStream rather than a PrintWriter to send binary data.
* Use the application server's caching facility.
* Session mechanisms from **fastest to slowest** are: HttpSession, Hidden fields, Cookies, URL rewriting, the persistency mechanism.
* Reduce the number of remote calls made by an application to improve performance.
* Cache remote objects locally where possible, rather than repeatedly fetching them.

1. **SQL Query optimization**

* Keep the transactions as short as possible. The longer a transaction is open, the longer it holds the locks on the resources acquired, and every other transaction must wait.
* Do not use DISTINCT clauses unnecessarily (i.e. when you know the rows will be unique).
* When possible, avoid using SELECT \*.
* Consider adding indexes to those columns causing full-table scans for your queries.
* Avoid using too many string functions (like SUBSTRING, LOWER, UPPER, and LIKE) or operators in your queries.
* Group all related search queries in a single JDBC call. The basic idea of performance tuning in DAO layer is to reduce the number of JDBC calls.
* Whenever your select query involves more than 6 tables, try to denormalize the table. To improve the performance, you need to reduce the number of tables.
* In case of batch operations, it is not a good idea to create SQL queries. SQL query will be compiled every time and then it is executed. You can use stored procedures. Stored procedures are compiled only once when they are executed for the first time.
* Avoid using views unless it is really needed. Querying from view takes more time than directly querying from the table.
* Try to use prepared statements as they are precompiled.

Performance tuning is **an ongoing process**. Simply put, good system performance depends on good design, good implementation, defined performance objectives and performance tuning. Mechanisms have to be implemented that provide performance metrics which can be compared against the performance objectives defined.

**Spring & Hibernate experience**

I am using **Spring Boot** for the POC work. Spring boot is new framework from the team at Pivotal, designed to simplify the bootstrapping and development of a new **Spring** application. The framework takes an opinionated approach to configuration, freeing developers from the need to define boilerplate configuration. My POC work is based on **MicroServices architecture**.

**Hibernate 3** I have used in the past project. And I've revised **Hibernate 5** before starting POC for Spring Boot related work.

**What did you do as Solution Architect?**

It was rather an **Application Architect** role.

As a Solution Architect, **I was responsible to** determine and propose solutions to the business problems based on the domain and app knowledge. For instance, for SRS reporting app, there was a new metrics data that Client was looking to make available on SRS UI. The high level steps involved were:

- Extracting the data from the source app.

- Making it available to SRS app to consume.

- Report on the consumed data.

My responsibility was to define processes, practices & rules of how these activities would be done by each of the interacting apps with more focus on SRS app more.

For instance above 3 activities were proposed to be done in the following way.

1. Data extraction - source app was free to choose way of. However the mode of file transfer was CSV files over SFTP (as the Company std was use of SFTP vs FTP).

2. Data consumption - SRS was already having a mechanism to process csv or flat files and store data to SRS DB. Asked them to either reuse existing mechanism or develop new as necessary.

3. Data Reporting - SRS had a custom report framework with which dynamic reports could be generated.

**How to count no of threads?**

1. There is a **static method on the Thread Class** that will return the number of active threads controlled by the JVM:

Thread.activeCount()

Returns the number of active threads in the current thread's “thread group”.

1. ManagementFactory.getThreadMXBean().getThreadCount() doesn't limit itself to thread groups as Thread.activeCount() does.
2. Additionally, external debuggers should list all active threads (and allow you to suspend any number of them) if you wish to monitor them in real-time.

Useful tool for debugging java programs, it gives the number of threads and other relevant info on them:

jconsole <process-id>

jconsole and jvisualvm are the mostly used tools.

1. This will give you the total number of threads in your VM :

int nbThreads = Thread.getAllStackTraces().keySet().size();

Now, if you want all threads currently executing, you can do that:

int nbRunning = 0;

for (Thread t : Thread.getAllStackTraces().keySet()) {

if (t.getState()==Thread.State.RUNNABLE) nbRunning++;

}

**How to take thread dump and why is it needed?**

**What is thread dump?**

A **thread dump** is a snapshot of the state of all threads that are part of the process. The state of each thread is presented with a so called **stack trace,** which shows the contents of a thread’s stack. Some of the threads belong to the Java application you are running, while others are JVM internal threads.

A thread dump reveals information about an application’s thread activity that can help you diagnose problems and better optimize application and JVM performance; **for example,** thread dumps automatically show the occurrence of a deadlock. Deadlocks bring some or all of an application to a complete halt.

**Why to take thread dump?**

**Main use -** for troubleshooting and diagnostics. Analyse dumps with other diagnostics tools, such as the JRockit Runtime Analyzer, which is part of Oracle JRockit Mission Control.

1. **Detecting Deadlocks -** The Oracle JRockit JVM automatically analyzes the thread dump information and detects whether there exists any circular (deadlocked) or blocked lock chains in it.

2. **Detecting Processing Bottlenecks** - If you discover (in a set of consecutive thread dumps) that one or more threads in your application is **temporarily stuck waiting for a lock** to be released, then you might have reason to look over the code of your Java application to see if the synchronization (serialization) is necessary or if the threads can be organized differently.

3. **Viewing The Runtime Profile of an Application** - By making several consecutive thread dumps, you might quickly get an overview of which parts of your Java application that are most heavily used.

When there is **an obstacle**, or when a Java based Web application is running much slower than expected, we need to use thread dumps.

**Thread contention** is a status in which one thread is waiting for a lock, held by another thread, to be lifted. Different threads frequently access **shared resources** on a web application. For example, to record a log, the thread trying to record the log must obtain a lock and access the shared resources.

**Deadlock** is a special type of thread contention, in which two or more threads are waiting for the other threads to complete their tasks in order to complete their own tasks.

Different issues can arise from thread contention. To analyze such issues, you need to use the thread dump. A thread dump will give you the information on the exact status of each thread.

**To summarize**, Thread dumps are essential diagnosis information used to analyze and troubleshoot performance related issues such as server hangs, deadlocks, slow running, idle or stuck applications, slow database interactions etc.

**How to take thread dump?**

1. **Using Jstack (since 1.5):** 
   * Command line tool that comes with JDK.
   * Get PID using jps -v
   * Run "jstack -l <pid> > <file-path>"
2. **Using JVisualVM (since 1.6 update 7):**

GUI Tool. Part of JDK - bin\jvisualvm.exe.

This tool also has the capability to capture thread dumps from the java processes that are running in remote host as well.

* + Launch the jvisualvm.
  + On the left panel, you will notice all the java applications that are running on your machine. Select one whose thread dump you want.
  + Go to the “Threads” tab. Click on the “Thread Dump” button.

1. **Using kill -3:**

In major enterprises for security reasons only JREs are installed in production machines. JDK tools to take thread dumps are not useful in such cases.

* + Obtain the process pid by using ps -ef command to check the pid of the currently running Java process.
  + Use the extracted pid as the parameter of "kill -3 <pid>" to obtain a thread dump.

When ‘kill -3’ option is used thread dump is sent to standard error stream. If you are running your application in tomcat, thread dump will be sent in to <TOMCAT\_HOME>/logs/catalina.out file.

1. **Java Mission Control (JMC) (since JDK 7 Update 40):**

Once you launch the tool, you will see all the Java processes that are running on your local or remote host.

On the left panel click on the “Flight Recorder” option that is listed below the Java process for which you want to take thread dumps. Here in the “Thread Dump” field, you can select the interval (like Every 60s) in which you want to capture thread dump. After the selection is complete start the Flight recorder. Once recording is complete, you will see the thread dumps in the “Threads” panel.

1. **Windows (Ctrl + Break):**

Will work only in Windows Operating system.

Select command line console window in which you have launched application.

Now on the console window issue the “Ctrl + Break” command.

The thread dump will be printed on the console window itself. To log to a particular file, launch app as below:

java -classpath . SampleThreadProgram > C:\workspace\threadDump.txt 2>&1

Thus when you issue “Ctrl + Break” thread dump will be sent to C:\workspace\threadDump.txt file.

1. **ThreadMXBean (since 1.5):**

Interface to use for programmatical thread dump generation.

1. **JCMD (Since 1.7):**

JCMD Tool has various capabilities such as identifying java process Ids, acquiring heap dumps, acquiring thread dumps, acquiring garbage collection statistics etc.

jcmd <pid> Thread.print > <file-path>

Ex. jcmd 37320 Thread.print > /opt/tmp/threadDump.txt

**Jstack & kill -3** are the best options because of Simpleness and Universalness (work with most of the versions).

**Thread Dumps using WebLogic Server**

1. From the **WLS Administration Console**, a thread dump can be created by navigating to Servers -> <Server Name> -> Monitoring -> Threads -> Dump Threads Stack OR Servers -> <Server Name> -> Monitoring -> Performance > Dump Threads Stack.

These methods could lead to truncated or incomplete thread dumps.

You can force garbage collecting through Servers -> <Server Name> -> Monitoring -> Performance > Garbage Collect.

1. Using WLST script with threadDump() command.

java weblogic.WLST connect("weblogic","weblogic","t3:<weblogicinstaceaddress>:<port>") threadDump()

Make sure you export weblogic.jar in your path.

1. Using **T3ServicesDef**

Can collect The Thread Dump remotely. This can be achieved using “weblogic.common.T3ServicesDef” and “weblogic.common.AdminServicesDef” weblogic APIs. To achieve it, we need to simply run “. ./setWLSEnv.sh” and run a program (search for it on web http://middlewaremagic.com/weblogic/?p=1379).

1. Using “weblogic.Admin” which is deprecated but still available:

Ex. java weblogic.Admin -url t3://AdminHostName:7001 -username weblogic -password weblogic THREAD\_DUMP

Thread Dumps will be generated in the Servers STDOUT file. Make sure you export weblogic.jar in your path.

**Vs Head Dump:**

A Heap Dump is a snapshot of the Java process heap memory at a given time. It's a useful tool to **troubleshoot memory leaks and other memory related issues**.

Heap dumps are to memory what thread dumps are to java threads.

**Experience with Multithreading?**

* Fine-tune the no of threads available for execution on WLS - to manage concurrent no of users. Normally, app server is well tested to handle multi-threading.
* Snippet of code marked with Synchronized block to take care of issues because multi-threading like use of Singleton DP while creating DB connection object.
* Otherwise Theoretical knowledge: You need that knowledge in understanding and debugging app issues related to concurrent execution of threads.

**What is Web 2.0?**

Web 2.0 refers to World Wide Web websites that emphasize user-generated content, usability (ease of use, even by non-experts), and interoperability (this means that a website can work well with other products, systems, and devices) for end users.

A Web 2.0 website may allow users to interact and collaborate with each other in a social media dialogue as creators of user-generated content in a virtual community, in contrast to the first generation of Web 1.0-era websites where people were limited to the passive viewing of content. Examples of Web 2.0 features include social networking sites and social media sites (e.g., Facebook), blogs, wikis, folksonomies ("tagging" keywords on websites and links), video sharing sites (e.g., YouTube), hosted services, Web applications ("apps"), collaborative consumption platforms, and mashup applications.

Web 1.0 is a retronym referring to the first stage of the World Wide Web's evolution. Here content creators were few in Web 1.0 with the vast majority of users simply acting as consumers of content. Personal web pages were common, consisting mainly of static pages hosted on ISP-run web servers, or on free web hosting services such as GeoCities.

With the advent of Web 2.0, it was more common for the average web user to have social networking profiles on sites such as Myspace and Facebook, as well as personal blogs on one of the new low-cost web hosting services or a dedicated blog host like Blogger or LiveJournal. The content for both was generated dynamically from stored content, allowing for readers to comment directly on pages in a way that was not previously common.

**Apache Axis:**

Apache Axis is an **implementation of the SOAP** ("Simple Object Access Protocol") submission to W3C.

Also called as a SOAP Engine.

SOAP is a lightweight protocol for **exchanging structured information in a decentralized, distributed environment**. It is an **XML based protocol** that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

Axis 1.1 has proven itself to be a reliable and stable base on which to implement Java Web services.

For Axis 1.2, we are focusing on our document/literal support to better address the WS-I Basic Profile 1.0 and JAX-RPC 1.1 specifications.

**FAQs related Axis:**

**What is a web service?**

It's common to have 2 programs on one computer talk to each other. One's the **server** (waiting and listening for requests), and the other's the **client** (contacting the server when it needs something done that the server does). The client and the server may talk to each other in a variety of ways: sockets, pipes, text files. The server and the client **don't have to be on the same machine**. You can have, say, Apache running on some web server machine, and Firefox running on your local machine. The client and the server talk to each other -- in the case of Firefox and Apache -- **using HTTP on top of TCP**.

**The idea** is, somewhere there's a **registry of web service descriptions**; structured descriptions telling exactly what services are provided by which web services (you might think of it as a list of objects and their public instance methods). When you want to make use of one of these services, you have a look at the web service description and then write a client that can talk to that service. There are, of course, tools out there that will turn the description into a bare-bones Java source code file (the skeleton of your client) for you to fill-in the implementation details.

The **grand plan at some point** was for big corps like Microsoft to provide a web service registry, and then you'd pay (on a per-use basis) for every web service you wanted to use (as opposed to having individual apps installed on your computer, that you may or may not have paid MS for). Companies like MS love the idea of ***pay-per-use***. Not sure - if this grand plan has panned out for them, but web services seem pretty useful regardless.

**How do web services work?**

When you create a web service -- one that you want other people to make use of -- you can use a WSDL description of it to register your web service with some web service registry. If you don't want anyone to be able to find out about your web service, then you don't have to register it -- it's optional. The "Universal Description, Discovery and Integration" protocol. A protocol for publishing web service descriptions. Somebody deploys a UDDI registry (it's just another web service), and then folks looking for some web service check with that registry to see what web services it knows about. Anybody can put up a web service registry.

SOAP is just the usual way that web services and their clients talk to each other -- they send XML text back and forth. You can watch it right on the wire (looking at the TCP packets with something like tcpmon) if you like.

**WSDL is a description language:** using XML, it describes exactly what your web service does. If you want a WSDL file for your web service, and you're using Apache Axis (mentioned below), you don't need to manually write the WSDL file -- you just have Axis create it for you from your Java code.

**What is Apache Axis? What's its relationship to Apache SOAP?**

As a client to a web service, encoding your requests to the web service, and decoding the responses you get back, to and from XML would be a pain (not to mention implementing the logic that deals with accepting requests and sending responses). The same goes if you're writing the web service yourself. Most folks use Apache Axis to do all that for them. You could write web service clients and servers without something like Axis, but it would be very tedious.

Axis was formerly known as "Apache SOAP". Currently, Axis is transitioning to version 2, but there's still a lot of Axis 1.2 users out there, and an Axis 2 stable release is not yet shipping.

Axis is an implementation of the SOAP protocol. It shields you from the details of dealing with SOAP and WSDL. You use Axis on the server side to write your web service (and deploy it as a Tomcat webapp), and you use Axis on the client side to make writing your client a snap.

Using Axis, you can choose from 4 different "styles" of web service: RPC, "Document", "Wrapped", and "Message". The Axis default is RPC -- and that's what we'll be using here. You'll often see the acronym JAX-RPC: Java over XML to make Remote Procedure Calls. SOAP is XML, and (by default) Axis sends SOAP messages to do RPC.

When using Axis to write your client, you don't need to directly deal with SOAP/XML/JAX-RPC. All of that is handled for you by Axis -- all you need to do is make the method calls on the web service object as if it were some local object. Same goes for your web service itself: just write the class and its instance methods, and let Axis take care of the rest. You install your web service just like any other Tomcat webapp. Your client that accesses the web service can just be a regular Java command-line program.

|  |  |
| --- | --- |
| **Apache SOAP** | **Axis** |
| really old | third generation |
| really slow | much faster, but not as fast as many |
| no WSDL support | WSDL support |
| proprietary API | JAX-RPC API |
| RPC/encoded only | RPC/encoded and Doc/literal |
| interoperability issues | very interoperable |
| extensibility issues | very extensible |
| low level API for headers | easy handler support for headers |

**Can other (non AXIS) clients access the AXIS web services? Eg. Perl's SOAP::Lite**

AXIS generates a WSDL for the web service and as long as there is a WSDL, client written in any language can consume the webservice.

**Installing, Configuring, and Deploying to AXIS**

You may install Tomcat and Axis according to the instructions that come with them.

The easy way (using a .jws file)

To install web service the simplest way possible, just rename the .java file to a .jws file and drop it in your webapps/axis directory, as described in the Axis docs.

The standard way (using a .wsdd file)

You can compile your .java file to a .class file, then put it into the webapps/axis/WEB-INF/classes directory on your server.

Remember this is a Fully Qualified Name - org.something.other becomes org/somthing/other.class.

Then use a .wsdd file (I used the one from samples/userguide/example3 as a starting point and modified it slightly) and deploy the web service using the AdminClient as described in the User's Guide.

**Verify that it's working**

To have Axis generate the WSDL for your service, you can click the link provided, which points to:

http://localhost:8080/axis/MyWebService.jws?wsdl

To actually use the web service, you'll need to run a web service client.

Axis comes with command line tools whose usage is pretty fundamental for a new user. These are wsdl2java, java2wsdl, tcpmon and adminclient. All of these tools need to be invoked via java, and need a fairly complex classpath to work correctly.

java <classpath> <fully qualified name of class> <parameters>

**How does Axis create my backend service objects? Can I control this?**

Axis supports a "scope" parameter on services, which can be set to "request" (make a new object to service each request - this is the default), "session" (associate a new object with each session), and "application" (all users share a singleton object). Refer to the Axis User's Guide under Scoped Services for details.

WARNING: If you set the scope to "session" or "application", it is possible that multiple threads of control may attempt to access your object's methods at the same time. It is your responsibility to ensure that your objects are thread-safe in these cases.

**Does Axis support sessions?**

**Yes.** We have a session abstraction which allows an extensible set of underlying implementations - take a look at the class org.apache.axis.session.Session for details. In particular, we currently support sessions based on HTTP cookies and also transport-independent sessions based on SOAP headers. It is up to some handler on the request chain to set up an appropriate Session implementation and attach it to the MessageContext with MessageContext.setSession() so that anyone who wants to use session semantics can get at it.

**How do I get access to the HttpServletRequest that called my service?**

Get this from the message context, as shown in EchoHeaders.jws :

private HttpServletRequest getRequest() {

MessageContext context = MessageContext.getCurrentContext();

HttpServletRequest req = (HttpServletRequest) context.getProperty(HTTPConstants.MC\_HTTP\_SERVLETREQUEST);

return req;

}

Note that the request will be null if the message came in over a different transport.

Hard coding your entire webapp to only support SOAP over HTTP may be very short-sighted.

**How do I deploy a service (i.e. update the server-config.wsdd) without a running Axis server?**

To deploy a service descriptor without a running Axis server, use the org.apache.axis.utils.Admin class:

cd webapps/axis/WEB-INF

java -cp $AXIS\_CLASSPATH org.apache.axis.utils.Admin server deploy.wsdd

The program will create or update the server-config.wsdd in the current directory, deploying the service(s) from the file deploy.wsdd. The copy of server-config.wsdd from axis.jar will be used as the default starting config if it has to create the file.

**How do I manually supply WSDL instead of returning automatically generated WSDL?**

In your WSDD file, you can specify the location of a WSDL file.

<service name="Test" provider="java:RPC" >

...

<wsdlFile>/org/someone/res/mywsdl.wsdl</wsdlFile>

<!-- <wsdlFile>WEB-INF/wsdls/mywsdl.wsdl</wsdlFile> -->

...

</service>

Axis will first look for a file of that name on the local path, then for a resource of that name on the classpath.

**How do I set a header when using WSDL2Java stubs?**

There are two styles of headers, explicit and implicit. Explicit headers are defined in the WSDL of the service. The WSDL2Java generation tool will recognize these headers in most cases and emit stub class methods that include the headers as arguments to the methods.

In other cases, you may want to set headers that are not explicitly called out in the WSDL. For instance, you want to do some custom processing in a handler or add security. In this case you can add headers to request before you invoke the stub method.

There are are two setHeader APIs on the org.apache.axis.client.Stub class. The first takes the namespace, name and value of the header.

setHeader(String namespace, String partName, Object headerValue)

The second takes a SoapHeaderElement:

setHeader(SOAPHeaderElement header)

Here is an example of using the first API

FooServiceLocator loc = new FooServiceLocator();

FooService binding = loc.getFooService();

org.apache.axis.client.Stub s = (Stub) binding;

s.setHeader("http://my.name.space/headers", "mysecurityheader", "This guy is OK");

result = binding.myOperation(...);

**How do I get response headers when using WSDL2Java stubs?**

You use the getResponseHeader() API in the Axis Stub object.

["SOAPHeaderElement"] getResponseHeader(String namespace, String partName)

Or to get the entire list of headers

["SOAPHeaderElement"][] getResponseHeaders()

Here is a code snippet that gets a header returned from myOperation().

FooServiceLocator loc = new FooServiceLocator();

FooService binding = loc.getFooService();

// Call web service operation that returns a header

result = binding.myOperation(...);

org.apache.axis.client.Stub s = (Stub) binding;

s.getResponseHeader("http://my.name.space/headers", "mysecurityheader");

**How do a get/set headers in my web service code?**

You use the MessageContext object .

Here is some example code

// Get current message context

MessageContext ctx = org.apache.axis.MessageContext.getCurrentContext();

// Get SOAP envelope of request

SOAPEnvelope env = ctx.getRequestMessage().getSOAPEnvelope();

// Get Header

SOAPHeaderElement soapHeaderElement = env.getHeaderByName(headerNamespace, headerName);

// Get value of header, could be a complex type!

Object value = soapHeaderElement.getObjectValue();`

**How do I set a header in the response to the client?**

Create a SOAPHeaderElement and add it to the SOAPEnvelope of the response message.

// Get current message context

MessageContext ctx = org.apache.axis.MessageContext.getCurrentContext();

// Get SOAP envelope of response

SOAPEnvelope env = ctx.getResponseMessage().getSOAPEnvelope();

// Create SOAP header object } } }

SOAPHeaderElement headerElement =

new org.apache.axis.message.SOAPHeaderElement(headerNamespace, headerName, headerValue);

headerElement.setMustUnderstand(mustUnderstand);

// Set Header

env.addHeader(headerElement);

**Where does Java2WSDL pick up the names for method parameters?**

The Java2WSDL documentation says that to generate a WSDL from a Java source

file, one should:

"Write and compile a Java interface (or class) that describes the web service interface.... (example) ... Note: If you compile your class with debug information, Java2WSDL will use the debug information to obtain the method parameter names."

Java only puts method parameter names into .class files for compiled classes (not interfaces), and only then for non-abstract methods. So Axis can only get method parameter names in certain situations.

However, Java2WSDL has an "-i" option that lets one specify an implementation class for an interface being processed, from which to extract method parameter names.

So, the documentation needs an update with qualifying the comment above, and maybe noting what the user must do to get method parameter names when generating a WSDL from an interface; it may save a little head-scratching.

**Creating web service client using Axis**

If you know the url of a web service (possibly one of your own at, say for this example, http://localhost:8080/axis/Bing.jws), and you want to write a client to use that web service, you can have Axis generate some Java source code files for you to make your client code simpler:

java org.apache.axis.wsdl.WSDL2Java http://localhost:8080/axis/Bing.jws?wsdl

Whichever directory you run that WSDL2Java command in, inside it you'll now see localhost/axis/Bing\_jws. Inside Bing\_jws you'll find 4 autogenerated .java files. Don't touch them. They are:

1. **Bing.java** -- an interface that extends java.rmi.Remote and declares some instance methods, the names of which correspond to the instance methods you created in your webapps/axis/Bing.jws file.
2. **BingService.java** -- another interface. This one extends javax.xml.rpc.Service and declares getBingAddress(), getBing(), and getBing( java.net.URL ).
3. **BingServiceLocator.java** -- a class that extends org.apache.axis.client.Service and implements BingService. You instaniate this class in your client. It's your Service object; you tell it to getBing(), and it gives you a Bing object on which you may call those instance methods you wrote in your Bing.jws class. :)
4. **BingSoapBindingStub** -- a class that extends org.apache.axis.client.Stub and implements Bing.

Finally, to actually create your client, you'll need to create your own new class that makes use of the ones listed above. It'll probably look something like:

import localhost.axis.Bing\_jws.\*;

public class MyBingClient

{

public static void main( String args[] ) throws Exception

{

BingService service = new BingServiceLocator();

Bing myBing = service.getBing();

// call instance methods on myBing, ex., say, myBing.foo()

}

}

**How To display SOAP Messages in Axis Client ?**

You have to enable the org.apache.axis.handlers.LogHandler declaring this handler in the client-config.wsdd configuration file.

**Notes:** client-config.wsdd must be dropped at the root of the classpath (e.g. under WEB-INF/classes). LogHandler.writeToConsole seems broken in Axis 1.3 (see AXIS-2455).

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Description** | **Optional** | **Default Value** |
| LogHandler.writeToConsole | If true, output SOAP messages in the console; otherwise, output in a file. | yes | False |
| LogHandler.fileName | Specifies the name of the output file when LogHandler.writeToConsole is false | yes | axis.log |

**How do I get the IP address of the client?**

Use the following:

String remoteIP = msgContext.getStrProp(Constants.MC\_REMOTE\_ADDR);

**Note:** your client IP address can be misleading sometimes because of proxy servers.

**How do I monitor SOAP traffic?**

You have several options available:

1. TCPMON included with Axis.
2. SOAP Monitor included with Axis.
3. SoapKnox Real-Time Web Service Monitoring Tool Is a commercially available plug-in that works natively inside Axis.
4. Soapmeter is an open source enhanced version of TCPMON.
5. SOAPscope is an inexpensive commercial product from Mindreef.
6. Microsoft Network Monitor included with Microsoft Windows Systems Management Server (SMS)
7. Microsoft SOAP Trace included with SOAP Toolkit 3.0.
8. AmberPoint Express is a commercial web services development managment tool. (Free for development).
9. Eclipse Web Service Validation Tools is an open source plugin for use with Eclipse.

**Apache Axis2™**

Apache Axis2™ is a **Web Services / SOAP / WSDL** **engine**, the successor to the widely used Apache Axis SOAP stack. There are two implementations of the Apache Axis2 Web services engine - Apache Axis2/Java and Apache Axis2/C

**Why Apache Axis2:**

A new architecture for Axis2 was introduced during the August 2004 Summit in Colombo, Sri Lanka. The new architecture on which Axis2 is based on is more flexible, efficient and configurable in comparison to Axis1.x architecture.

Apache Axis2 not only supports SOAP 1.1 and SOAP 1.2, but it also has integrated support for the widely popular REST style of Web services.

Apache Axis2 is more efficient, more modular and more XML-oriented than the older version. It is carefully designed to support the easy addition of plug-in "modules" that extend their functionality for features such as security and reliability.

Apache Axis2 is built on Apache AXIOM (Axis2 comes with its own light-weight object model, AXIOM, for message processing which is extensible, highly performant and is developer convenient), a new high performant, pull-based XML object model.

Refer http://axis.apache.org/axis2/java/core/docs/quickstartguide.html