JavaOne℠

Java™ Servlet 3.0: Empowering Your Web Applications With Async, Extensibility and More

Rajiv Mordani, Jan Luehe
Sun Microsystems
Greg Wilkins
Webtide / Jetty

Agenda

- Overview
- Ease of Development
- Dynamic registration of Servlets and Filters
- Pluggability
- Asynchronous support
- Security enhancements
- Miscellaneous

Overview

- Java Servlet 3.0 API – JSR 315
- About 20 members in the expert group
  - Good mix of representation from major Java EE vendors, web container vendors and individual web framework authors
- Main areas of focus
  - Ease of Development
  - Pluggability
  - Asynchronous support
  - Security

Status

- Specification in Proposed Final Draft
- Final release aligned with Java EE 6

Agenda

- Overview
- Ease of Development
- Dynamic registration of Servlets and Filters
- Pluggability
- Asynchronous support
- Security enhancements
- Miscellaneous

Ease of Development (EoD)

- Focus on Ease of Development (EoD) in the Servlet 3.0 API
- Enhance API to use the new language features introduced since J2SE 5.0
- Annotations for declarative style of programming
  - No web.xml needed
- Generics for type safety in the API without breaking backwards compatibility
- Better defaults and convention over configuration

Ease of Development
Use of Annotations

- Annotations to declare Servlets, Filters, Listeners and security constraints
  - `@WebServlet` – Define a Servlet
  - `@WebFilter` – Define a Filter
  - `@WebListener` – Define a Listener
  - `@WebInitParam` – Define init params
  - `@MultipartConfig` – Define fileupload properties
- Can use web.xml to override values specified in the annotations

Ease of Development
Use of Annotations (contd)

- `@WebServlet` for defining a Servlet
  - The annotation MUST have at a minimum the URL pattern for the Servlet
  - All other fields optional with reasonable defaults
  - For example, the default name of the Servlet is the fully qualified class name
  - Class MUST still extend **`HttpServlet`**
    - Method contracts for **`doGet, doPost`** inherited from abstract class

Servlet 2.5 example

```
public class SimpleSample
extends HttpServlet {

    public void doGet
    (HttpServletRequest req,
     HttpServletResponse res)
    {


    }

}
```

web.xml

(intentionally left unreadable)

```
<web-app>
  <servlet>
    <servlet-name>                MyServlet
    </servlet-name>
    <servlet-class>
     samples.SimpleSample
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      MyServlet
    </servlet-name>
    <url-pattern>
      /MyApp
    </url-pattern>
  </servlet-mapping>
...
```

Servlet 3.0 example

```
@WebServlet(“/foo”)
public class SimpleSample extends
HttpServlet

{

    public void doGet(HttpServletRequest
                req,HttpServletResponse res)

    {


    }

}
```

Servlet 3.0 example

```
@WebServlet(urlPatterns=“/foo”,
  name=”MyServlet”, asyncSupported=true)
public class SimpleSample extends
HttpServlet

{

    public void doGet(HttpServletRequest
                 req,HttpServletResponse res)

    {


    }

}
```

Agenda

- Overview
- Ease of Development
- Dynamic registration of Servlets and Filters
- Pluggability
- Asynchronous support
- Security enhancements
- Miscellaneous

Dynamic registration of Servlets and Filters

Register

- Performed during ServletContext initialization
- `ServletContext#add[Servlet|Filter]`
  - Overloaded versions take [Servlet|Filter] name and
    - Fully qualified [Servlet|Filter] class name OR
    - `Class<? extends [Servlet|Filter]>` OR
    - [Servlet|Filter] instance
  - Use returned **`Registration`** handle to configure all aspects of [Servlet|Filter]

Dynamic registration of Servlets and Filters

Create and Register

- `ServletContext#create[Servlet|Filter]`
  - Takes `Class<? extends [Servlet|Filter]>` argument
  - Container responsible for instantiating the `[Servlet |Filter]`
  - Supports resource injection by container
  - Returned [**`Servlet|Filter`**] instance may be fully customized before it is registered via the **`ServletContext#add[Servlet|Filter`**] methods

Dynamic registration of Servlets and Filters

Lookup

- `ServletContext#get[Servlet|Filter]Registration`
    - Takes [Servlet|Filter] name as argument
    - Returned **`Registration`** handle provides subset of configuration methods
        - May only be used to add initialization parameters and mappings
        - Any conflicts returned as **`java.util.Set`**

Dynamic registration of Servlets/Filters

Register Example

```
ServletRegistration.Dynamic dynamic =
    servletContext.addServlet(
        "DynamicServlet",
  "com.mycom.MyServlet");
dynamic.addMapping("/dynamicServlet");
dynamic.setAsyncSupported(true);
```

Dynamic registration of Servlets/Filters

Lookup Example

```
ServletRegistration declared =

  servletContext.getServletRegistration("Declar
  edServlet");

declared.addMapping("/declaredServlet");

declared.setInitParameter("param", "value");
```

Agenda

- Overview
- Ease of Development
- Dynamic registration of Servlets and Filters
- Pluggability
- Asynchronous support
- Security enhancements
- Miscellaneous

Pluggability

- Enable use of libraries and framework without boiler plate configuration in deployment descriptors
  - Put the burden on the framework developer
- Modularize **`web.xml`** to allow frameworks to be self-contained within their own JAR file
- Programmatic configuration APIs
- Use of annotations

Pluggability

Motivation for web.xml modularization

- Use of framework requires (possibly complex) configuration in **`web.xml`**
- For example
  - Declare a controller Servlet
  - Logging and security Filters
  - Declare Listeners to perform actions at various points in the lifecycle of the application
- Can get complex as dependencies increase
- Frameworks also need to document all the configuration that needs to be done

Pluggability
web-fragment.xml

- **`web-fragment.xml`** is descriptor for framework / library
- Included in **`META-INF`** directory
- Container responsible for discovering fragments and assembling the effective deployment descriptor
- Almost identical to **`web.xml`**
  - Ordering related elements different
- Only JAR files in **`WEB-INF/lib`** considered as fragments

Pluggability

web-fragment.xml example

```
 <web-fragment>
  <servlet>
    <servlet-name>welcome</servlet-name>
    <servlet-class>com.mycom.WelcomeServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>welcome</servlet-name>
    <url-pattern>/Welcome</url-pattern>
  </servlet-mapping>
  ...
</web-fragment>
```

Pluggability
Ordering

- Compatible with JavaServer™ Faces
- Fragments identified by **`<name>`**
- **`web.xml`** may declare **absolute** ordering of fragments via **`<absolute-ordering>`**
- Fragments may declare ordering preferences **relative** to other fragments via **`<ordering>`** with nested **`<before>`** and **`<after>`**
  - Ignored if **`<absolute-ordering>`** specified
- Special **`<others/>`** element moves fragment to beginning or end of list of sorted fragments

Pluggability
Resource sharing

- Static and JavaServer™ Pages (JSP) resources no longer confined to web application's document root
- May be placed inside **`WEB-INF/lib/[*.jar]/META-INF/resources`**
- Container must honor this new location when processing HTTP requests and calls to **`ServletContext#getResource[AsStream]`**
- Resources in document root take precedence over those in bundled JAR files

Pluggability
Resource sharing: Example

`mywebapp.war` packaging:

```
/index.jsp

/WEB-INF/lib/shared.jar!/META-
 INF/resources/shared.jsp
```

Request for:

`http://localhost:8080/mywebapp/shared.jsp`

will be served from:

```
 /path/to/mywebapp/WEB-
INF/lib/shared.jar!/META-
INF/resources/shared.jsp
```

Pluggability
Shared libraries

- Support plugging in of container installed JAR files
    - Examples: JSF, JAX-WS, Spring
- Libraries may provide implementation of **`ServletContainerInitializer`**
- Looked up via the JAR Services API in JDK 6
- Invoked before any Listeners during the initialization of the application

Pluggability
Shared libraries (contd)

- **`ServletContainerInitializer`** expresses interest in Classes via **`@HandlesTypes`**
- Container discovers classes that match **`@HandlesTypes`** and passes them to **`ServletContainerInitializer`**
- **`ServletContainerInitializer`** inspects passed in Classes and may register Servlets and Filters based on them

Pluggability

ServletContainerInitializer example

```
@HandlesTypes(WebService.class)
public  class  JAXWSInitializer  implements
 ServletContainerInitializer {
   public void onStartup(Set<Class<?>> c,
                                   ServletContext ctx)
   {
      ctx.addServlet(“JAXWSServlet”,
                   “com.sun.jaxws.JAXWSServlet”);


   }
}
```

Agenda

- Overview
- Ease of Development
- Dynamic registration of Servlets and Filters
- Pluggability
- Asynchronous support
- Security enhancements
- Miscellaneous

Why Asynchronous Servlets?

- Not for Async IO!
  - Requests mostly small (single packet)
  - Hard to asynchronously produce large responses
  - Async IO support waiting for NIO2 (Servlet 3.1?)

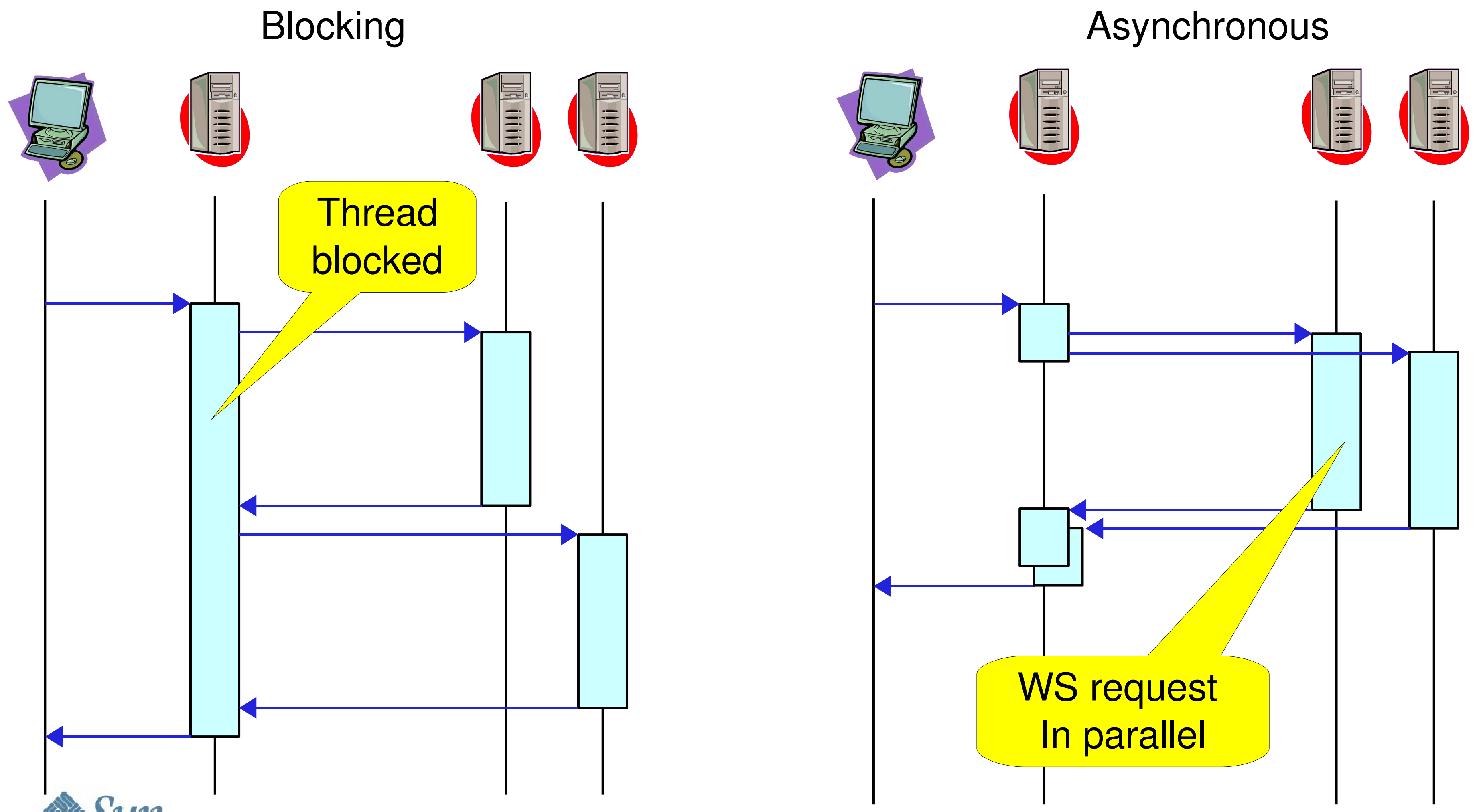
- Async Servlets are for:
  - Waiting for resources (eg JDBC connection)
  - Waiting for events (eg Chat)
  - Waiting for responses (eg web services, QoS)

Blocking waiting consumes resources

- Web Application using remote web services
  - Handling 1000 requests / sec
  - 50% requests call remote web service
  - 500 threads in container thread pool

- If remote web service is slow (1000ms)
  - Thread starvation in 1 second!
  - 50% of requests use all 500 threads

Waiting for Web Services



Asynchronous API

ServletRequest

- `ServletRequest#isAsyncSupported()`
    - **`True`** if ALL [Filter|Servlet]s support async in
        - the Filter chain
        - the RequestDispatch chain
- Configured in
    - **`web.xml`**
        - `<async-supported>true</async-supported>`
    - With annotation
        - `@WebServlet(asyncSupported=true)`
    - Programmatic
        - `registration.setAsyncSupported(boolean)`

Asynchronous API

ServletRequest

- **`AsyncContext ServletRequest#startAsync()`**
  - Called by [Filter|Servlet]
  - Response is NOT commited on return of:
    - **`Servlet.service(request,response)`**
    - `Filter chain`
- **`AsyncContext ServletRequest#startAsync (ServletRequest req, ServletResponse res)`**
  - Variation that preserves wrappers

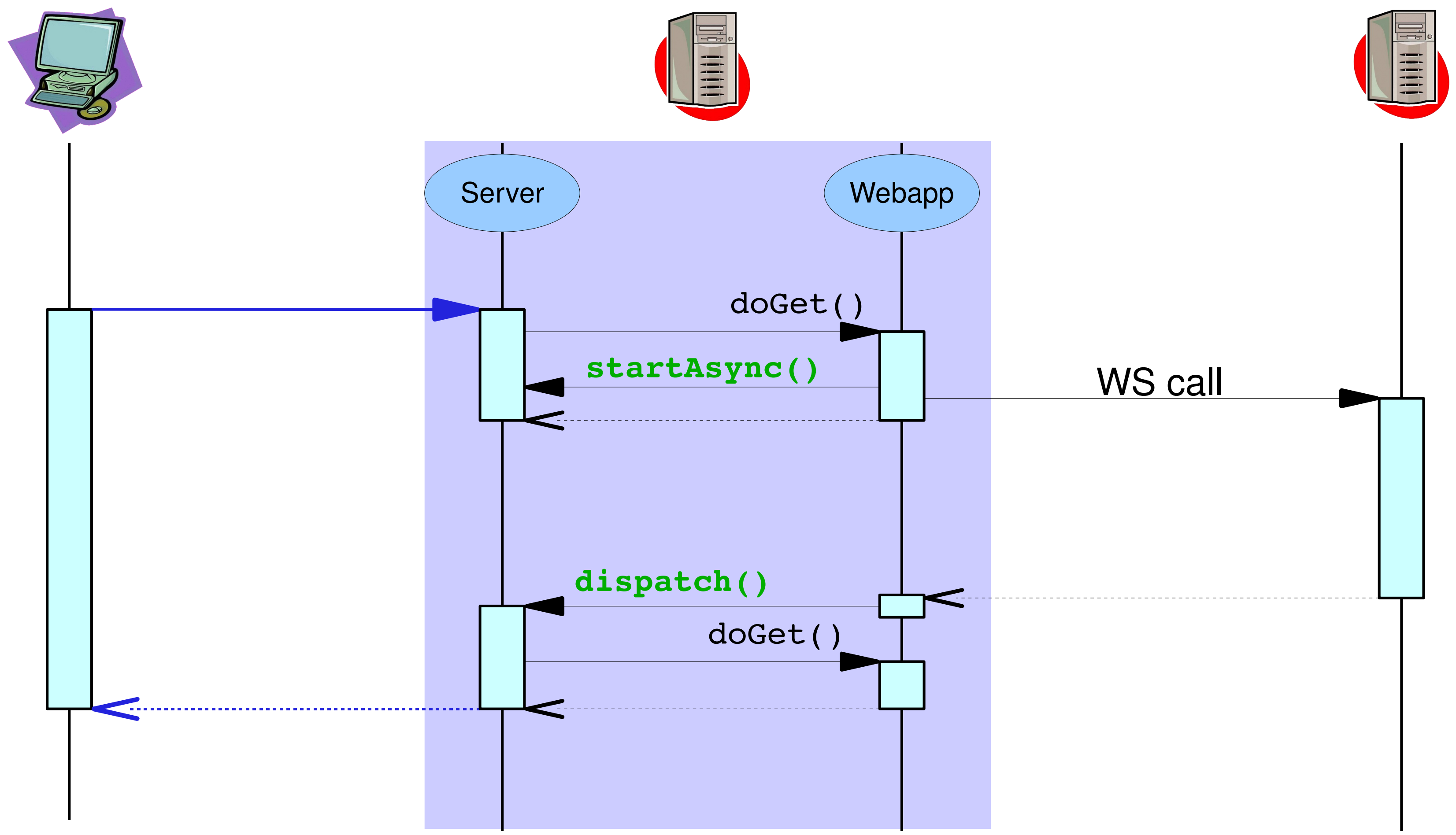
Asynchronous API
AsyncContext

- `AsyncContext#dispatch()`
  - Called by your asynchronous handler
  - Schedule async dispatch: `DispatcherType.ASYNC`
  - Response generated by [Filter|Servlet] using:
    - container thread pool
    - JSP, JSF or other frameworks usable
    - JNDI, JTA, EJBs usable
- `AsyncContext#dispatch(String path)`
  - Variation to async dispatch to specific Servlet

Asynchronous API
AsyncContext

- **`AsyncContext#complete()`**
  - Called by your asynchronous handler
  - Response has been generated asynchronously
    - without Servlet features, or
    - with **`AsyncContext#start(Runnable r)`**
      - for JNDI, classloader

Asynchronous Web Service



Multiple Usage Styles

- **`startAsync()    …    dispatch()`**
  - Retry request after async wait
  - Filters re-applied if on **`DispatcherType.ASYNC`**
- **`startAsync()    …    dispatch(path)`**
  - Use specific Servlet handling after async wait
- **`startAsync()    …    complete()`**
  - Generate response asynchronously

Multiple Usage Styles

- **`startAsync(req,res)… dispatch()`**
  - Retry request after async wait
  - Wrappers are kept
  - **`RequestDispatcher#forward`** target used
- **`startAsync(req,res)… dispatch(path)`**
  - Specific Servlet handling after async wait
- **`startAsync(req,res)… complete()`**
  - Generate wrapped response asynchronously

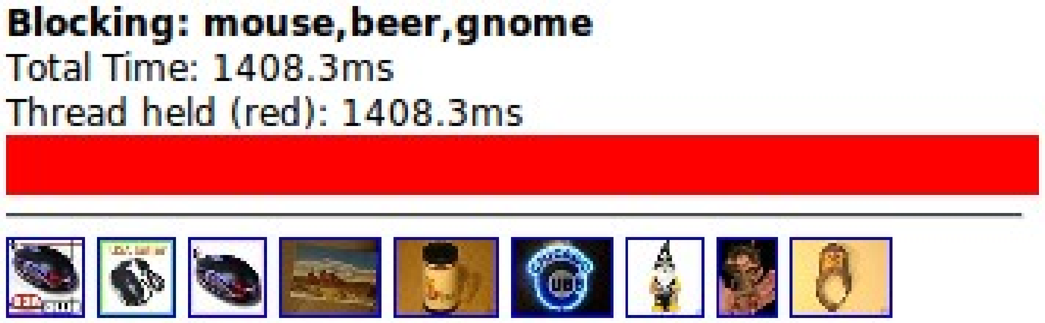
Asynchronous API Details

- Timeouts
  - `ServletRequest#setAsyncTimeout(long ms)`
  - By default error dispatch on timeout
- Listeners
  - `AsyncListener#OnTimeout`
  - `AsyncListener#OnComplete`

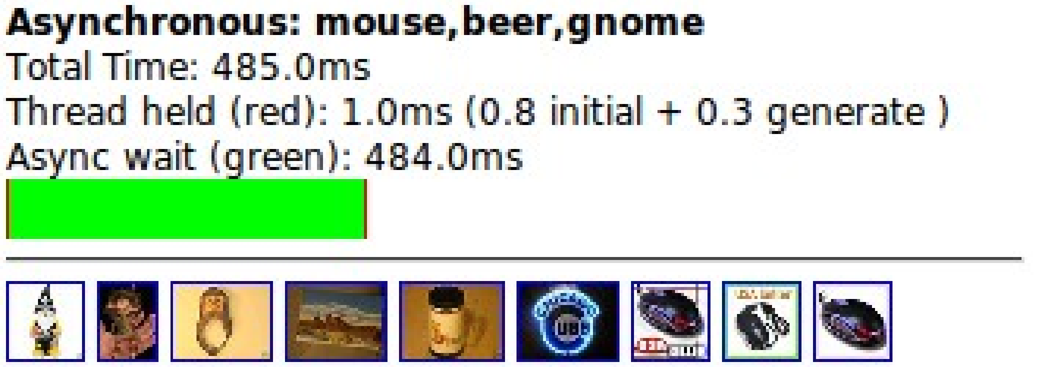
Demonstration

Asynchronous eBay Web Service

- EoD packaging
  - META-INF
    - web-fragment.xml
    - Resources/*
- Glassfish Container
  - Async Serlvet
- Jetty HTTP Client
  - Async Client



Blocking: mouse,beer,gnome
Total Time: 1408.3ms
Thread held (red): 1408.3ms



Asynchronous: mouse,beer,gnome
Total Time: 485.0ms
Thread held (red): 1.0ms (0.8 initial + 0.3 generate)
Async wait (green): 484.0ms

Agenda

- Overview
- Ease of Development
- Dynamic registration of Servlets and Filters
- Pluggability
- Asynchronous support
- Security enhancements
- Miscellaneous

Security

Security constraints via common annotations

- Support for common annotations
  - **`@RolesAllowed`** -> **`auth-constraint`** with roles
  - **`@DenyAll`** -> Empty **`auth-constraint`**
  - **`@PermitAll`** -> No **`auth-constraint`**
  - **`@TransportProtected`** -> **`user-data-constraint`**
- Annotations enforced on **`javax.http.Servlet`** class and **`doXXX`** methods of **`HttpServlet`**
- Method-targeted annotations take precedence over class-targeted annotations

Security

Security constraints via common annotations (contd)

- Security constraints in **`web.xml`** override annotations, **`metdata-complete`** disables annotations
- **`web-resource-collection`** enhanced with **`http-method-omission`** to
  - Allow constraints to be specified on non-enumerable HTTP method subsets (i.e., all other methods)

Security

Programmatic container authentication and logout

- **`HttpServletRequest#login(String username, String password)`**
    - Replacement for FBL
    - Application supervises credential collection
- **`HttpServletRequest#authenticate(HttpServletResponse)`**
    - Application initiates container mediated authentication from a resource that is not covered by any authentication constraints
    - Application decides when authentication must occur

Security

Programmatic container authentication and logout (contd)

- **`HttpServletRequest#logout`**
- Integration of additional container authentication modules via Servlet Profile of JSR 196 recommended

Agenda

- Overview
- Ease of Development
- Dynamic registration of Servlets and Filters
- Pluggability
- Asynchronous support
- Security enhancements
- Miscellaneous

Miscellaneous Features / APIs

- Session tracking cookie configuration
  - Via **`web.xml`**
  - Programmatic via **`javax.servlet.SessionCookieConfig`**
- Support for **`HttpOnly`** cookie attribute
  - Example: **`servletContext.getSessionCookieConfig().setHttpOnly(true)`**
- Default error page

Miscellaneous Features / APIs (contd)

```
ServletRequest#getServletContext
ServletRequest#getDispatcherType
Servlet[Request|
  Response]Wrapper#isWrapperFor
HttpServletResponse#getStatus
HttpServletResponse#getHeader
HttpServletResponse#getHeaders
HttpServletResponse#getHeaderNames
```

Miscellaneous Features / APIs (contd)

File upload APIs

`ServletRequest#getParts`

`ServletRequest#getPart`

`@MultipartConfig`

Changes to **`web.xml`**

Summary

- Major revision since Servlet 2.4
- Comprehensive set of new features enable modern style of web applications and greatly increases developer productivity
- Simplifies assembly of large applications from reusable components

JavaOne

GlassFish Community
Open Source and Enterprise Ready

- **GlassFish V3 Preview Available now!**
  - Java EE 6 reference implementation
  - Modular OSGi architecture – easy to develop & deploy
  - Runs in-process and easy to extend
  - Support for Ruby-on-Rails, Groovy and Grails, Python and Django
- **GlassFish V2 – Production Ready**
  - Best price/performance open source App server with Clustering, High Availability, Load Balancing
  - Secure, Reliable, Transactional, .NET-interop Web svcs
  - Support for Ajax and Comet
- **GlassFish ESB**
  - SOA and Business Integration platform
- **GlassFish Communications App Server**
  - SIP servlet technology for converged services

- **24x7 Enterprise and Mission Critical Support**
  - **sun.com/glassfish**
- **Tools Integration**
  - NetBeans and Eclipse

glassfish.org

Always free to download, deploy and distribute

Webtide & Jetty

- Status update
- http://eclipse.org/jetty

JavaOne℠

Thank You

Rajiv Mordani
rajiv.mordani@sun.com
Jan Luehe
jan.luehe@sun.com
Greg Wilkins
gregw@webtide.com