

Sviluppo di Web Service Java RESTful con JAX-RS e Jersey

Indice generale

PANORAMICA SUI WEB SERVICE RESTFUL.....	3
PANORAMICA SU JAX-RS.....	4
CREAZIONE DEL PRIMO WEB SERVICE REST.....	5
PRODURRE VARI TIPI DI RAPPRESENTAZIONI DAI WEB SERVICE REST.....	6
DOWNLOAD DEL DEL DATABASE H2.....	8

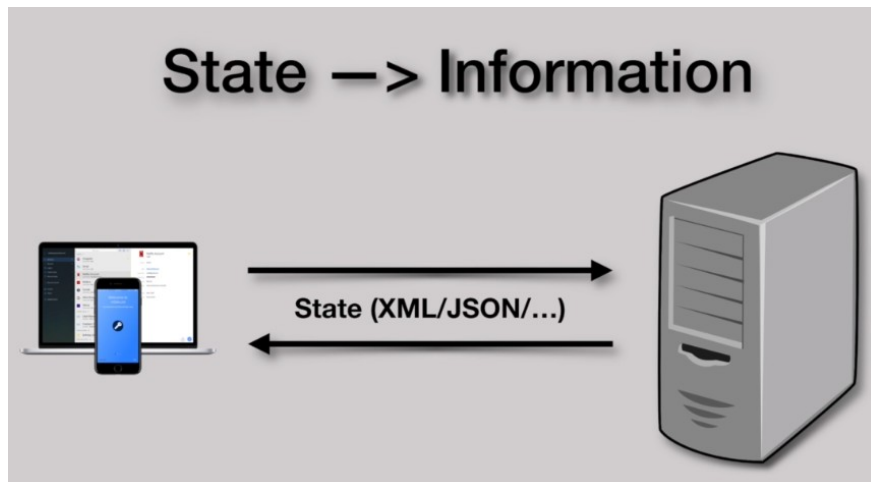
PANORAMICA SUI WEB SERVICE RESTFUL

Per prima cosa, cerchiamo di capire il significato del termine REST.

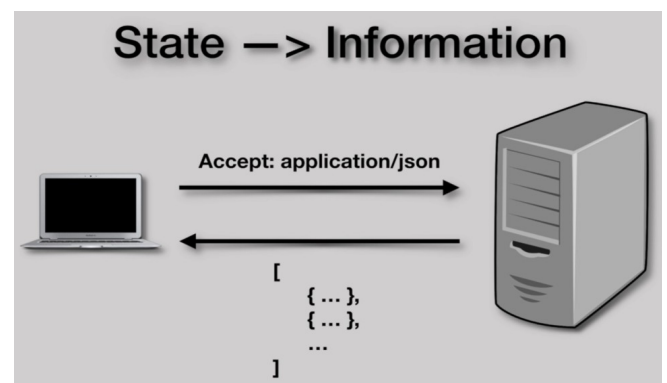
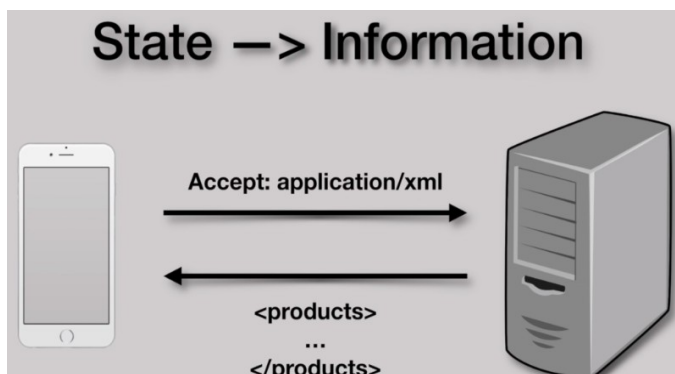
REST è l'acronimo di REpresentational State Transfer, che in italiano si traduce come "trasferimento di stato in rappresentazioni diverse".

Il termine "stato" qui si riferisce allo stato di un insieme di oggetti, denominato risorsa. Per esempio i prodotti, i clienti, gli ordini e gli utenti possono essere considerati tutti come risorse.

In generale ogni risorsa ha alcune informazioni (che definiscono il suo stato) che possono essere scambiate o trasferite in formati diversi, dette rappresentazioni.



Ad esempio, un'applicazione mobile può inviare richieste al server in formato XML, mentre un'altra web application può inviare richieste in formato JSON.



È possibile effettuare una negoziazione sul formato delle informazioni scambiate tra chi ha la risorsa che effettua una richiesta e chi ha la risorsa che eroga il servizio richiesto, in modo tale da avere due rappresentazioni uguali tra loro.

Per i web service REST è necessario:

- utilizzare un URI (Uniform Resource Identifier) per accedervi;
- utilizzare i metodi HTTP (GET, POST, PUT, DELETE...), che rappresentano le azioni da eseguire su una risorsa. Ad esempio, si può effettuare una richiesta GET al server per ottenere informazioni sullo stato di uno specifico ordine, oppure una richiesta POST nel caso in cui si voglia aggiungere un nuovo stato ad una risorsa esistente.

PANORAMICA SU JAX-RS

JAX-RS è una specifica standardizzata da JCP (Java Community Process). JCP è l'istituzione che si occupa di regolare lo sviluppo della tecnologia JAVA.

JAX-RS supporta la creazione di web service REST, che è semplificata grazie all'utilizzo delle annotation presenti all'interno di questa specifica. Una nota importante è che per utilizzare JAX-RS non è richiesta alcuna configurazione.

JAX-RS ha alcune annotation che ci permettono di mappare una classe JAVA come risorsa web. Diamo un'occhiata alle annotation più comunemente utilizzate:

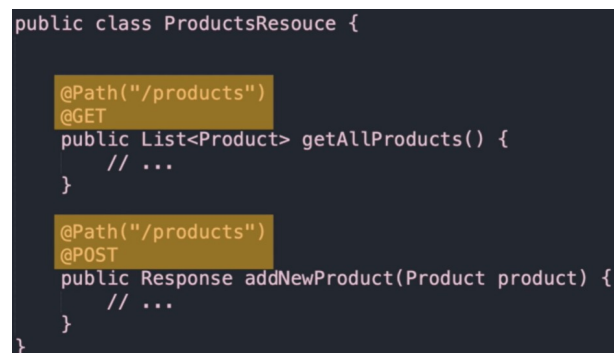
- `@Path`: corrisponde al path a cui risponde un metodo del servizio;



The diagram illustrates the mapping between a URL and a Java class. At the top, the URL `http://example.com/api/products` is shown, with `products` highlighted in a yellow box. Below it, a Java class `ProductsResource` is shown with a method `getAll()`. The `@Path("/products")` annotation is highlighted in a yellow box, indicating its role in mapping the URL path to the class method.

```
public class ProductsResource {  
    @Path("/products")  
    public List<Product> getAll() {  
        // ...  
    }  
}
```

- `@GET`, `@POST`, `@PUT`, `@DELETE`: specificano il tipo di richiesta HTTP della risorsa;



The diagram shows a Java class `ProductsResource` with two methods. The first method `getAllProducts()` is annotated with `@Path("/products")` and `@GET`, both highlighted in yellow boxes. The second method `addNewProduct()` is annotated with `@Path("/products")` and `@POST`, also highlighted in yellow boxes. This demonstrates how different HTTP methods are mapped to different methods in the resource class.

```
public class ProductsResource {  
    @Path("/products")  
    @GET  
    public List<Product> getAllProducts() {  
        // ...  
    }  
    @Path("/products")  
    @POST  
    public Response addNewProduct(Product product) {  
        // ...  
    }  
}
```

- `@Produces`: specifica il tipo di risposta restituita. Il tipo dell'informazione è noto come MIME type.



The diagram shows a Java class `ProductsResource` with a method `getOne()`. The method is annotated with `@Path("/products/{id}")`, `@GET`, and `@Produces({"application/xml", "application/json"})`. The `@Produces` annotation and its value are highlighted in a yellow box, showing how it specifies the MIME types of the response.

```
public class ProductsResource {  
    @Path("/products/{id}")  
    @GET  
    @Produces({"application/xml", "application/json"})  
    public Product getOne(@PathParam("id") int id) {  
        // ...  
    }  
}
```

- `@Consumes`: specifica il tipo di richiesta accettata;

CREAZIONE DEL PRIMO WEB SERVICE REST

Per poter usare le API di JAX-RS, si devono scaricare e inserire all'interno del progetto in cui andremo a sviluppare i servizi REST. Se si sta utilizzando una build automation (ad esempio Maven o Gradle) si devono inserire le dipendenze, altrimenti va fatto tutto a mano.

Nel nostro caso utilizziamo Maven, quindi possiamo importare la seguente dipendenza all'interno del file pom.xml:

```
<dependencies>
  <dependency>
    <groupId>javax.ws.rs</groupId>
    <artifactId>javax.ws.rs-api</artifactId>
    <version>2.0</version>
  </dependency>

  <dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-server</artifactId>
    <version>1.17</version>
  </dependency>

  <dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-servlet</artifactId>
    <version>1.17</version>
  </dependency>

  <dependency>
    <groupId>com.fasterxml.jackson.jaxrs</groupId>
    <artifactId>jackson-jaxrs-json-provider</artifactId>
    <version>2.9.5</version>
  </dependency>
</dependencies>
```

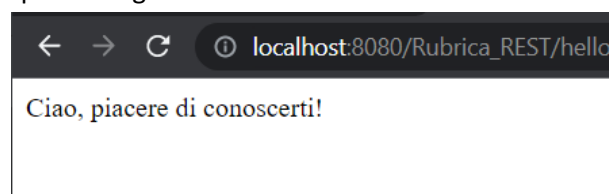
Ora possiamo iniziare a creare il nostro primo servizio REST:

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

@Path("/hello")
public class RisorsaHello {

    @GET
    public String saluto(){ //http://localhost:8080/Rubrica_REST/hello
        return "Ciao, piacere di conoscerti!";
    }
}
```

Il risultato di questo esempio è il seguente:



PRODURRE VARI TIPI DI RAPPRESENTAZIONI DAI WEB SERVICE REST

Per definire qual è il tipo di rappresentazione dell'informazione rilasciata da una risorsa, si deve utilizzare l'annotation `@Produces` sopra i vari metodi del servizio, come nell'esempio seguente:

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

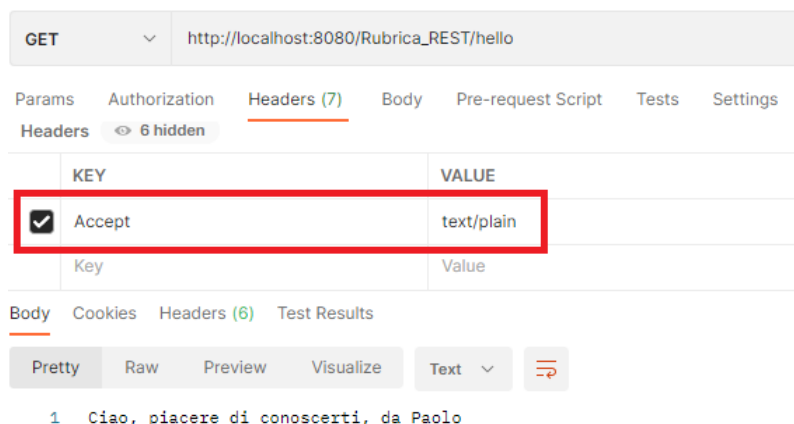
@Path("/hello")
public class RisorsaHello {

    @GET
    @Produces({"text/plain"})
    public String saluto(){ //http://localhost:8080/Rubrica_REST/hello
        return "Ciao, piacere di conoscerti, da Paolo";
    }

    @GET
    @Produces({"application/xml"})
    public String salutoXML(){
        return "<?xml version='1.0' ?>\n" +
            "\n" +
            "<saluto>\n" +
            "    <messaggio>Ciao, piacere di conoscerti</messaggio>\n" +
            "    <da>Paolo</da>\n" +
            "</saluto>";
    }

    @GET
    @Produces({"application/json"})
    public String salutoJSON(){
        return "{\n" +
            "  \"messaggio\":\"Ciao. piacere di conoscerti\"\n" +
            "  \"da\":\"Paolo\" \n" +
            "}";
    }
}
```

I risultati di questo esempio sono i seguenti:



The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: http://localhost:8080/Rubrica_REST/hello
- Headers tab selected, showing 7 headers (6 hidden).
- Header table:

KEY	VALUE
<input checked="" type="checkbox"/> Accept	text/plain
Key	Value

Below the headers, the response body is shown in the 'Body' tab, displaying the text: "Ciao, piacere di conoscerti, da Paolo".

GET ▼ http://localhost:8080/Rubrica_REST/hello

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Headers 6 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	Accept	application/xml
	Key	Value

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize XML ▼

```
1 <?xml version="1.0" ?>
2 <saluto>
3   <messaggio>Ciao, piacere di conoscerti</messaggio>
4   <da>Paolo</da>
5 </saluto>
```

GET ▼ http://localhost:8080/Rubrica_REST/hello

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Headers 6 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	Accept	application/json
	Key	Value

Body Cookies Headers (6) Test Results

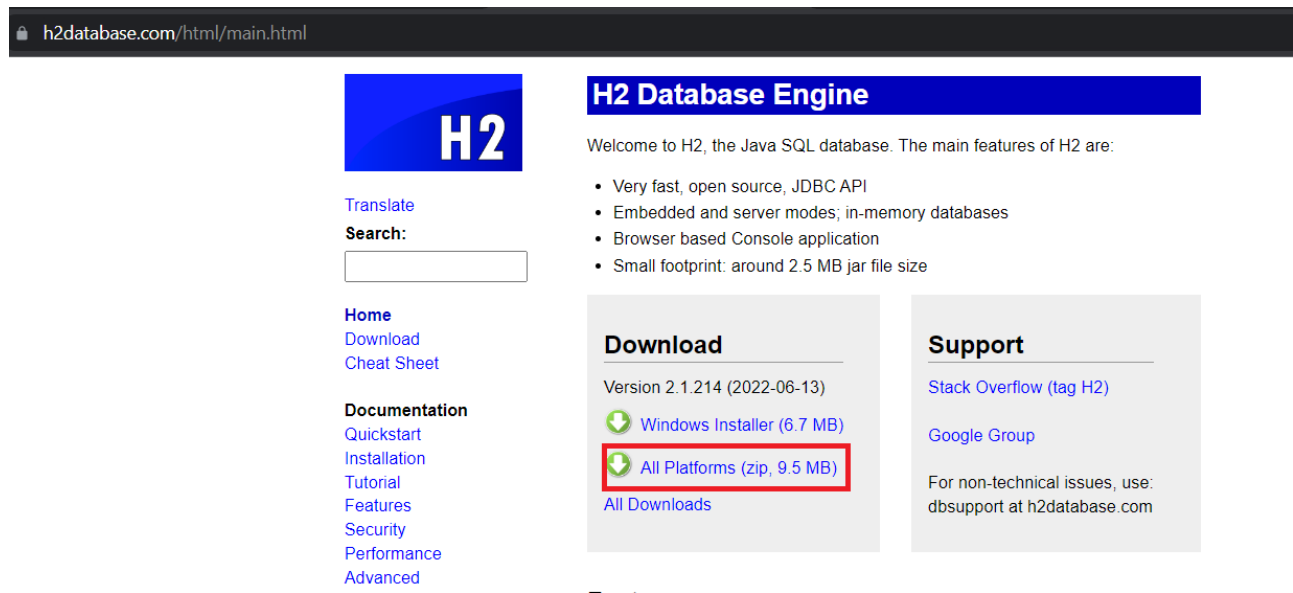
Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "saluto": {
3     "messaggio": "Ciao. piacere di conoscerti"
4     "da": "Paolo"
5   }
6 }
```

DOWNLOAD DEL DEL DATABASE H2

Per questo corso utilizzeremo un database chiamato H2. Questo database non ha bisogno di essere configurato, ma verrà utilizzato quasi subito dopo il download.

Vediamo tutti i passaggi per il download di H2 e la creazione di un database:



The screenshot shows the H2 Database Engine website. The header includes the URL `h2database.com/html/main.html`. The main content area features the H2 logo, a search bar, and a list of links under 'Home' and 'Documentation'. The 'Download' section highlights the 'All Platforms (zip, 9.5 MB)' option, which is circled in red. The 'Support' section provides links to Stack Overflow and Google Group.

H2 Database Engine

Welcome to H2, the Java SQL database. The main features of H2 are:

- Very fast, open source, JDBC API
- Embedded and server modes; in-memory databases
- Browser based Console application
- Small footprint: around 2.5 MB jar file size

Download

Version 2.1.214 (2022-06-13)

- Windows Installer (6.7 MB)
- All Platforms (zip, 9.5 MB)**

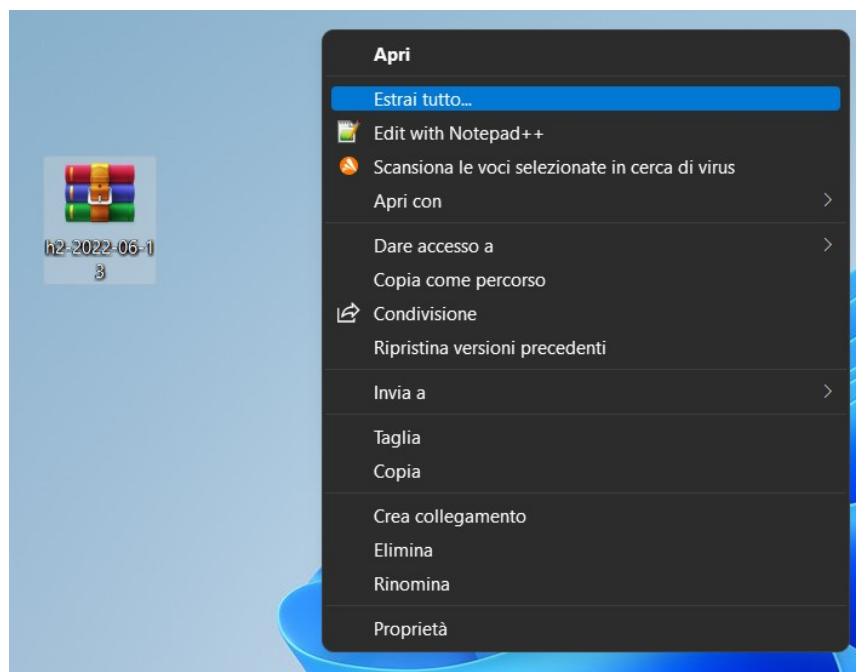
[All Downloads](#)

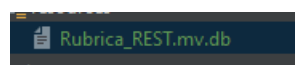
Support

[Stack Overflow \(tag H2\)](#)

[Google Group](#)

For non-technical issues, use: [dbsupport at h2database.com](mailto:dbsupport@h2database.com)





Creare un file con estensione .mv.db e inserirlo all'interno del progetto

Non sicuro | 192.168.1.9:8082/test.do?jsessionid=264bc8a6a112

English ▼ Preferenze Accessori Aiuto

Accesso

Configurazioni salvate: 1 Generic H2 (Embedded) ▼

Nome configurazione: 2 Generic H2 (Embedded) Salva Rimuovi

Classe driver: 3 org.h2.Driver

JDBC URL: 4 jdbc:h2:./devdb

Nome utente: 5 sa

Password: **opzionale**

7 Connetti Prova la connessione 6

Connezzione riuscita

[illegible]