



ORACLE

Lucky Number 6 For Java EE

Mike Keith
Oracle Corporation

About Me

- Many years experience in distributed and server-side transaction-oriented systems
- Member of Java EE 6 (JSR 316) and various other JCP expert groups
- Java Architect for Oracle
- Member of OSGi Alliance Enterprise Expert Group (EEG)
- Presenter at numerous conferences and events

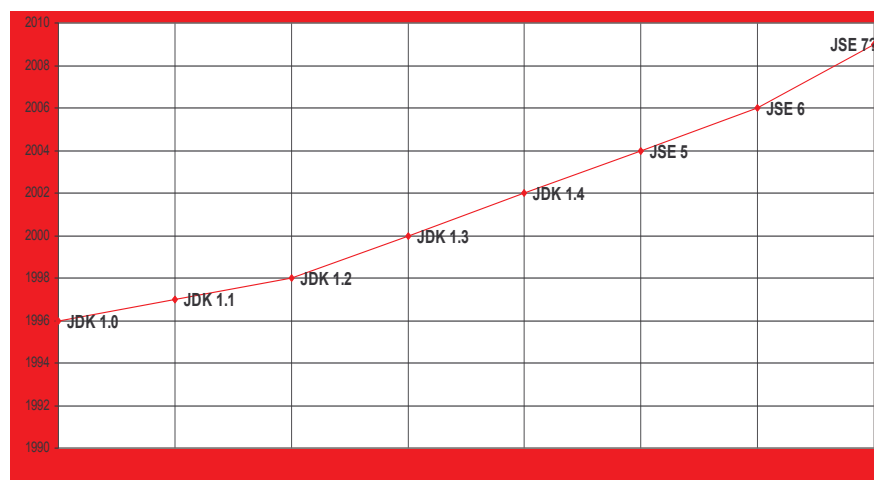
ORACLE

About You

- ❖ Who has already used/knows Java EE 5?
- ❖ How many people have heard about some of the features being introduced in Java EE 6?
- ❖ Who is excited about the new features?
- ❖ Who thinks the Java EE release schedule is too long and should be shortened?
- ❖ Who believes that it doesn't matter anyway because Microsoft is going to kill Java EE?

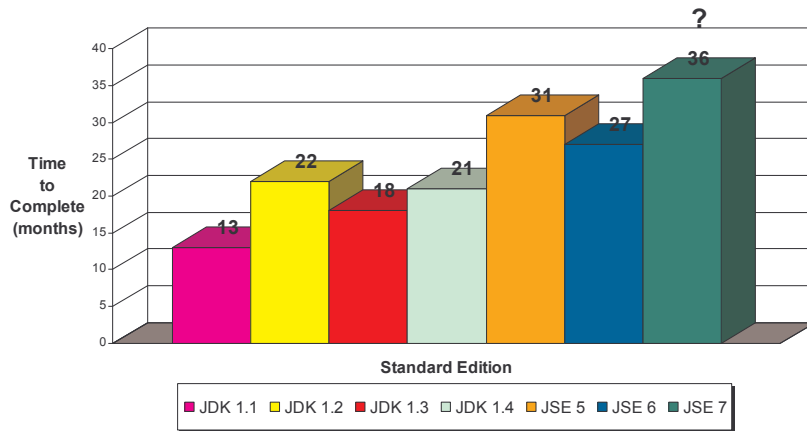
ORACLE

The Java SE Timeline



ORACLE

Java SE Release Cycle Length



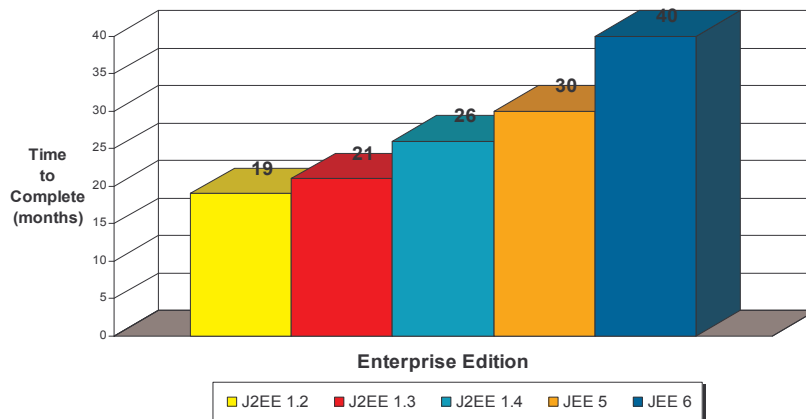
ORACLE

The Java EE Timeline



ORACLE

Java EE Release Cycle Length



ORACLE

Java EE 6 Goals and Highlights

- Add features already in popular use
- Include current technologies
- Make the platform easier to use
- Improve extensibility
- Provide more flexible packaging and delivery options
- Make old technologies optionally supported
- Establish a process for keeping the platform clean

ORACLE

Java EE 6 – Major New Features

- JSF 2.0 (JSR 314)
- Servlet 3.0 (JSR 315)
- EJB 3.1 (JSR 318)
- JPA 2.0 (JSR 317)
- JAX-RS 1.1 (JSR 311)
- Java Connectors 1.6 (JSR 322)
- JCDI 1.0 (Possible) (JSR 299)
- Bean Validation 1.0 (JSR 303)

ORACLE

Java EE 6 – Minor Changes

- JSP 2.2
- EL 1.2
- JAXB 2.2
- JAX-WS 2.2

ORACLE

Java EE 6 – Profiles

- Subset of enterprise technologies
- User can write to a minimal technology set
- Targets a specific application category
- Vendors may implement multiple profiles
- Vendors may augment profile with additional technologies
- Vendors may still augment technologies with additional features

ORACLE

Java EE 6 – Web Profile

- Targeted at web applications
- Presentation layer technologies:
 - Servlet 3.0, JSP 2.2, EL 1.2, JSTL 1.2, JSF 2.0
- Component layer technologies:
 - EJB Lite 3.1, JPA 2.0
- Infrastructure technologies
 - JTA 1.1, Common Annotations
- Possible additions:
 - JAX-RS 1.1, JCDI 1.0

ORACLE

Java EE 6 - Namespaces

- Introduce 3 new JNDI namespaces
 - java:global – shared by all applications in the server
 - java:app – shared by all modules in an application
 - java:module – shared by all components in a module
- May specify any of these when declaring a resource ref
- Some component (e.g. EJBs) refs will be placed in JNDI in each scope

ORACLE

Servlets 3.0 – Annotations

- New annotations to declare components
 - @WebServlet
 - @ServletFilter
 - @WebServletContextListener
- Still need to extend HttpServlet, implement Filter or ServletContextListener
- Search in WEB-INF/classes and all JARs in WEB-INF/lib for annotated classes

ORACLE

Servlets 3.0 – Annotations

- Examples:

```
@WebServlet(name="TestServlet",
            urlPatterns={"/foo", "/bar"})
public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                     HttpServletResponse resp) {...}
}

@ServletFilter("/foo")
public class MyFilter implements Filter {
    public void doFilter(HttpServletRequest req,
                       HttpServletResponse resp) {...}
}
```

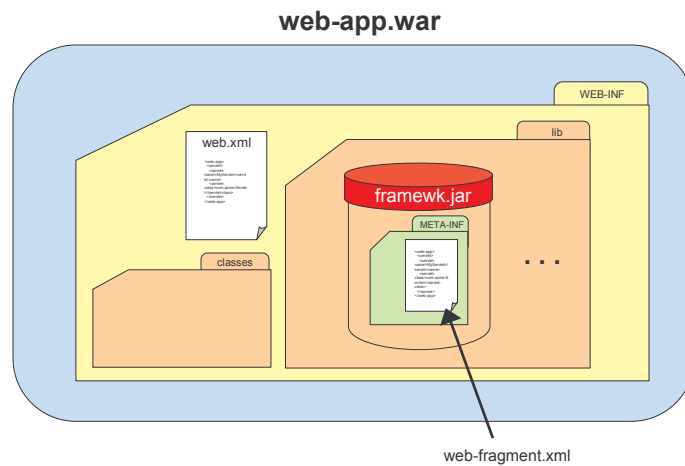
ORACLE

Servlets 3.0 – Web Fragments

- Web-fragments allow specification of independent bits of web.xml in multiple places
 - increased modularity
 - framework pluggability
- Web-fragments exist as web-fragment.xml files in META-INF folder of JARs in WEB-INF/lib
- Frameworks can add fragments into their jar to cause framework servlets to be deployed
- Apps do not have to edit the parent web.xml file

ORACLE

Servlets 3.0 – Web Fragments



ORACLE

Servlets 3.0 – Web Fragments

```
<web-fragment>
  <servlet>
    <servlet-name>
      AcmeServlet
    </servlet-name>
    <servlet-class>
      com.acmeframework.AdminServlet
    </servlet-class>
  </servlet>
  <listener>
    <listener-class>
      com.acmeframework.RequestListener
    </listener-class>
  </listener>
</web-fragment>
```

ORACLE

Servlets 3.0 – Async Processing

- Asynchronous request processing for AJAX, etc.
 - `@WebServlet(urlPatterns={...}, asyncSupported=true)`
- Call `servletRequest.startAsync()` to prevent container from committing response on method exit
- Can further control async operation by invoking on the `AsyncContext` API
- Can add async event listener to get `timeout()` or `complete()` event notification

ORACLE

Servlets 3.0 – Async Processing

```
@WebServlet("/longCall", asyncSupported=true)
public class MyServlet extends HttpServlet {

    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp) {

        AsyncContext async = req.startAsync(req, resp);
        ServletContext ctx = req.getServletContext();
        ((Queue<AsyncContext>) ctx.getAttribute("work"))
            .add(async);
    }
}
```

ORACLE

Servlets 3.0 – Async Processing

```
@WebServletContextListener
public class WorkProc implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {
        Queue<AsyncContext> workQ =
            new ConcurrentLinkedQueue<AsyncContext>();
        sce.getServletContext().setAttribute("work", workQ);
        Executor executor = Executors.newFixedThreadPool(10);
        while(true) {
            if(!workQ.isEmpty()) {
                AsyncContext async = workQ.poll();
                executor.execute(new Runnable() {
                    public void run() {
                        ServletRequest request = async.getRequest();
                        // Do some task that may block or be slow...
                        async.forward("/returnResults.jsp");
                    }
                });
            }
        }
    } // contextDestroyed() method, etc.
}
```

ORACLE

EJB 3.1 – No-interface View

- Bean class used as client interface
 - Public methods form logical business interface

```
@Stateless
public class ServiceBean {

    public Object serviceMethod(Object param) {
        // Perform some exciting public service
    }

    private void supportMethod() {
        // Private code not exported to client
    }
}
```

ORACLE

EJB 3.1 – Singletons

- Single instance per JVM
 - Useful for read-only or shared state
 - Can also mark as a startup initialization class

```
@Singleton @Startup
public class SharedBean {

    SharedData shared;

    @PostConstruct void initialize() {
        // Initialize the shared data structure
    }
    public SharedData getSharedData() {return shared; }
}
```

ORACLE

EJB 3.1 – Concurrency

- Flexible concurrency management options
 - Container or bean-managed concurrency
 - READ and WRITE container settings

```
@Singleton
public class Counter {

    int counter;

    @Lock(WRITE) public void increment() {
        counter = counter + 1;
    }
    @Lock(READ) public int getCount() {return counter;}
}
```

ORACLE

EJB 3.1 – Asynchronicity

- Asynchronous method calls - `@Asynchronous`
 - May return `void` or `java.util.concurrent.Future<V>`

```
public interface LongRemoteCall {
    @Asynchronous
    Future<Integer> call(Object param);
}

@Stateless
@Local(LongRemoteCall.class)
public class LongRemoteCallBean {

    public Integer call(Object param) {
        return ... // return result of long call
    }
}
```

ORACLE

EJB 3.1 – Asynchronicity

- Clients can use the returned `Future` object to:
 - Check the call status
 - Get the result
 - Cancel the call

```
@EJB LongRemoteCall remoteCall;

Future<Integer> future = remoteCall.call(param);
// Do something else for a while...

// Now we need the result, so go get it
try { future.get() }
catch (InterruptedException ex) {throw ex.getCause();}
```

ORACLE

EJB 3.1 – Timer Service

- Updated timer service - supports “cron”-like syntax
- Timers can be created automatically at deployment
- Timers may be persistent or transient

```
@Singleton
public class CacheBean {
    Cache data;

    // Automatically refresh cache every 10 minutes
    @Schedule(minute="*/10",hour="*",persistent=false;)
    public void refresh() { ... }
    ...
}
```

ORACLE

EJB 3.1 – EJB Lite

- Proper subset of all possible EJB features
- Simplest and most useful features
- Supported by embeddable container
- Required by Web Profile
- Includes:
 - Stateful, stateless, singleton session beans (local only)
 - Interceptors
 - Transactions and Security

ORACLE

EJB 3.1 – Embeddable Container

- Allows EJB to run in Java SE
 - More agile testing and development
 - Supports EJB Lite
 - Modules can be detected or specified using properties
 - JNDI lookups in client code, injection in beans

```
EJBContainer ec = EJBContainer.createEJBContainer();
Context ctx = ec.getContext();
MyBean bean = ctx.lookup("java:app/MyBean");
bean.doStuff();
ec.close();
```

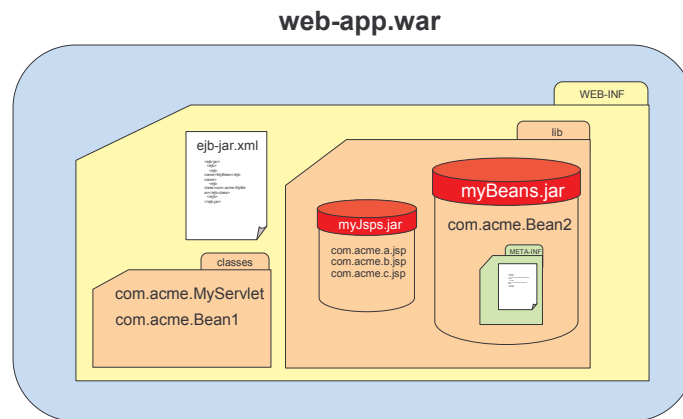
ORACLE

EJB 3.1 – Packaging

- Simpler to package EJB classes in web archive
- **ejb-jar.xml** file is optional, but if present may be in **WEB-INF**, or in a JAR in **WEB-INF/lib**
- Bean classes can be placed in **WEB-INF/classes** or in a JAR in **WEB-INF/lib**
- One or more **ejb** JAR files can be placed in **WEB-INF/lib**
- All components share same module namespace
- No component-specific namespace exists in WAR

ORACLE

EJB 3.1 – Packaging



ORACLE

JPA 2.0 – Mappings

- Embeddables may contain embedded and relationship mappings
- Collections of basic and embeddable types
- Persistently ordered Lists
- Maps keys and values may be any combination of basic, embedded or entity types
- More flexibility with join tables (can use them more, or can use them less)

ORACLE

JPA 2.0 – Mappings

```
@Entity
public class Employee {
    @Id int id;

    @ElementCollection
    @CollectionTable(name="EMP_NAMES")
    Set<String> nickNames;

    @OneToOne @JoinTable(name="EMP_CUB")
    Cubicle cubicle;

    @OneToMany(mappedBy="emp")
    @MapKeyColumn(name="P_TYPE")
    Map<String,Phone> phoneNumbers;
    ...
}
```

EMP_NAMES

EMPLOYEE_ID	NICKNAMES

EMP_CUB

EMPLOYEE_ID	CUBICLE_ID

PHONE

ID	P_TYPE	EMP_ID	...

ORACLE

JPA 2.0 – Criteria API

- Complement to string-based JP QL
- Java objects represent JP QL concepts and are used as building blocks to build the query
- Dynamic query creation w/o string manipulation
- Can use either of two variants:
 - string-based – use attribute strings, more dynamic
 - strongly typed – more compile-time safety
- Strongly typed version uses generated metamodel

ORACLE

JPA 2.0 – Strongly-typed Criteria

```
QueryBuilder qb = em.getQueryBuilder();
CriteriaQuery q = qb.create();

Root<Customer> cust = q.from(Customer.class);
Join<Customer,Order> order =
    cust.join(Customer_.orders);
Join<Customer,Address> addr =
    cust.join(Customer_.address);

q.where(
    qb.equal(addr.get(Address_.state), "CA"),
    qb.gt(order.get(Order_.quantity), 5));

q.select(
    order.get(Order_.quantity),
    qb.prod(order.get(Order_.totalCost), 1.08),
    addr.get(Address_.zipCode));
```

ORACLE

JPA 2.0 – String-based Criteria

```
QueryBuilder qb = em.getQueryBuilder();
CriteriaQuery q = qb.create();

Root cust = q.from(Customer.class);
Join order = cust.join("orders");
Join addr = cust.join("address");

q.where(
    qb.equal(addr.get("state"), "CA"),
    qb.gt(order.get("quantity"), 5));

q.select(
    order.get("quantity"),
    qb.prod(order.get("totalCost"), 1.08),
    addr.get("zipCode"));
```

ORACLE

JPA 2.0 – Other Features

- More standardized properties
- Full support for all derived identifier configurations
- Mixed access mode in single class/hierarchy
- Pessimistic and other locking modes
- Interface for controlling shared L2 cache
- Enhancements to JP QL
- Additional EntityManager API

ORACLE

Java EE 6 – Cleanup Process

- Technologies become outdated, no longer used
 - Enter the “pruning” stage
- Any technology in pruning stage may be deemed “optional” in a future spec release
- Optional means vendors do not have to include technology in implementation
- Technologies in pruning stage in Java EE 6:
 - JAX-RPC, EJB entity beans, JAXR, JSR-88 Deployment

ORACLE

Java EE 6 – Status

- Most JSR expert groups are in PFD (Proposed Final Draft) stage
- Some twiddling is still being done
- Currently looking at re-factoring JSR 299 to come up with a platform component model
- Java EE 6 (and all constituent component technologies) target release date is **Sept 2009**

ORACLE

Summary

- ✓ Lots of user input during the spec process
- ✓ Java EE 6 even easier to use
- ✓ Support for modern and popular technologies
- ✓ Decoupling enables simpler testing/development
- ✓ Platform profiles – take what you need
- ✓ Pruning prevents platform bloat
- ✓ Industrial strength Reference Implementations

ORACLE

Links and Resources

- Java EE 6 (JSR 316)
<http://www.jcp.org/en/jsr/detail?=316>
- Component JSRs
<http://www.jcp.org/en/jsr/all>
- Glassfish v3 (Java EE 6, Servlet 3.0, EJB 3.1 RI)
<http://glassfish.dev.java.net>
- EclipseLink (JPA 2.0 RI)
<http://eclipse.org/eclipselink>

ORACLE