Alma Mater Studiorum
Università di Bologna
Scuola di Ingegneria

Tecnologie Web T
A.A. 2019–2020

Esercitazione 3
Servlet

Home Page del corso: http://www-db.disi.unibo.it/courses/TW/
Versione elettronica: L.03.Servlet.pdf
Versione elettronica: L.03.Servlet-2p.pdf

Agenda

- Importazione e modifica di un progetto di esempio
  - class-path a tempo di compilazione ed esecuzione
  - deployment ed esecuzione
  - descrittore *web.xml*
  - interazione con l'applicazione
- Creazione di un nuovo progetto
  - servlet e mantenimento dello stato
  - avvio e deployment direttamente da Eclipse

Per cominciare

- Il file ***03a_TecWeb.zip*** contiene lo scheletro di un semplice progetto di esempio basato sull'uso di Servlet
  - creato con Eclipse, contiene già tutti i descrittori necessari per essere riconosciuto e configurato correttamente
  - una volta corretti ***i piccoli "errori" creati ad arte*** per la prima parte di questa esercitazione, può essere riutilizzato come base per altri progetti futuri di applicazioni Web, non solo all'interno di questo corso

- Importare il progetto come visto nelle precedenti esercitazioni
  - *File → Import → General → Existing Projects into Workspace → Next → Select archive file*

Progetto Eclipse: struttura dell'applicazione Web

La directory ***web*** contiene l'esatta struttura dell'applicazione che verrà eseguita all'interno del server

- risorse “statiche” (dal punto di vista del server): pagine HTML, immagini, fogli di stile CSS, script Javascript, …
- metadati dell'applicazione
  - *WEB-INF/web.xml*

  (per ora tralasciamo questa parte)
- bytecode (file *.class*) delle classi Java che costituiscono l'applicazione Web
  - *WEB-INF/classes*

  (direttorio inizialmente vuoto, usato come destinazione dei sorgenti compilati attraverso il build file di ANT)
- librerie ***necessarie a tempo di esecuzione, ma non presenti tra le librerie rese disponibili dal server***
  - *WEB-INF/lib*

  (direttorio i cui archivi *.jar* sono da aggiungere al build-path di Eclipse, **se necessari anche a tempo di compilazione**)

Progetto Eclipse: build file di Ant

Oltre alle normali operazioni, comuni ai progetti di applicazioni “tradizionali”, il file di build che useremo per lo sviluppo di applicazioni Web prevede:

- packaging in formato WAR (Web Archives Repository)
- deployment
  - **copia dell'archivio WAR o dell'equivalente direttorio esploso in una apposita directory del server**, al fine della attivazione dell'applicazione Web
- aggiornamento delle sole risorse statiche dell'applicazione Web
  - richiede il deploy in formato “esploso”
  - evita di ricreare da zero l'archivio WAR in caso di modifiche che non coinvolgono classi Java e descrittori
  - permette quindi di non “spegnere” e “riavviare” l'applicazione sul server (e quindi di **non perdere eventuali informazioni di sessioni attive**)
  - può richiedere di **cancellare la cache del browser** (specialmente IE)

Inoltre (opzionale ma consigliato):

- avvio del tunnel TCP/IP per monitorare il traffico HTTP in ingresso e uscita dalle pagine dell'applicazione

Apache Tomcat: struttura su file system

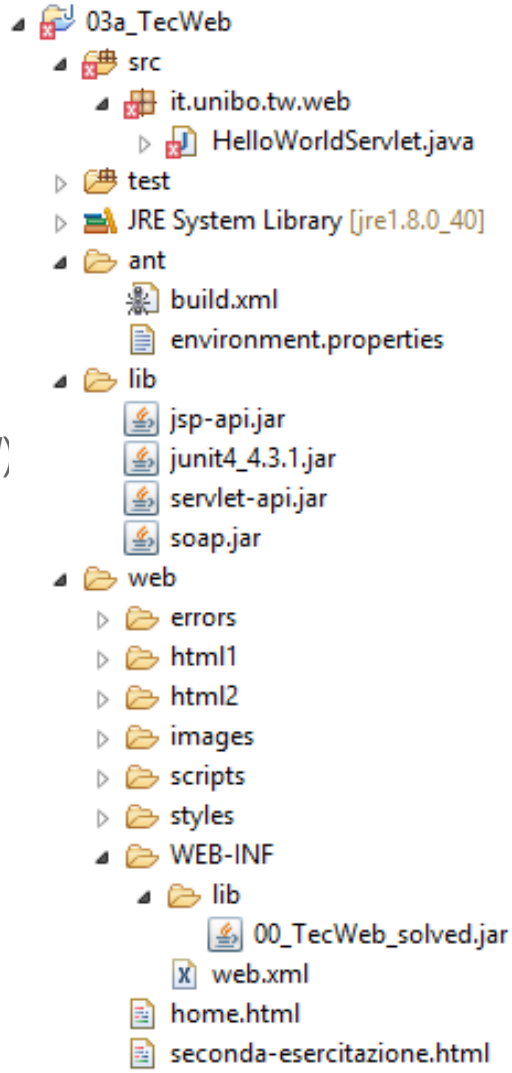
- **bin**: script e comandi di avvio
- **common**: librerie Java visibili e condivise da tutte le applicazioni Web in esecuzione sul server
- **conf**: configurazione di porte, permessi e altre risorse
- **logs**: file di log (da creare a mano se non esiste)
- **server**: codice del server
- **webapps**: pubblicazione delle applicazioni Web
- **temp**, **work**: directory per le operazioni del server (salvataggio dei dati di sessione, compilazione delle pagine JSP, ...)

Build-path

- Problemi di compilazione in Eclipse
  - aggiungere al build-path le librerie necessarie a compile-time, ma fornite dal container a run-time
    - lib/servlet-api.jar
    - ...
  - aggiungere al build-path le librerie necessarie a compile-time e da fornire al container a run-time
    - web/WEB-INF/lib/00_TecWeb_solved.jar

    (l'applicazione della prima esercitazione con *HelloWorld*)

- Il file di build **ant/build.xml** è invece in grado di funzionare perfettamente
  - il classpath usato da Ant è indipendente da quello dell'IDE e viene definito dallo stesso file di build
  - gli script di Ant possono perciò eseguire in maniera autonoma, anche in assenza di un IDE...
  - ...a patto che le proprietà relative all'ambiente di esecuzione siano impostate correttamente
    - ant/environment.properties (controllare!)



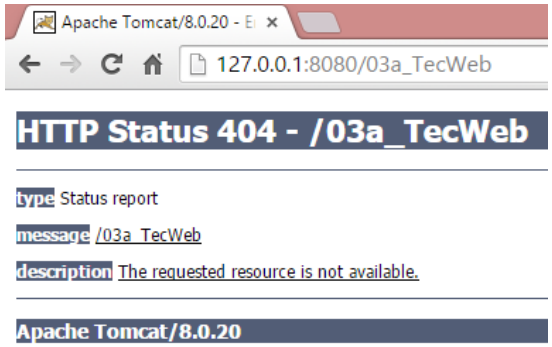
Deployment

- Il progetto contiene
  - il jar relativo alla prima esercitazione (*00_TecWeb_solved.jar*)
  - la soluzione alla seconda esercitazione (*seconda-esercitazione.html*)
  - una semplice pagina HTML iniziale che “intrattiene” l'utente intanto che le classi dell'applicazione vengono caricate in memoria, al primo accesso (*home.html*)
  - una classe che estende *HttpServlet* e riutilizza il materiale della prima esercitazione per produrre il più classico degli “hello world”
  - fogli di stile, immagini, script, pagine di errore, ...
  - un descrittore XML che specifica al Web server cosa fare con tutto ciò

- Primo passo, per il momento
  - avviare Tomcat (in modalità “esterna” a Eclipse) – nell’esercizio useremo Dynamic Web Project
    - TOMCAT_HOME/bin/startup.sh/ oppure startup.bat
  - controllare i file di log - TOMCAT_HOME/logs/catalina.x.out
  - compilare, creare WAR manualmente e pubblicare l'applicazione Web
    - se il comando **ant** è disponibile da riga di comando:
      ant -f $PROJECT_ROOT/ant/build.xml 09a.deploy.war
    - altrimenti tramite la view "Ant" di Eclipse, utilizzando il file build.xml

Primi passi

- Provate a seguire le seguenti istruzioni step-by-step
  - lanciate il server ed eseguite il deployment dell'applicazione
  - accedete alla pagina http://localhost:8080/03a_TecWeb/seconda-esercitazione.html
  - avviate il tunnel TCP/IP ed eseguite la stessa operazione attraverso il tunnel
  - cancellate il contenuto del tunnel (clear)
  - modificate il contenuto della pagina seconda-esercitazione.html e aggiornate la sua versione sul server per mezzo di Ant
  - eseguite la stessa richiesta (attraverso il tunnel)
- Sondaggio
  - quanti hanno visto passare nuovo traffico HTTP nel tunnel?
  - quanti usavano Chrome? Internet Explorer? Firefox?
- Modificate le impostazioni relative all'uso della cache (oppure cancellatela) nel browser e riprovate
- Infine, accedete al contesto Web dell'applicazione
  - http://localhost:8080/03a_TecWeb/



Riflessioni e... aspetti da sistemare

- Infine, nonostante l'archivio WAR contenga delle Servlet, il Servlet container (Tomcat) **non conosce a quali URL devono essere associate** e in mancanza di tale informazione, non può renderle disponibili

- Inoltre l'utente...
  - **non può sapere da quale pagina iniziare la navigazione**
  - **non deve ricevere messaggi di errore tecnici (404?)**

- Il Web server, generalmente, ci viene incontro...
  - presentando automaticamente le pagine di benvenuto di default, se presenti, a fronte della richiesta del solo contesto dell'applicazione Web
    - *index.html, index.jsp, ...*
    - ma per complicare le cose, la *"homepage"* di questo progetto si chiama *home.html*
  - creando pagine di errore di default, in caso di problemi

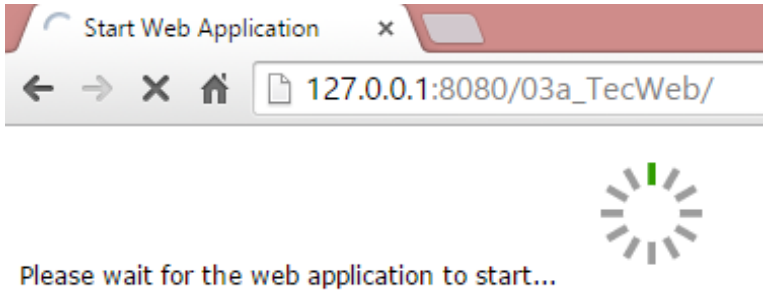
- ***I descrittori XML sono la chiave per risolvere questi problemi*** specificando al server...
  - come è fatta l'applicazione Web contenuta nel file *.war*
  - come gestire aspetti quali pagine di benvenuto ed errore, criteri di sicurezza, risorse utilizzate, ...

Modifichiamo il file web/WEB-INF/web.xml

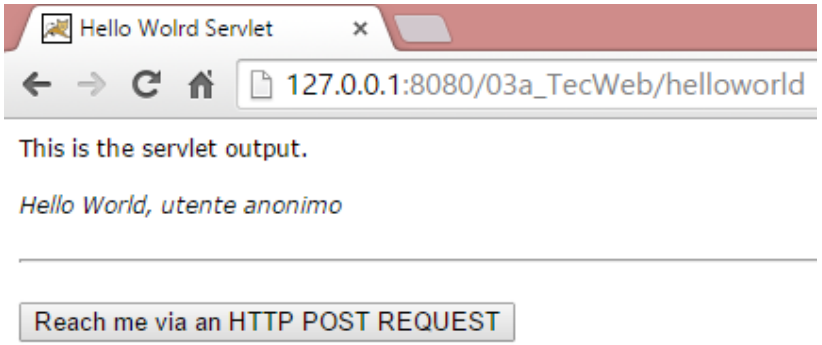
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">

   <!-- 1) General -->
   <!-- Name the application -->
   <display-name>03a_TecWeb</display-name>
   <description> A servlet-based project to use
        as a template for your owns </description>

   <!-- 2) Servlets -->
   <servlet>
      <servlet-name>HelloWorld</servlet-name>
      <servlet-class>
            it.unibo.tw.web.HelloWorldServlet
      </servlet-class>
   </servlet>

   <!-- Map some URL's to the  servlet -->
   <servlet-mapping>
      <servlet-name>HelloWorld</servlet-name>
      <url-pattern>/helloworld</url-pattern>
   </servlet-mapping>

   <!-- 3) Welcome Files -->
   <!-- Define, in order of preference, which file to
   show when no filename is defined in the path -->
   <welcome-file-list>
      <welcome-file>test.html</welcome-file>
      <welcome-file>home.html</welcome-file>
   </welcome-file-list>

   <!-- 4) Error Handler -->
   <!-- Define an error handler for 404 pages -->
   <error-page>
      <error-code>404</error-code>
      <location>/errors/notfound.html</location>
   </error-page>
   <!-- Define an handler for java.lang.Exception -->
   <error-page>
      <exception-type>
            java.lang.Exception
      </exception-type>
      <location>/errors/exception.html</location>
   </error-page>
</web-app>
```

Nuovo deployment

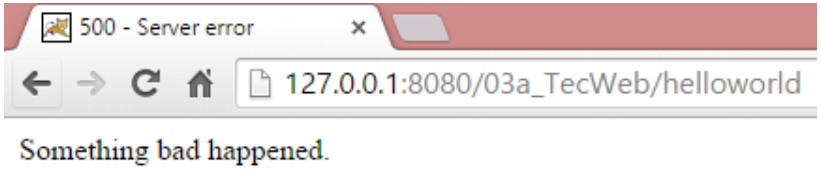
- Accesso al contesto dell'applicazione Web



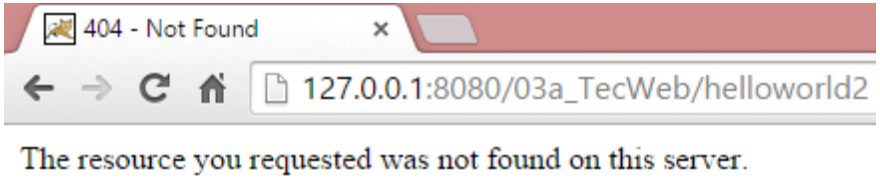
- Caricamento della servlet *hello world*



- Accesso via HTTP POST... errore (graceful)!



- Accesso a una risorsa che non esiste... errore (graceful)!



Nuovo esercizio: mantenimento dello stato

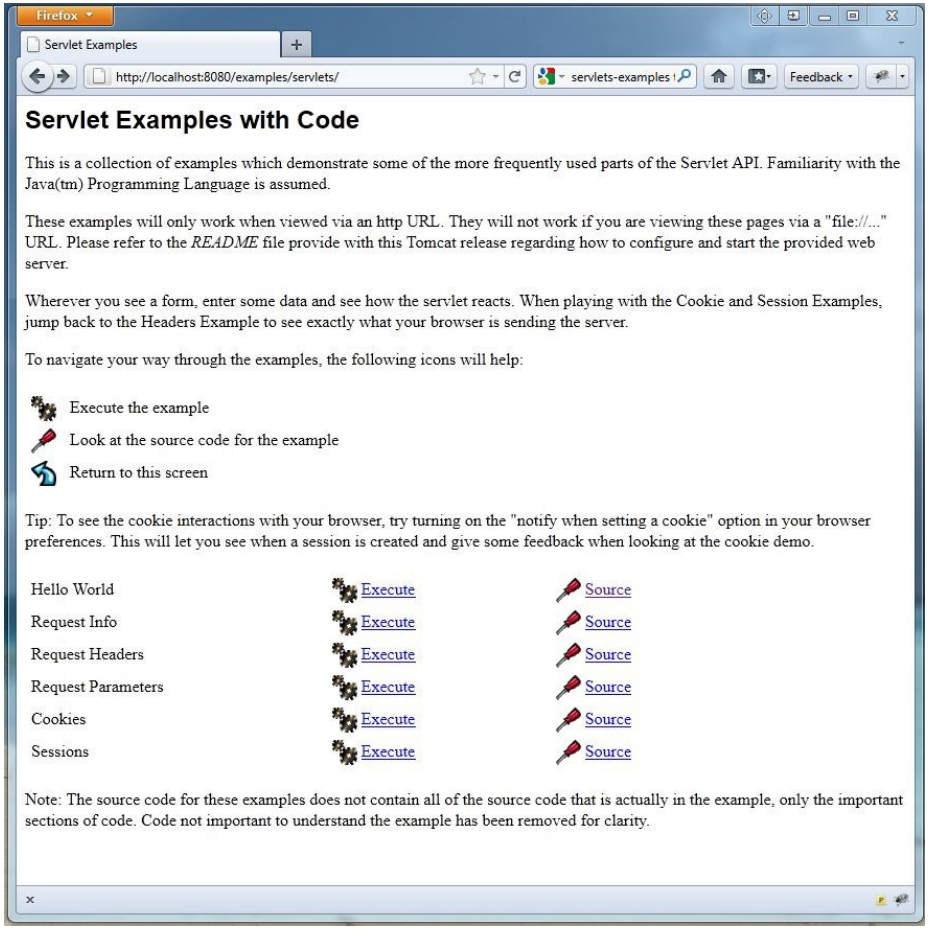
Sfruttando quanto appreso a lezione e in laboratorio

- creare un progetto Eclipse di tipo "Dynamic Web" (o modificare quello dell'esercitazione) e realizzare una Servlet in grado di servire richieste HTTP come segue
  - HTTP GET:
    - presentazione di un form per l'invio di testo al server (mediante HTTP POST)
    - valorizzazione del campo di input del form con l'eventuale testo già inviato dall'utente in precedenti interazioni con la stessa Servlet
  - HTTP POST:
    - visualizzazione del testo ricevuto nella pagina HTML di risposta
    - memorizzazione e mantenimento del testo ricevuto (stato)
- per il mantenimento dello stato, scegliere uno tra i seguenti meccanismi
  - salvataggio di attributi in **sessione**, lato server
  - salvataggio di cookie sul **browser**, lato client

Appendice 1: ulteriori esempi di Servlet

Tomcat fornisce out-of-the-box alcuni esempi relativi all'utilizzo delle Servlet (e anche JSP), **molto utili** come riferimento

- accessibili a partire da *http://localhost:8080/examples*
- funzionamento ed estratti del codice sorgente
- il codice sorgente completo è comunque disponibile su file system, nella directory di deployment che corrisponde al contesto *"examples"*



Appendice 2: Alcune linee guida sull'uso di Ant in Eclipse

Alla pagina **Laboratorio** del sito del corso *http://www-db.disi.unibo.it/courses/TW/* è disponibile un progetto Eclipse *Ant-based*

- *build.xml* include nel classpath tutti i file jar presenti nella directory **lib** → inserire in questa directory i file jar necessari in fase di **compilazione**

  il build path del progetto Eclipse (*Properties → Java Build Path*) viene **completamente ignorato** da Ant
- i file jar necessari in fase di **esecuzione** devono risiedere nella directory **web/WEB-INF/lib**, altrimenti non verranno inclusi nel file war
- ricordarsi di modificare opportunamente il file **ant/environment.properties**

Per creare una Servlet, creare una classe Java standard ed includere nel build path i file jar necessari alla compilazione delle Servlet

- ricordarsi di modificare opportunamente il file web.xml

Appendice 2: Alcune linee guida sull'uso di Ant in Eclipse

- È possibile lanciare Ant da riga di comando (se Ant è installato)
  - cd $PROJECT_HOME/ant
  - ant <nome_obiettivo>
- È possibile lanciare Ant dall'interno di Eclipse
  - *Windows → Show view → Other... → Ant → Ant*
  - trascinare il file *build.xml* nella nuova vista ed eseguire un obiettivo tramite double-click

Attenzione! quando Ant viene eseguito dall'interno di Eclipse, Ant eredita le impostazioni di Eclipse per quanto riguarda **JAVA_HOME**. Se compare l'errore *Perhaps JAVA_HOME does not point to the JDK. It is currently set to "C:\Program Files\Java\jre6",* modificare la JRE/JDK di default di Eclipse:

- *Eclipse → Windows → Preferences → Java → Installed JREs*
- in questa pagina aggiungere e selezionare una jdk al posto della jre di default

Appendice 3: Alcune linee guida sull'uso di Dynamic Web Project

- Apposita *perspective* per la creazione di applicazioni Web
  - *Windows → Open Perspective → Other... → Web*
- Creazione di un progetto Web dinamico
  - *File → New → Other... → Web → Dynamic Web Project*
  - nel wizard specificare **2.5** in *Dynamic web module version*
  - nel wizard selezionare *Generate web.xml deployment descriptor*
- Creazione Servlet/JSP
  - *New → Other... → Web → Servlet/JSP File* (file web.xml modificato automaticamente)
- Avvio di Tomcat
  - per attivare la view *Servers*: *Window → Show View → Other → Server → Servers*
  - per creare un nuovo server, view "Servers": *File → New → Other... → Server → Apache ...*
  - strumenti base: avviare/fermare Tomcat, avvio in modalità debug locale
  - deploy/undeploy di applicazioni Web: *tasto destro del mouse sul nome del server → Add and Remove...*
  - inoltre **redeploy automatico** ad ogni compilazione di servlet e/o JSP

Appendice 3: Alcune linee guida sull'uso di Dynamic Web Project

Attenzione! A default Eclipse effettua deploy delle applicazioni in una directory diversa da TOMCAT_HOME/webapps, non consente deploy tramite interfaccia Web e utilizza file di configurazione propri. **Per utilizzare proprietà e directory presenti in TOMCAT_HOME**

- fare doppio click sul nome del server
- selezionare *Use Tomcat installation (takes control of Tomcat installation)*
- tale opzione è selezionabile solo senza applicazioni Web in fase di deployment

Se il build path del progetto non contiene le librerie relative a Servlet e JSP, si hanno errori in fase di compilazione, ad esempio *The import javax.servlet cannot be resolved*.

- per aggiungere la libreria *Properties → Java Build Path → Libraries → Add Library... → Server Runtime → ...*