ORACLE®

Understanding the Nuts & Bolts of Java EE 6

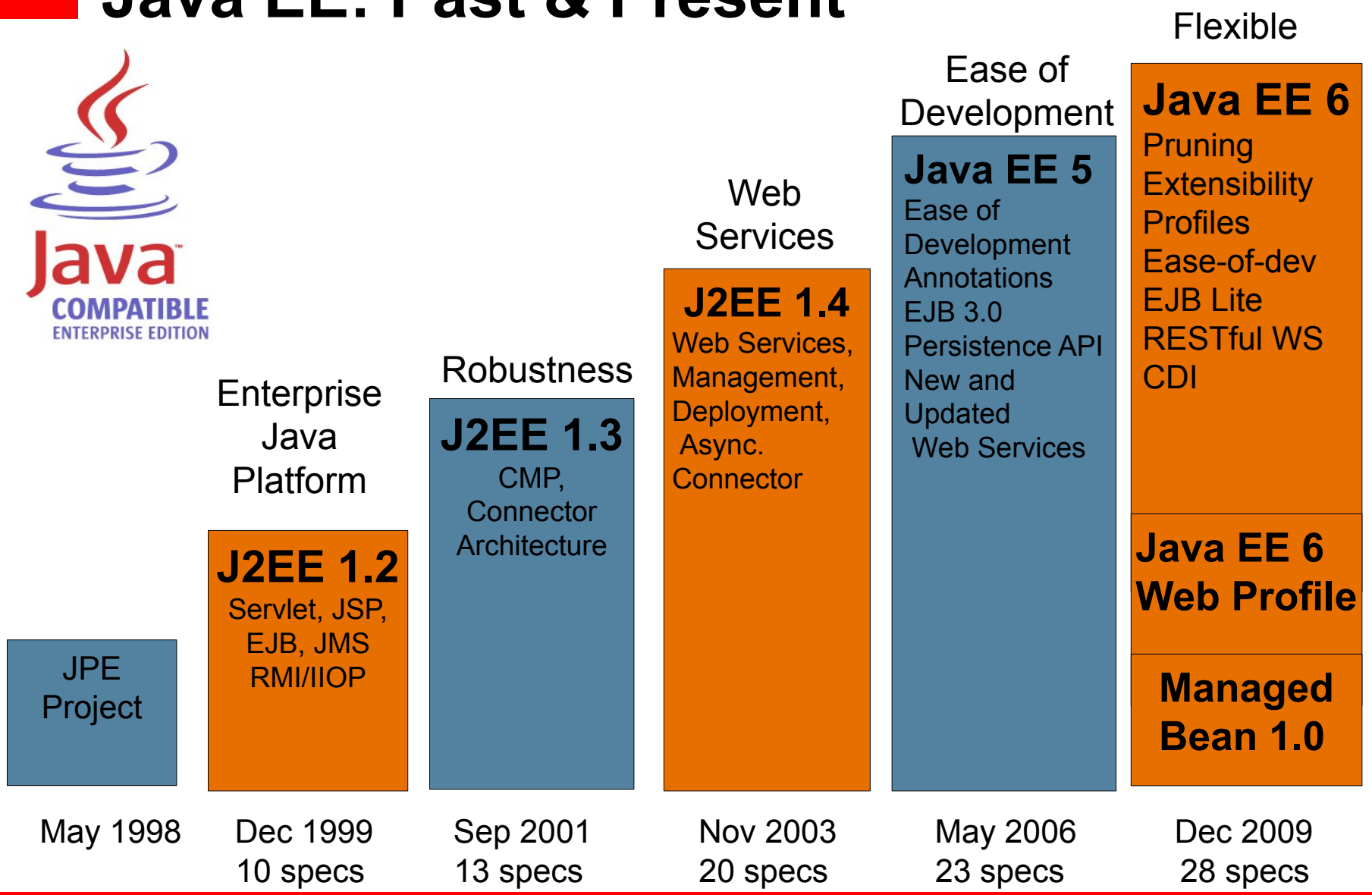
Arun Gupta, Java EE & GlassFish Guy
blogs.sun.com/arungupta, @arungupta

The following/preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Are you tweeting ?

#glassfish
#jfokus

Java EE: Past & Present



Compatible Java EE 5 Impl

http://java.sun.com/javaee/overview/compatibility-javaee5.jsp

Compatible Java EE 6 Impls

Today:

Tmax Soft

JBoss a division of Red Hat

Web Profile Only

Java COMPATIBLE ENTERPRISE EDITION

Announced:

ORACLE FUSION MIDDLEWARE WEBLOGIC

caucho

WebSphere

APACHE GERONIMO

Light-weight

- Java EE 6 Web Profile
- Pruning
  - Pruned today, means
    - Optional in the next release
    - Deleted in the subsequent releases
  - Technologies marked in Javadocs
    - EJB 2.x Entity Beans, JAX-RPC, JAXR, JSR 88

- EJB-in-WAR
- No-interface EJB
- Optional “web.xml”/”faces-config.xml”
- Annotation-driven
  - @Schedule
  - @Path
  - @Inject
  - . . .


<web-fragment>
<filter>
<filter-name>wicket.helloworld</filter-name>
<filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
<init-param>
<param-name>applicationClassName</param-name>
<param-value>...</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>wicket.helloworld</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
</web-fragment>

Java EE 6 - Done

- Specifications approved by the JCP
- Reference Implementation is GlassFish Server Open Source Edition 3
- TCK

Dec 2009

Java EE 6 Specifications

- The Platform
- Java EE 6 Web Profile 1.0
- Managed Beans 1.0

Java EE 6 Specifications
New

- Contexts and Dependency Injection for Java EE (JSR 299)
- Bean Validation 1.0 (JSR 303)
- Java API for RESTful Web Services (JSR 311)
- Dependency Injection for Java (JSR 330)

Java EE 6 Specifications
Extreme Makeover

- Java Server Faces 2.0 (JSR 314)
- Java Servlets 3.0 (JSR 315)
- Java Persistence 2.0 (JSR 317)
- Enterprise Java Beans 3.1 & Interceptors 1.1 (JSR 318)
- Java EE Connector Architecture 1.6 (JSR 322)

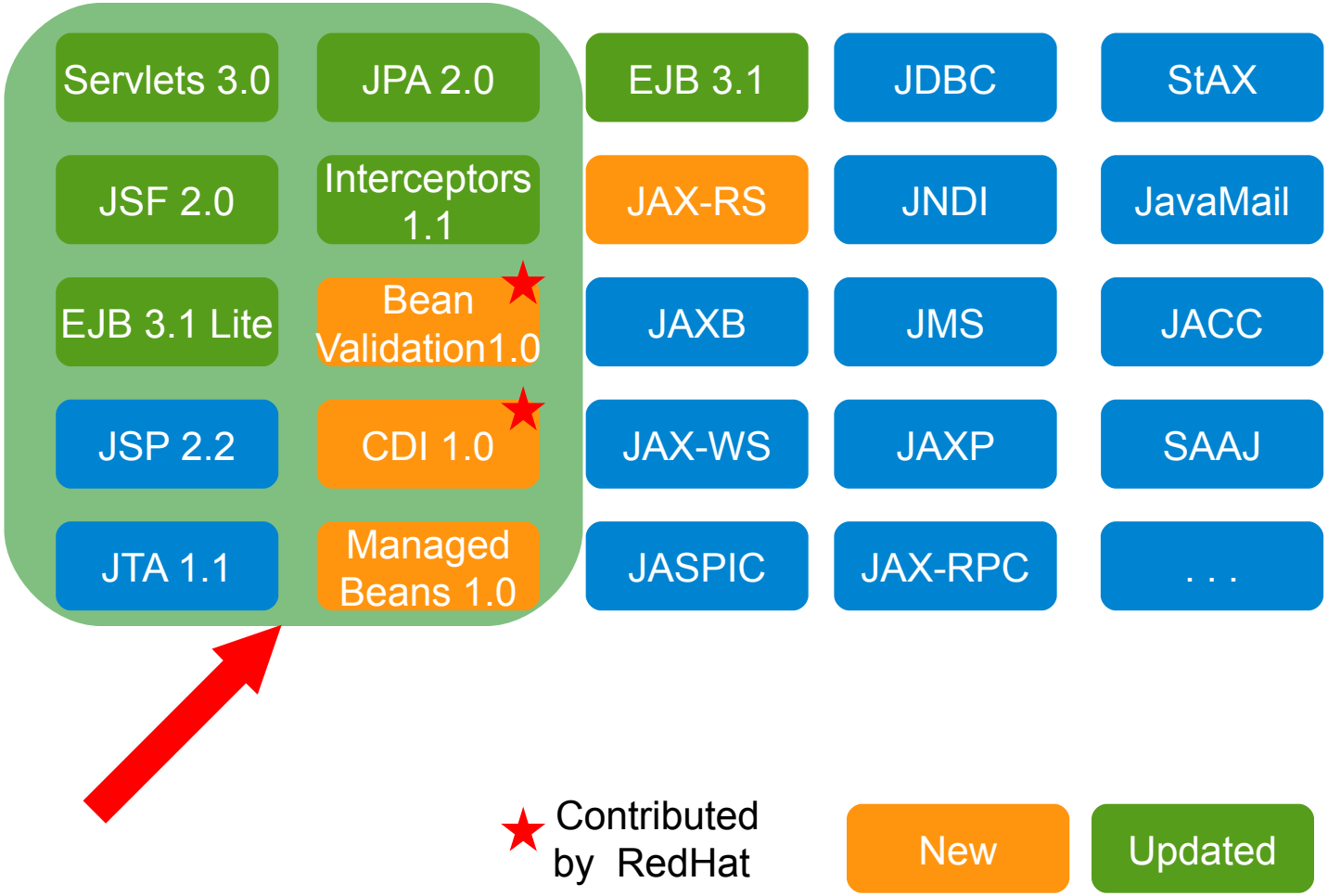
Java EE 6 Specifications
Updates

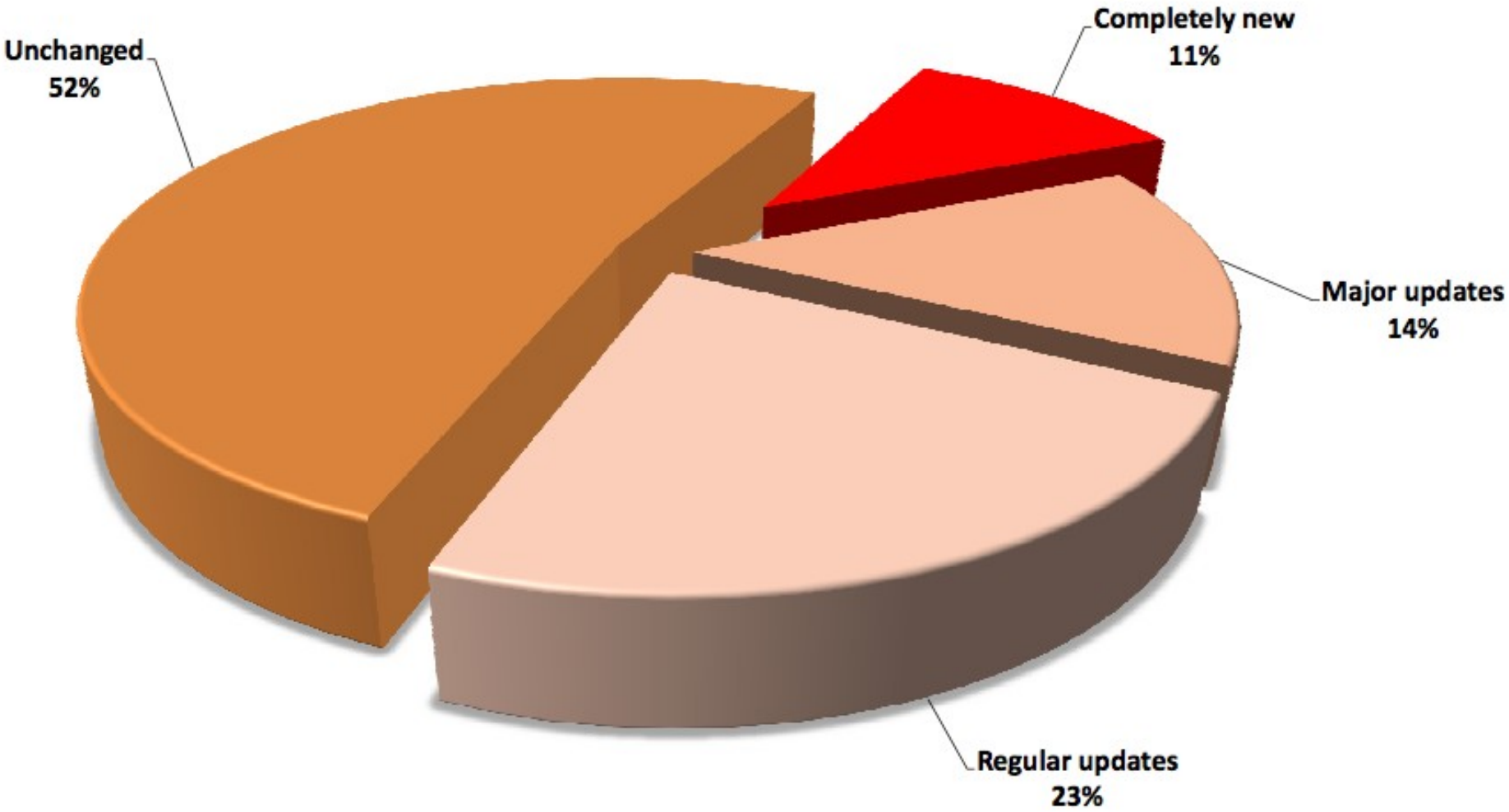
- Java API for XML-based Web Services 2.2 (JSR 224)
- Java API for XML Binding 2.2 (JSR 222)
- Web Services Metadata MR3 (JSR 181)
- JSP 2.2/EL 2.2 (JSR 245)
- Web Services for Java EE 1.3 (JSR 109)
- Common Annotations 1.1 (JSR 250)
- Java Authorization Contract for Containers 1.3 (JSR 115)
- Java Authentication Service Provider Interface for Containers 1.0 (JSR 196)

Java EE 6 Specifications
As is

- JDBC 4.0 API
- Java Naming and Directory Interface 1.2
- Java Message Service 1.1
- Java Transaction API 1.1
- Java Transaction Service 1.0
- JavaMail API Specification 1.4
- JavaBeans Activation Framework 1.1
- Java API for XML Processing 1.3
- Java API for XML-based RPC 1.1
- SOAP with Attachments API for Java 1.3
- Java API for XML Registries 1.0
- Java EE Management Specification 1.1 (JSR 77)
- Java EE Deployment Specification 1.2 (JSR 88)
- Java Management Extensions 1.2
- Java Authentication and Authorization Service 1.0
- Debugging Support for Other Languages (JSR 45)
- Standard Tag Library for JSP 1.2 (JSR 52)
- Streaming API for XML 1.0 (JSR 173)

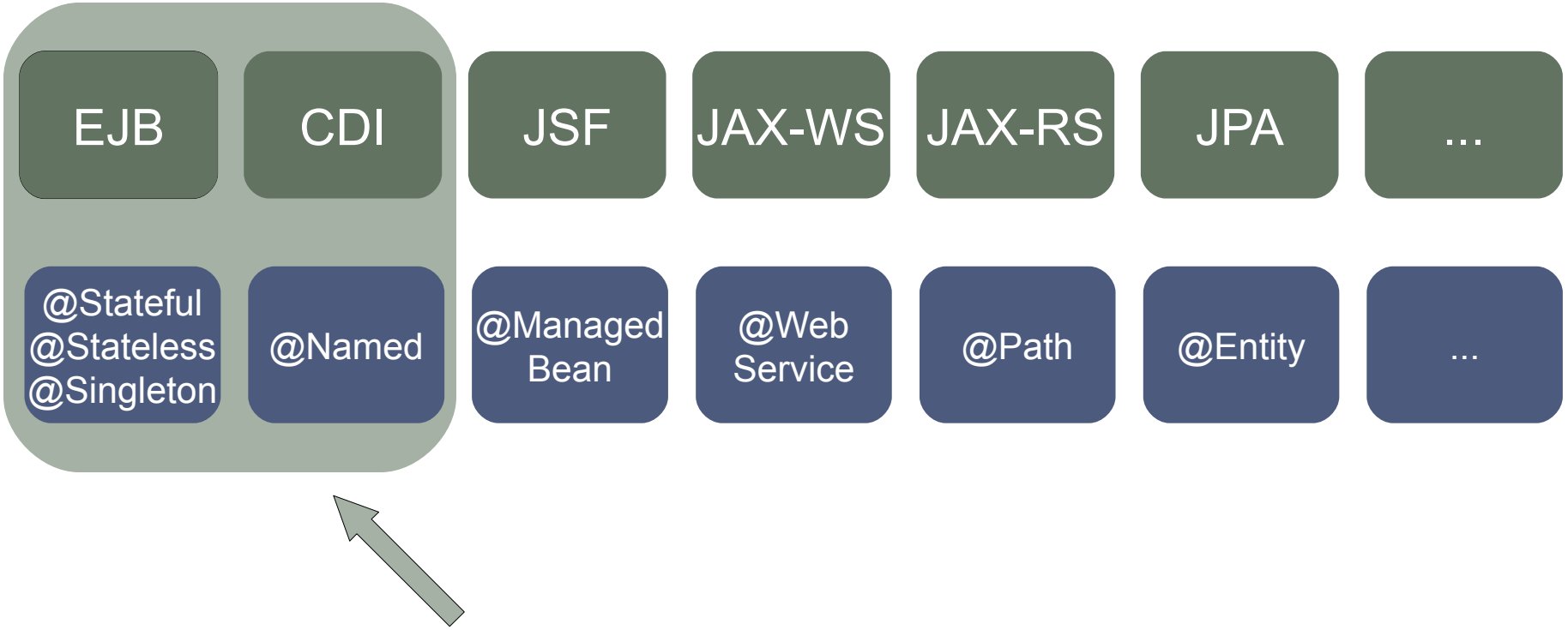
Java EE 6 Web Profile 1.0





http://blog.eisele.net/2010/12/who-is-afraid-of-java-ee-6-get-rid-of.html

Managed Beans 1.0



@javax.annotation.ManagedBean

Managed Beans 1.0

- POJO as managed component for the Java EE container
  - JavaBeans component model for Java EE
  - Simple and Universally useful
  - Advanced concepts in companion specs
- Basic Services
  - Resource Injection, Lifecycle Callbacks, Interceptors
- Available as
  - @Resource / @Inject
  - java:app/<module-name>/<bean-name>
  - java:module/<bean-name>

Demo #1
Managed Beans

Managed Beans 1.0 - Sample

```
@javax.annotation.ManagedBean
public class MyManagedBean {
  @PostConstruct
  public void setupResources() {
    // setup your resources
  }

  @PreDestroy
  public void cleanupResources() {
     // collect them back here
  }

  public String sayHello(String name) {
    return "Hello " + name;
  }
}
```

```
@Resource
MyManagedBean bean;

@Inject
MyManagedBean bean;
```

http://blogs.sun.com/arungupta/entry/totd_129_managed_beans_1

Interceptors 1.1

- Interpose on invocations and lifecycle events on a target class
- Defined
  - Using annotations or DD
  - Default Interceptors (only in DD)
- Class & Method Interceptors
  - In the same transaction & security context
- Cross-cutting concerns: logging, auditing, profiling

Demo #2
Interceptors

Interceptors – Business Method (Logging)

```
@InterceptorBinding
@Retention(RUNTIME)
@Target({METHOD,TYPE})
public @interface
LoggingInterceptorBinding {
}
```

```
@LoggingInterceptorBinding
public class MyManagedBean {
  . . .
}
```

```
@Interceptor
@LoggingInterceptorBinding
public class @LogInterceptor {
  @AroundInvoke
  public Object log(InvocationContext context) {
     System.out.println(context.getMethod().getName());
     System.out.println(context.getParameters());
     return context.proceed();
  }
}
```

Why Interceptor Bindings ?

- Remove dependency from the interceptor implementation class
- Can vary depending upon deployment environment
- Allows central ordering of interceptors

Interceptors – Business Method (Transaction)

```
@InterceptorBinding
@Retention(RUNTIME)
@Target({METHOD,TYPE})
public @interface Transactional {
}
```

```
@Transactional
public class ShoppingCart { . . . }
```

```
public class ShoppingCart {
  @Transactional public void
checkOut() { . . . }
```

```
@Interceptor
@Transactional
public class TransactionInterceptor {
  @Resource UserTransaction tx;
  @AroundInvoke
  public Object manageTransaction(InvocationContext context) {
    tx.begin()
    context.proceed();
    tx.commit();
  }
}
```

http://blogs.sun.com/arungupta/entry/totd_151_transactional_interceptors_using

Servlets in Java EE 5

At least 2 files

```
<!--Deployment descriptor
   web.xml -->
<web-app>
   <servlet>
     <servlet-name>MyServlet
             </servlet-name>
       <servlet-class>
         com.sun.MyServlet
       </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet
      </servlet-name>
    <url-pattern>/myApp/*
      </url-pattern>
  </servlet-mapping>
    ...
</web-app>
```

```
/* Code in Java Class */

package com.sun;
public class MyServlet extends
HttpServlet {
public void
doGet(HttpServletRequest
req,HttpServletResponse res)
{
…
}
…
}
```

Servlets 3.0 (JSR 315)

Annotations-based @WebServlet

```
package com.sun;
@WebServlet(name="MyServlet", urlPatterns={"/myApp/*"})
public class MyServlet extends HttpServlet {
      public void doGet(HttpServletRequest req,
                        HttpServletResponse res)
   {
            ...
   }
```

```
<!--Deployment descriptor web.xml -->
<web-app>
   <servlet>
     <servlet-name>MyServlet</servlet-name>
     <servlet-class>
       com.sun.MyServlet
     </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/myApp/*</url-pattern>
  </servlet-mapping>
   ...
</web-app>
```

http://blogs.sun.com/arungupta/entry/screencast_37_java_ee_6

Demo #3
Servlets

Servlets 3.0

Annotations-based @WebServlet

```
@WebServlet(name="mytest",
      urlPatterns={"/myurl"},
      initParams={
        @InitParam(name="n1", value="v1"),
        @InitParam(name="n2", value="v2")
      }
)
public class TestServlet extends
       javax.servlet.http.HttpServlet {
         ....
}
```

Servlets 3.0

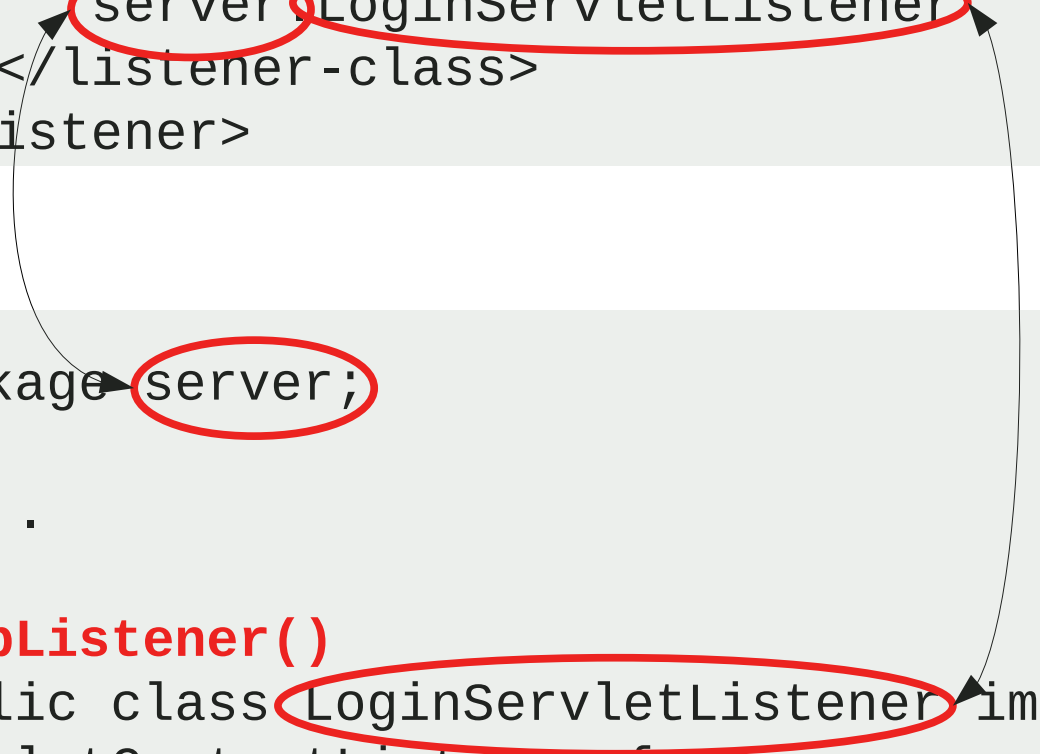
Annotations-based @WebListeners

```
<listener>
   <listener-class>
      server.LoginServletListener
   </listener-class>
</listener>
```

```
package server;

. . .

@WebListener()
public class LoginServletListener implements
ServletContextListener {
```



Servlets 3.0
Asynchronous Servlets

- Useful for Comet, long waits
- Must declare @WebServlet(asyncSupported=true)

```
AsyncContext context = request.startAsync();
context.addListener(new AsyncListener() { … });
context.dispatch(“/request.jsp”);
//context.start(Runnable action);
. . .
context.complete();
```

http://blogs.sun.com/arungupta/entry/totd_139_asynchronous_request_processing

Servlets 3.0
Extensibility

- Plugin libraries using *web fragments*
  - Modular web.xml
- Bundled in framework JAR file in META-INF directory
- Zero-configuration, drag-and-drop for web frameworks
  - Servlets, servlet filters, context listeners for a framework get discovered and registered by the container
- Only JAR files in WEB-INF/lib are used

Servlets 3.0
Extensibility


```
<web-fragment>
    <filter>
        <filter-name>wicket.helloworld</filter-name>
        <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
        <init-param>
            <param-name>applicationClassName</param-name>
            <param-value>...</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>wicket.helloworld</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-fragment>
```

http://blogs.sun.com/arungupta/entry/totd_91_applying_java_ee

Servlets 3.0
Extensibility - ServletContainerInitializer

- Provided by the apps or container
- Discovered using the service provider API
- Expresses interest in classes via @HandlesTypes
- Who uses it ?
  - Mojarra (JSF2) is bootstrapped into GlassFish
    - No “faces-config.xml” or “web.xml”
  - Jersey (JAX-RS) registers root Application
    - No (or portable) “web.xml”

Servlets 3.0
Dynamic Registration – Java Server Faces

```
@SuppressWarnings({"UnusedDeclaration"})
@HandlesTypes({
      ManagedBean.class,
      FacesComponent.class,
      FacesValidator.class,
      FacesConverter.class,
      FacesBehaviorRenderer.class,
      ResourceDependency.class,
      ResourceDependencies.class,
      ListenerFor.class,
      ListenersFor.class,
      UIComponent.class,
      Validator.class,
      Converter.class,
      Renderer.class

})
public class FacesInitializer implements ServletContainerInitializer {

    // NOTE: Loggins should not be used with this class.

    private static final String FACES_SERVLET_CLASS =
FacesServlet.class.getName();
```

Servlets 3.0
Dynamic Registration – Java Server Faces

```
    public void onStartup(Set<Class<?>> classes, ServletContext servletContext)
          throws ServletException {

        if (shouldCheckMappings(classes, servletContext)) {

            Map<String,? extends ServletRegistration> existing =
servletContext.getServletRegistrations();
            for (ServletRegistration registration : existing.values()) {
                if (FACES_SERVLET_CLASS.equals(registration.getClassName())) {
                    // FacesServlet has already been defined, so we're
                    // not going to add additional mappings;
                    return;
                }
            }
            ServletRegistration reg =
                  servletContext.addServlet("FacesServlet",
                                            "javax.faces.webapp.FacesServlet");
            reg.addMapping("/faces/*", "*.jsf", "*.faces");
            servletContext.setAttribute(RIConstants.FACES_INITIALIZER_MAPPINGS_ADDED,
Boolean.TRUE);
```

Servlets 3.0
Dynamic Registration – Create, Register, Lookup

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

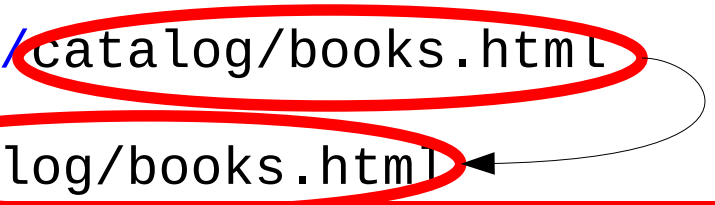
```
ServletRegistration dynamic = servletContext.
     getServletRegistration(“DynamicServlet”);
dynamic.addMapping(“/dynamicServlet”);
dynamic.setInitParameter(“param”, “value”);
```

Servlets 3.0
Resource Sharing

- Static and JSP no longer confined to document root of the web application
- May be placed in **`WEB-INF/lib/[*.jar]/META-INF/resources`**
- Resources in document root take precedence over those in bundled JAR

```
myapp.war
  WEB-INF/lib/catalog.jar
                /META-INF/resources/catalog/books.html

http://localhost:8080/myapp/catalog/books.html
```



Servlets 3.0
Much more ...

- Programmatic authentication, login, logout
  - `HttpServletRequest.authenticate`
  - `HttpServletRequest.login`
  - `HttpServletRequest.logout`
- File upload support
- Servlet Security
  - @ServletSecurity
- Default Error Page
  - Any HTTP status code
  - http://blogs.sun.com/arungupta/entry/totd_136_default_error_page

```
<error-page>
    <error-code>...</error-code>
    <exception-type>...</exception-type>
    <location>/404.html</location>
</error-page>
```

Java Persistence API 2 (JSR 317)
Sophisticated mapping/modeling options

- Collection of basic types

```
@Entity
public class Person {
    @Id protected String ssn;
    protected String name;
    protected Date birthDate;
    . . .
    @ElementCollection
    @CollectionTable(name="ALIAS")
    protected Set<String> nickNames;
}
```

Java Persistence API 2
Sophisticated mapping/modeling options

- Collection of embeddables

```
@Embeddable public class Address {
    String street;
    String city;
    String state;
    . . .
}

@Entity public class RichPerson extends Person {
    . . .
    @ElementCollection
    protected Set<Address> vacationHomes;
    . . .
}
```

Java Persistence API 2
Sophisticated mapping/modeling options

- Multiple levels of embedding

```
@Embeddable public class ContactInfo {
    @Embedded Address address;
    . . .
}

@Entity public class Employee {
    @Id int empId;
    String name;
    ContactInfo contactInfo;
    . . .
}
```

Java Persistence API 2
Sophisticated mapping/modeling options

- Improved Map support

```
@Entity public class VideoStore {
    @Id Integer storeId;
    Address location;
    . . .
    @ElementCollection
    Map<Movie, Integer> inventory;
}

@Entity public class Movie {
    @Id String title;
    @String director;
    . . .
}
```

Java Persistence API 2
Expanded JPQL

- Support for all new modeling/mapping features
- Operators and functions in select list
- INDEX, KEY/VALUE ENTRY

// Inventory is Map<Movie, Integer>

```
SELECT v.location.street, KEY(i).title, VALUE(i),
FROM VideoStore v JOIN v.inventory i
WHERE KEY(i).director LIKE '%Hitchcock%'
    AND VALUE(i) > 0
```

Java Persistence API 2
Expaded JPQL – CASE Expression

```
UPDATE Employee e
SET e.salary =
    CASE e.rating
        WHEN 1 THEN e.salary * 1.05
        WHEN 2 THEN e.salary * 1.02
        ELSE e.salary * 0.95
    END
```

Java Persistence API 2
Metamodel

- Abstract “schema-level” model over managed classes of a Persistence Context
  - Entities, Mapped classes, Embeddables, ...
- Accessed dynamically
  - EntityManager or EntityManagerFactory.getMetamodel()
- And/or statically materialized as metamodel classes
  - Use annotation processor with javac

Demo #4
Java Persistence API

Java Persistence API 2

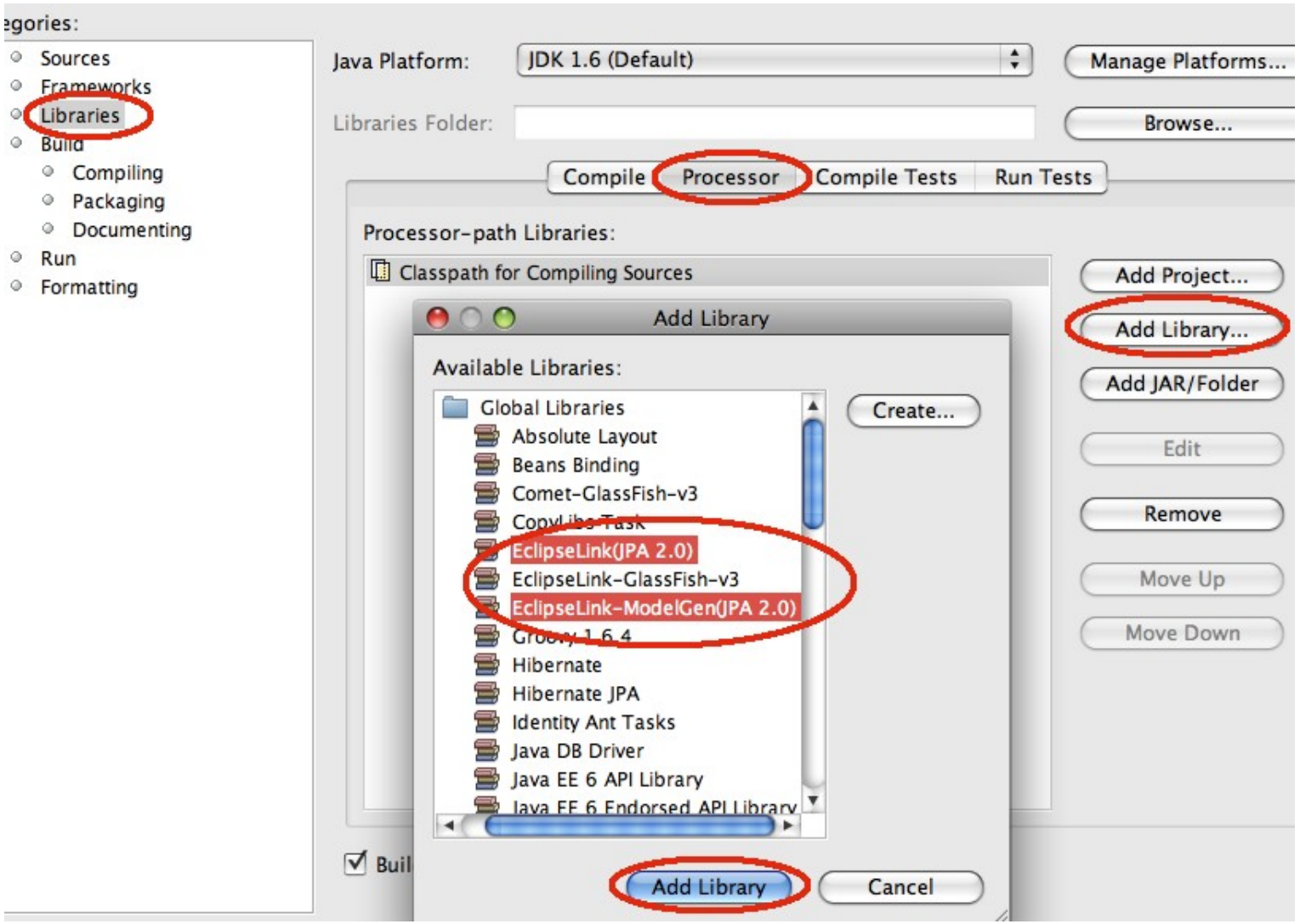
Metamodel Example

```
@Entity
public class Customer {
  @Id Integer custId;
  String name;
  ...
  Address address;
  @ManyToOne SalesRep rep;
  @OneToMany Set<Order> orders;
}
```

```
import javax.persistence.metamodel.*;

@StaticMetamodel(Customer.class)
public class Customer_ {
  public static SingularAttribute<Customer, Integer> custId;
  public static SingularAttribute<Customer, String> name;
  public static SingularAttribute<Customer, Address> address;
  public static SingularAttribute<Customer, SalesRep> rep;
  public static SetAttribute<Customer, Order> orders;
}
```

Canonical Metamodel using NetBeans



http://blogs.sun.com/arungupta/entry/totd_148_jpa2_metamodel_classes

Java Persistence API 2
Type-safe Criteria API

```
EntityManager em = …;
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<ResultType> cquery =
    cb.createQuery(ResultType.class);
Root<MyEntity> e = cquery.from(MyEntity.class);
Join<MyEntity, RelatedEntity> j = e.join(…);
…
cquery.select(…)
      .where(…)
      .orderBy(…)
      .groupBy(…);

TypedQuery<ResultType> tq = em.createQuery(cquery);
List<ResultType> result = tq.getResultList();
```

Java Persistence API 2
Caching

- 1st-level Cache by PersistenceContext
  - Only one object instance for any database row
- 2nd-level by “shared-cache-mode”
  - ALL, NONE
  - UNSPECIFIED – Provider specific defaults
  - ENABE_SELECTIVE - Only entities with Cacheable(true)
  - DISABLE_SELECTIVE - All but with Cacheable(false)
  - Optional feature for PersistenceProvider
- Properties for find, refresh, setProperty
  - CacheRetrieveMode – USE, BYPASS

Java Persistence API 2
Much more ...

- New locking modes
  - PESSIMISTIC_READ – grab shared lock
  - PESSIMISTIC_WRITE – grab exclusive lock
  - PESSIMISTIC_FORCE_INCREMENT – update version
  - `em.find(<entity>.class, id, LockModeType.XXX)`
  - `em.lock(<entity>, LockModeType.XXX)`
- Standard configuration options
  - javax.persistence.jdbc.[driver | url | user | password]

JPA 2.1 Candidate Features

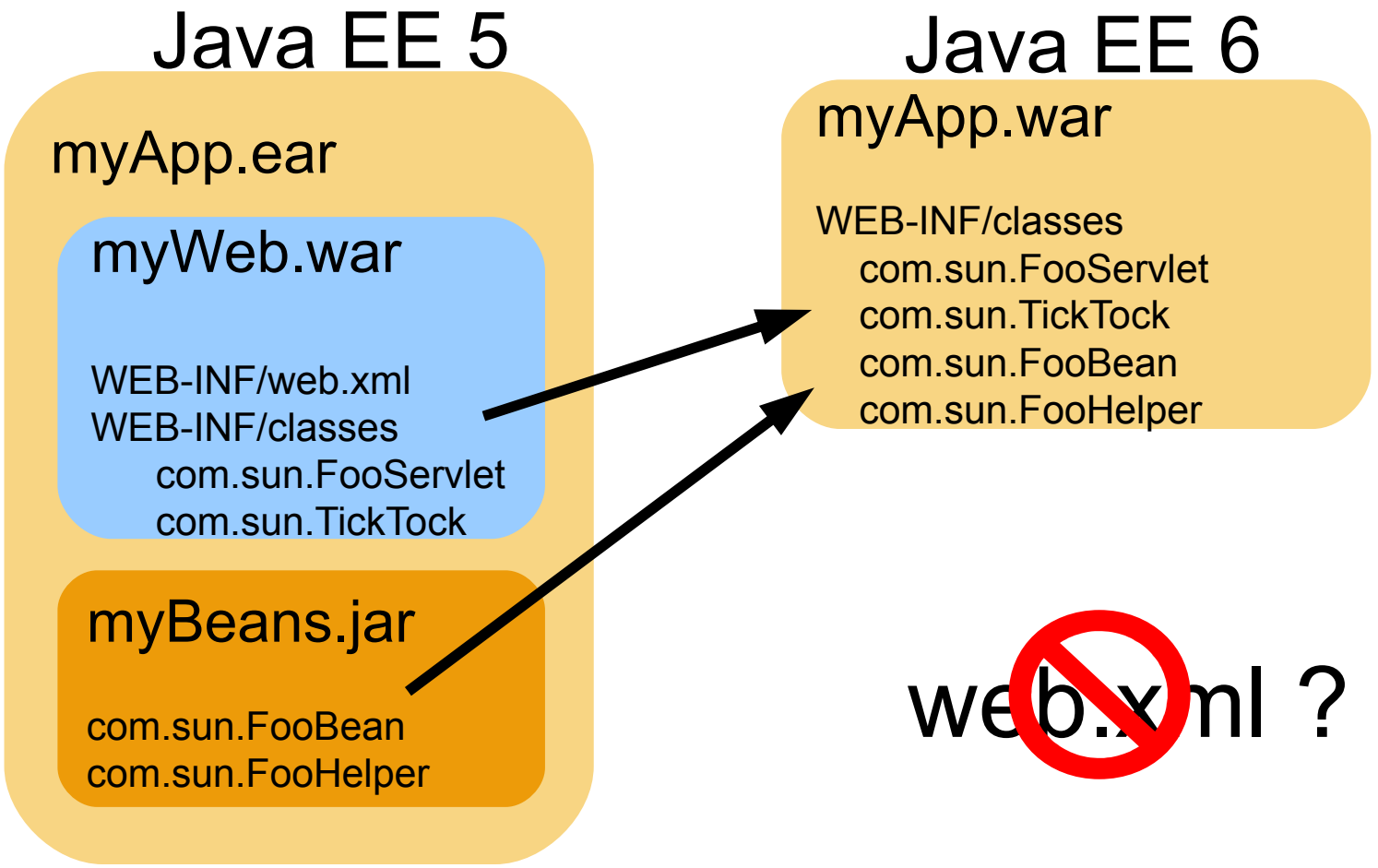
http://jcp.org/en/jsr/detail?id=338

- Converters/Custom types
- Fetch groups/plans
- Immutable attributes and read-only entities
- Support for schema generation
- Generated values for non-PKs
- Multi-tenancy
- UUID generator type
- Persistence Context synchronization control
- Dynamic definition of PU
- Additional event listeners

JPA 2.1 Candidate Features

- Support for stored procedures
- Database and vendor function invocation
- Update and Delete Criteria queries
- Support for mapping between JPQL and Criteria queries
- Improved support for result type mapping of native queries
- Downcasting
- Query by Example

EJB 3.1 (JSR 318)
Package & Deploy in a WAR



http://blogs.sun.com/arungupta/entry/screencast_37_java_ee_6

Demo #5

EJB 3.1

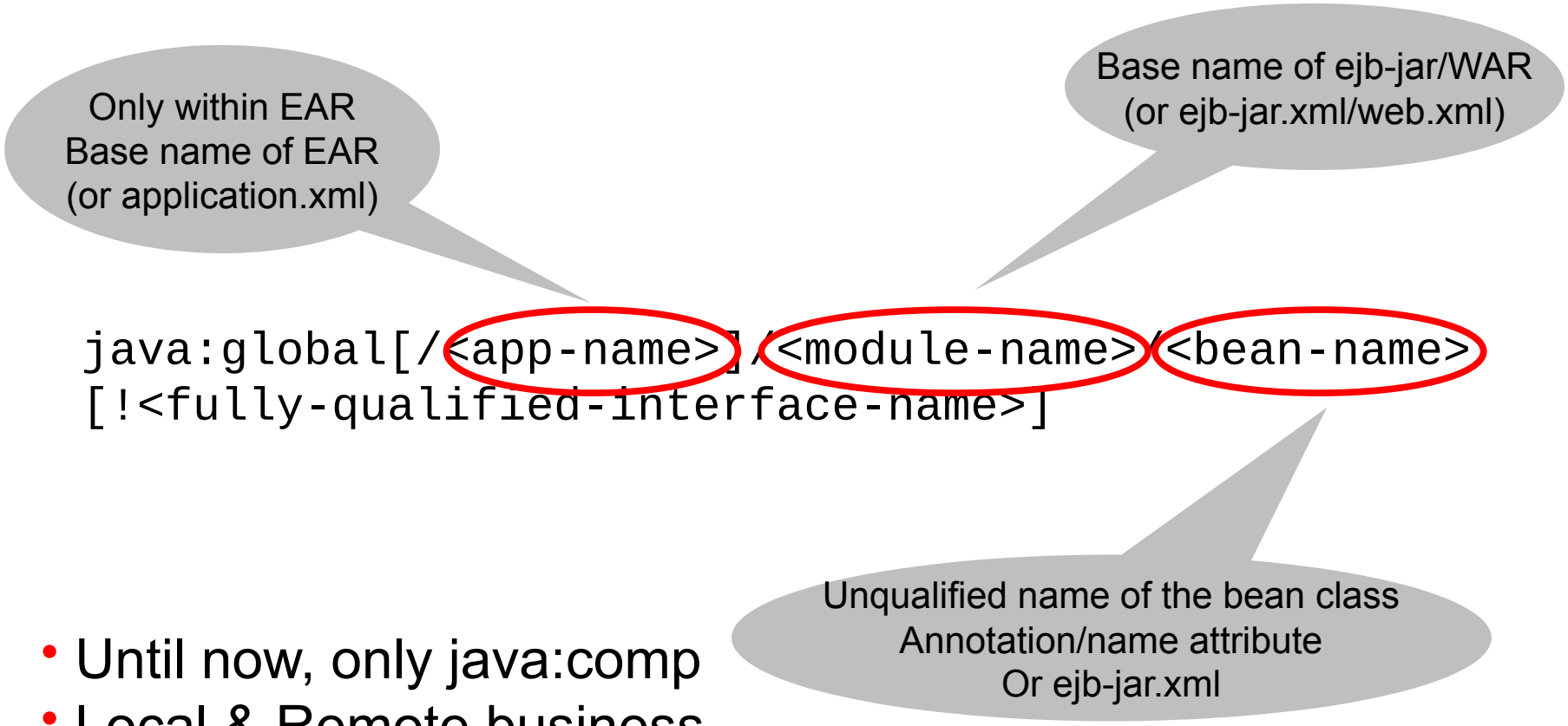
EJB 3.1

- No interface view – one source file per bean
  - Only for Local and within WAR
  - Required for Remote
  - No location transparency
- Component initialization in @PostConstruct
  - No assumptions on no-arg ctor

EJB 3.1
Portable Global JNDI Name Syntax

- Portable
- Global
- Application/Module-scoped
- Derived from metadata such as name, component name etc.

EJB 3.1
Portable Global JNDI Name Syntax



- Until now, only java:comp
- Local & Remote business
- No-interface
- Also in java:app, java:module

EJB 3.1
Embeddable API – Deploy the Bean

```
public void testEJB() throws NamingException {
        EJBContainer ejbC =
             EJBContainer.createEJBContainer();
        Context ctx = ejbC.getContext();
        App app = (App)
             ctx.lookup("java:global/classes/App");
        assertNotNull(app);
        String NAME = "Duke";
        String greeting = app.sayHello(NAME);
        assertNotNull(greeting);
        assertTrue(greeting.equals("Hello " + NAME));
        ejbC.close();
    }
```

http://blogs.sun.com/arungupta/entry/totd_128_ejbcontainer_createejbcontainer_embedded

EJB 3.1
Singleton Beans

- One instance per app/VM, not pooled
  - Useful for caching state
  - CMT/BMT
  - Access to container services for injection, resource manager, timers, startup/shutdown callbacks, etc.
  - Enable eager initialization using @Startup
  - Define initialization ordering using @Depends

```
@Singleton
public class MyEJB {
  . . .
}
```

EJB 3.1
Asynchronous Session Beans

- Light-weight JMS
- Control returns to the client before the container dispatches the invocation to a bean instance
- @Asynchronous – method or class
- Return type – void or Future<V>
- Transaction context does not propagate
  - REQUIRED → REQUIRED_NEW
- Security principal propagates

EJB 3.1

Asynchronous Session Beans – Code Sample

```
@Stateless
@Asynchronous
public class SimpleAsyncEJB {
    public Future<Integer> addNumbers(int n1, int n2) {
        Integer result;

        result = n1 + n2;
        try {
            // simulate JPA queries + reading file system
            Thread.currentThread().sleep(2000);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }

        return new AsyncResult(result);
    }
}
```

http://blogs.sun.com/arungupta/entry/totd_137_asynchronous_ejb_a

EJB 3.1
Timers

- Automatically created EJB Timers
- Calendar-based Timers – cron like semantics
  - Every Mon & Wed midnight
    `@Schedule(dayOfWeek="Mon,Wed")`
  - 2pm on Last Thur of Nov of every year
    `(hour="14", dayOfMonth="Last Thu", month="Nov")`
  - Every 5 minutes of every hour
    `(minute="*/5", hour="*")`
  - Every 10 seconds starting at 30
    `(second="30/10")`
  - Every 14th minute within the hour, for the hours 1 and 2 am
    `(minute="*/14", hour="1,2")`

EJB 3.1
Timers

- Can be chained
  - ```
    @Schedules({
      @Schedule(hour="6",dayOfWeek="Tue,Thu,Fri-Sun"),
       @Schedule(hour="12",dayOfWeek="Mon,Wed")
    })
    ```
- May be associated with a TimeZone
- Single persistent timer across JVMs
- Non-persistent timer, e.g. Cache
  - `@Schedule(..., persistent=false)`
    ```

EJB 3.1
EJB 3.1 Lite

A proper subset of the full EJB 3.1 API that includes a small, powerful selection of EJB features suitable for writing portable transactional business logic

. . .

suitable for inclusion in a wider range of Java products, many of which have much smaller installation and runtime footprints than a typical full Java EE implementation

EJB 3.1

EJB 3.1 Lite – Feature Comparison

TABLE 2. EJB LITE AND EJB 3.1 FEATURES COMPARISON

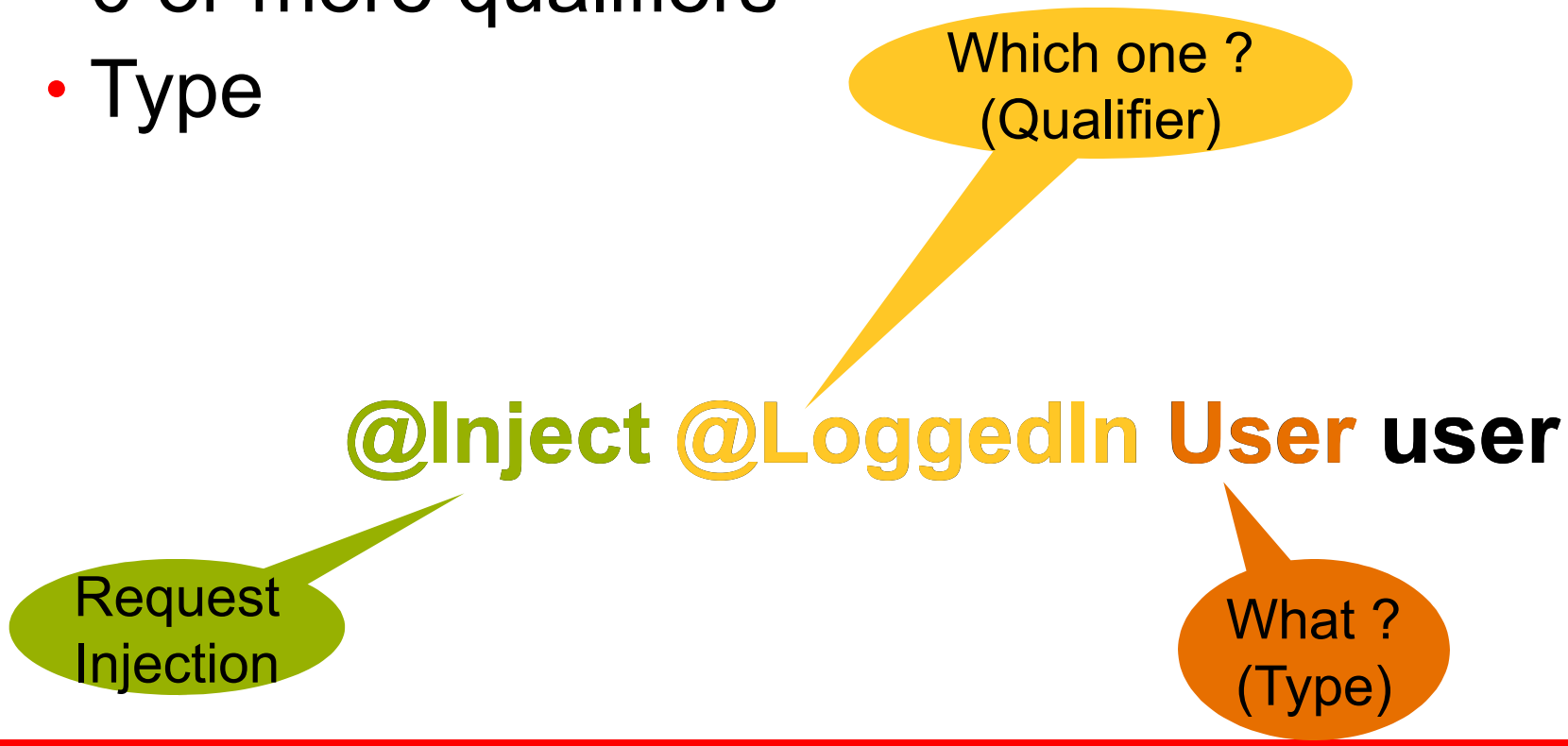
FEATURE	EJB LITE	EJB 3.1
Local session beans	X	X
Declarative security	X	X
Transactions (CMT/BMT)	X	X
Interceptors	X	X
Security	X	X
Message-driven beans		X
RMI/IIOP interoperability and remote view		X
EJB 2.x backward compatibility		X
Time service		X
Asynchronous method invocations		X

Contexts & Dependency Injection - CDI (JSR 299)

- Type-safe Dependency Injection
  - No String-based identifiers
- Strong typing, Loose coupling
  - Events, Interceptors, Decorators
- Context & Scope management – extensible
- Portable Extensions
- Bridge EJB and JSF in the platform
- Works with Java EE modular and component architecture
  - Integration with Unified Expression Language (UEL)

CDI
Injection Points

- Field, Method, Constructor
- 0 or more qualifiers
- Type



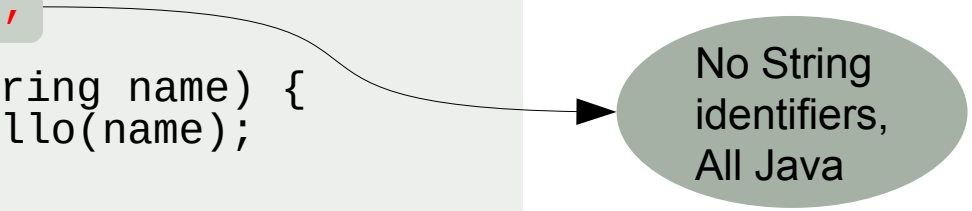
Basic – Sample Code

```
public interface Greeting {
    public String sayHello(String name);
}
```

```
public class HelloGreeting implements Greeting {
    public String sayHello(String name) {
        return “Hello “ + name;
    }
}
```

```
@Stateless
public class GreetingService {
    @Inject Greeting greeting;

    public String sayHello(String name) {
        return greeting.sayHello(name);
    }
}
```



CDI – Sample Client Code
Field and Method Injection

```
public class CheckoutHandler {

  @Inject @LoggedIn User user;

  @Inject PaymentProcessor processor;

  @Inject void setShoppingCart(@Default Cart cart) {
     …
  }

}
```

CDI – Sample Client Code
Constructor Injection

```
public class CheckoutHandler {

  @Inject
  CheckoutHandler(@LoggedIn User user,
                  PaymentProcessor processor,
                  @Default Cart cart) {
    ...
  }

}
```

- Only one constructor can have @Inject

CDI - Sample Client Code

Multiple Qualifiers and Qualifiers with Arguments

```
public class CheckoutHandler {

  @Inject
  CheckoutHandler(@LoggedIn User user,
                  @Reliable
                  @PayBy(CREDIT_CARD)
                  PaymentProcessor processor,
                  @Default Cart cart) {
    ...
  }

}
```

CDI - How to configure ?

There is none!

- Discovers bean in all modules in which CDI is enabled
- “beans.xml”
  - WEB-INF of WAR
  - META-INF of JAR
  - META-INF of directory in the classpath
- Can enable groups of bean selectively via a descriptor

Demo #6
CDI Qualifiers

CDI - Scopes

- Beans can be declared in a scope
  - Everywhere: @ApplicationScoped, @RequestScoped
  - Web app: @SessionScoped
  - JSF app: @ConversationScoped
    - Transient and long-running
  - Pseudo-scope (default): @Dependent
  - Custom scopes via @Scope
- CDI runtime makes sure the right bean is created at the right time
- Client do NOT have to be scope-aware

CDI - Named Beans

Built-in support for the Unified EL

- Beans give themselves a name with `@Named("cart")`
- Then refer to it from a JSF or JSP page using the EL:

```
<h:commandButton
        value="Checkout"
        action="#{cart.checkout}"/>
```

Events – More decoupling

- Annotation-based event model
  - Based upon “Observer” pattern
- A “producer” bean fires an event
- An “observer” bean watches an event
- Events can have qualifiers
- Transactional event observers
  - IN_PROGRESS, AFTER_SUCCESS, AFTER_FAILURE, AFTER_COMPLETION, BEFORE_COMPLETION

Events – Sample Code

```
@Inject @Any Event<PrintEvent> myEvent;

void print() {
   . . .
   myEvent.fire(new PrintEvent(5));
}
```

```
void onPrint(@Observes PrintEvent event){…}
```

```
public class PrintEvent {
   public PrintEvent(int pages) {
      this.pages = pages;
   }
   . . .
}
```

```
void addProduct(@Observes(during = AFTER_SUCCESS) @Created
Product product)
```

CDI
Much more ...

- Producer methods and fields
- Bridging Java EE resources
- Alternatives
- Interceptors
- Decorators
- Stereotypes

Java Server Faces 2.0 (JSR 314)

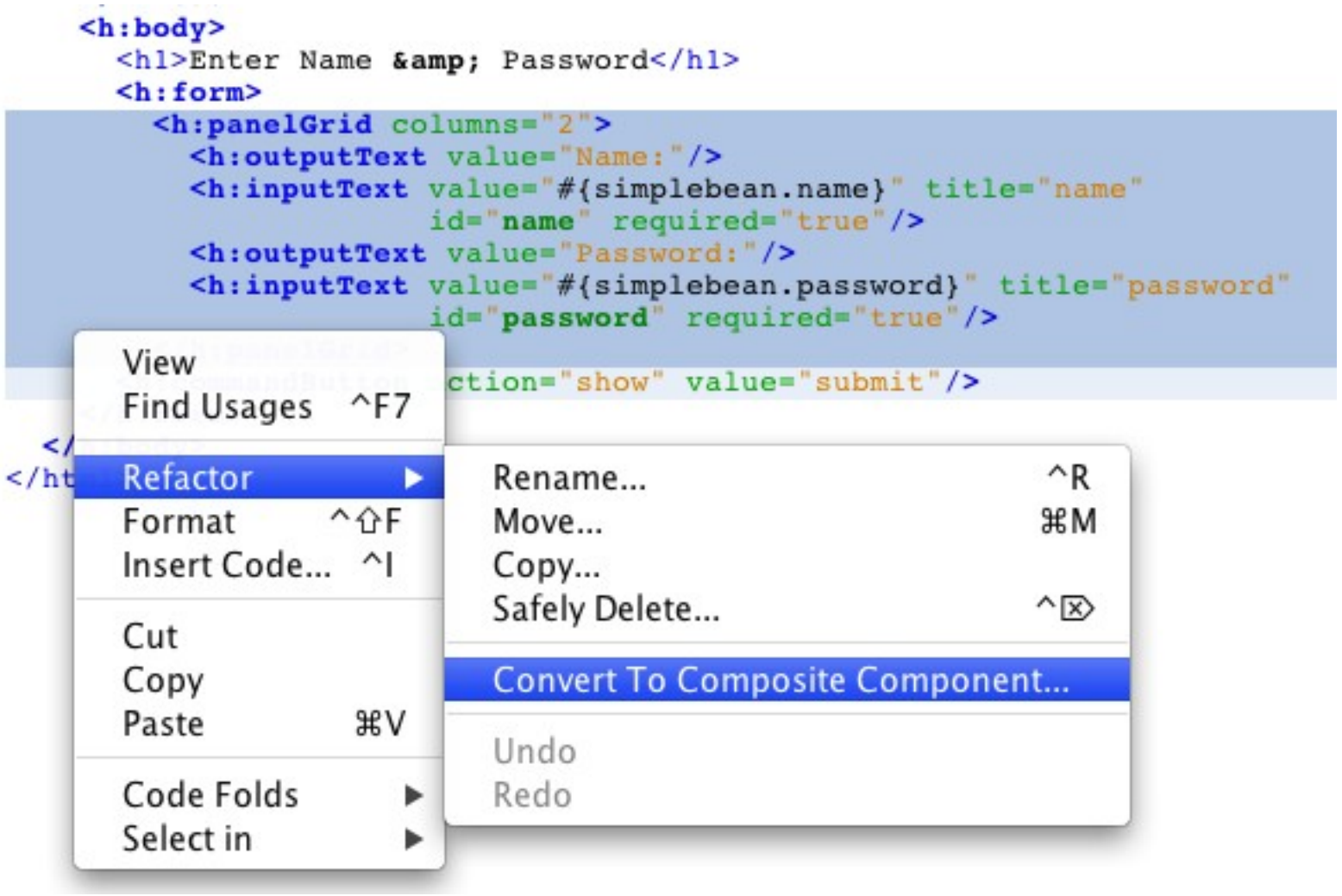
- Facelets as “templating language” for the page
  - Custom components much easier to develop

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Enter Name &amp; Password</title>
  </h:head>
  <h:body>
    <h1>Enter Name &amp; Password</h1>
    <h:form>
      <h:panelGrid columns="2">
        <h:outputText value="Name:"/>
        <h:inputText value="#{simplebean.name}" title="name"
                     id="name" required="true"/>
        <h:outputText value="Password:"/>
        <h:inputText value="#{simplebean.password}" title="password"
                     id="password" required="true"/>
      </h:panelGrid>
      <h:commandButton action="show" value="submit"/>
    </h:form>
  </h:body>
</html>
```

Demo #7

JSF 2.0

JSF 2 Composite Components



JSF 2 Composite Components

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ez="http://java.sun.com/jsf/composite/ezcomp">
  <h:head>
    <title>Enter Name &amp; Password</title>
  </h:head>
  <h:body>
    <h1>Enter Name &amp; Password</h1>
    <h:form>
      <ez:username-password/>
      <h:commandButton action="show" value="submit"/>
    </h:form>
  </h:body>
</html>
```

```
. . .
WEB-INF
index.xhtml
resources/
  ezcomp/
    username-password.xhtml
```

http://blogs.sun.com/arungupta/entry/totd_147_java_server_faces

Java Server Faces 2.0
Integrated Ajax Support

- f:ajax

```
<h:commandButton
      actionListener="#{sakilabean.findActors}"
      value="submit">
      <f:ajax execute="length"
          render="actorTable totalActors"/>
    </h:commandButton>
```

http://blogs.sun.com/arungupta/entry/totd_123_f_ajax_bean

Java Server Faces 2.0

- “faces-config.xml” optional in common cases
  - <managed-bean> → @ManagedBean or @Named
  - Validator, Renderer, Listener, ...
  - Default navigation rules – match a view on the disk

```
@Named(“simplebean”)
public class SimpleBean {
. . .
}
<h:commandButton action="show" value="submit"/>
```

Java Server Faces 2.0
Much more ...

- Runs on Servlet 2.5+
- Conditional navigation
- Project Stages
  - Development, UnitTest, SystemTest, Production
- Custom Scopes for Managed Beans
- Bookmarkable URLs
  - h:link, h:button

```
<navigation-case>
  <from-outcome>success</from-outcome>
  <to-view-id>/page2.xhtml</to-view-id>
  <!-- Only accept if the following condition
is true -->
  <if>#{foo.someCondition}</if>
</navigation-case>
```

```
<h:link outcome="viewEntry" value="Link">
   <f:param name="entry" value="#{aBean.entry}"/>
</h:link>

<a href="http://localhost:8080/myapp/viewEntry.xhtml?
entry=entry1">Link</a>
```

Bean Validation (JSR 303)

- Tier-independent mechanism to define constraints for data validation
  - Represented by annotations
  - javax.validation.* package
- Integrated with JSF and JPA
  - JSF: f:validateRequired, f:validateRegexp
  - JPA: pre-persist, pre-update, and pre-remove
- @NotNull(message="..."), @Max, @Min, @Size
- Fully Extensible
  - @Email String recipient;

Bean Validation
Integration with JPA

- Managed classes may be configured
  - Entities, Mapped superclasses, Embeddable classes
- Applied during pre-persist, pre-update, pre-remove lifecycle events
- How to enable ?
  - “validation-mode” in persistence.xml
  - “javax.persistence.validation.mode” key in Persistence.createEntityManagerFactory
- Specific set of classes can be targeted
  - javax.persistence.validation.group.pre-[persist|update|remove]

Bean Validation
Integration with JSF

- Individual validators not required
- Integration with EL
  - f:validateBean, f:validateRequired

```
<h:form>
  <f:validateBean>
    <h:inputText value="#{model.property}" />
    <h:selectOneRadio value="#{model.radioProperty}" > …
    </h:selectOneRadio>
    <!-- other input components here -->
  </f:validateBean>
</h:form>
```

Demo #8
Bean Validation

JAX-RS 1.1

- Java API for building RESTful Web Services
- POJO based
- Annotation-driven
- Server-side API
- HTTP-centric

JAX-RS 1.1
Code Sample - Simple

```
@Path("helloworld")
public class HelloWorldResource {
    @Context UriInfo ui;

    @GET
    @Produces("text/plain")
    public String sayHello() {
        return "Hello World";
    }

    @GET
    @Path("morning")
    public String morning() {
        return “Good Morning!”;
    }
}
```

JAX-RS 1.1

Code Sample – Specifying Output MIME type

```
@Path("/helloworld")
@Produces("text/plain")
public class HelloWorldResource {
    @GET
    public String doGetAsPlainText() {
      . . .
    }

    @GET
    @Produces("text/html")
    public String doGetAsHtml() {
      . . .
    }
}
```

```
@GET
@Produces({
  "application/xml",
  "application/json"})
public String doGetAsXmlOrJson() {
  . . .
}
```

Demo #9

JAX-RS

JAX-RS 1.1

Code Sample – Specifying Input MIME type

```
@POST
@Consumes("text/plain")
public String saveMessage() {
    . . .
}
```

JAX-RS 1.1
Code Sample

```
import javax.inject.Inject;
import javax.enterprise.context.RequestScoped;

@RequestScoped
public class ActorResource {
    @Inject DatbaseBean db;

    public Actor getActor(int id) {
        return db.findActorById(id);
    }
}
```

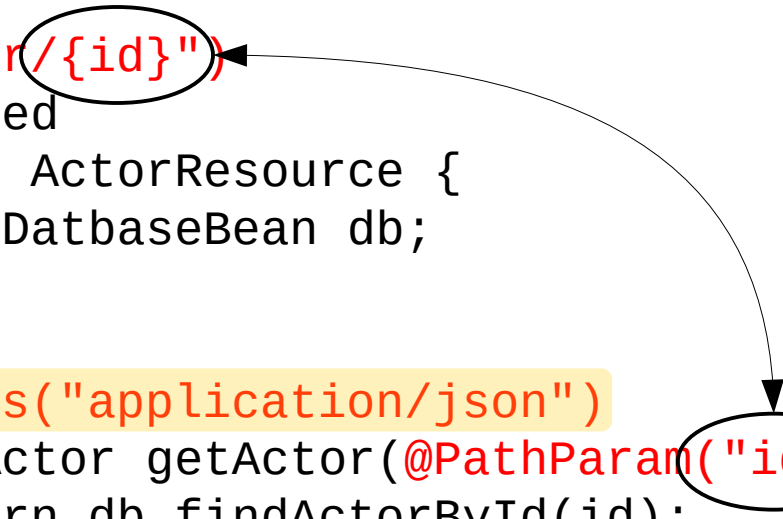
JAX-RS 1.1

Code Sample

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.PathParam;
import javax.inject.Inject;
import javax.enterprise.context.RequestScoped;

@Path("/actor/{id}")
@RequestScoped
public class ActorResource {
    @Inject DatbaseBean db;

    @GET
    @Produces("application/json")
    public Actor getActor(@PathParam("id") int id) {
        return db.findActorById(id);
    }
}
```



http://blogs.sun.com/arungupta/entry/totd_124_using_cdi_jpa

JAX-RS 1.1
Integration with Java EE 6 – Servlets 3.0

- No or Portable “web.xml”

```
<web-app>
  <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>
      com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>com.foo.MyApplication</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/resources/*</url-pattern>
  </servlet-mapping>
</web-app>
```

```
@ApplicationPath(“resources”)
public class MyApplication
    extends
    javax.ws.rs.core.Application {
}
```

JAX-RS 1.1
Jersey Client-side API

- Consume HTTP-based RESTful Services
- Easy-to-use
  - Better than HttpURLConnection!
- Reuses JAX-RS API
  - Resources and URI are first-class citizens
- Not part of JAX-RS yet
  - com.sun.jersey.api.client

JAX-RS 1.1

Jersey Client API – Code Sample

```
Client client = Client.create();

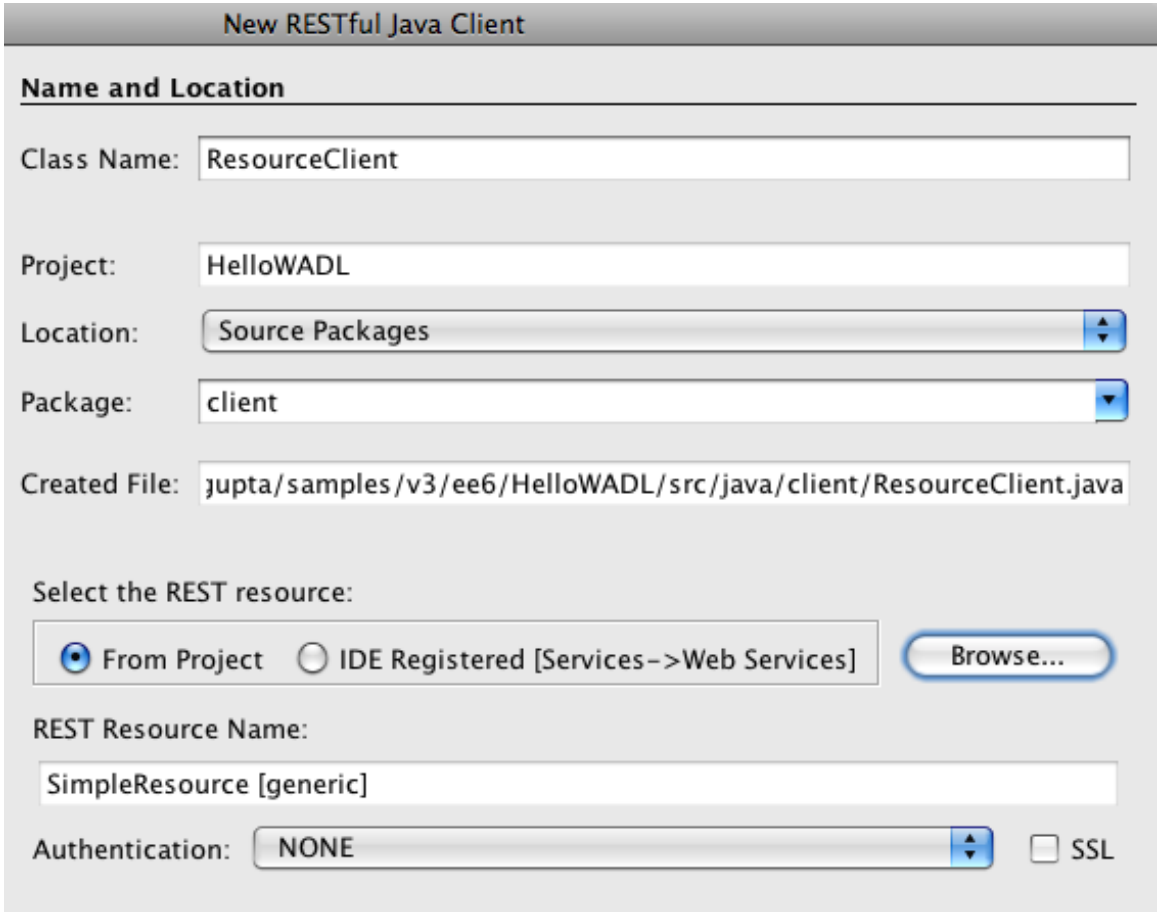
WebResource resource = client.resource(“...”);

//curl  http://example.com/base
String s = resource.get(String.class);

//curl -HAccept:text/plain http://example.com/base
String s = resource.
     accept(“text/plain”).
     get(String.class);
```

http://blogs.sun.com/enterprisetechtips/entry/consuming_restful_web_services_with

JAX-RS 1.1
Jersey Client API – NetBeans Code Generation



JAX-RS 1.1
WADL Representation of Resources

- Machine processable description of HTTP-based Applications
- Generated OOTB for the application

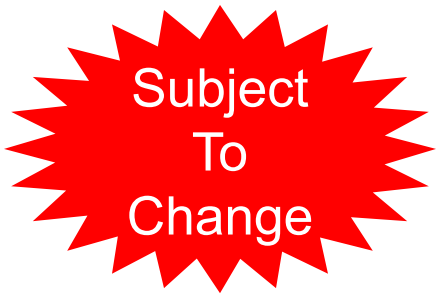
```
<application xmlns="http://research.sun.com/wadl/2006/10">
  <doc xmlns:jersey="http://jersey.dev.java.net/"
       jersey:generatedBy="Jersey: 1.1.4.1 11/24/2009 01:37 AM"/>
  <resources base="http://localhost:8080/HelloWADL/resources/">
    <resource path="generic">
      <method id="getText" name="GET">
        <response>
          <representation mediaType="text/plain"/>
        </response>
      </method>
      <method id="putText" name="PUT">
        <request>
          <representation mediaType="text/plain"/>
        </request>
      </method>
    </resource>
  </resources>
</application>
```

JAX-RS 1.1
Much more ...

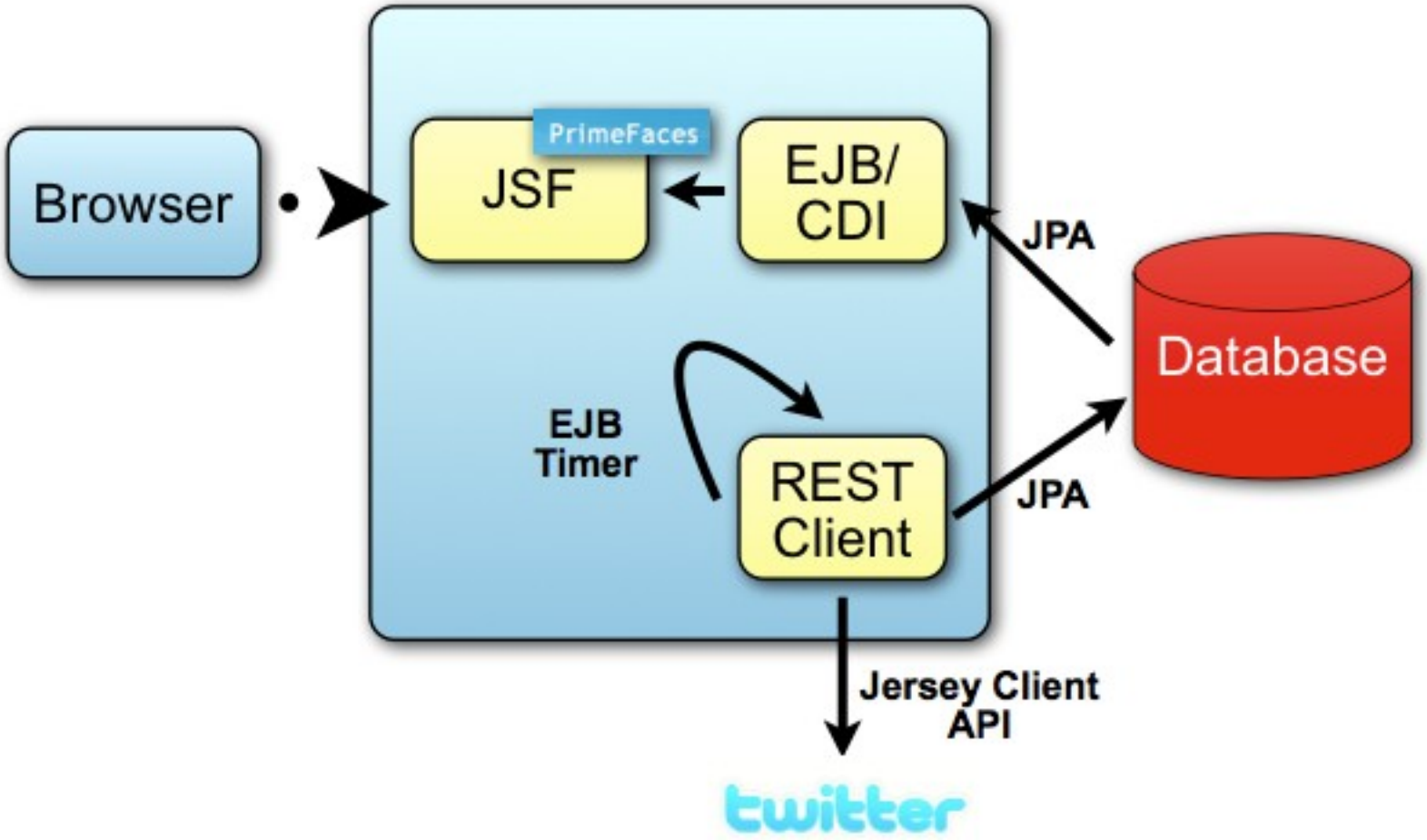
- Jersey is the Reference Implementation, included in GlassFish
  - RESTEasy, Restlet, CXF, Wink
- Hypermedia support (only in Jersey)
- Integration with Spring, Guice, Atom, …
- WADL representation
  - Complete, Per resource
- Jersey 1.2 modules are OSGi compliant

JAX-RS 2.0

http://markmail.org/thread/wgm3hj3rrva3j6jo



- Client API
  - Low level using Builder pattern, Higher-level
- Hypermedia
- MVC Pattern
  - Resource controllers, Pluggable viewing technology
- Bean Validation
  - Form or Query parameter validation
- Closer integration with @Inject, etc.
- Server-side asynchronous request processing
- Server-side content negotiation



http://blogs.sun.com/arungupta/entry/java_ee_6_twitter_demo

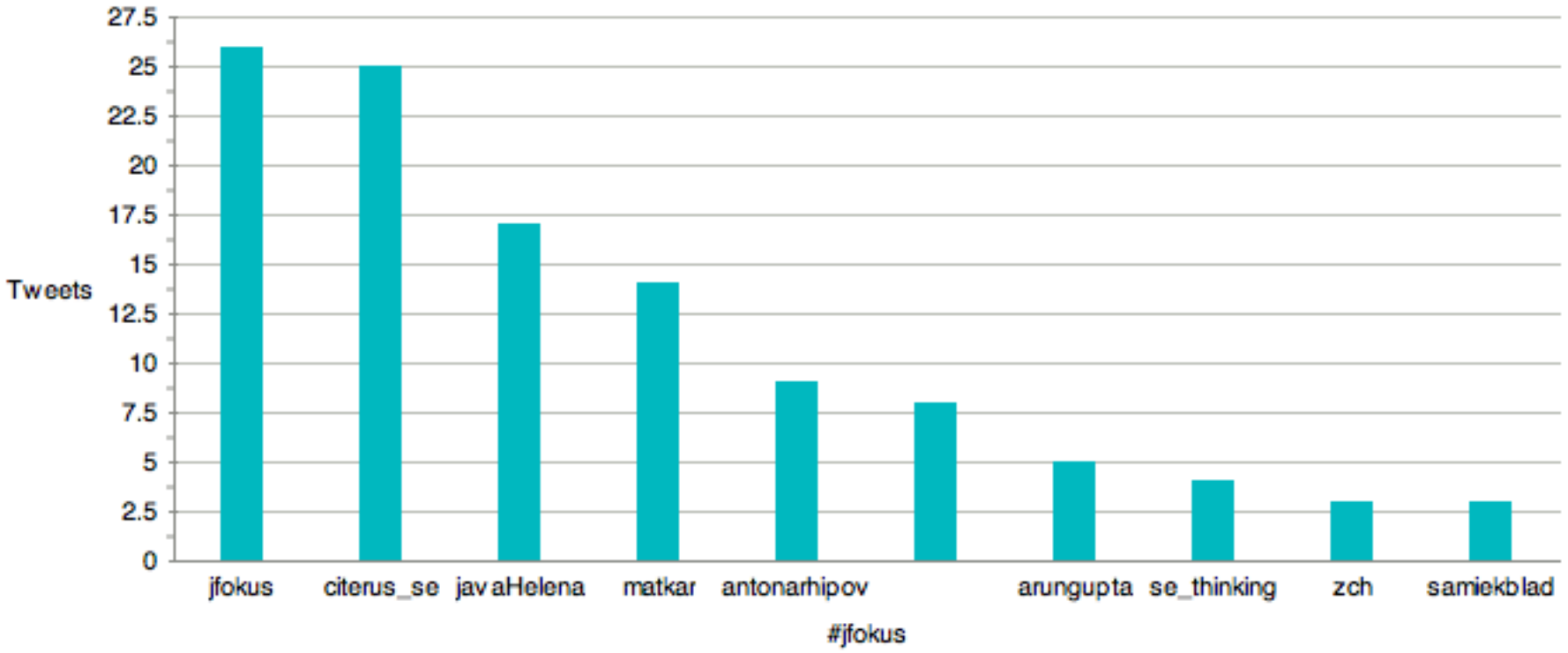
Demo #10

Twitter App

Java EE 6 Demo - Twitter Trends

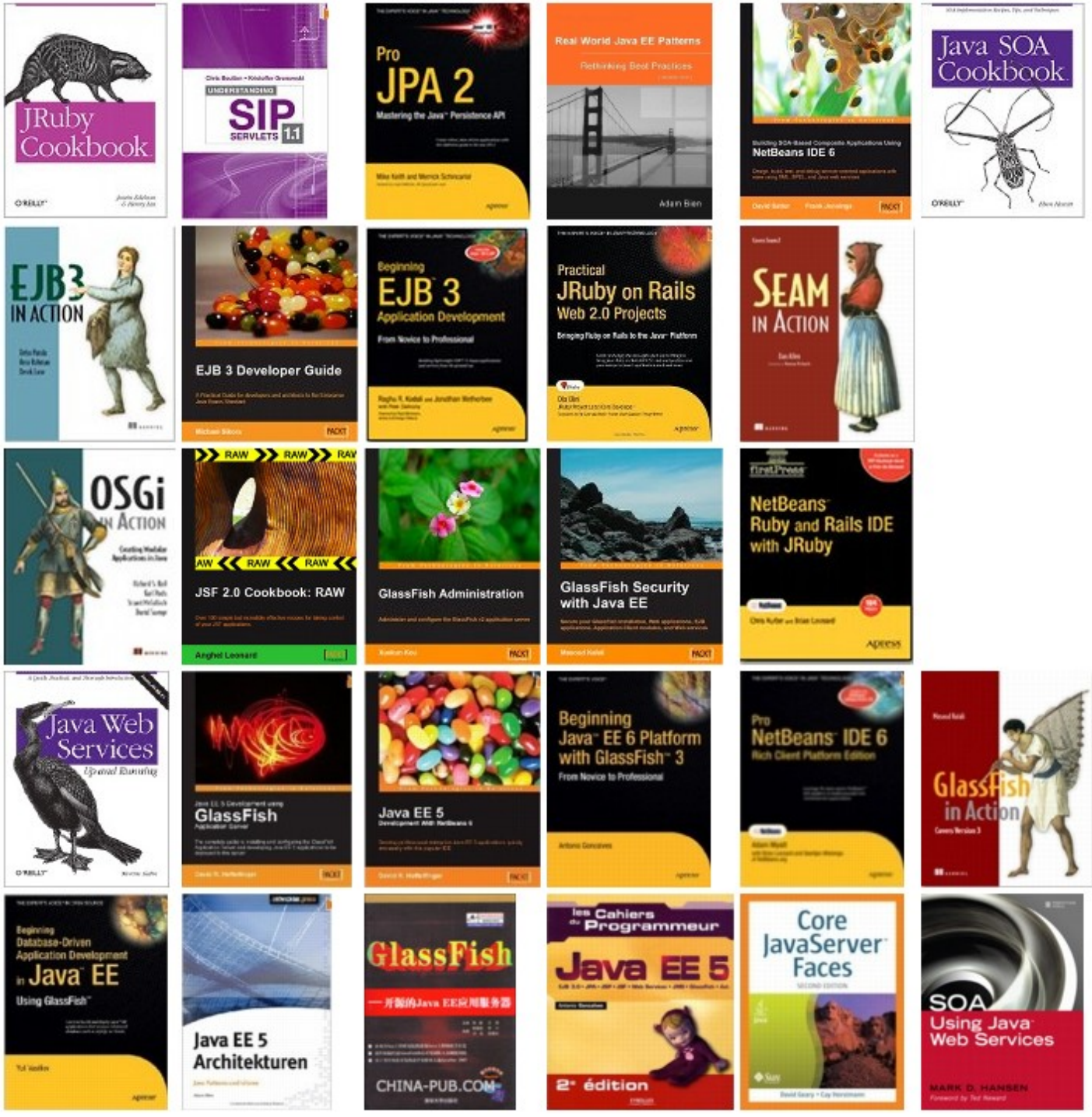
<Trends> <Tags> <Source>

#eclipsecon | #glassfish | #javaee6 | #javaone | #jfokus



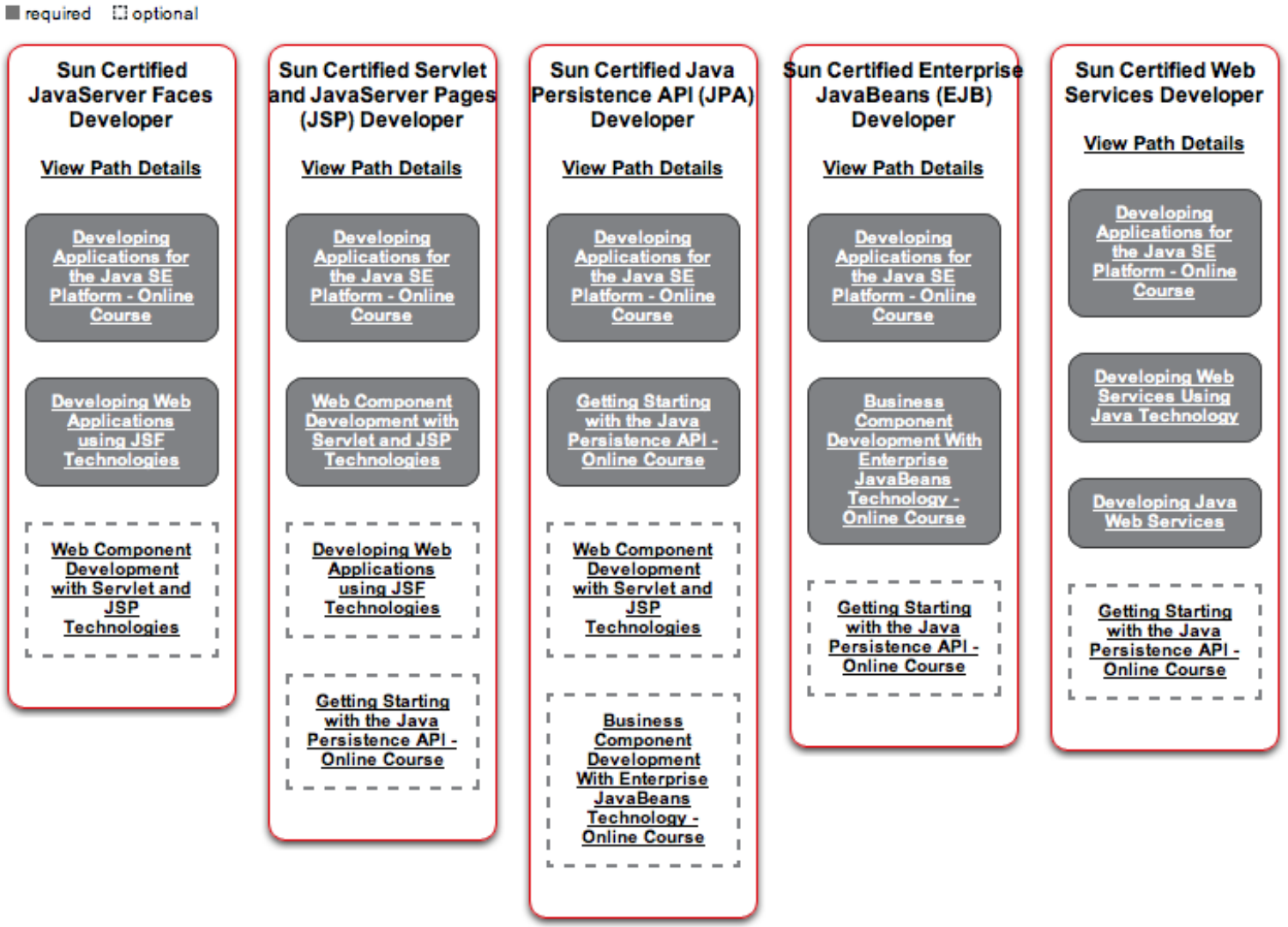
Cannot view these charts ? Try a **JavaScript version**.

IDE Support for Java EE 6


JRuby Cookbook
SIP Servlets 1.1
Pro JPA 2
Mastering the Java Persistence API
Real World Java EE Patterns
Rethinking Best Practices
Adam Bien
NetBeans IDE 6
Java SOA Cookbook
EJB3 IN ACTION
EJB 3 Developer Guide
Beginning EJB 3
Application Development
From Novice to Professional
Practical JRuby on Rails
Web 2.0 Projects
SEAM IN ACTION
OSGi IN ACTION
JSF 2.0 Cookbook: RAW
Anghel Leonard
GlassFish Administration
GlassFish Security with Java EE
NetBeans Ruby and Rails IDE with JRuby
Java Web Services
GlassFish
Java EE 5
Beginning Java EE 6 Platform with GlassFish 3
From Novice to Professional
Pro NetBeans IDE 6
Rich Client Platform Edition
GlassFish in Action
Beginning Database-Driven Application Development in Java EE
Using GlassFish
Java EE 5 Architekturen
GlassFish
CHINA-PUB.COM
Les Cahiers du Programmeur
Java EE 5
2e édition
Core JavaServer Faces
SOA
Using Java Web Services
MARK D. HANSEN

Enterprise Application Development with Java EE6 - Learning Paths

These learning paths provide knowledge and skills for jobs that require expertise in Java Enterprise Edition 6 (Java EE6) platform develo



http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getlppage?page_id=212&path=SADJ

From the real users ...

Developers can concentrate on business logic, Java EE 6 is providing a standard for the infrastructure.

Jigsaw puzzle, Modular, standard, less xml, **easy, easy**, have I said **easy**?

Standards compliance, vendor independence, **milliseconds and kilobyte deployment**

Higher integrated specs, simple and annotation driven, single-classloader WARs, **next level of industry standard**

Faster development, **less frameworks/complexity**, more great code shipped

http://blogs.sun.com/arungupta/tags/community+feedback

Avoid “framework explosion”

In selecting an application server our main goal was to avoid the framework explosion that happens when you use a **"custom" Enterprise stack like Tomcat + Spring + Hibernate + Myfaces** +... Java EE 6 had 80% of what we needed out of the box: strong persistence support (JPA), inversion of control (CDI), and a lightweight component model (EJB 3.1)

http://blogs.sun.com/stories/entry/egesa_engineering_avoids_framework_explosion

What is GlassFish ?

- A community
  - Users, Partners, Testers, Developers, ...
  - Started in 2005 on java.net
- Application Server
  - Open Source (CDDL & GPL v2)
  - Java EE Reference Implementation

GlassFish Server Chronology

2006	2007	2008	2009	2010	…

GlassFish v1

Java EE 5, Single Instance

GlassFish v2

Java EE 5, High Availability

GlassFish Server 3

Java EE 6, Single Instance

GlassFish Server 3.1

Java EE 6, High Availability

GlassFish Server Distributions

Distribution	License	Features
GlassFish Server Open Source Edition 3.1 *Web Profile*	CDDL & GPLv2	• Java EE 6 compatibility • Web Profile support • In-memory replication / clustering • Centralized Administration
GlassFish Open Source Edition 3.1	CDDL & GPLv2	• Java EE 6 compatibility • Full Java EE distribution • In-memory replication / clustering • Centralized Administration

Oracle GlassFish Server 3.1
Web Profile
	Commercial	• Adds
		  • Oracle GlassFish Server Control
		  • Patches, support, knowledge base

Oracle GlassFish Server 3.1	Commercial	• Adds
		  • Oracle GlassFish Server Control
		  • Patches, support, knowledge base

GlassFish 3.1 >= 3.0 + 2.1.1

- Main Features
  - Clustering and Centralized Administration
  - High Availability
- Other ...
  - OSGi/EE RFCs
  - Application Versioning
  - Application-scoped Resources
  - SSH-based remote management and monitoring
  - Embedded (extensive)
  - Admin Console based on RESTful API

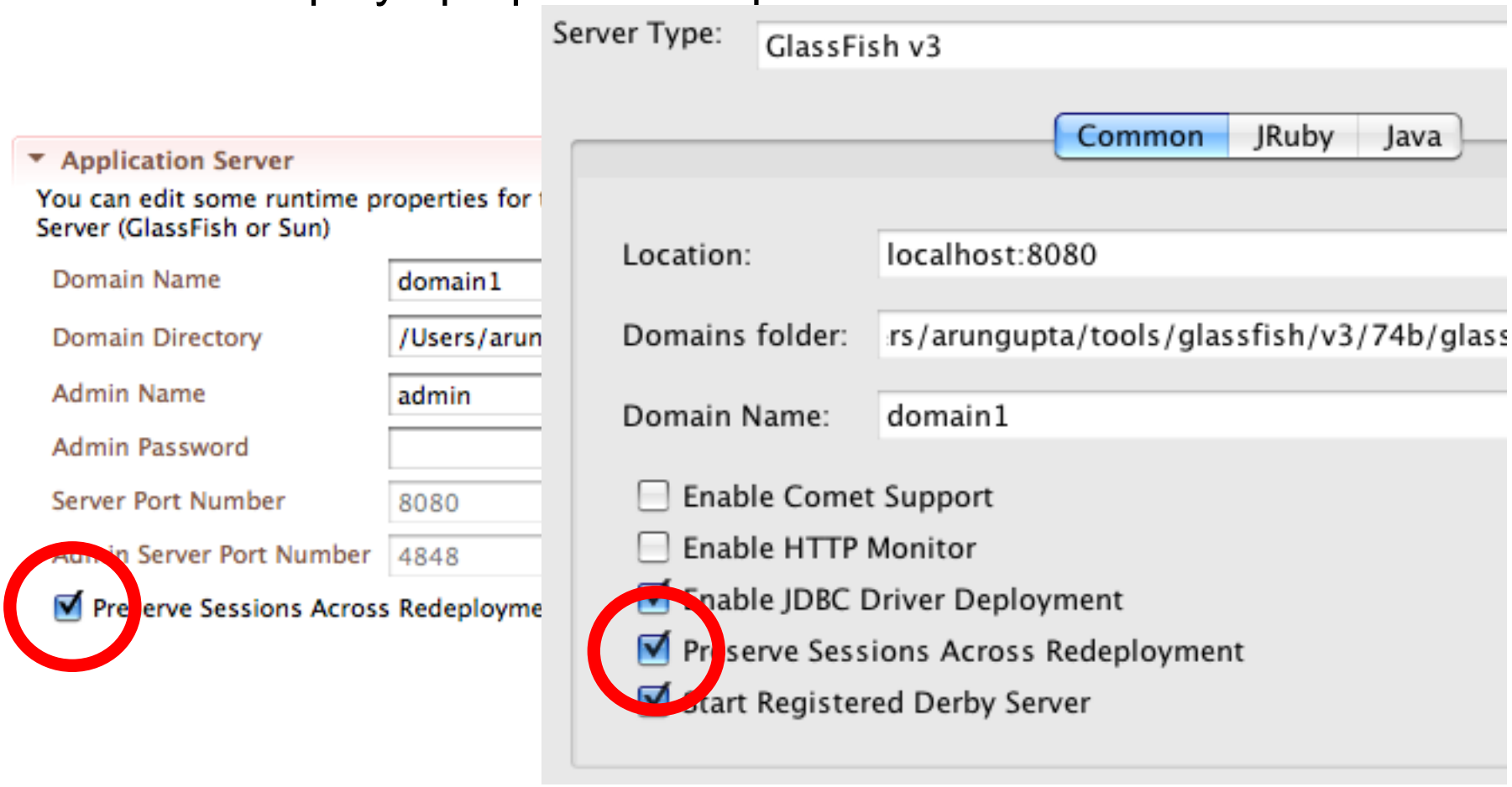
http://blogs.sun.com/arungupta/entry/glassfish_3_1_glassfish_2


Home
Version
User: admin | Domain: domain1 | Server: localhost
GlassFish™ v3 Administration Console
Tree
Common Tasks
Registration
GlassFish News
Enterprise Server
Applications
Lifecycle Modules
Resources
JDBC
Connectors
Resource Adapter Configs
JMS Resources
JavaMail Sessions
JNDI
Configuration
JVM Settings
Logger Settings
Web Container
EJB Container
Ruby Container
Java Message Service
Security
Registration
GlassFish News
Subscriptions
Deployment
List Deployed Applications
Deploy an Application
Quick Start Guide
Administration Guide
Developer's Guide
Application Deployment Guide
Update Center
Installed Components
Available Updates
Available Add-Ons
Other Tasks
Create New JDBC Connection Pool
Run GlassFish Faster
Download the white paper and learn how to optimize GlassFish performance in a production environment. » Get It Now
Get production support.
Stay current with GlassFish news at The Aquarium or learn about GlassFish deployments.
Get community support, Developer Expert Assistance or Expert-to-engineer web training for Java EE 5.
Join Project GlassFish
Learn about upcoming Java EE 6 Release in this webinar.

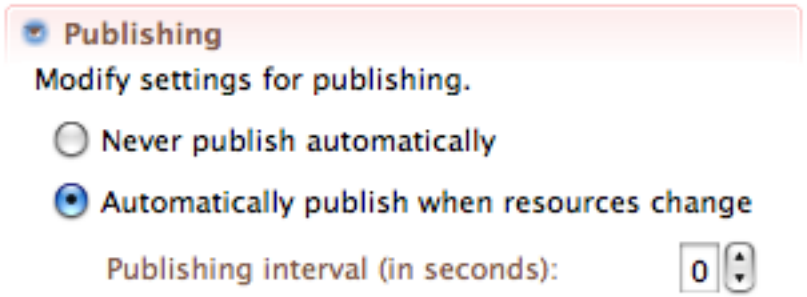
Boost your productivity

Retain session across deployment

asadmin redeploy –properties keepSessions=true helloworld.war



Boost your productivity
Deploy-on-Save



Fusion Middleware Integration Strategy

- Commercial version will have integrations with Fusion Middleware
  - Certification on JRockit
  - Integration with Coherence and TopLink
- Fusion Middleware and Fusion Applications currently not planned to be certified on GlassFish
- Initial integrations will be interoperability
  - Web services, Web services policy, Identity Management (OAM)

What does Java EE offer to Cloud ?

- Containers
- Injectable services
- Scale to large clusters
- Security model
- . . .

What can Java EE do for Clouds ?

- Tighter requirements for resource/state
- Better isolation between applications
- Support for multi-tenant applications
- Potential standard APIs for NRDBMS, Caching, WebSockets, JSON, HTML5
- Common management and monitoring interfaces
- Better packaging
  - Apps/Data are (multiple) versioned, Upgrades, Expose/Connect to services, QoS attributes, ...
- Evolution, not revolution!

References

- glassfish.org
- blogs.sun.com/theaquarium
- oracle.com/goto/glassfish
- youtube.com/user/GlassFishVideos
- Follow @glassfish

ORACLE®

Understanding the Nuts & Bolts of Java EE 6

Arun Gupta, Java EE & GlassFish Guy
blogs.sun.com/arungupta, @arungupta