RESTful Web Service

Andrea Pinazzi

10 Maggio 2010

Che cos'è REST?

REST, Representational State Transfer, è uno stile architetturale per sistemi software distribuiti. Il termine è stato introdotto e definito nel 2000 da Roy Fielding, uno dei principali autori delle specifiche del protocollo HTTP. Indica una serie di principi architetturali per la progettazione di Web Service.

Resources

Concetto centrale per un sistema RESTful è quello di risorsa. Una risorsa è qualunque entità che possa essere indirizzabile tramite Web, cioè accessibile e trasferibile tra client e server. Spesso rappresenta un oggetto appartenente al dominio del problema che stiamo trattando.

- un articolo di un sito di notizie
- uno studente di una qualche università
- un'immagine in una web gallery
- gli ultimi 20 tweets di un certo utente
- ...

Resources

Durante un'interazione tra client e server quello che viene trasferito è una rappresentazione dello stato interno della risorsa. Ad esempio l'ultimo post in un blog può essere servito ad un browser come una pagina in formato html, mentre ad un news reader come un documento xml.

HTTP

Inizialmente REST venne descritto da Fielding nel contesto del protocollo HTTP e questo è il protocollo a livello Applicazione maggiormente utilizzato; ma un sistema RESTful si può appoggiare ad un qualunque altro protocollo che fornisca un vocabolario altrettando ricco.
A differenza di altre specifiche per Web Service (es. SOAP) REST sfrutta infatti a pieno la semantica e la ricchezza dei comandi HTTP e le sue funzionalità come, ad esempio, la negoziazione dei contenuti.

HTTP, esempi

Richiesta	Significato
GET articles/3	recupera una rappresentazione dell'articolo desiderato
POST articles/	crea un nuovo articolo, la cui rappresentazione è nel body della richiesta
PUT articles/54	modifica l'articolo esistente
DELETE articles/	rimuove l'intera collezione di articoli

Vincoli

Perchè un Web Service sia conforme alle specifiche REST deve avere alcune specifiche caratteristiche:

- architettura basata su client e server
- stateless. Ogni ciclo di request\response deve rappresentare un'interazione completa del client con il server. In questo modo non è necessario per mantenere informazioni sulla sessione utente, minimizzando l'uso di memoria del server e la sua complessità.
- uniformemente accessibile. Ogni risorsa deve avere un'indirizzo univoco e ogni risorsa di ogni sistema presenta la stessa interfaccia, precisamente quella individuata dal protocollo HTTP.

Jersey REST Framework

Jersey è l'implementazione di riferimento della specifica JAX-RS (JSR 311) per la realizzazione di Web Service RESTful su piattaforma Java.
Jersey permette di creare risorse semplicemente così come sviluppiamo POJO (Plain Old Java Object) utilizzando delle specifiche annotazioni per i metodi e le classi. In altre parole il framework si occupa di gestire le richieste HTTP e la negoziazione della rappresentazione e noi possiamo concentrarci alla soluzione del problema.
Per utilizzarlo è sufficiente che siano disponibili nel classpath le librerie jersey-core.jar, jersey-server.jar, jsr311-api.jar e asm.jar.

HelloWorldResource.java

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// class will be at addressable at the URI "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {
  // The java method will process HTTP GET requests
  @GET
  /* The Java method will produce content identified by the
   * MIME Media type "text/plain"
   */
  @Produces("text/plain")
  public String getMessage() {
    return "Hello World";
  }
}
```

HelloWorldResource.java

HelloWorldResource è il più semplice esempio di risorsa realizzabile tramite Jersey. Si tratta di una semplice classe java e il framework ricava informazioni sulla sua gestione tramite le annotazioni utilizzate.

- @Path indica la URI a cui la risorsa sarà raggiungibile, in questo caso sarà, ad esempio `http://www.example.com/helloworld`.
  L'annotazione è posta al livello della dichiarazione della classe, quindi helloworld sarà il path di base per tutti i suoi metodi.
- @Produces specifica uno o più MIME-Type con cui lo stato della risorsa può essere trasferito al client che ne fa richiesta. Se il client non ne supporta neanche uno il framework risponderà con l'opportuno codice di errore HTTP.
- @GET evidenzia il comando HTTP che il metodo è incaricato di gestire.

HelloWorldResource.java, HTTP Request and Response

```
GET /helloworld HTTP/1.1
User-Agent: curl/7.20.0 (i686-pc-linux-gnu)
Host: localhost:9999
Accept: */*

HTTP/1.1 200 OK
server: grizzly/1.9.8
Content-Type: text/plain
Transfer-Encoding: chunked
Date: Tue, 27 Apr 2010 14:05:54 GMT

Hello World
```

Deploy

Il framework Jersey comprende, oltre all'implementazione di JAX RS, anche vari connettori per diverse tipologie di web server e application server. Ad esempio è possibile utilizzarlo in un'istanza embedded di Grizzly oppure all'interno di una web application servita da Tomcat oppure GlassFish. Da notare che nel caso in cui l'applicazione RESTful fosse messa in produzione come servlet allora i vari @Path sarebbero da intendersi relativi al mapping della stessa servlet: l'HelloWorldResource vista in precedenza sarebbe raggiungibile, ad esempio, all'indirizzo `http://www.example.com/my-servlet/helloworld/`.
In entrambi i casi l'unico parametro di configurazione essenziale per Jersey è il package in cui sono racchiuse le classi delle nostre risorse.

Deploy Standalone

```
public class Main {
  public static void main( String[] args ) throws IOException {

    final String baseURI = "http://localhost:9999/";
    final Map<String,String> initParams = new HashMap<String,String>();

    // tell Jersey the package containing Resources
    initParams.put( "com.sun.jersey.confing.property.packages",
        "com.sun.jersey.samples.helloworld.resources" );

    System.out.println("Starting grizzly...");

    // start Grizzly Web Server
    SelectorThread threadSelector =
      GrizzlyWebContainerFactory.create(baseURI,initParams);

    System.out.println("Application started at " + baseURI );
    System.out.prinln("Press Enter to stop");

    System.in.read();
    threadSelector.stopEndpoint();
  }
}
```

Deploy come Web Application

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" 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">
    <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>
          com.sun.jersey.spi.container.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>
              com.sun.jersey.samples.helloworld.resources
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

IscrizioREST

IscrizioREST è un esempio di web service RESTful per gestire le iscrizioni degli studenti ai vari esami di un'università.
Il sistema prevede la gestione di una lista di studenti, di materie e di esami. Oltre alle risorse relative al problema è anche disponibile, al path "/client", un semplice client per il servizio basato su html, css e javascript.

Maven

Il progetto è stato gestito mediante maven. Si può vedere questo tool come una specie di ant "on steroids": fornisce un metodo standard e uniforme per creare e gestire varie tipologie di progetto Java con le loro dipendenze. Tutta la magia è nel file pom.xml situato nella root dei sorgenti in cui si specificano e configurano nome, versione, dipendenze e plugin utilizzati.
Esiste un notevole numero di plugin che permettono di gestire tutti gli aspetti della vita di un progetto, dalla compilazione alla pacchettizzazione, dal testing al deploy alla generazione dei javadoc.
I progetti creati con maven sono gestibili nativamente da NetBeans, mentre per eclipse è necessario installare un apposito plugin e istruire maven affinchè generi i file di progetto necessari.

Maven

Una piccola lista dell'utilizzo di maven nel progetto:

- mvn archetype:generate genera uno scheletro di progetto basandosi sulla tipologia, o archetipo, desiderato e sull'input dell'utente.
- mvn clean pulisce la directory del progetto.
- mvn compile compila i sorgenti.
- mvn exec:java esegue l'applicazione, la MainClass è specificata nel pom.xml
- mvn package genera il .jar del progetto. Mediante l'utilizzo dell'assembly plugin viene creato anche un file .jar contenente tutti i file .class delle dipendenze del progetto, in modo da avere un file autoeseguibile portabile.

DBMS

L'accesso e la memorizzazione dei dati avviene utilizzando come DMBS Apache Derby attraverso le librerie OpenJPA che sono una delle implementazioni disponibili delle Java Persistence API.
JPA è un ORM specifico della piattaforma Java che permette di gestire la persistenza delle entità e le loro relazioni mediante annotazioni. Le entità altro non sono che un oggetto del nostro domain model che devono essere persistenti, cioè memorizzati e recuperabili, tra un'esecuzione e l'altra della nostra applicazione.
JPA è indipendente dal database scelto (si basa su JDBC) e fornisce anche un linguaggio per effettuare query, JPQL, molto simile a SQL.

SubjectResource

```
@Path("/subjects")
public class SubjectResource {
  // retrieves the xml list of subjects
  @GET
    public JAXBElement<SubjectListType> getSubjectsList() {
      Controller c = AppConfig.getInstance().getController();
      return c.getSubjectsList();
    }
  // adds a subject to the database
  @POST
    public Response addSubject(SubjectType s) {
      try {
        Controller c = AppConfig.getInstance().getController();
        Long id = c.addSubject(s);
        URI uri = new URI(AppConfig.URI + id.toString());
        return Response.created(uri).build();
      } catch (URISyntaxException ex) {
        return Response.serverError().build();
      } catch( InvalidXmlException ixe ) {
        throw new WebApplicationException(400);
      }
    }
}
```

SubjectResource

- nel metodo getSubjectList vediamo come sia possibile restituire in risposta anche dati complessi e non solamente i tipi primitivi di Java. In questo caso viene dato in risposta un Java Bean che verrà trasformato in xml (o json) automaticamente da JAXB con cui Jersey è perfettamente integrato.
- addSubject è un esempio di gestione di input da parte dell'utente. Il body della richiesta HTTP viene passato al metodo come argomento. Anche in questo caso si possono gestire sia tipi primitivi sia oggetti più complessi. SubjectType è un Java Bean creato e inizializzato a partire da un frammento xml.
- è possibile inoltre vedere come sia possibile generare le appropriate risposte http in funzione del flusso di esecuzione che la richiesta incontra.

SubjectResource

```
POST /subjects HTTP/1.1
User-Agent: curl/7.20.0 (i686-pc-linux-gnu)Host: localhost:9999
Accept: */*
Content-type:text/xml
Content-Length: 307

<?xml version="1.0" encoding="UTF-8"?>
[cut]

HTTP/1.1 201 Created
server: grizzly/1.9.8
Location: http://localhost:9999/subjects/3
Content-Type: text/plain; charset=iso-8859-1
Content-Length: 0
Date: Wed, 28 Apr 2010 14:45:18 GMT
```

StudentResource

```
@Path("/students")
public class StudentResource {
    /**
     * gets the career of a student
     * @param id the id of the student
     * @return the xml representation of every exam of a student
     */
    @Path("{id:[0-9]+}/career")
    @GET
    @Produces(MediaType.TEXT_XML )
    public JAXBElement<StudentCareerType> getStudentCareer(
            @PathParam("id") Long id
            ) {
        Controller c = AppConfig.getInstance().getController();
        try {
            return c.getStudentCareer(id);
        } catch( EntityNotFoundException enf ) {
            throw new NotFoundException(enf.getMessage());
        }
    }
}
```

StudentResource

- getStudentCareer illustra la realizzazione di una tipica struttura a sottodirectory di REST. Se a `http://wwww.example.com/students/2` è possibile recuperare l'anagrafica dello studente con matricola "2", a `http://www.example.com/students/2/career` è reperibile la rappresentazione della sua carriera universitaria.
- mediante l'annotazione @PathParam è possibile passare ai metodi delle parti dell'URL a cui è indirizzata la richiesta.
- è possibile specificare espressioni regolari per caratterizzare il formato del path voluto; se la richiesta non rispetta questi vincoli Jersey si occupa di rispondere appropriatamente.