



# Evolving a Platform

Lessons from Java EE 6 Development

Roberto Chinnici

Java EE 6 Specification Lead

Sun Microsystems, Inc.



# Java Enterprise Edition Platform

**More Than 5 Million Developers**

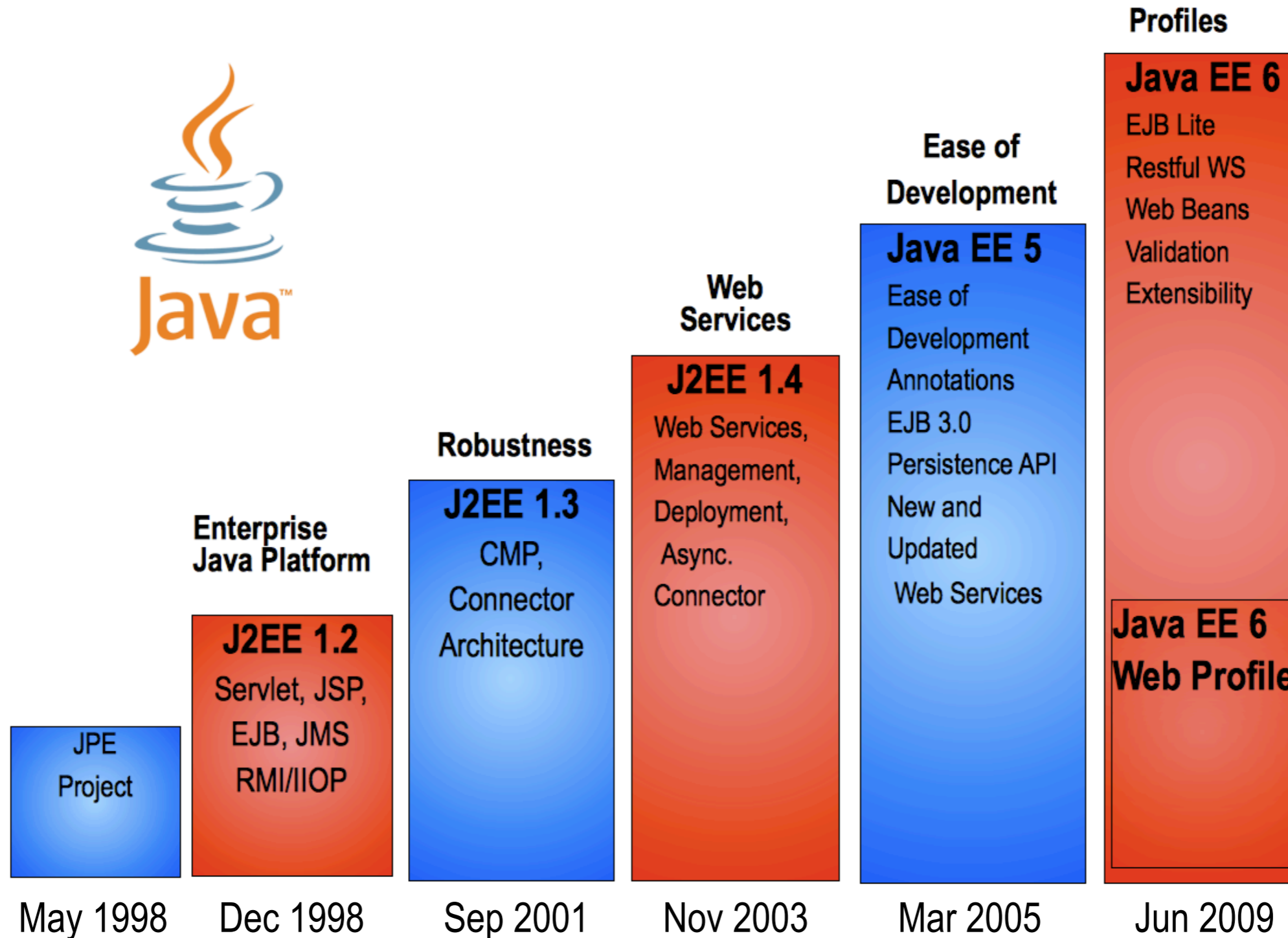
# Healthy Competition Between Vendors



# A Vibrant Ecosystem



# A History of Enterprise Java



# What Have We Learned in the Past 11 Years?

**Up to J2EE 1.4:  
Power At Any Price**

**Starting with Java EE 5:  
Ease of Development**

**Can't We Just Fix Everything That  
Proves To Be "Wrong"?**

# The Importance of Compatibility

# Dimensions of Compatibility

## Product compatibility

Applications deployed on servers from different vendors

## Backward compatibility

Applications from 1999 running on the latest servers

## Binary compatibility

Class file, jar archive format

Java Language Specification Chapter 13

# Compatibility in Java EE 6

- All APIs are binary-compatible with previous versions
  - Interfaces implemented by apps are effectively immutable
  - E.g. javax.servlet.Servlet interface cannot ever be changed
  - Abstract classes better than interfaces
- Deployment descriptors are an evolution of previous ones
  - Older descriptors are still valid
  - Behavior does not depend on descriptor version

# How Do We Innovate Then?

# Our Strategy

1. Simplify programming model
2. Make different APIs work better together
3. Introduce new APIs in key areas
4. Enable timely evolution

# 1. Simplify Programming Model

# Capture Common Patterns

# New Functionality in EJB 3.1

## Singleton components

Hard to to without container support

`@Singleton public class ...`

## Asynchronous method invocation

Workarounds are very verbose

`@Asynchronous public Future<Result> compute(...)`

# Fix Inconsistencies

# Redesigning JNDI Scopes

Problem: java:comp works differently in EJB and Web modules

Introduce new, appropriately named scopes:

java:module, java:application, java:global

Resource sharing done at the appropriate level

```
@Resource(name="java:module/env/customerDB")
```

```
DataSource db;
```

Allows referring to components in different apps:

```
java:global/myApp/myModule/MyBean!com.acme.MyInterface
```

# Adopt What Works

# Facelets in JSF 2.0

Problem: JSP was never meant to be a page description language

Facelets successful alternative in open source

Declarative page description language based on XHTML

Standardize facelets in JSF 2.0

## 2. Make Different APIs Work Better Together

# EJB Components in the Web Tier

Problem: using EJBs in the web tier requires two levels of packaging (ejb jar and war file inside an ear file)

EJB 3.1 components can be defined in WEB-INF/classes

No interface mode means 1 EJB component takes 1 class

All usual transactional, security annotations OK  
`@TransactionAttribute(REQUIRES_NEW)`

# Bridge EJB and JSF

JCDI (née Web Beans) makes EJB components usable contextually

Beans can be tied to a scope (request, session, conversation)

```
@ConversationScoped @Stateful  
public class ShoppingCart { ...}
```

New ELResolver makes them accessible by name from JSF pages

```
{shoppingCart.checkout}
```

# 3. Introduce New APIs

# Look at Multiple Use Cases

# Pluggability in Servlet 3.0

Problem: Lots of web apps need libraries

Libraries have to be configured manually in XML

Define new general pluggability API

Static and dynamic version

Also allows overriding of static servlet/filter configuration

Enabler for scripting support in Java EE 6

# New Paradigms Benefit From New APIs

# Java API for RESTful Web Services

REST paradigm becoming more common in web apps

Existing servlet support for HTTP is too low-level

JAX-RS models HTTP concepts directly:

`@Path("/widget/{id}")`

`@HttpMethod("GET")`

`@GET`

`@Produces("text/plain")`

`@Consumes("application/x-www-form-urlencoded")`

Extensible provider framework for automated type conversions

# JAX-RS as a “Modern” API

Declarative, annotation-based layer: javax.ws.rs package

Deeper, programmatic layer: javax.ws.rs.core package

Nice helpers like ResponseBuilder

```
ResponseBuilder.status(...).tag(...).cacheControl(...).entity(...).build();
```

Request type complements HttpServletRequest

```
Request.evaluatePreconditions(Date, EntityTag)
```

# A New API as a Significantly Better Way To Perform a Task

# Criteria API in JPA 2.0

Problem: dynamic queries have to be built as strings

```
SELECT o.quantity, a.zipcode
```

```
FROM Customer c JOIN c.orders o JOIN c.address a
```

```
WHERE a.state = 'CA'
```

```
ORDER BY o.quantity, a.zipcode
```

New typesafe criteria API in JPA 2.0

Takes advantage of Java language generics

`from(...)`, `join(...)`, `where(...)`, `orderBy(...)` methods

```
Join<Customer, Order> o = queryBuilder.join(Customer_.orders)
```

# Criteria API in JPA 2.0

Problem: dynamic queries have to be built as strings

```
SELECT o.quantity, a.zipcode
```

```
FROM Customer c JOIN c.orders o JOIN c.address a
```

```
WHERE a.state = 'CA'
```

```
ORDER BY o.quantity, a.zipcode
```

## New typesafe criteria API in JPA 2.0

Takes advantage of Java language generics

from(...), join(...), where(...), orderBy(...) methods

```
Join<Customer, Order> o = queryBuilder.join(Customer_.orders)
```

**Listen to your Users.  
“Why isn’t there an API for ...”**

# New Bean Validation JSR

Validation API that works across JPA, JSF, JAX-RS, ...

Constraints are placed directly on classes

## Declarative layer

```
@NotNull Size(max=40) private String streetName;
```

```
@NotNull @Valid private Country country;
```

## Programmatic layer

```
Set<ConstraintViolation> Validator.validate(Object)
```

# Sometimes Extending an Existing API is Hard

# Asynchronous API in Servlet 3.0

Motivated by Comet, chat rooms, Ajax clients, legacy systems  
Need to break the one-request-one-thread model

Servlet API very old, nearly evolution-proof

Created new async mode represented by an `AsyncContext`  
Application may schedule a task asynchronously or delegate to another servlet

Opt-in system

```
@WebServlet(asyncSupported=true)
```

```
@WebFilter(asyncSupported=true)
```

# 4. Enable Timely Evolution

# Can We Ever Remove Anything From the Platform?

# Fixing Historical Errors

Problem: some technologies have become obsolete

JAX-RPC, EJB 2.1 Entity Beans, JSR-88

Often superseded by newer ones:

JAX-WS, JPA

Pruning process lets us remove technologies from the platform

They remain optional to preserve backward compatibility

# What If One Size Does Not Fit All?

# Profiles

Bundles of technologies targeting one class of applications and one developer community

Created and evolved independently of the platform

New versions can appear at their own pace (e.g. every 6 months)

Free to add, remove components from the platform

Very few constraints built-in

# Java EE 6 Web Profile

Targets web applications

Intermediate, fully functional profile (“batteries included”)

Required technologies:

Servlet 3.0, JSP 2.1, EL 2.1, JSTL 1.2

JSF 2.0

EJB Lite 3.1, JTA 1.1

JPA 2.0

Bean Validation 1.0

# Recap

# New in Java EE 6

## Profiles

- Web Profile

## Pruning process

## Extensibility in the web tier

- Zero-configuration use of third-party libraries, frameworks

## Several new APIs

- To cover new areas (validation, criteria API, etc.)

- To make existing APIs work better together (JCDI)

## Simpler packaging for EJBs in web apps

And much more!

# Final Release September 2009

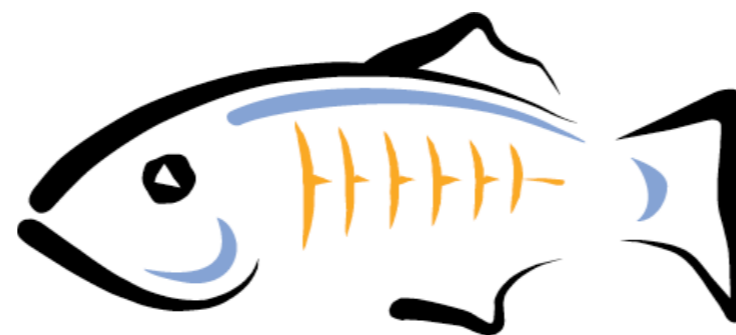
# How Can I Find Out More?

Blogs, articles appearing regularly

Expert groups starting to work in an open way

All specifications on the JCP.org web site

Implementation work as open source in GlassFish V3





Roberto Chinnici  
roberto.chinnici @ sun.com

