Università degli Studi di Bologna
Facoltà di Ingegneria

Sistemi Distribuiti M
A.A. 2013 – 2014

Esercitazione Java EE
Strumenti di sviluppo

Ing. Stefano Monti
Ing. Samuele Pasini

stefano.monti@epocaricerca.it

Agenda

Parte prima - Strumenti

JBoss AS

installazione, configurazione e startup

deploy di applicazioni e amministrazione

Tomcat

installazione, configurazione e startup

deploy di applicazioni e amministrazione

MySQL

strumenti di gestione

Eclipse

Caratteristiche generali

Gestione progetti (importazione, compilazione, esecuzione,debug...)

ANT

Concetti fondamentali

Esempi di utilizzo

Agenda

Parte seconda – Introduzione alle esercitazioni

Introduzione generale

Principi generali

Dominio del problema

Come procedere

Esercitazione JPA

Obiettivi

Dettagli tecnologici

Esercitazione EJB3

Obiettivi

Dettagli tecnologici

Parte prima

Strumenti di sviluppo

JBoss Application Server

JBoss AS - Introduzione

Application Server Java EE-compliant

progetto open source

acquisito da RedHat (offre un servizio professionale di supporto, a pagamento)

Versioni *stable* (attenzione alla versione JDK 5/6)

4.2.x → *architettura Microkernel*

Microkernel basato su JMX

Servizi “agganciati” al microkernel

5.x → *architettura Microcontainer*

Supporto a deployment di POJO

Microcontainer utilizzabile al di fuori dell'AS (es. testing)

Versione *unstable (milestone 2)*

6.0 → supporto ad alcune feature di Java EE 6

JSF 2 + Servlet 3.0 (JSR 315)

Bean Validation (JSR 303)

CDI (JSR 299)

JPA 2.0 (JSR 317)

JBoss AS – Installazione

Scaricare l'archivio (.tar.gz o .zip)

Decompattare l'archivio in una opportuna directory (es. Linux /opt)

L'archivio viene espanso nelle seguenti directory (JBoss 4.2.x)

- **bin** : script vari (startup, shutdown, ecc...) e relativi file di configurazione
- **client**: librerie jar e file utili per accedere a JBoss dall'esterno (es. chiamata ad EJB)
- **docs** : documenti
  - ESEMPI di codice!!
  - DTD e XSD usati da JBoss
- **lib** : librerie jar necessarie per il funzionamento del server
- **server** :
  - differenti **configurazioni** (insiemi di servizi) dell'AS
  - ciascuna sottocartella è una configurazione

JBoss AS – Configurazioni Server

Architettura JBoss a microkernel (v.4.2.x)

- un **JMX Mbean server** iniziale (microkernel)
- uno o più servizi pluggabili

Possibilità di realizzare differenti **configurazioni**

- gruppi di servizi
- adattabilità a differenti requisiti operativi
- allo startup del server, scelta della configurazione da lanciare (opzione -c), oppure *default*
- possibilità di creare nuove configurazioni oltre a quelle preesistenti

Il direttorio *server* contiene alcune configurazioni "preconfezionate" in altrettante omonime sottocartelle

- **minimal**: solo *logging*, *JNDI Server* e *URL deployment scanner*
- **default**: configurazione tipica per app J2EE
- **all**: tutti i servizi disponibili, incluso clustering e RMI/IIOP (Corba)

JBoss AS – Esempi di Server config

...

server

...

default

***conf**: file di configurazione*

***lib**: librerie necessarie per la configurazione*

***deploy**: cartella di hot-deploy delle applicazioni*

***data**: cartella contenente dati operativi che devono sopravvivere al reboot*

***tmp**: cartella contenente dati temporanei che non sopravvivono al reboot*

***work**: cartella di lavoro per il servlet container (Tomcat)*

***log**: cartella di log del server*

all

***farm**: hot deploy per applicazioni in cluster*

***deploy-hasingleton**: hot deploy per applicazioni single node (non in cluster)*

JBoss AS – Startup e shutdown

Nella cartella bin

- **run.sh** (oppure run.bat x sist. Win)
  - opzione **-c <configurazione>**

es. ./run.sh -c all

  - opzione **-b <ind_ip>** (a default, localhost)

es. ./run.sh -b 192.168.1.100

  - a default, lancia JBoss in foreground: CTRL_C per terminare
  - È necessario impostare la variabile JAVA_HOME al JDK in uso
- **shutdown.sh -S** (oppure shutdown.bat x sist. Win)
  - termina una istanza di JBoss (es. remota e/o lanciata in background)
  - varie modalità di terminazione (vd. shutdown.sh senza parametri)
- file **run.conf** (solo sistemi Unix/Linux)

opzioni aggiuntive per la JVM, ad esempio:

  - impostare modalità debug
  - impostare heap size (max/min)
  - sistemi Win:parametri cablati direttamente in run.bat

JBoss AS – Logging

A default

- output su console
- file *<JBOSS_HOME>/server/<CONFIG>/log/server.log* con informazioni più approfondite
- uso della piattaforma Apache Log4j

E' possibile modificare le strategie di logging

- file *<JBOSS_HOME>/server/<CONFIG>/conf/jboss-log4j.xml*
- *modifiche ricaricate al volo!!*
- creazione nuovi appender (non solo file o console, anche JMS, SMTP, SNMP, ecc...)
- filtri sui log: solo determinate info su determinati appender
- proprietà varie: es. file rolling

JBoss AS – Deploy/undeploy

Hot deploy/undeploy di applicazioni: applicazioni caricate/rimosse a server attivo

Deploy == **copia** di archivi delle applicazioni (es. .ear, .war, .jar) in

<JBOSS_HOME>/server/<CONFIG>/deploy

- archivi compressi e/o decompattati

Undeploy == **rimozione** archivi delle applicazioni da

<JBOSS_HOME>/server/<CONFIG>/deploy

Alcune (molte) applicazioni già presenti, es. (config *default*)

- *jboss-web.deployer : Tomcat*
- *jmx-console.war: strumento di amministrazione*
- *Altri servizi e/o componenti (es. datasource, ...)*

Altre metodologie di deployment (es. JMX bean opportuno)

JBoss AS – Strumenti amministrazione

Alcuni basilari strumenti Web di amministrazione all'indirizzo

http://localhost:8080/

(oppure IP diverso se startup con opzione *-b*)

- Tomcat status: info su webapp caricate nel sistema
- JBoss Web Console: info e azioni sul server
- JMX Console

JMX Console (http://localhost:8080/jmx-console/)

- interfaccia Web per gli Mbean attualmente attivi nel sistema
- form HTML generate automaticamente a partire da metodi/proprietà degli Mbean
- MOLTO utile come strumento per amministrare/configurare anche le proprie applicazioni
- Tra gli Mbean esposti: JNDIView
  - registro delle applicazioni (e dei componenti) → utile per operazioni di lookup

JBoss AS - Riferimenti

Link download:

http://www.jboss.org/jbossas/downloads/

Documentazione Application Server:

http://www.jboss.org/jbossas/docs.html

Community JBoss (forum, wiki, blog, ecc...)

http://community.jboss.org/

Blog di alcuni *core developer*

http://in.relation.to/

Tomcat Servlet Container

Tomcat – Introduzione e installazione

Un Web Server, interamente scritto in Java

- ambiente di esecuzione per applicazioni Web scritte in accordo alle specifiche Java Servlet e JSP
- permette di pubblicare anche risorse HTML statiche

Installazione del server

- download dal sito ufficiale http://tomcat.apache.org
- estrazione del contenuto del file ZIP

Tomcat – Struttura su file system

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*

server: *librerie del server*

webapps: *cartella di hot deploy delle applicazioni Web*

temp, ***work***: *direttori per le operazioni del server (salvataggio dei dati di sessione, compilazione delle pagine JSP, ...)*

...

Tomcat – Startup/shutdown e debug

Nella cartella *bin*

- *startup.sh* (o startup.bat x Win)
  - startup del server
  - processo lanciato in background
  - necessità di impostare la variabile di ambiente JAVA_HOME al JRE/JDK
  - wrapper per ***catalina.sh***
  - *modalità debug*
    - *export JPDA_ADDRESS=8000*
    - *export JPDA_TRANSPORT=dt_socket*
    - *exec "$PRGDIR"/"$EXECUTABLE"* ***jpda start "$@"***
- *shutdown.sh* (o shutdown.bat x Win)
  - shutdown del server
- ***catalina.sh***
  - *lancio del server*
  - *impostazione parametri della JVM*
  - *gestisce le opzioni di avvio del server (es. debug)*

Tomcat – Configurazione

Nella cartella *conf*

- file di configurazione *server.conf*
  - *configurazione del server (es.host, porte, valvole, ecc...)*
- file di configurazione *logging.properties*
  - *gestione dei log*
- file di configurazione *web.xml*
  - *proprietà comuni a tutte le webapp caricate su questa istanza di Tomcat*

Tomcat – Deploy/undeploy

Hot deploy di applicazioni nell'istanza correntemente attiva

Due possibilità (sia per deploy che undeploy)

- locale
  - copia (rimozione) di un archivio webapp (.war) nella directory di hot deploy ***webapps***
- remota via Web
  - interfaccia web (form upload) di caricamento (o rimozione) degli archivi webapp
  - a default: http://localhost:8080/manager/html

Tomcat – Logging

Cartella *logs*

Diversi file di log (a default in *daily rolling*)

- catalina.out
  - raccoglie STDOUT e STDERR e, in generale, errori non gestiti
- *localhost<DATA>.log*
  - *log applicativi*

Tomcat – Strumenti amministrazione

http://localhost:8080/manager/html

Interfaccia web di amministrazione delle webapp caricate

- lista delle applicazioni caricate
- hot deploy/undeploy applicazioni
- stop/restart applicazioni
- gestione durata della sessione

http://localhost:8080/manager/status

Interfaccia web di gestione del servlet container

- informazioni sul HW/SW del server (RAM,JVM, Architettura, SO,...)
- statistiche di funzionamento (tempi di richiesta/risposta, byte scambiati, ecc...)

Tomcat - Riferimenti

Link download

http://tomcat.apache.org/

Tomcat user guide

http://tomcat.apache.org/tomcat-6.0-doc/index.html

MySQL

MySQL - Architettura

Server DBMS

- usualmente installato come servizio
- accessibile via rete (a default porta 3306)

Differenti *client* di gestione

- a riga di comando
- via interfaccia Web (es. PHPMyAdmin)
- via interfaccia GUI (es. MySQLAdmin)

MySQL – Client a riga di comando

A riga di comando:

- connessione

mysql -u<USER> -p (digitare la pwd quando richiesto)

- creazione database e *grant* privilegi di accesso

create database <NOME_DB>;

grant all privileges on <NOME_DB>.* to <USER>@localhost identified by '<USER>';

- selezione database ed elenco tabelle

use <NOME_DB>;

show tables;

- poi usuali query SQL

desc <NOME_TABELLA>;

select * from <NOME_TABELLA>;

truncate|drop <NOME_TABELLA>; //ATTENZIONE!!

insert ...

update ...

delete ...

MySQL – Integrazione con JBoss

Java Database Connnectivity - JDBC

- API Java standard
- feature di connessione a DBMS non dipendenti dal DB
- ciascun DBMS offre *driver* (librerie *.jar*) JDBC per il proprio DBMS server

Installare *driver* MySQL su JBoss,

necessario rendere visibile il *driver* alle app JEE che ne necessitano

→ copiare *mysql-connector-xxx.jar* in <JBOSS_HOME>/server/<CONFIG>/lib

→ analogamente per Tomcat...

Istruzioni di riferimento

- http://community.jboss.org/wiki/SetUpAMySQLDatasource

MySQL – Riferimenti

Link download

http://dev.mysql.com/downloads/mysql/5.5.4.html

Documentazione

http://dev.mysql.com/doc/refman/5.0/en/index.html

Eclipse

Eclipse - Introduzione

Ambiente integrato di sviluppo (IDE)

interamente scritto in Java

multipiattaforma (Win/Mac/Linux/…)

multilinguaggio (tool anche per il C)

open source (controllato dalla Eclipse Foundation)

Architettura basata su tecnologie core e plug-in

Fortemente modulare ed espandibile

Adattabile (e adattato!) alle più diverse esigenze attraverso l'installazione di cosiddetti “plug-in”

Eclipse - Primo impatto

Avviare Eclipse per la prima volta

scelta del direttorio per il **Workspace** (dove verranno salvati i progetti)

Welcome... eccetera: → *close*

dovesse mai servire di nuovo: *Help → Welcome*

Workbench (area di lavoro) costituita da un insieme di V**iew** (viste)

Package View (struttura logica dei progetti)

Navigator View (struttura dei file su disco)

Java Editor (scrittura del codice)

Outline View (struttura del file aperto nell'editor)

Console (stdout e stderr prodotti dalle attività eseguite)

Problems (*dove guardare quando qualcosa va storto!!!*)

…e tante altre: *Window → Show view*

Perspective (prospettiva) come associazione di un preciso insieme di viste, in precise posizioni, per affrontare determinate operazioni (codifica, debug, condivisione su SVN, ...)

Windows → Open perspective

Eclipse - Perché un IDE

Numerose funzionalità “di comodo” per velocizzare la scrittura del codice e garantire la sua correttezza a tempo di compilazione

evidenziazione (parole chiave del linguaggio, errori, …)

messaggi di errore e consigli per risoluzione (a volte automatica)

autocompletamento (parentesi, nomi delle variabili, modificatori di tipo, …): si attiva da solo dopo un istante, o su comando: *Ctrl+Space*

generazione automatica di codice (costruttori, metodi getter/setter, …)

supporto per il refactor (nomi di package, classe, metodi, variabili, ...)

...

Veramente un sacco di funzionalità

right-click dovunque :)

menu *Help → Search*

sito di eclipse, tutorial on-line (spesso persino animati)

Ricerche specifiche su Google

Eclipse - Gestione dei progetti

Creazione

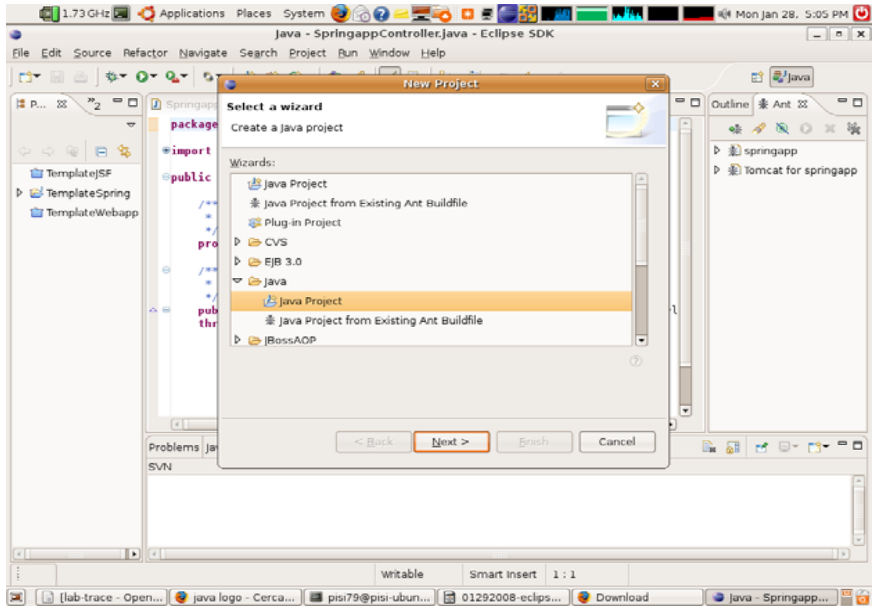
File → New → Java Project / Project…

Importazione da file zip (esempi del corso)

*File → Import → General → **Existing Projects into Workspace** → Next → **Select archive file***

***Nota Bene**: nel workspace non possono esistere più progetti con lo stesso nome!*

Occorre cancellare o rinominare quello già esistente, prima di importarne uno con lo stesso nome: diversamente, il progetto "omonimo" contenuto nel file ZIP non viene neanche visualizzato tra i progetti individuati nell'archivio



Eclipse - Debug di applicazioni “remote”

Applicazioni J2EE NON “girano” all'interno dell'IDE...

...nonostante alcune funzionalità dell'IDE inducano in errore

Per effettuare debug di applicazioni J2EE

- processo per server J2EE

lanciare JBoss abilitando opzione debug remoto...

-Xdebug -Xrunjdwp:transport=dt_socket,address=$PORT,server=y,suspend=n

- processo per IDE

creare (e poi avviare) una apposita “Debug configuration” nell'IDE

Run → Debug configurations... → Remote Java application → right-click → New...

...e indicare la stessa porta di ascolto $PORT

specificare opportuni breakpoint sul sorgente da “debuggare”

- coordinamento/comunicazione mediante socket → processi anche su macchine remote!!

Eclipse - Funzionalità Debug

Attraverso l'IDE, è possibile seguire passo-passo il flusso di un programma:

specificare opportuni ***breakpoint*** nei quali interrompere e monitorare l'esecuzione

...left-click oppure *right-click → toggle breakpoint* sulla fascia grigia a sinistra del codice, nell'editor principale

...ed eseguire il programma in modalità debug dall'interno dell'IDE stesso

right click → Debug come... → Java application sulla classe contenente il metodo main()

In caso di successo, la prospettiva corrente dell'IDE si modifica per esporre le tipiche funzionalità da debug

Possibili operazioni:

comandi *Play/Pause & STEP INTO, STEP OVER, STEP RETURN*

controllare/modificare il valore run-time delle variabili nella vista *Variables*

Eccetera... eccetera...

cambiare il valore di una variabile,

ispezionare il risultato di un'espressione,

Eclipse - Riferimenti

Link download

http://www.eclipse.org/downloads/

Documentazione Eclipse

http://eclipsetutorial.sourceforge.net/

http://eclipsetutorial.sourceforge.net/totalbeginner.html

ANT

ANT – necessità di build tool

Lo sviluppo di un'applicazione richiede di eseguire tipiche sequenze di operazioni

Scrittura del codice sorgente, compilazione, collaudo, packaging, distribuzione, ...

Tali operazioni sono ripetitive e la loro esecuzione può richiedere azioni diverse in ambienti di sviluppo diversi

Posizioni e convenzioni dei file su disco e convenzioni di nome

Posizione e nome di menu e pulsanti nei diversi ambienti di sviluppo (e spesso anche in diverse versioni dello stesso ambiente)

...

Gli strumenti di sviluppo come ANT, detti *build tool*, permettono invece di

definire una volta per tutte le operazioni da compiere

eseguire tali operazioni in maniera automatica

fare tutto questo in maniera indipendente dall'IDE utilizzato

ANT – build.xml

ANT è realizzato in Java e configurato mediante file XML

Permette di definire in maniera leggibile e facilmente modificabile un insieme di obiettivi (***target***) il cui raggiungimento permette di completare le diverse fasi di sviluppo del progetto

inizializzazione, compilazione, collaudo, packaging, ...

relazioni di dipendenza

definizione di proprietà *(**property**)* mediante variabili di tipo write-once che è possibile riferire all'interno dei diversi obiettivi

Non esistono obiettivi predefiniti, ma ciascuno è definito attraverso l'indicazione di una o più operazioni (***task***)

copia di file, compilazione, creazione di archivi, ...

ANT rende disponibili una serie di operazioni predefinite (***core task***) e prevede una serie di operazioni opzionali (***optional task****)* dipendenti da librerie di terze parti

è inoltre possibile definire nuovi “task”, attraverso apposite classi Java

ANT - Uso ai fini delle esercitazioni

Istruzioni per l'uso:

ant / build.xml: definizione degli obiettivi da completare (nonostante sia possibile modificare ed estendere tale file a piacimento, esso è concepito per poter essere usato senza alcuna modifica)

ant / environment.properties: proprietà richiamate da *build.xml* che differiscono da macchina a macchina e sono quindi DA MODIFICARE per poter completare l'esercitazione

Accorgimenti:

E' possibile lanciare *ant* da riga di comando

cd $PROJECT_HOME/ant

ant <nome_obiettivo>

E' possibile lanciare *ant* dall'interno di Eclipse (in questo caso, se ne eredita la JAVA_HOME):

Windows → Show view → Other.. → Ant → Ant

Trascinare il file *build.xml* nella nuova vista

Eseguire un obiettivo tramite *double-click*

Struttura del progetto

All'interno del direttorio radice

src: sorgente (file *.java*) dell'applicazione da sviluppare

test: sorgente delle routine di test (opzionali) che verificano il corretto funzionamento dell'applicazione

LIBRERIE *(la visualizzazione può variare da versione a versione di Eclipse)**:*** *codice fornito da terze parti necessario allo sviluppo*

JRE le classi base del runtime di Java (es: java.lang.String)

API e loro eventuali **implementazioni**. riferite dall'applicazione

ant: strumenti per l'esecuzione automatica di operazioni

compilazione, esecuzione dei test, packaging, distribuzione, ...

lib: direttorio che fisicamente contiene gli archivi *.jar* delle librerie in uso nel progetto *(nota: alcune versioni di Eclipse "nascondono" le librerie aggiunte al build-path, onde evitare di visualizzare informazioni "doppie")*

resources: altre risorse da allegare alla versione distribuibile del progetto (immagini, file multimediali, …)

tmp: direttorio per scopi temporanei

ANT - Riferimenti

Link download

http://ant.apache.org/bindownload.cgi

Documentazione

http://ant.apache.org/manual/index.html

Parte seconda

Esercitazioni

Introduzione generale

Principi generali

Uno stesso dominio applicativo alla base di tutti gli esercizi proposti

- Gestione di una biblioteca
- Estrapolato e ridotto da una applicazione Web distribuita reale

Occasione per sperimentare diverse tecnologie di integrazione

- ***Hibernate*** (mapping object-relational)
- ***EJB3*** (integrazione e remoting)

Sviluppo di componenti software in grado di eseguire

- All'interno di applicazioni Web
  - Offerta di servizi a chi effettua delle richieste
  - Esecuzione supportata da un Web server o da un application server
- All'interno di ambienti di test
  - Lotti batch di operazioni e verifica di correttezza
  - Esecuzione supportata da una suite di test
- All'interno di applicazioni stand-alone
  - Dotate di proprio *main()*
  - Ad esempio impiegate per l'inizializzazione dei database con dati di prova

Dominio del problema

Modellazione delle seguenti entità e reciproche relazioni

- *Autori* $n \leftrightarrow m$ *Libri* $n \leftrightarrow 1$ *Editori*

Necessità di disporre di componenti in grado di fornire accesso allo strato di persistenza

- Racchiudere, isolare e nascondere le modalità di accesso al database
- Astrarre le diverse tipologie di realizzazione (MySql, Hsqldb, …)
- Permettere agli altri componenti di trattare i dati in termini di *Plain Old Java Object* (POJO), senza curarsi di come questi siano mappati

Adozione del pattern DAO

- Interfacce che stabiliscono i metodi di lettura/scrittura
- Implementazione di tali interfacce in accordo a diverse tecnologie e DB
- Pattern creazionali (es: factory) per l'ottenimento delle implementazioni

http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

Obiettivi

Utilizzare le tecnologie citate per

- Implementare gli oggetti DAO veri e propri, dove necesario
  - Fornita un'implementazione JDBC di esempio
  - Richieste re-implementazioni via Hibernate e/o JPA
- Permettere la loro integrazione con le altri parti del sistema
  - Funzionalità proprie della logica di business
  - Componenti Web
  - Suite di test
  - Ambiente runtime
  - Java Console
  - …
- Verificare le potenzialità e criticità di ciascuna tecnologia
  - Innalzamento del livello di astrazione
  - Semplificazione del codice
  - Migliorata manutenibilità
  - …
  - Difficoltà di adozione
  - Aumento della complessità dell'applicazione
  - …

Come procedere

Tutti gli esercizi prendono spunto dallo stesso codice sorgente iniziale

- Modellazione del dominio del problema
  - Specifica delle interfacce DAO
  - Definizione degli oggetti POJO scambiati attraverso di esse
- Inizializzazione della base di conoscenza
  - Classi con metodi *main()*
  - Utilizzo dei pattern DAO e factory per accedere al DB
  - Scrittura, cancellazione e rilettura di informazioni predefinite
- Set minimale di test
  - Basati su JUnit4
  - “Declinabili” per ciascuna tecnologia
  - Estensibili su iniziativa dello studente
- Applicazione Web minimale
  - Una singola pagina JSP
  - Accesso al DB in lettura e scrittura
  - Presentazione di informazioni e risultati all’utente dotato di browser
  - Base per sviluppare (opzionalmente) altri componenti relativi alla logica di business e presentazione

Dettagli tecnologici

Il sorgente iniziale è fornito tramite un file ZIP contenente un progetto gestibile con ANT e/o all'interno dell'ambiente di sviluppo Eclipse

- Importazione in Eclipse (grazie alla presenza di metadati "specifici")

File → Import → General → Existing Project into Workspace →

Select archive file → [fill in the blank *or* browse] *→ Finish*

- File di build di ANT pronti all'uso

Modificare con i dati relativi alla propria macchina ***solamente***

il file `$PROJECT_ROOT/ant/environment.properties`

- Ovviamente è possibile/consigliato importare in Eclipse ***e*** lanciare i target di ANT dall'interno dell'IDE
  - sfruttare funzionalità proprie dell'IDE
    - parsing, autocompilazione, refactoring, autocompletamento, …
  - ottimizzare l'esecuzione di operazioni sempre uguali e ripetitive via ANT
    - compilazione, packaging, deployment, undeployment, …
- Uno sguardo al progetto vale più di mille parole/slide

Un esempio

In formato analogo al sorgente iniziale, viene fornita una possibile implementazione degli oggetti DAO, basata su tecnologia ***JDBC***

- API unificata
  - Diverse librerie ("connettori") forniscono i "driver" che implementano le API per l'accesso ai rispettivi tipi di database server
    - *MySql*, *Hsqldb*, *DB2*, *PostgreSQL*, …
  - Operazioni eseguite passate stringhe SQL agli oggetti
    - Il mapping tra oggetti Java e tabelle è completamente fatto "a mano"!
  - Database diversi spesso supportano "dialetti" diversi per le stesse operazioni, quindi richiedono stringhe SQL diverse
    - Una implementazione del DAO, basata su JDBC, per ciascuno!
- Il progetto d'esempio supporta
  - *MySQL* (connettore fornito, server da installare)
  - *Hsqldb* (connettore fornito, server scritto in Java e contenuto nella stessa libreria del connettore: target di ANT per avviarlo!)
- Scopo del progetto è semplicemente poter disporre di un possibile esempio di implementazione delle specifiche!

Esercitazione JPA

Es. JPA - Obiettivo 1: mapping O/R

Data la modellazione del dominio dei dati per l'applicazione Web di esempio, relativa alla gestione di *"Libri"*, *"Autori"* ed *"Editori"* in una ipotetica biblioteca…

- Utilizzare le funzionalità di JPA per permettere un mapping object-relational automatico tra oggetti Java e tabelle di database
  - per mezzo di annotazioni Java conformi allo standard JPA
- Utilizzare le funzionalità di JPA per implementare gli oggetti DAO
  - per mezzo di API compatibili con lo standard JPA e quindi basate sul concetto di `EntityManager`

Es. JPA - Obiettivo 2: transazionalità

Supponendo che ad ogni richiesta HTTP ricevuta dal Web server debba corrispondere un insieme atomico di operazioni su database, che inizia con l'ottenimento di una nuova istanza di factory DAO e termina con la restituzione della risposta al client…

- realizzare una implementazione DAO alternativa alla precedente, in cui
  - i singoli metodi degli oggetti DAO sono sollevati dalla responsabilità di dichiarare l'inizio e la fine delle transazioni al proprio interno (semantica = "*una transazione per ogni richiesta di operazione su database ai DAO*")
  - tale responsabilità è assegnata all'istanza della factory* che li ha generati (semantica = "*una transazione per ogni richiesta formulata da un client*")

* Si introduce a tal fine, per praticità, una versione estesa delle specifiche della DAOFactory, che prevede anche l'operazione di *release()* della factory stessa, oltre a quella di *get()*.

Es. JPA – Dettagli tecnologici

Il progetto contenente il codice su cui basare lo sviluppo…

- Contiene una applicazione Web minimale, una suite di test e un insieme di classi per l'inizializzazione della base di conoscenza
- Permette, attraverso ANT, di eseguire il deployment della applicazione su un'installazione del **Web Server Tomcat**
- Contiene una versione estesa delle specifiche della factory DAO
  - Metodi per l'ottenimento e la restituzione delle factory concrete
  - Obbligo per le factory concrete di implementare un metodo per la terminazione delle transazioni
- Contiene versioni modificate dei test, per gestire le semantiche transazionali

Es. JPA - Riferimenti

JPA Tutorial

http://java.sun.com/developer/technicalArticles/J2EE/jpa/

JPA Implementation patterns: Saving (detached) entities

http://blog.xebia.com/2009/03/23/jpa-implementation-patterns-saving-detached-entities/

Esercitazione EJB 3

Es. EJB3 – Obiettivo 1: mapping O/R

Data l'applicazione Web di esempio, per la gestione di *"Libri"*, *"Autori"* ed *"Editori"* in una ipotetica biblioteca…

Realizzare la parte di logica di accesso al database (pattern DAO) mediante componenti Enterprise Java Beans 3.0 (server JBoss AS).

In particolare:

- Utilizzare mapping Object-Relational tramite componenti Entity Bean
- Realizzare DAO tramite opportuni componenti Session Bean

Es. EJB3 – Obiettivo 2: uso JMS

Inoltre, estendere la logica applicativa già descritta, aggiungendo un meccanismo di logging (potenzialmente) remoto:

- Ciascun metodo che preveda scritture su DB (aggiunta di nuovi libri, autori, ecc…) deve inviare un messaggio JMS ad opportuno componente di logging
- Componente di logging (realizzato come Message Driven Bean) scrive su opportuno log (anche solo stdout) il messaggio ricevuto

Es. EJB3 – Criticità ed estensioni

- Gestire correttamente il mapping O-R tramite (annotazioni) JPA
- Selezionare la tipologia di Session Bean più adatta alla realizzazione di componenti DAO
- Gestire correttamente il tipo di modello di messaggistica (a coda o a topic) per il componente di logging
- Considerare la possibilità di utilizzare Interceptor per “decorare” con logica di invio messaggi i metodi DAO interessati (scrittura su DB) invece di cablare la logica di invio all’interno del metodo stesso

Es. EJB3 – Dettagli tecnologici

Il progetto contenente il codice su cui basare lo sviluppo…

- Contiene una applicazione Web minimale
- Permette, attraverso ANT, di eseguire il deployment della applicazione su un'installazione dell'application Server JBoss (in versione 4.2.x)
- Contiene scheletri degli opportuni descrittori XML per
  - Deployment della applicazione con archivio in formato .EAR
  - Binding della base di dati a database MySQL
- Una suite di test per verificare la correttezza delle routine DAO
  - Le routine di test invocano direttamente i componenti all'interno dell'application server: quale impatto sull'uso di interfacce locali/remote?

Es. EJB3 – Riferimenti

Java Message Service Tutorial

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

JBoss MDB examples

http://www.mastertheboss.com/en/jboss-server/69-jboss-mdb.html

http://www.jboss.org/jbossejb3/docs/tutorial/mdb/mdb.html

Interceptors

http://www.jboss.org/jbossejb3/docs/tutorial/interceptor/interceptor.html

Note specifiche alle installazioni in Lab2

Installazione “da produzione”

server JBoss, Tomcat e MySQL installati come servizi (su S.O. GNU/Linux)

Mysql (v. 5.1)

Avvio e shutdown: sudo /etc/init.d/mysql start (oppure stop)

DB per le esercitazioni: sd10db

Utente: sd10user

Password: sd10pwd

Tomcat (v. 6)

Dir di installazione: /opt/tomcat6

Cartella di hot deploy: /opt/tomcat6/deploy

Avvio e shutdown: sudo /etc/init.d/tomcat6 start (oppure stop)

JBoss (v. 4.2.3)

Dir di installazione: /opt/jboss-4.2.3.GA

Cartella di hot deploy:/opt/jboss-4.2.3.GA/server/default/deploy

Avvio e shutdown: sudo /etc/init.d/jboss start