

# **Applicazioni Web**

## **Intro ai Web Container**

### **Tomcat**

**Danilo Croce**

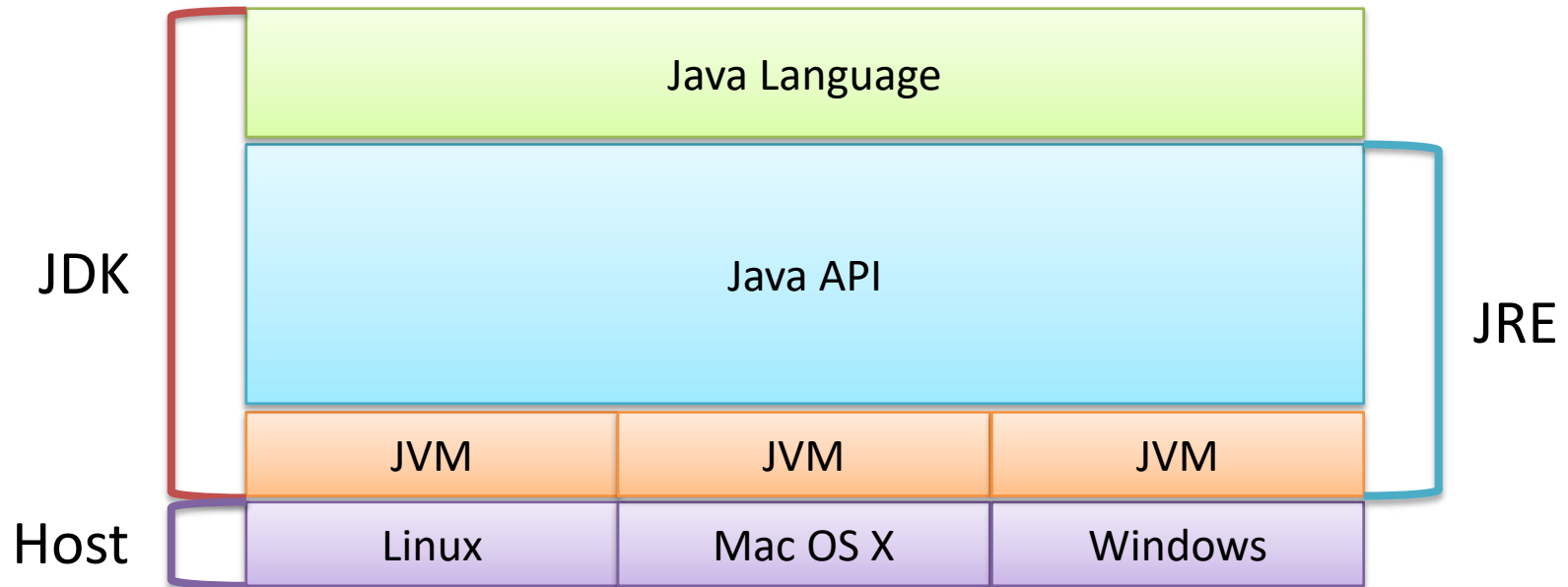
(parzialmente ispirato da Java Enterprise Edition di Gabriele Tomei)

**Maggio 2019**

# Le Piattaforme Java

- Tutte le piattaforme Java consistono di
  - Java Virtual Machine (JVM) + Application Programming Interface (API)
- La **JVM** è un particolare programma (**interprete**) eseguito su uno specifico sistema ospite (**host**) che consente l'esecuzione di programmi Java
  - JVM interpreta il codice intermedio (**bytecode** contenuto in file .class) risultato della compilazione del codice sorgente (.java)
- Esistono varie implementazioni JVM, una per ciascun sistema host supportato
  - Linux x86/x64
  - Mac OS X x64
  - Win x86/x64
  - ...
- La **API** è una collezione di componenti software “standard” messi a disposizione degli sviluppatori Java per creare nuovi componenti e/o applicazioni

# Java: Linguaggio + Piattaforma



# Java SE

- È la piattaforma di riferimento quando si parla di Java
- Java SE API fornisce le funzionalità “core” del linguaggio
  - tipi nativi (ad es., int, boolean, char, etc.)
  - classi e oggetti base (ad es., Class, Object, String, etc.)
  - classi e oggetti per gestire
    - I/O
    - Security
    - Database
    - Graphical User Interface (GUI)
    - XML
    - ...

# Java EE

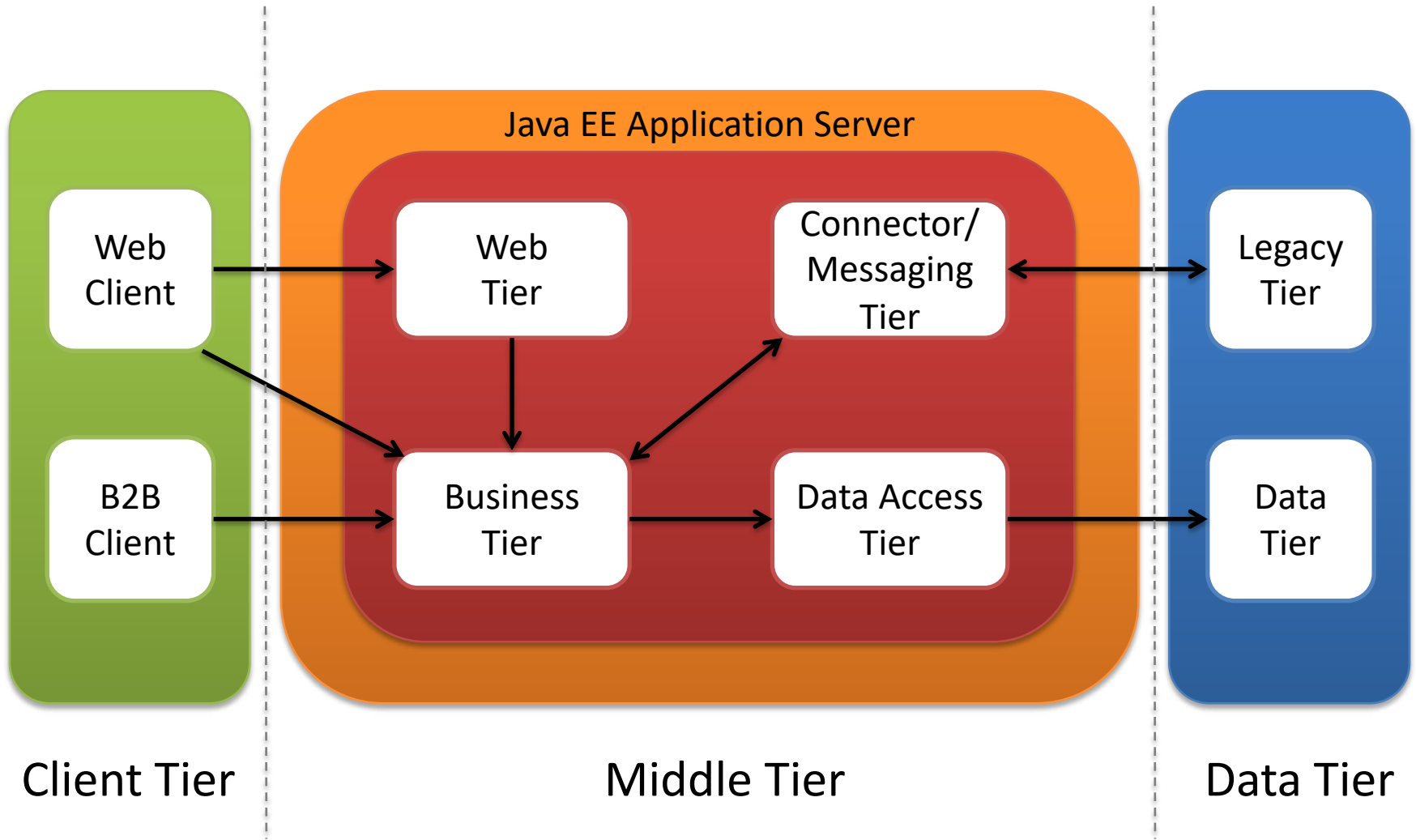
- Realizza una piattaforma “standard” per lo sviluppo, l’esecuzione e la gestione di **applicazioni enterprise**:
  - **Multi-tier** → strutturate a “livelli”
  - **Web-enabled** → accessibili via Web
  - **Server-centric** → eseguite in uno specifico ambiente server
  - **Component-based** → costituite da componenti sw in esecuzione su una o più istanze server distribuite
- Si basa sulla piattaforma Java SE a cui aggiunge specifiche e strumenti (API) ad hoc
- Condivide i vantaggi delle applicazioni Java SE:
  - 1 specifica standard vs. molte implementazioni
  - implementazioni disponibili per la maggior parte di sistemi host
  - portabilità, facilità di sviluppo, riuso, sicurezza, etc.

# Architettura Multi-tier



- **Modello** architetturale “astratto” per applicazioni enterprise
  - indipendente dalle scelte tecnologiche (linguaggio, piattaforma, etc.)
- Le funzionalità dell’applicazione sono suddivise in 3 “livelli” isolati (**Tiers**):
  - **Client Tier** → esegue richieste al Middle-tier
  - **Middle Tier** → gestisce le richieste provenienti dai clients e processa i dati dell’applicazione
  - **Data Tier** → mantiene i dati in strutture di memorizzazione permanenti
- Java EE è una particolare realizzazione del modello che si concentra sul Middle Tier → Java EE Application Server

# Java EE: Architettura Multi-tier



# Java EE: Web Tier

Tecnologia	Scopo
Servlets	Classi Java che processano le richieste HTTP e generano dinamicamente le risposte (HTML)
JavaServer Faces	Framework per il design dell'interfaccia utente di applicazioni Web
JavaServer Faces Facelets	Particolari applicazioni JavaServer Faces che usano pagine XHTML anziché JSP
Expression Language	Insieme di tags standard usati in JSP e Facelets per riferirsi a componenti Java EE
JavaServer Pages (JSP)	Documenti testuali compilati e trasformati in Servlets per aggiungere contenuto dinamico a pagine HTML
JavaServer Pages Standard Tag Library	Tag library che raccoglie funzionalità comuni a pagine JSP
JavaBeans Components	Oggetti Java per la memorizzazione temporanea dei contenuti di un'applicazione



# Java EE: Business Tier

Tecnologia	Scopo
Enterprise JavaBeans (EJB)	Componenti gestite dall'Application Server che incapsulano le funzionalità principali dell'applicazione
JAX-RS RESTful Web Services	API per la creazione di Web Services REST (via HTTP GET e POST)
JAX-WS Web Service Endpoints	API per la creazione ed il consumo di Web Services XML/SOAP
Java Persistence API Entities	API per il mapping tra i dati contenuti nei sistemi di memorizzazione persistente e corrispondenti oggetti Java
Java EE Managed Beans	Essenzialmente EJB che non richiedono requisiti di sicurezza/transazionalità

# Java EE: Data Tier

Tecnologia	Scopo
Java Database Connectivity API (JDBC)	API a basso livello per l'accesso ed il recupero dei dati memorizzati su supporti permanenti. Tipicamente usata per eseguire query SQL ad un particolare RDBMS
Java Persistence API	API per la creazione di Web Services REST (via HTTP GET e POST)
Java EE Connector Architecture	API per la creazione ed il consumo di Web Services XML/SOAP
Java Transaction API (JTA)	API per la definizione e la gestione delle transazioni tra sorgenti dati multiple e distribuite

# Java EE Application Servers

- Server che **implementa** la piattaforma **Java EE**
- Ospita i componenti **Middle Tier** di un'applicazione enterprise multi-tiered
- Fornisce i **servizi standard** specificati da Java EE a questi componenti sottoforma di **container**:
  - gestione della concorrenza, scalabilità
  - sicurezza
  - persistenza, transazioni
  - gestione del ciclo di vita dei componenti sw
- Java EE servers “famosi”: GlassFish (Oracle), JBoss AS (Red Hat), WebLogic (Oracle-BEA), WebSphere (IBM), **Tomcat (Apache Software Foundation)**
  - [http://en.wikipedia.org/wiki/Comparison\\_of\\_application\\_servers#Java](http://en.wikipedia.org/wiki/Comparison_of_application_servers#Java)

# Java EE Containers

- Interfaccia tra un componente dell'applicazione e le funzionalità di “basso livello” fornite dalla piattaforma per supportare quel componente
- Le funzionalità di un container sono specificate dalla piattaforma
- Un tipo di container per ciascun tipo di componente
- Java EE Server fornisce ai vari containers un ambiente omogeneo in cui è garantito il funzionamento di ciascun componente dell'applicazione

# Web Container

- Interfaccia tra le componenti web ed il server web
- Un componente web può essere una **Servlet**, una pagina **JSF** o **JSP**
- Gestisce il ciclo di vita del componente
- Smista le richieste ai vari componenti dell'applicazione
- Fornisce interfacce verso “dati contestuali” (ad es. informazioni sulla richiesta corrente)

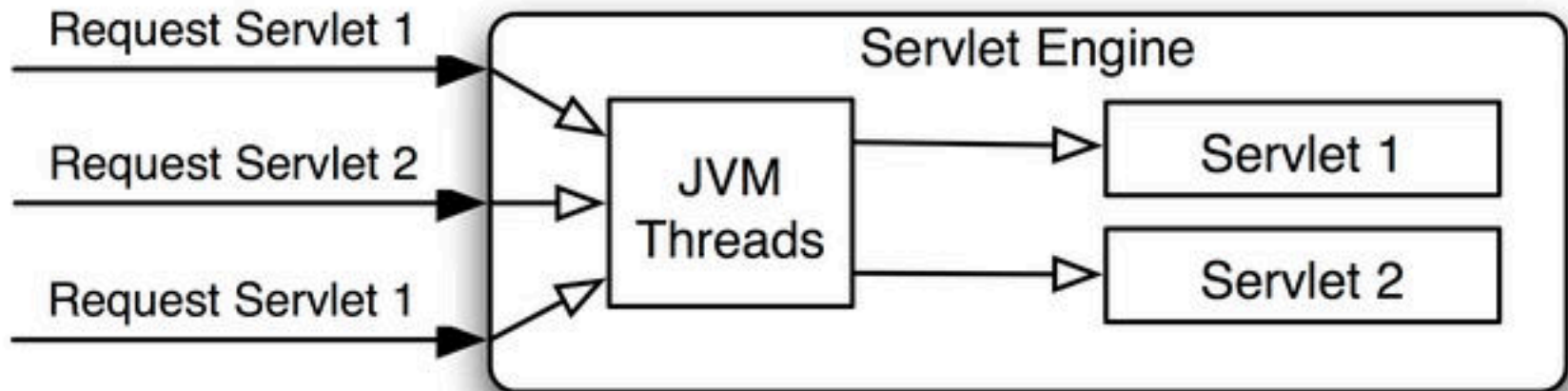
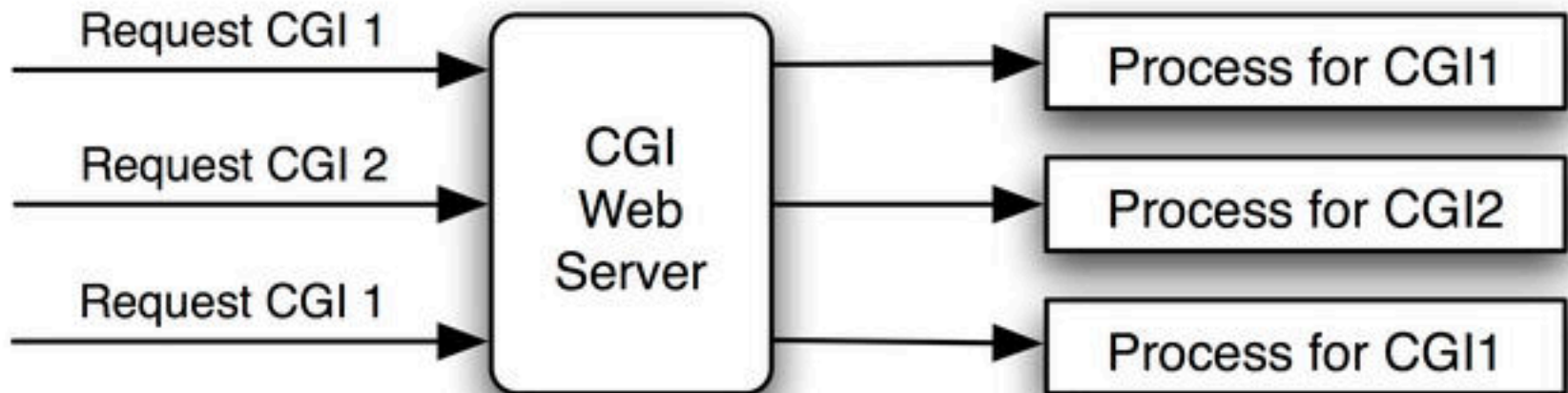
# Application Client Container

- Interfaccia (**gateway**) tra le **applicazioni** client Java EE ed il **server** Java EE
- I clients sono particolari applicazioni Java SE che usano i componenti server Java EE
- In esecuzione su macchine client (generalmente diverse dal server Java EE)

# Java Servlet/Java Server Pages (JSP)

- Sviluppato da Java Community Process
- Parte dello standard Java EE
- OO, platform-independent, efficiente, scalabile,...
- Consente la separazione tra il livello di presentazione (interfaccia) e la logica applicativa

# CGI vs. Servlet/JSP





# CGI vs. Servlet/JSP

- CGI
  - 1 richiesta client → 1 processo server
- Servlet/JSP
  - 1 richiesta client → 1 thread all'interno dello stesso processo server JVM
- Ottimizzazione delle risorse
  - Processo vs. Thread

# Java Servlet: Vantaggi

- Condividono tutti i vantaggi del sw scritto in Java:
  - Portabilità, OO, riuso, supporto di librerie già esistenti, efficienza, sicurezza, etc.
- Si basano su una ben consolidata API specifica per il protocollo HTTP:
  - processing delle richieste
  - generazione delle risposte
  - gestione delle sessioni e dei cookies

# Servlet e Applicazioni Web (2)

- Sia le Servlets che le JSPs sono eseguite all'interno di archivi Web (WAR)
- I WAR a loro volta sono in esecuzione su un Servlet Container (parte delle specifiche Java EE server)
- Il Servlet Container coincide con il servizio Apache Tomcat

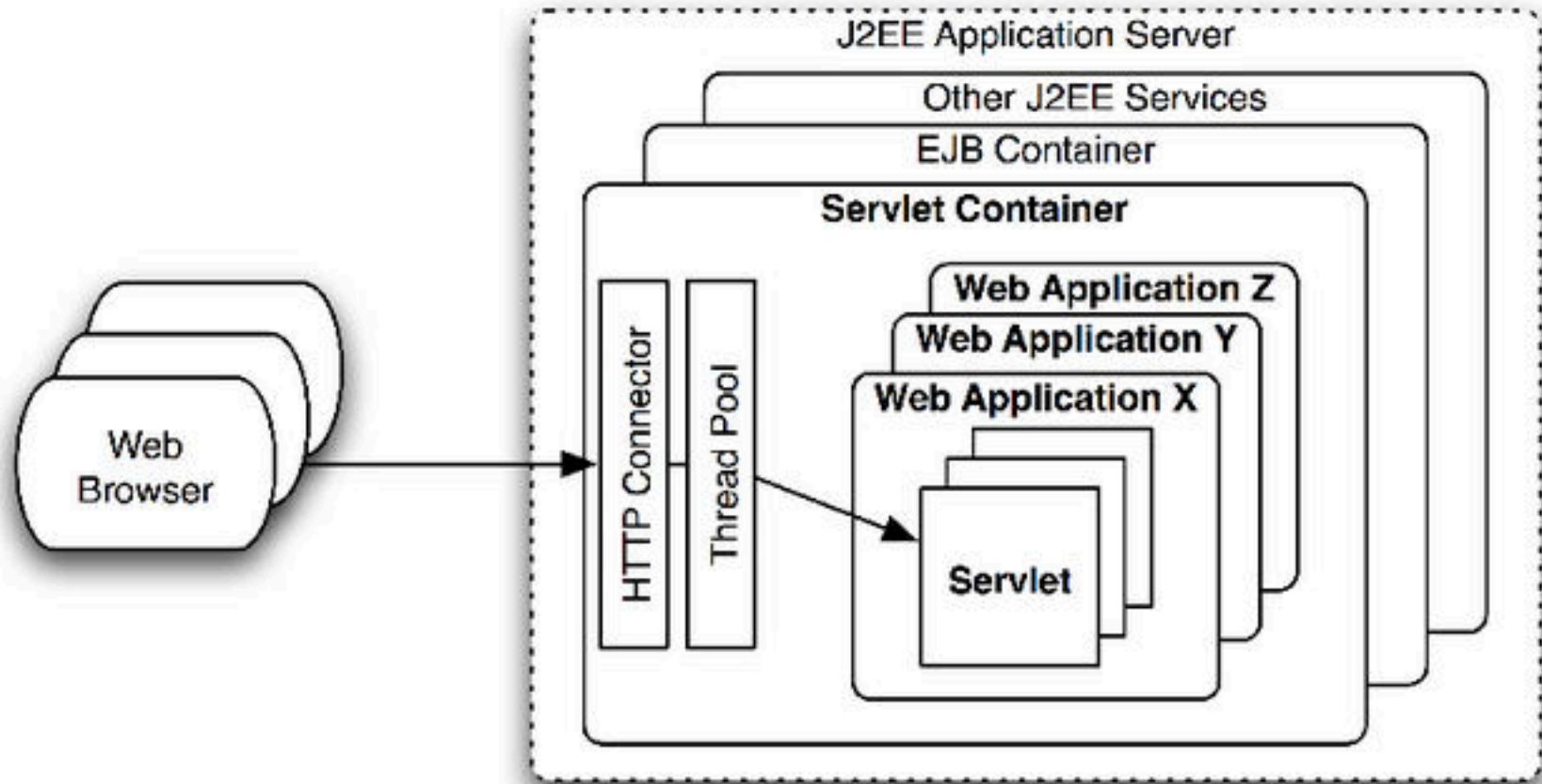
# Servlet e Applicazioni Web (3)

- Le applicazioni web sono isolate l'una dall'altra all'interno dello stesso Servlet Container
- Il Servlet Container fornisce tutti quei servizi di “basso livello” necessari per il ciclo di vita di Servlets e JSPs:
  - gestione delle connessioni HTTP, sessioni, threading, sicurezza, gestione delle risorse, monitoring, deployment, etc.

# Ma che cos'è una Java Servlet?

- È una normale classe Java che consente l'interazione richiesta/risposta con un'applicazione secondo il modello client/server
- Le Servlets sono progettate per gestire qualunque tipo di protocollo richiesta/risposta
- Tipicamente vengono usate per l'implementazione di applicazioni che interagiscono secondo il protocollo HTTP (richiesta/risposta via Web)

# Servlet e Applicazioni Web



# Tomcat Basics

## What is Tomcat?

- Web container
  - Manage Java Servlets
- HTTP Web server
  - Deliver web pages on the request to clients.
  - Receive content from clients.
  - Create HTML documents dynamically ("on-the-fly")
- How to download it
  - <https://tomcat.apache.org/download-90.cgi>

# Which version?

https://tomcat.apache.org/download-90.cgi

[Yahoo!](#)
[Unicredit](#)
[Huric](#)
[Reveal](#)
[didattica](#)
[PA](#)
[Semeval](#)
[Google Maps](#)
[absita](#)
[I4ALL](#)
[YouTube](#)

You are currently using **http://us.mirrors.quenda.co/apache/**. If you encounter a problem with this mirror, please select another mirror. If the mirror is failing, there are *backup* mirrors (at the end of the mirrors list) that should be available.

Other mirrors:  [Change](#)

## 9.0.20

Please see the [README](#) file for packaging information. It explains what every distribution contains.

### Binary Distributions

- Core:
  - [zip](#) ([pgp](#), [sha512](#))
  - [tar.gz](#) ([pgp](#), [sha512](#))
  - [32-bit Windows zip](#) ([pgp](#), [sha512](#))
  - [64-bit Windows zip](#) ([pgp](#), [sha512](#))
  - [32-bit/64-bit Windows Service Installer](#) ([pgp](#), [sha512](#))
- Full documentation:
  - [tar.gz](#) ([pgp](#), [sha512](#))
- Deployer:
  - [zip](#) ([pgp](#), [sha512](#))
  - [tar.gz](#) ([pgp](#), [sha512](#))
- Embedded:
  - [tar.gz](#) ([pgp](#), [sha512](#))
  - [zip](#) ([pgp](#), [sha512](#))

### Source Code Distributions

- [tar.gz](#) ([pgp](#), [sha512](#))



# Some details: who does what?

## **Servlet Container: Catalina**

- Catalina implements Sun Microsystems's specifications for servlet and JavaServer Pages (JSP).

## **Connector: Coyote**

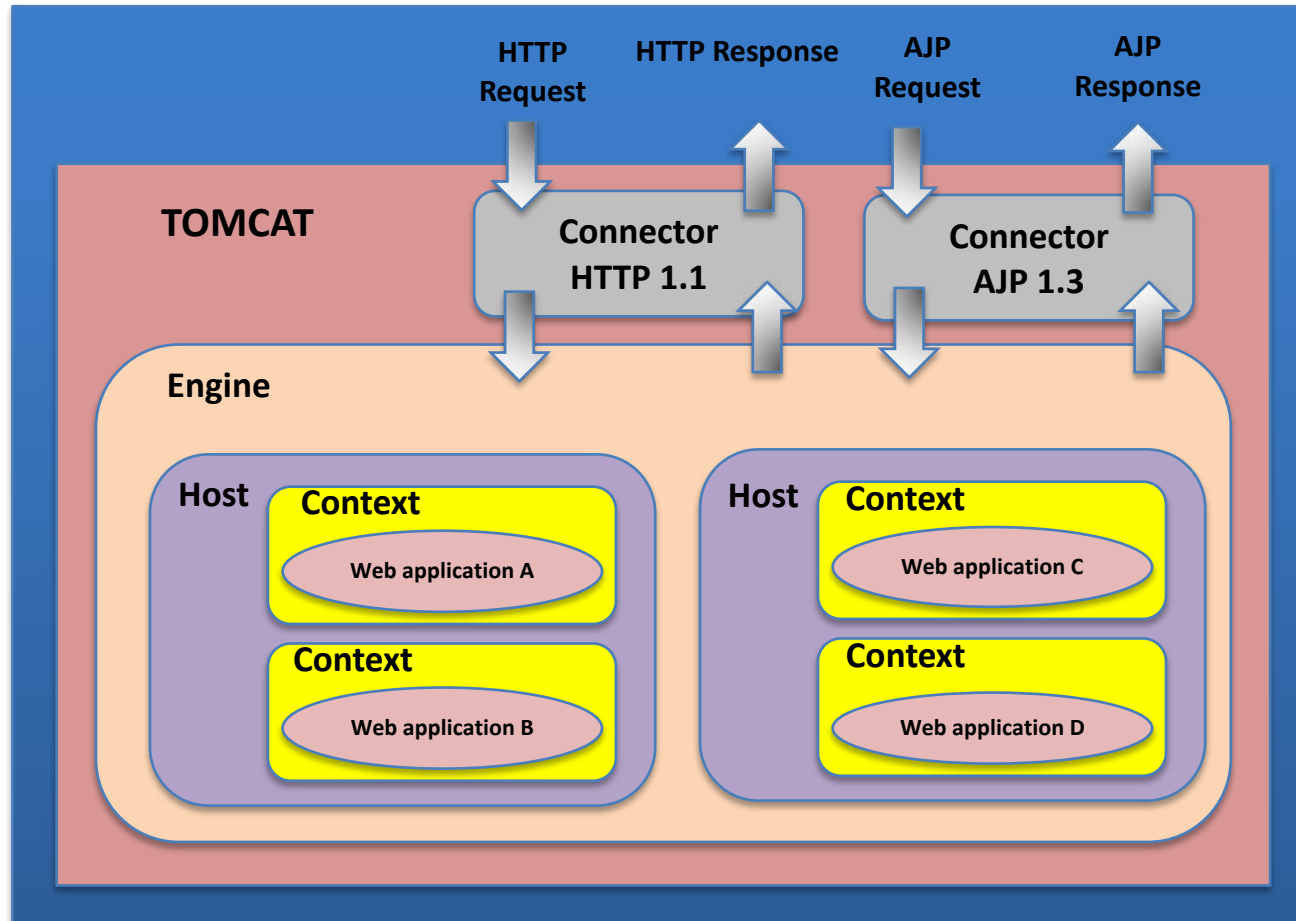
- Coyote is a Connector component for Tomcat that supports the HTTP 1.1 protocol as a web server.
- This allows Catalina, nominally a Java Servlet or JSP container, to also act as a plain web server that serves local files as HTTP documents.
- Coyote listens for incoming connections to the server on a specific TCP port and forwards the request to the Tomcat Engine to process the request and send back a response to the requesting client.

# Some details: who does what? (2)

## JSP Engine: Jasper

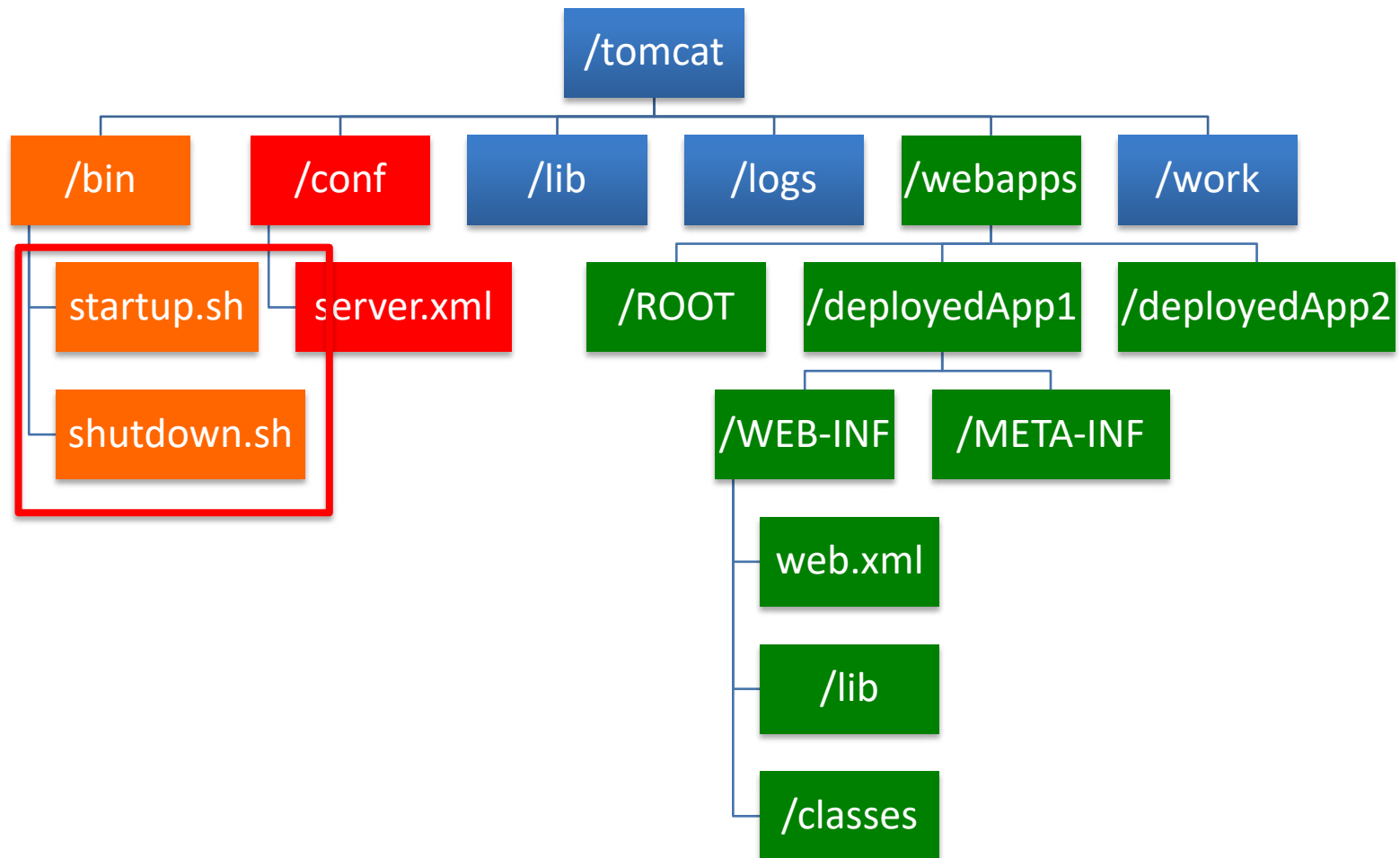
- Jasper is Tomcat's JSP Engine. Jasper parses JSP files to compile them into Java code as servlets (that can be handled by Catalina).
- At runtime, Jasper detects changes to JSP files and recompiles them

# Tomcat Architecture



More about *server.xml*: <http://www.mulesoft.com/tomcat-configuration#.URM9VlqWkSg>

# Look inside the Tomcat directory



# Start Tomcat Server

- Windows

> startup.bat

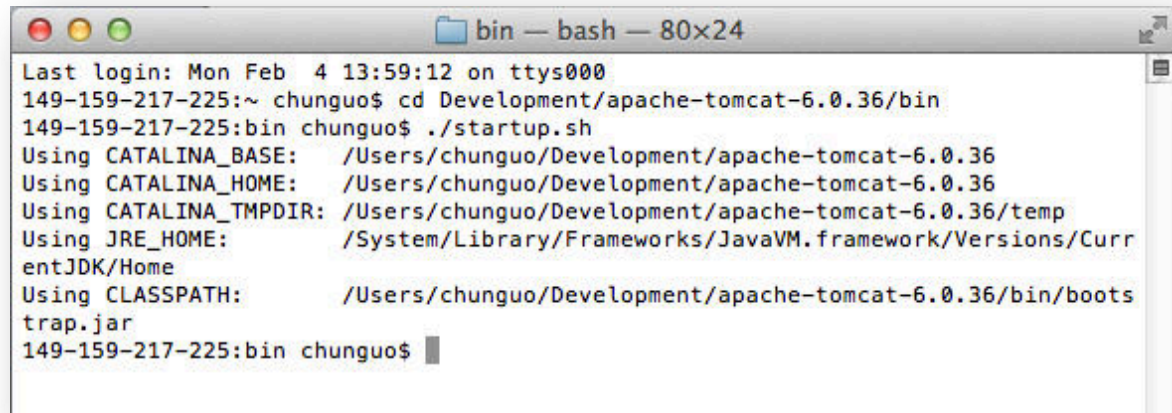
- Mac

\$ cd "TOMCAT\_HOME"/bin

\$ ./startup.sh

- Test

– <http://localhost:8080>



```

bin — bash — 80x24
Last login: Mon Feb  4 13:59:12 on ttys000
149-159-217-225:~ chunguo$ cd Development/apache-tomcat-6.0.36/bin
149-159-217-225:bin chunguo$ ./startup.sh
Using CATALINA_BASE:   /Users/chunguo/Development/apache-tomcat-6.0.36
Using CATALINA_HOME:   /Users/chunguo/Development/apache-tomcat-6.0.36
Using CATALINA_TMPDIR: /Users/chunguo/Development/apache-tomcat-6.0.36/temp
Using JRE_HOME:        /System/Library/Frameworks/JavaVM.framework/Versions/CurrentJDK/Home
Using CLASSPATH:       /Users/chunguo/Development/apache-tomcat-6.0.36/bin/bootstrap.jar
149-159-217-225:bin chunguo$
  
```

# Shut Down Tomcat Server

- Windows

- > shutdown.bat

- Mac

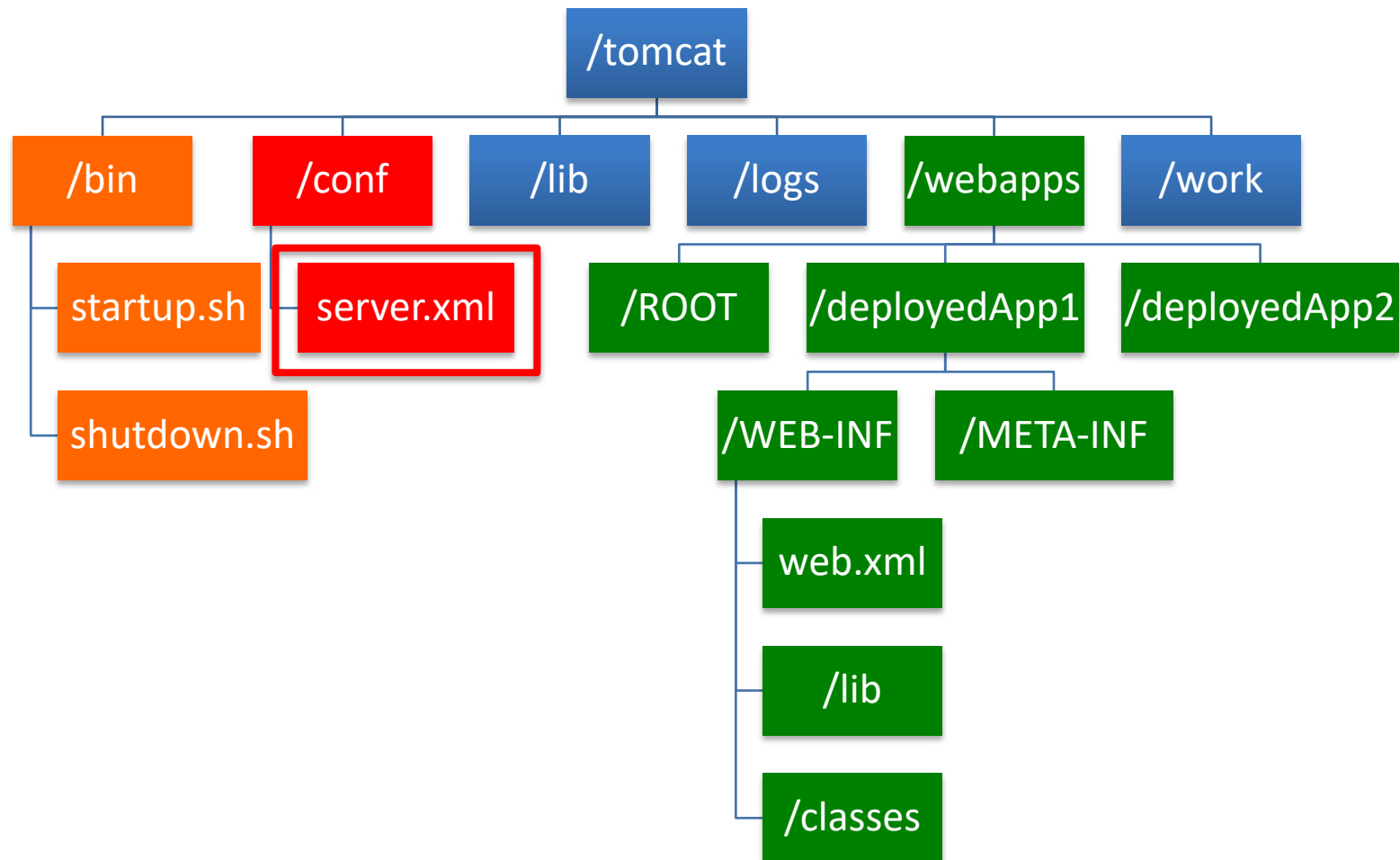
- \$ cd "TOMCAT\_HOME"/bin

- \$ ./shutdown.sh

- Test

- <http://localhost:8080>

# Look inside the Tomcat directory



# Host, Context

- Host

`http://chausie.slis.indiana.edu/`

`http://localhost:8080/`

- Context

`http://chausie.slis.indiana.edu/ROOT`

`http://localhost:8080/myApp`



# Tomcat configuration file (server.xml)

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
      redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443"/>
    <Engine name="Catalina" defaultHost="localhost">
      <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
        resourceName="UserDatabase"/>
      <Host name="localhost" appBase="webapps" unpackWARs="true"
        autoDeploy="true"
        xmlValidation="false" xmlNamespaceAware="false">
      </Host>
    </Engine>
  </Service>
</Server>
```

# Change Default Port

- In “server.xml”

... ..

```
<Connector port="8080" protocol="HTTP/1.1"  
           connectionTimeout="20000"  
           redirectPort="8443" />
```

... ..

# Change Default Port

- In “server.xml”

... ..

```
<Connector port="8080" protocol="HTTP/1.1"  
            connectionTimeout="20000"  
            redirectPort="8443" />
```

... ..

# To establish Tomcat integration in Eclipse:

1. From the **Eclipse** main menu choose File > New > Other...
2. Select **Server > Server**.
3. Click Next. Figure: Define a New **Server**.
4. Select **Tomcat vx.x Server**.
5. Click Next. Figure: Choose **Tomcat** version.
6. Browse to the folder of your **Tomcat** installation.  
...
7. Select Finish.