JAVA ED IL WEB PER GENTE PRATICA

scrivere con Regola

Autori:
Nicola Santi, Marco Rosi

Supervisore:
Lorenzo Bragaglia

16 aprile 2008

Indice

Prefazione

Capitolo 1

Getting Started

Questo capitolo è una cura per gli impazienti; seguendo le istruzioni dei paragrafi seguenti sarete in grado di installare e predisporre una prima applicazioni web con Regola.

1.1 Installare Regola kit

Le applicazioni realizzate con Regola kit utilizzano Maven 2 per tutta la gestione del ciclo di build (compilazione, esecuzione dei test, creazione dei file war, ...). Quindi come prima cosa bisogna installare Maven 2 scaricandolo dal sito maven.apache.org e seguendo le istruzioni.

Finita l'installazione di Maven 2 verificate che tutto sia andato a buon fine aprendo una console di terminale e lanciando il seguente comando.

```
1 nicola@casper:~# mvn -version
  Maven version: 2.0.8
3 Java version: 1.6.0_03
  OS name: "linux" version: "2.6.22-14-generic" arch: "i386" Family:
  "unix"
```

Se tutto è corretto Maven 2 risponde al comando restituendo la sua versione, quella di Java ed infine alcune informazioni circa il sistema operativo in uso.

Regola kit non richiede nessuna installazione particolare (anche se è possibile scaricare un pacchetto contente documentazione e comandi di utilità): quindi finita l'installazione di Maven 2 siete pronti già per utilizzare Regola kit.

■ Per maggiori informazioni su come installare Regola kit sulle vostre macchine di sviluppo si rimanda al capitolo 2 nella pagina 13. ■

1.2 Predisporre il database

Per questo tutorial ipotizziamo di avere a disposizione un database di tipo MySql già installato sulla macchina dove intendiamo creare il progetto. Assicuratevi che il database stia funzionando e digitate il comando seguente:

```
nicola@casper:~# mysql -u root -p
```

Vi troverete dentro la shell di amministrazione di MySql. Approfittatene per creare un nuovo database che sarà utilizzato dall'applicazione digitando il comando.

```
1 mysql> create database clienti;
```

(Attenzione al punto e virgola in fondo al comando). A questo punto non ci resta che creare anche l'utente utilizzato dalla nostra applicazione per accedere al database (nell'esempio porta il mio nome *nicola*).

```
1 mysql> grant all on clienti.* to 'nicola'@'localhost';
```

Bene, con il database abbiamo finito. Digitate questo ultimo comando per uscire dalla shell di amministrazione di MySql e passare al paragrafo seguente.

```
1 mysql> exit
```

■ Regola kit è in grado di utilizzare diversi DBMS (ad esempio Oracle, Microsoft Sql, PostgreSQL, Hypersonic, ...). Per sapere come configurare la vostra applicazione per utilizzare DBMS diversi da MySql si rimanda al capitolo 5 nella pagina 19. ■

1.3 Creare un progetto con Regola kit

Posizionatevi nella cartella dove volete creare il vostro nuovo progetto e digitate un comando simile al seguente con l'accortezza però di modificare il parametro gruopId (nell'esempio *com.acme*) con il nome del vostro package di default ed il parametro artifactId (nell'esempio *clienti*) con il nome del nuovo progetto.

```
1 nicola@casper:~# mvn archetype:create -DarchetypeGroupId=org.regola \
-DarchetypeArtifactId=regola-jsf-archetype \
3 -DarchetypeVersion=1.1-SNAPSHOT -DgroupId=com.acme \
-DartifactId=clienti
```

Attenzione: il comando qui sopra è riparito su diverse righe per chiarezza tipografica, deve invece essere digitato su una sola riga.

La prima volta che lanciate questo comando Maven 2 scarica tutte le librerie necessarie (la cosa potrebbe prendere un po' di tempo) e crea una sotto cartella col nome del progetto (nell'esempio la cartella clienti). Il progetto è questo punto è già stato creato, posizioniatevi all'interno della cartella *clienti* col comando:

```
nicola@casper:~# cd clienti
```

1.4 Collegarsi al database

Prima di lanciare la nostra nuova applicazione è necessario informarla circa le coordinate del database da utilizzare, per farlo bisogna apportare un modifica al file src/test/resources/jetty/env.xml con il vostro editor di testo preferito. Dovete inserire cambiare solo il nome del database (alla riga 6) e lo username (riga 8) da utilizzare per ottenere qualcosa di simile al frammento di xml seguente:

```
1  ...
   <New id="jira-ds" class="org.mortbay.jetty.plus.naming.Resource">
3    <Arg>jdbc/Datasource</Arg>
     <Arg>
5      <New
       class="org.enhydra.jdbc.standard.StandardConnectionPoolDataSource">
         <Set name="Url">jdbc:mysql://localhost/clienti</Set>
7        <Set name="DriverName">com.mysql.jdbc.Driver</Set>
         <Set name="User">nicola</Set>
9      </New>
     </Arg>
11 </New>
   ...
```

■ Per avere maggiori informazioni sulle diverse configurazioni relative alle connessioni al database si rimanda al capitolo 4 ■

1.5 Avviare l'applicazione

Ora tutto è pronto per avviare l'applicazione, se avete lasciato la cartella principale del progetto tornateci e da lì lanciate il comando seguente (e lasciate aperta la console):

```
nicola@casper:~/projects/clienti# mvn jetty:run
```

Sullo schermo si susseguiranno diverse righe per informarvi che l'applicazione è stata inizializzata e quando, infine, apparirà la dicitura *Started Jetty Server* saprete che tutto è pronto.

Lasciando sempre aperta la console aprite un'istanza del vostro browser e collegatevi all'indirizzo localhost:8080/clienti per vedere la pagina di benvenuto della vostra applicazione.

Complimenti, avete appena fatto il primo passo nel mondo delle applicazioni Regola kit!

■ Per visualizzare l'applicazione state utilizzando un piccolo (ma molto completo) application server di nome Jetty. Per la messa in produzione però si consiglia di utilizzare dei container diversi (ad esempio Tomcat o JBoss). Per imparare a come creare i pacchetti per questi application server si rimanda al capitolo 6 nella pagina 21 ■

1.6 Struttura di un progetto Regola kit

La struttura della cartella di un progetto Regola kit si impronta alla struttra standard di un progetto web di Maven 2. Al primo livello troviamo:

pom.xml	il file di configurazione di Maven 2
src/	la cartella dei sorgenti
target/	contiene i file compilati ed i pacchetti per le consegne

La cartella target contiene quanto i pacchetti pronti per la consegna con la classi compilate ed i descrittori. Si tratta di una cartella il cui contenuto è ricreato ogni volta si lanci il comando:

```
1 nicola@casper:~/projects/clienti# mvn package
```

La cartella src contiene i sorgenti (html, js, css e java) dell'applicazione. Al suo interno potete trovare:

main/	contiene i sorgenti dell'appplicazione
test/	contiene i sorgenti dei test
site/	reportistica generata da Maven 2

La cartella main e la cartella test contengono entrambe i sorgenti java (nella sottocartella java) e le altre risorse (nella cartella risorse). Queste ultime sono i file di configurazione, i mappaggi orm e, in generale, tutto quello che non sono sorgenti Java ma devono finire comunque nel classpath. La differenza tra la cartella main e quella test e che il contenuto di quest'ultima non finisce mai nei pacchetti destinati alla produzione ma è usato esclusivamente per l'esecuzione dei test. Infine la cartella main contiene anche la sottocartella webapp dove si trova la webroot, ovvero le pagine web dell'applicazione ed il file web.xml.

■ Per una descrizione completa dei file e delle cartelle standard di un progetto Regola kit si rimanda al capitolo 3 nella pagina 15 ■

1.7 Persistenza su database

Regola kit abbraccia una metodologia di sviluppo incentrata sull'analisi del dominio del problema (Domain Driven Development) per cui il primo passo è quello di creare le classi di modello. Spesso queste classi sono persistite sul database per cui si potrebbe iniziare scrivendo la classe e poi creando la corrispondente tabella sul database. Oppure, al contrario come faremo tra poco, creando prima la tabella del database e facendoci poi creare in automatico la classe Java (nel prossimo paragrafo 1.9 nella pagina 10). Colleghiamoci nuovamente al database clienti.

```
1 nicola@casper:~/projects/clienti# mysql -u root -p clienti
```

Creiamo una piccola tabella con la sola chiave primaria (id) ed un campo di descrizione (label).

```
mysql> create table prodotti (id int(11) not null auto_increment,
label varchar(80) not null, primary key (id) );
mysql> describe prodotti;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| label | varchar(80) | NO   |     |         |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.02 sec)
```

Inseriamo qualche dato nella tabella, ad esempio alcune descrizioni di esempio per verificare poi il funzionamento dell'applicazione.

```
mysql> insert into prodotti values (null, 'book');
Query OK, 1 row affected (0.05 sec)

mysql> insert into prodotti values (null, 'bottle');
Query OK, 1 row affected (0.00 sec)

mysql> insert into prodotti values (null, 'paper');
Query OK, 1 row affected (0.00 sec)

mysql> select * from prodotti;
+----+--------+
| id | label  |
+----+--------+
|  1 | book   |
|  2 | bottle |
|  3 | paper  |
+----+--------+
3 rows in set (0.01 sec)
```

Adesso il database contiene una tabella con dei dati che possiamo usare per persistere le nostre classi di modello. Usciamo dal database e torniamo all'applicazione.

```
mysql> exit
```

1.8 (Ri)collegarsi al database

Al paragrafo 1.4 nella pagina 7 abbiamo configurato l'applicazione per utilizzare il nostro database in fase di esecuzione. Adesso dobbiamo fare in modo che anche in fase di sviluppo si possa accedere al database (ad esempio per usare i generatori di codice o lanciare la batteria di test). Il file da modificare è src/test/resources/designtime.properties e deve essere aggiornato in modo da contenere lo username, la password ed il nome del database. Il risultato finale deve risultare simile al seguente:

```
...
hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.connection.driver_class = com.mysql.jdbc.Driver
hibernate.connection.url = jdbc:mysql://localhost/clienti
hibernate.connection.username = nicola
```

```
hibernate.connection.password =
...
```

1.9 Classi di modello

Siete ora in grado di scrivere la classe di modello che sarà persistita sulla tabella prodotti... oppure potete farvela generare automaticamente e poi modificare convenientemente le classi prodotte. Userete gli Hibernate Tools che sono già configurati all'interno delle applicazioni prodotte con Regola kit e trovano nell'unico file src/test/resources/hibernate.reveng.xml la configurazione di tutto il processo di generazione inversa, a partire cioè dal database. Specificate il nome della tabella da cui partire (*prodotti* alla riga 2), il nome della classe (*Prodotto*, al singolare e con la prima lettera maiuscola nella riga 4), il package da utilizzare (*com.acme.model* sempre alla riga 2).

```
1 ...
  <table-filter match-name="prodotti"  package="com.acme.model"
  exclude="false"/>
3
   <table name="prodotti" class="Prodotto" >
5     <primary-key property="id" />
   </table>
7 ...
```

Ora avviate la generazione: posizionantevi nella cartella principale del vostro progetto ed utilizzate il plugin di Maven 2 Hibernate3 che consente di generare le classi java (il goal hbm2java), i file di mappaggio di hibernate (hbm2hbmxml) e la configurazione generale di hibernate (hbm2cfgxml).

```
1 nicola@casper:~/projects/clienti# mvn hibernate3:hbm2java
  hibernate3:hbm2hbmxml hibernate3:hbm2cfgxml
```

Il primo file generato si trova nella posizione src/main/java/com/acme/-model/Prodotto.java e contiene la classe Java:

```
1 public class Prodotto  implements java.io.Serializable {
3     private Integer id;
5     public Prodotto() {
      }
7
      public Integer getId() {
9         return this.id;
      }
11
      public void setId(Integer id) {
13        this.id = id;
      }
15 }
```

Poi è stato creato il file con i mappaggi di hibernate src/main/resource-s/com/acme/model/Prodotto.hbm.xml, molto semplice in questo caso:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 14-apr-2008 12.23.38 by Hibernate Tools 3.2.0.CR1 -->
<hibernate-mapping>
    <class name="com.acme.model.Prodotto" table="prodotti"
    catalog="clienti">
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="identity" />
        </id>
    </class>
</hibernate-mapping>
```

Ed infine è stato inserito un riferimento a quest'ultimo file di mappaggio dentro la configuraizone principale di Hibernate src/main/resources/hibernate.cfg.xml (alla riga 13).

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
        name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.password"></property>
        <property
        name="hibernate.connection.url">jdbc:mysql://localhost/clienti</property>
        <property
        name="hibernate.connection.username">nicola</property>
        <property
        name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

       <mapping resource="com/acme/model/Prodotti.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

Attenzione: il goal hibernate3:hbm2cfgxml cancella e riscrive ogni volta questo file ed inoltre vi aggiunge delle configurazioni che a runtime non sono usate (come username, password, url e driver class). Nell'impiego di tutti i giorni di Regola kit il nostro team di sviluppo non utilizza il goal hibernate3:hbm2cfgxml e si occupa di aggiungere manualmente i mappaggi delle risorse. Naturalmente le configurazioni non usate non costituiscono problema, per cui alla fine la scelta di impiegare o meno hibernate3:hbm2cfgxml è lasciata alla vostra discrezione.

■ Esistono altri goal disponibili, ad esempio la generazione degli script sql per creare le tabella a partire dalla configurazione delle classi. Si rimanda, per approfondimenti, al capitolo 5 nella pagina 19. ■

1.10 Dal modello alla presentazione

Adesso che avete creato la classe di modello è il momento di realizzare il codice per leggere e scrivere oggetti (della classe Prodotto) sul database,

le pagine web che elenchino questi oggetti così come la pagine di dettaglio per effettuare modifiche. Questo codice può essere scritto a mano oppure potete partire facendovi generare automaticamente della classi di default che utilizzerete come modello di partenza per le vostre modifiche.

Per avviare il generatore di Regola kit utilizzate il seguente comando:

```
1 nicola@casper:~/projects/clienti# mvn exec:java -Dexec.args="-c
com.acme.model.Prodotto -m"
```

Noterete tra i parametri passati al comando il nome della classe attorno a cui costruire i vari livelli e l'opzione *m* che specifica di utilizzare un'ampia catena di generatori, in particolare:

dao	produce il custom dao
modelPattern	produce la classe necessaria a Model Pattern
properties	aggiunge le chiavi per la localizzazione
list-handler	genera il controller dietro la pagina di lista
list	genera la pagina di lista
form-handler	genera il controller dietro la pagina di dettaglio
form	genera la pagina di dettaglio

■ I generatori possono anche essere avviati individualmente. Per scoprire come e conoscere anche altri generatori forniti con Regola kit si rimanda al capitolo 7.4 nella pagina 23. ■

Capitolo 2

Installazione

2.1 Maven 2

TODO: Qui si spiega che Regola kit si fonda su Maven 2 per tutta la gestione del ciclo di build.

2.2 Librerie

TODO: Dove si elencano i moduli di cui si compone

2.3 Dipendenze

TODO: Dove si elencano le dipendenze, ricordando però che sono gestite da Maven 2

2.4 Eclipse IDE

TODO: Come lavorare ad un progetto di Regola utilizzando Eclipse 3.3 ed i plugin necessari per il funzionamento

Capitolo 3

Struttura di un progetto

Questo capitolo funziona un po' come una cartina stradale e rimanda ad altri capitoli per l'approfondimento.

/	pom.xml	il file principale di Maven 2
src/main/resources/	runtime.properties	proprietà di runtime
src/main/resources/	log4j.xml	la verbosità dei log
src/main/resources/	hibernate.cfg.xml	la configurazione principale di Hibernate
src/main/resources/	ApplicationResources.properties	i testi tradotti nelle varie lingue
src/main/resources/	applicationContext-*.xml	le configurazioni di Spring
src/main/resources/	<stesse cartelle dei packages>	i mappaggi di Hibernate
src/test/resources/	designtime.properties	proprietà a design time
src/test/resources/	log4j.xml	la verbosità dei log
src/main/resources/	hibernate.reveng.xml	Hibernate tools
src/main/resources/	applicationContext-*-test.xml	le configurazioni di Spring
src/main/resources/	jetty/env.xml	datasource per Jetty

3.1 Lo standard Maven 2

TODO: Si descrive lo standard utilizzato e si fa una prima panoramica delle cartelle coinvolte

3.2 L'iniezione delle dipendenze

TODO: La distinzione tra i tre livelli di bean

3.3 La localizzazione

TODO: I file di localizzazione

3.4 Le connessioni al database

TODO: I file di localizzazione

3.5 Verbosità dei log

TODO: log4j e come interagisce con JBoss

3.6 La sezione dei test

TODO: come configurare i test e quali file utilizzano.

3.7 Applicazione Servlet

TODO: Davvero in breve la struttura

Capitolo 4

Database

4.1 Configurazione di run-time

TODO: si spiegano i datasource e perché conviene usarli. Si rimanda al capitolo della Messa In Produzione per esempi di configurazione con Tomcat e JBoss e si spiega come configurare Jetty.

4.2 Configurazione di design-time

TODO: come realizzarla e come funziona.

Capitolo 5

Persistenza

5.1 Hibernate

TODO: Qui si dice che l'orm predefinito da Regola kit è Hibernate, perché è estremamente flessibile e su database preesistenti consente di gestire anche i casi più anomali. Perché non usiamo ancora JPA (immaturo).

5.2 Configurazione

TODO: I file hibernate.cfg.xml ed i mappaggi. Alcuni esempi di base per questi file.

5.3 Generatori automatici

TODO: Esempi di configurazione di hibernate.reveng.xml ed esempi (tutti) dei vari goals disponibili a riga di comando.

5.4 Altri ORM

TODO: Si parla del supporto sperimentale per gli altri orm.

Capitolo 6

Messa in produzione

6.1 Produrre i pacchetti war ed ear

TODO: Qui si spiega come produrre i pacchetti e cosa contengano al loro interno

6.2 Application Server

TODO: Perché non usare Jetty ed esempi di configurazione per Tomcat e JBoss. Si rimanda al capitolo dei database per la configurazione dei datasource.

6.3 Integrazione continua

TODO: Perché serve e come configurare i file standard dei progetti di Regola kit

Capitolo 7

Sviluppo

Questo capitolo apre la parte dedicata alle funzionalità offerte da Regola kit relativamente allo sviluppo. Lasciandosi alle spalle le configurazioni.

7.1 Domain Driven Development

TODO: Molto in breve si spiega come ci si incentri sul modello.

7.2 Livelli

TODO: Si presentano i vari livelli e si rimanda a capitoli successivi per i dettagli

7.3 Model Pattern

TODO: Si descrive la principale innovazione di Regola kit nell'ambito dello sviluppo (cioè in tutto quello che non riguarda sistemistica e configurazione). Qui si descrive l'idea di base del Model Pattern che sarà poi dettagliato nei paragrafi successi.

7.4 Generatori

TODO: Qui si elencano i generatori disponibili e cosa scrivano. Se il paragrafo diventasse troppo lungo allora lo mettiamo su un capitolo a parte.

Capitolo 8

Dao

8.1 Scopo

TODO: A cosa serve DAO?

8.2 GenericDao

TODO: Se ne descrivono interfacce e si presentano esempi d'uso (dei test d'unità) che si snodano tra i vari paragrafi.

8.3 Creare un custom dao

TODO: l'interfaccia, l'implementazione ed infine la configurazione di Spring. Si ricorda che esiste un generatore per questo.

8.4 Ricerche con Model Pattern

TODO: Qui si spiega per filo e per segno tutte le configurazioni di una classe ModelPattern con tutte le annotazioni e un bel po' di esempi.

Capitolo 9

Servizio

9.1 Scopo

TODO: A cosa serve il livello di Servizio?

9.2 GenericManager

TODO: Come crearlo. Si ricorda che esiste un generatore per questo.

9.3 Transazioni

TODO: come sono demarcate e come aggiungere politiche diverse per aprire e chiudere transazioni

9.4 Politiche di detach

TODO: come usare oggetti detacciati, la sessione lunga, il lazy loading ecc.

9.5 Web Services

TODO: esempi di configurazioni già pronte dentro Regola kit

Capitolo 10

Presentazione Web

10.1 Scopo

TODO: Di cosa si occupa questo livello?

10.2 Tecnologie

TODO: Panoramica brevissima su JSF, Spring WebFlow e Spring MVC.

10.3 Pagina di lista

TODO: Partendo da un esempio si provvede a spiegare quali sono i file coinvolti e come devono essere configurati. Si ricorda che esiste un generatore per questo.

10.4 Pagina di form

TODO: Partendo da un esempio si provvede a spiegare quali sono i file coinvolti e come devono essere configurati. Si ricorda che esiste un generatore per questo.

Bibliografia

Indice analitico