Java Servlets

Servlets provide extensions of functionalities of servers, most commonly HTTP servers.
They are precompiled Java programs that are executed on the server side.
Servlets provide the following benefits:
- Efficiency
- Persistence
- Portability
- Robustness
- Extensibility
- Security

Servlet Engine

In order to execute a servlet, a servlet engine is needed.
Clients can only invoke servlets via a servlet engine.
A servlet engine is a key component in an application server.
The main responsibility of a servlet engine is to accept and process requests that invoke servlets.
Servlet engines are also known as servlet containers.

Tomcat

A Java Servlet, Java ServerPage (JSP) container.
A reference implementation of servlet and JSP.
It is small, installs quickly, and free.
Version 3.2 supports Servlet 2.2 and JSP 1.1
Version 4.0 supports Servlet 2.3 and JSP 1.2
Tomcat can be used as a:
- stand-alone container
- for development and debugging
- add-on to a web server
- for deployment
- supports Apache, IIS, and Netscape web servers
- in-process or out-of-process

Installing Tomcat

Download:
http://jakarta.apache.org/tomcat
Current version:
4.0.4
Unpack in a directory, for example
C:\foo\ (Windows)
/home/jia/foo/ (Linux)
A new subdirectory will be created:
jakarta-tomcat-4.0.4
This directory will be called Tomcat root or Tomcat home.
It will be referred to as <tomcat home>
Installing Tomcat (cont'd)

- Subdirectories of Tomcat home:
  
  **bin**  scripts to start/shutdown Tomcat etc.
  **conf**  server configuration files
  **common**
    - **common/classes**  unpacked classes
    - **common/lib**  jar files
  **classes**  unpacked classes available to all web apps
  **lib**  jar files available to all web apps
  **logs**  log files
  **server**  class/jar files for Tomcat internal use only
  **webapps**  web applications (include ROOT, examples)
  **work**  temporary working directory

- Read documentations:
  
  - README.txt
  - RUNNING.txt

- Set environment variables:
  
  - `JAVA_HOME`  the root directory of JDK
  - `CATALINA_HOME`  the root directory of Tomcat

  On Windows: (AUTOEXEC.BAT)
  ```
  set JAVA_HOME=C:\j2sdk1.4.0
  set CATALINA_HOME=C:\foo\jakarta-tomcat-4.0.4
  ```

  On Linux: (bash: ~/.bash_profile)
  ```
  export JAVA_HOME=/usr/local/j2sdk1.4.0
  export CATALINA_HOME=/home/jia/foo/jakarta-tomcat-4.0.4
  ```

Running Tomcat

- Start up Tomcat
  
  default port: 8080

- Point browser to
  
  `http://localhost:8080/`

- The default page served is located at
  
  `<tomcat home>/webapps/ROOT/index.html`

- Each subdirectory under
  
  `<tomcat home>/webapps/`
  
  represents a `web application`, which consists of a collection of HTML pages, servlets, JSP pages, Java classes, and other resources.

- If things go wrong, check the log file:
  
  `<tomcat home>/logs/catalina.out`
A Simple Servlet

Response page contents are sent in plain text.

```java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Hello extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                   throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello! The time is " +
                       new Date());
    }
}
```

Copyright © 2001-2002, Xiaoping Jia.

A Simple Servlet (cont'd)

- Compile servlet:
  ```
  javac -classpath <tomcat home>/common/lib/servlet.jar
         Hello.java
  ```
- Deploy servlet:
  ```
  copy Hello.class to
  <tomcat home>/webapps/ROOT/WEB-INF/classes/
  ```
- Start/restart Tomcat
- Invoke servlet:
  ```
  http://localhost:8080/servlet/Hello
  ```

Write Content in HTML

```java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloHTML extends HttpServlet {
    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("Hello! The time is " +
                     new Date());
    }
}
```

Copyright © 2001-2002, Xiaoping Jia.
Write Content in HTML (cont'd)

```java
out.println("<html>");
out.println("<head><title>Hello</title></head>");
out.println("<body>\n<br><h1>Hello!</h1>\nThe time is <i>" +
    new Date() + "</i>\n</body>\n</html>
```

Invoke:

```
http://localhost:8080/servlet/HelloHTML
```

Servlet API Packages

Package `javax.servlet`

- contains interfaces and abstract classes for protocol-independent generic servlets.

Package `javax.servlet.http`

- contains interfaces and abstract classes for servlets using the HTTP protocol.
- extends the interfaces and classes in `javax.servlet`.

A servlet engine provides implementation of these two packages:

```
tomcat home\common\lib\servlet.jar
```

Servlet API documentations are available at:

```
http://localhost:8080/tomcat-docs/servletapi/index.html
```

Servlet Framework

Design pattern: **Template Methods**

```
GenericServlet

HttpServlet
```

Copyright © 2001-2002, Xiaoping Jia.
Servlet Framework (cont'd)

interface javax.servlet.Servlet
- Defines methods that all servlets must implement.
  - `init()`, `service()`, `destroy()`

abstract class javax.servlet.GenericServlet
- Defines a generic, protocol-independent servlet.
  - `init()`, `service()`, `destroy()`

abstract class javax.servlet.http.HttpServlet
- Defines a servlet using the HTTP protocol.
  - `doGet()`, `doPost()`, etc.

interface javax.servlet.SingleThreadModel
- Ensures that servlets handle only one request at a time.
  - Marker interface. Defines no methods.

Servlet Lifecycle

The life cycle of a servlet is controlled by the servlet engine.

Servlets are instantiated (loaded) and destroyed (unloaded) by the servlet engine.

For a generic servlet, the servlet engine calls the following methods:

- `init()` when the servlet is first loaded.
- `service()` when each request is received.
- `destroy()` just before the servlet is unloaded.

void init()
- Called once when the servlet is first loaded.
- Not called for each request.
- To perform one-time initialization or configuration of the servlet, e.g.
  - reading initialization parameters
  - reading configuration parameters from property files or other resources
  - setting up databases
- To be overridden by subclasses. No need to call `super.init()`.

Servlet Framework (cont'd)

interface javax.servlet.ServletRequest
- Provides client request information to a servlet.

interface javax.servlet.ServletResponse
- Assists a servlet in sending a response to the client.

interface javax.servlet.http.HttpServletRequest
- Extends `javax.servlet.ServletRequest`
- Provides request information for HTTP servlets.

interface javax.servlet.http.HttpServletResponse
- Extends `javax.servlet.ServletResponse`
- Provides HTTP-specific functionality in sending a response.
**Servlet Life Cycle:** `service()`

`abstract
void service(ServletRequest request,
            ServletResponse response)`

- Called when each request is received.
- To be overridden by subclasses to handle requests.
- Executed in a new thread for each request.
- Multiple threads may execute this method at the same time.
- However, you may choose the `single thread model` to prevent multiple threads to execute this method at the same time.
- `implements interface SingleThreadModel`

**Servlet Life Cycle:** `destroy()`

`void destroy()`

- Called just before the servlet is unloaded.
- Not called after each request.
- To release resources.
- Servlets may be unloaded by the servlet container at any time.

**Http Servlet Methods**

- The `service()` method is overridden to dispatch requests to `doGet()`, `doPost()`, etc.
- `allow service to be added incrementally.
- `provide default behavior for HEAD, TRACE, TRACE
- `requests
- Do not override the `service()` method.
- `Override `get`, `post`, etc. to handle requests
- `get`, `post`, etc. to handle requests

Copyright © 2001-2002, Xiaoping Jia.
Web Applications

- A web application consists of a collection of resources including HTML pages, servlets, JSP's, other Java classes, images, and audio/video files.
- All the resources of a web application are located under a directory known as the document root of the web application.

Web Applications and Their Contexts

- A web application is mapped to a specific URI prefix, so that an application server (servlet engine) can recognize and forward any request with that URI prefix to the web application.
- A mapping of a URI prefix to a web application creates a context in the application server.
- To deploy a new web application in an application server, a new context needs to be added to the application server.

Configuring Tomcat

Tomcat server configuration file is an XML document:

```xml
<Server>
  <Service name="Tomcat-Standalone">
    <Connector port="8080" ... />
    <Engine name="Standalone" ...
    <Host name="localhost" debug="0" appBase="webapps" unpackWARs="true">
      <Context path="/examples" docBase="examples" debug="0" reloadable="true">
        ...
        (add your context here)
      </Context>
    </Host>
  </Service>
</Server>
```

A Little XML

- eXtensible Markup Language
- Developed by World Wide Web Consortium (W3C)
- http://www.w3c.org
- Similar to HTML
- XML and HTML are both based on SGML
- XML is influenced by HTML
- HTML documents are not well-formed XML documents
- XHTML is an XML version of HTML
Components of an XML Document

- **XML declaration**
  ```xml
  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
  ```

- **Elements**
  ```xml
  <course> ... </course>
  <student /> 
  ```

- **Attributes**
  ```xml
  <student id="1234567" />
  ```

Components of an XML Document (cont'd)

- **Document type definition (DTD)**
  ```xml
  <!DOCTYPE books [  ...  ]>
  ```

- **Comments**
  ```xml
  <!-- This is a comment -->
  <!-- This is an -- illegal -- comment -->
  ```

- **Processing instructions (PI)**
  ```xml
  <? ... ?>
  ```

XML Elements

- An XML document is a text document with *markups*, or *tags*.
- **start-tag** `<Server>`
- **end-tag** `</Server>`
- **empty-tag** `<Context/>`

- An *element* is denoted by
  - an empty tag, e.g. `<Context/>`, or
  - a matching pair of start and end tags e.g.
    ```xml
    <Server> ... </Server>
    ```

- Elements can be nested.
- An XML document consists of one or more elements.

XML Attributes

- Empty-tags and start-tags may have *attributes*
  ```xml
  <Cont ext path="/examples" docBase="/examples" debug="0"
  reloadable="true" />
  ```

- Each attribute is a name-value pair.
- Attribute values must be enclosed between quotation marks (".")
Well-Formed XML Documents

- An XML document is *well-formed* if
  - It has a single root element.
  - All start-tags are matched by end-tags, and *vice versa*
  - All start-tags and end-tage are properly nested.
- The following segments are not well-formed:
  - `<a> <b> <!-- comment --> </c>`
  - `<a> <b> <a> </b> </a>`
  - `<a> <b> </a> </b>`
- A well-formed XML document forms a tree of elements.

Valid XML Documents

- An XML document may impose further restrictions on its structure using DTD (Document Type Definition) or schema.
- An XML document is *valid*, if it is well-formed and satisfies the restrictions imposed by a DTD or schema.

Tomcat Configuration

- The elements in `<tomcat home>/conf/server.xml`
- The `<Server>` element represents the entire servlet container.
- The Tomcat server supports multiple *virtual hosts*.
- Each virtual host can have its own host name.
- Each virtual host is represented by a `<Host>` element.
- Each web application is assigned to a virtual host.
- Each virtual may contain one or more contexts.
- Each context is represented by a `<Context>` element inside `<Host>`

Specifying A Host

- The `<Host>` element may have the following attributes:
  - `appBase`
  - The *application base* directory of this virtual host.
  - This attribute is required.
  - Can be an absolute path or a path relative to the `<tomcat home>`
  - `name`
  - The host name.
Specifying A Context

- The `<Context>` element may have the following attributes:
  - `path`
    - The context path of the web application.
    - The prefix of the request URI for the web application.
    - This attribute is required.
    - Must start with `/` character.
    - Must be unique.

Specifying A Context (cont'd)

- `docBase`
  - This attribute is required.
  - Can use an absolute path or a path relative to the `appBase` of the enclosing `<Host>` element.
  - Examples:
    - `MyApps` relative to `<tomcat home>/webapps/`
    - `/home/jia/MyApps` absolute path on Unix
    - `C:\MyApps` absolute path on Windows
  - On a Windows platform, the drive letter and a colon (e.g. `c:`) is required when specifying an absolute path.

Specifying A Context (cont'd)

- `reloadable`
  - If true, instruct Tomcat to reload servlets whenever they are updated.
  - When developing and debugging a server, it is very convenient to have Tomcat reload it upon change, this lets you fix bugs and have Tomcat test the new code without the need to shutdown and restart.
  - Default value: `false`
  - While this feature is very useful during development, it requires overhead to do the checking. This capability should generally not be used in deployed production applications.

Specifying A Context (cont'd)

- `cookie`
  - Whether to use cookies for session identification.
  - Default value: `true`

- `debug`
  - Debug level used for logging debug messages.
    - `from 0 to 9`
    - Default value: `0`
Deploy Web Apps in Tomcat

Adding new contexts in

```xml
<Server>
  ...
  <Host name="localhost" debug="0" appBase="webapps" unpackWARs="true">
    ...
    <Context path="/se452" docBase="se452" debug="0" reloadable="true" />
    <Context path="/myapps" docBase="/home/jia/MyApps" debug="0" reloadable="true" />
  </Host>
</Server>
```

Copyright © 2001-2002, Xiaoping Jia.

Directory Structure of Web Apps

- The directory structure is J2EE standard.
- Assume the document root directory of the web app to be deployed is called `<WebApp>`.
- The contents of the document root directory:
  - Static html pages and other resources such as images files may be stored directly in this directory or its subdirectories, e.g.,
    - `<WebApp>/index.html`
    - `<WebApp>/images/coolpic.gif`
  - A subdirectory called `WEB-INF`
  - It contains servlets and the resources used by the servlets.
  - The contents of this subdirectory are protected.

The WEB-INF Directory

- The configuration file of the web application, known as the deployment descriptor
- `<WebApp>/WEB-INF/web.xml`
- It contains the unpacked class files of servlets.
- The subdirectory structure of this directory must coincide with the package structure of the servlets.
- `<WebApp>/WEB-INF/classes/*`
  - It contains the jar files of other Java classes used by the web application, such as third-party packages.

Creating Document Root

To test the new context:
- Create the document root directory and subdirectories under the document root:
  - For context `se452`
    ```
    <tomcat home>/webapps/se452
    <tomcat home>/webapps/se452/WEB-INF
    <tomcat home>/webapps/se452/WEB-INF/classes
    ```
  - For context `myapps`
    ```
    / home/jia/MyApps
    / home/jia/MyApps/WEB-INF
    / home/jia/MyApps/WEB-INF/classes
    ```

Copyright © 2001-2002, Xiaoping Jia.
Creating `web.xml`

- Create a deployment descriptor.

A minimum `web.xml`:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
</web-app>
```

Copyright © 2001-2002, Xiaoping Jia.

---

Deploying Web App

- Copy `web.xml` to `<doc root>/WEB-INF`
  - Context `se452`
    - `<doc root>/webapps/se452/WEB-INF/`
  - Context `myapps`
    - `/home/jia/MyApps/WEB-INF/`

- Copy servlet class files (`HelloHTML.class`) to
  - Context `se452`
    - `<doc root>/WEB-INF/classes`
  - Context `myapps`
    - `/home/jia/MyApps/WEB-INF/classes/`

Testing the New Context

Invoke the web apps in the new contexts as follows:

- `http://localhost:8080/myapps/servlet/HelloHTML`
- `http://localhost:8080/se452/servlet/HelloHTML`
- `http://localhost:8080`

Copyright © 2001-2002, Xiaoping Jia.

---

Configure Web Apps

The `<servlet>` and a `<servlet-mapping>` element in deployment descriptor (`web.xml`).

- `<servlet>` element: *servlet definition*
  - It defines a name for a servlet by mapping the name to the class name of the servlet.
  - It sets initialization parameters.
  - Different servlet names can be mapped to the same servlet class.

- `<servlet-mapping>` element: *servlet mapping*
  - It maps a URL or a URL pattern to a servlet name.
  - Different URL can be mapped to the same servlet name.
Servlet Definition and Mapping

An example of servlet definition and a servlet mapping:

```xml
<web-app>
  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>HelloHTML</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Using Servlet Definition

A servlet can be invoked with the name specified in the servlet definition:

- `http://localhost:8080/myapps/servlet/hello`
- `http://localhost:8080/se452/servlet/hello`
- `http://localhost:8080` (protocol, host, and port)
- `/se452,/myapps` (the context prefix)
- `/hello` (the servlet url)

Using Servlet Mapping

A servlet can be invoked with the URL specified in the servlet mapping:

- `http://localhost:8080/myapps/hello`
- `http://localhost:8080/se452/hello`
- `http://localhost:8080` (protocol, host, and port)
- `/se452,/myapps` (the context prefix)
- `/hello` (the servlet url)

Initialization Parameters

Sometimes we want to initialize a servlet when it is first loaded by the servlet engine.
- configuration
- specialization
- customization

Initialization parameters are usually handled in the `init()` method.
The `init()` Methods

void init()

- Override this one to initialize a servlet

void init(ServletConfig config)

- Do not override this one.
- If you do override, you must call
  
  super.init(config)

HelloHTML Revisited

Enhancements to the HelloHTML servlet.

- Takes two initialization parameters

  bgcolor, fgcolor

- It is placed in a package named
  
  myservlets

HelloHTML Revisited (cont'd)

package myservlets;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloHTML extends HttpServlet {
  protected String bgcolor;
  protected String fgcolor;

  public void init() {
    bgcolor = getInitParameter("bgcolor");
    fgcolor = getInitParameter("fgcolor");
  }

  <doGet () method>
}
HelloHTML Revisited (cont'd)

```java
if (fgcolor != null) {
    out.println("<h1><font color="" + fgcolor + ">
    Hello! </font></h1>);
} else {
    out.println("<h1> Hello! </h1>);
}
out.println("The time is " +
    new Date() + "</i>");
out.println("The request URI is " +
    request.getRequestURI() + "</i>");
out.println("</body>");
out.println("</html>");
```

Deploy Revised HelloHTML

- Copy the HelloHTML.class file to
  `<WebApp>/WEB-INF/classes/myservlets/HelloHTML.class`
- Edit the deployment descriptor
  `<WebApp>/WEB-INF/web.xml`

Servlet Definition

Add a servlet definition in `web.xml`:

```xml
<servlet>
    <servlet-name>hello-green</servlet-name>
    <servlet-class>myservlets.HelloHTML</servlet-class>
    <init-param>
        <param-name>bgcolor</param-name>
        <param-value>white</param-value>
    </init-param>
    <init-param>
        <param-name>fgcolor</param-name>
        <param-value>green</param-value>
    </init-param>
</servlet>
```

Servlet Definition (cont'd)

Add another servlet definition:

```xml
<servlet>
    <servlet-name>hello-blue</servlet-name>
    <servlet-class>myservlets.HelloHTML</servlet-class>
    <init-param>
        <param-name>bgcolor</param-name>
        <param-value>ivory</param-value>
    </init-param>
    <init-param>
        <param-name>fgcolor</param-name>
        <param-value>blue</param-value>
    </init-param>
</servlet>
```
Servlet Mappings

Add three servlet mappings in `web.xml`:

```xml
<servlet-mapping>
  <servlet-name>hello-green</servlet-name>
  <url-pattern>/green</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>hello-blue</servlet-name>
  <url-pattern>/blue</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>hello-blue</servlet-name>
  <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

Invoke HelloHTML

Invoke HelloHTML using URL `/green`:

```
http://localhost:8080/myapps/green
```

Hello!

The time is Wed Jan 10 16:20:28 CST 2001
The request URI is /myapps/green

Invoke HelloHTML (cont'd)

Invoke HelloHTML using URL `/blue`:

```
http://localhost:8080/myapps/blue
```

Hello!

The time is Wed Jan 10 16:21:14 CST 2001
The request URI is /myapps/blue

Invoke HelloHTML (cont'd)

Invoke HelloHTML using a undefined URL `/hello/redish`:

```
http://localhost:8080/myapps/hello/redish
```

Hello!

The time is Wed Jan 10 16:21:55 CST 2001
The request URI is /myapps/hello/redish

Copyright © 2001-2002, Xiaoping Jia.