Universita' degli Studi di Bologna
Facolta' di Ingegneria

Anno Accademico 2008-2009

Laboratorio di Tecnologie Web

Introduzione a Tomcat

http://www-lia.deis.unibo.it/Courses/TecnologieWeb0809

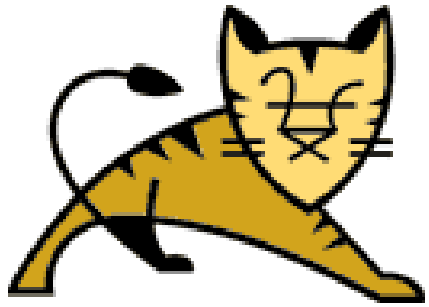
Sviluppo di applicazioni web

- JEE tutorial:
  - "*Le applicazioni Java EE sono* ***costituite da componenti***"
  - "*Un componente Java EE rappresenta una* ***unità software funzionale autocontenuta*** *che viene* ***assemblata all'interno di una applicazione enterprise*** *e che comunica con altri componenti*"
  - "*I componenti Java EE sono scritti in linguaggio Java e vengono* ***compilati allo stesso modo di ogni altro programma*** *scritto in tale linguaggio"*
  - *La differenza tra componenti Java EE e normali classi Java è che i primi sono* ***assemblati*** *all'interno di un'applicazione Java EE,* ***rispondono a precise specifiche*** *e* ***utilizzano opprtune funzionalità dell'ambiente nel quale operano****, in quanto* ***sono eseguite e gestite da un server*** *Java EE*"

(http://java.sun.com/javaee/5/docs/tutorial/doc/)

Un semplice web server JEE (Tomcat)

- Fornisce l'ambiente di esecuzione (*container*) per I componenti JEE scritti in accordo alle specifiche **Servlet** e **JSP**
- Download ed installazione (estrazione del contenuto di un file zip)
  - http://tomcat.apache.org
  - ..oppure dal sito del corso
- Struttura del server su file system
  - **webapps** → direttorio per il deploy delle applicazioni web
  - **conf** → file di configurazione (porte, permessi, …)
  - **log** → direttorio per i file di log (da creare a mano se non esiste!)
  - **bin** → file di avvio
  - **lib** → librerie che realizzano il server e le sue funzionalità (API di Servlet e JSP da aggiungere ai path in Eclipse per lo sviluppo di applicazioni web)



Avvio di Tomcat

- Lancio in background
  - `TOMCAT_HOME$ bin/startup.sh`
- Controllo dei log
  - `TOMCAT_HOME$ tail -f logs/catalina.out`
  - `TOMCAT_HOME$ tail –f logs/localhost-OGGI.log`
- Pagina principale (http://localhost:8080/)
  - Esempi
  - Link utili
- Gestione delle applicazioni web (http://localhost:8080/manager/html)
  - Credenziali in $TOMCAT_HOME/conf/users.xml

Una semplice applicazione web

- Per ottenerla…
  - …scaricare lo zip dal sito del corso, senza estrarre nulla
  - …importare come “existing project” in formato archivio
- Note sulla struttura del progetto
  - **lib** → possibili librerie aggiuntive necessarie (connettori a DB, funzionalità extra, ..)
    - a runtime sul server (verranno impacchettate insieme alla applicazione web)

      e/o

    - a buildtime in Eclipse (da aggiungere al buildpath del progetto in Eclipse e al classpath usato nei file di build di Ant)
  - **war** → direttorio che rappresenta la struttura dell’applicazione web
    - risorse già finali (descrittori XML, immagini, pagine HTML, CSS, scripts, …) collocati direttamente qui
    - risorse prodotte dalla compilazione (classi, bundle di messaggi, ..) aggiunte in maniera automatica mediante i target di Ant

Troubleshooting

- Una X rossa contrassegna il progetto…
  - …errori nel codice?
  - …librerie referenziate ma mancanti?
  - …errori nella struttura del progetto?

La vista Problems descrive l'errore: manca la cartella src, che figura come "source folder" nelle proprietà del progetto.

Il progetto non ha codice, serve solo a dimostrare la struttura che useremo per le webapp; così facendo, tuttavia, la cartella vuota non è stata inclusa nello zip! Errore!

Creiamola! *New* → *Folder* oppure via ANT: `$ ant prepare`

Compilazione, packaging, deployment via ANT

- Usiamo il file *build.xml* per
  - **compilare (`build`)**

    dipende da altri target, li esegue prima di procedere ai propri task

    nel nostro caso non c'è niente da compilare: l'applicazione consta del solo file HTML e dei descrittori XML per tomcat)
  - **deployare (`deploy.as.dir, deploy.as.war`)**

    crea un archivio *.war* che contiene le risorse ed (eventualmente) il codice da installare sul web server

    copia tale archivio (eventualmente esplodendolo) sul server
  - **undeployare (`undeploy`)**

    rimuove l'applicazione dal server cancellandone l'archivio/direttorio
- Affinché possa funzionare tutto correttamente occorre modificare i file *.properties*
  - *environment.properties* → percorsi di server e librerie sulla macchina in uso
  - *project.properties* → nome da dare al progetto per il deployment

Gestione di Tomcat attraverso ANT

- La libreria *catalina-ant.jar* definisce una serie di funzioni che possiamo richiamare attraverso ANT per intervenire sullo stato di Tomcat
  - è presente all'interno delle librerie di Tomcat
  - possiamo associare alcune sue classi a nuovi task personalizzati (quali sono queste classi lo conosciamo perché abbiamo letto la documentazione ☺)

```
<taskdef name="list"
classname="org.apache.catalina.ant.ListTask">
    <classpath>
        <path location="${appserver.home}/server/lib/catalina-
ant.jar"/>
    </classpath>
</taskdef>
```

  - e richiamare questi task all'interno dei target che ci interessano

```
<list    url="${tomcat.manager.url}"
         username="${tomcat.manager.username}"
          password="${tomcat.manager.password}"
/>
```

(quali attributi / parametri si attende il task è riportato nella documentazione)

Sebbene li definiamo con taskdef, noi invochiamo task che si basano su classi già esistenti, implementate da altri sviluppatori; il manuale di ANT spiega tuttavia anche come scrivere le classi Java per implementare task personalizzati totalmente nuovi

Usiamo ANT per costruire un tunnel/monitor TCP (1)

- La libreria *soap.jar* (v. sito del corso) contiene una classe che opera da tunnel TCP (`org.apache.soap.util.net.TcpTunnel`) ed una che mostra visivamente cosa passa nel tunnel (`org.apache.soap.util.net.TcpTunnelGui`)
  - l'HTTP si basa sulla trasmissione di caratteri! Possiamo osservare la comunicazione tra client e server (header, parametri di GET e POST, sequenza delle request/response, chunking, …)
  - per lanciare il tunnel/monitor è sufficiente esegire il metodo `main()` delle relative classi, passando come parametri la porta TCP locale di ascolto e la coppia indirizzo IP + porta TCP remota di destinazione
- Attraverso ANT possiamo
  - lanciare il `main()` attraverso il *core task* **`java`**
  - raccogliere I parametri attraverso una semplice interfaccia grafica offerta dal *core task* **`input`**

Usiamo ANT per costruire un tunnel/monitor TCP (2)

```
<target name="run.tcptunnel" >
                <input
                    message="Please enter source port (default = 8880):"
                    addproperty="source.port"
                    defaultvalue="8880" />
                <input
                    message="Please enter destination host  (default =
localhost):"
                    addproperty="destination.host"
                    defaultvalue="localhost" />
                <input
                    message="Please enter destination port  (default = 8080):"
                    addproperty="destination.port"
                    defaultvalue="8080" />
                <java
                        classname="org.apache.soap.util.net.TcpTunnelGui"
                        fork="true"
                >
                        <classpath>
                                <pathelement location="ant/lib/soap.jar"/>
                        </classpath>
                        <arg value="${source.port}"/>
                        <arg value="${destination.host}"/>
                        <arg value="${destination.port}"/>
                </java>
        </target>
```

Ora basta indicare le coordinate di un qualsiasi server (anche di uno vero!) e dirigere le richieste sulla porta locale anziché verso di esso per osservare il traffico di informazioni scambiate

Come opera il tunnel HTTP

- Ascolta su una porta locale (es: *8888*) dell'host su cui è lanciato
- Ripropone le chiamate a un endpoint remoto (es: *www.google.it:80*)
- Stabilisce una sola destinazione remota, all'avvio

```
$ java -classpath ./soap.jar
org.apache.soap.util.net.TcpTunnel 8888 www.google.it 80
```

- Può fare da tunnel per qualsiasi dato su TCP/IP, non solo lo stream di caratteri usato dal protocollo HTTP
  - ma esige che le richieste siano dirette all'endpoint locale, non remoto
  - e per quanto riguarda i link presenti sulla pagina remota…
    - …se diretti a risorse con path relativo a quello della pagina corrente continuano a venire gestiti attraverso il tunnel (che a noi appare direttamente come l'host remoto, non come quello intermedio!)
    - …se diretti a risorse con path assoluto, non passano più nel tunnel