Chapter No. 1
"Getting Started"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.1 "Getting Started"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Sunil Gulabani is a software engineer based in Ahmedabad, Gujarat (India). He completed his graduation in Commerce from S M Patel Institute of Commerce (SMPIC) and Masters in Computer Applications from Ahmedabad Education Society Institute of Computer Studies (AESICS). He has been a top ranker while pursuing his master's degree. He has also presented a paper "Effective Label Matching For Automated Evaluation of Use Case Diagrams" on Technology For Education (T4E)—IIIT. Hyderabad, an IEEE Conference, along with senior lecturers, Vinay Vachharajani and Dr. Jyoti Pareek.

He has been working since 2011 as a software engineer, and is Cloud technology savvy. He has experience in developing enterprise solutions using Java (EE), Apache SOLR, RESTful web services, GWT, SmartGWT, Amazon web services (AWS), Redis, Memcache, MongoDB, and so on. He has a keen interest in system architecture and integration, data modeling, relational databases, and mapping with NoSQL for high throughput.

For More Information:
Apart from that, he takes interest in writing tech blogs and is actively involved in a knowledge-sharing community named Java User Group Ahmadabad (JUG-Ahmedabad).

You can visit him online at http://www.sunilgulabani.com and follow him on Twitter at twitter.com/sunil_gulabani, or reach him directly at sunil_gulabani@yahoo.com.

I would like to express my heartiest thanks to my parents and family members who supported me at each and every level of my career. I would also like to convey thanks to my friends and colleagues without whom jumping to the next step of my career would not be possible. And thanks to the Packt Publishing team, who gave me an opportunity to author this book.
Developing RESTful Web Services with Jersey 2.0

This book is for developing the JAX-RS 2.0 RESTful web services using Java. It provides an understanding of new features of JAX-RS 2.0, along with practical examples for the server and client side. It also covers implementation of different media representations, such as JSON, XML, and multipart. Apart from this, we have also included the modern HTML5-feature of Server-Sent Events (SSE). SSE is basically used for real-time applications, where a server pushes events to the client. Lastly, we described how to generate an XML specification named Web Application Description Language (WADL) of the web services.

What This Book Covers

Chapter 1, Getting Started, gives a brief note regarding the new features that have been introduced in JAX-RS 2.0, such as client API, filters and interceptors, and client-side and server-side asynchronous. It also contains some other prominent features, such as listing of modules and dependencies that are used for implementing JAX-RS 2.0.

Chapter 2, Server API, covers how to create resource classes and methods, usage of parameter annotations to access user-defined values, and subresources to consume specific resource methods. We will see when the root-resource classes are accessible, and the rules for injecting the path annotations to access values. Lastly, we will learn how to load the resource classes using different methods of the Application model, ResourceConfig, and Without Application Model.

Chapter 3, Client API, shows how to consume the RESTful web services that are using the JAX-RS client API. We will cover how to call web services for different HTTP headers and *Param annotations, so that we can then perform all the CRUD operations using this client API. JAX-RS is a wrapper class of the HTTP, so we can use any web services that are based on the HTTP protocol. JAX-RS client API follows the uniformity that implements the REST architecture.

For More Information:  
Chapter 4, Common Media-Type Representations, covers different representations of the data. Data representation is the primary decision for any application. We need to decide an appropriate representation on the basis of the client that will consume the web services. We will also go through the implementation of different representations on the server side as well as the client side.

Chapter 5, Server-Sent Events (SSE), covers how to create a connection between the client/server and maintain the connection at the server's end. This is needed to push the data from the server to the client without any new request initiated by the client. This type of mechanism is basically used for applications such as chatting, stock market, or any real-time data-providing applications.

Chapter 6, WADL, describes Web Application Description Language (WADL), which is a skeleton of the deployed RESTful web service.

For More Information:
Getting Started

This chapter provides readers with a brief introduction on JAX-RS 2.0. It also includes the principles required to be followed in RESTful web services and the new features that have been introduced in JAX-RS 2.0, such as client API, filters and interceptors, client-side and server-side asynchronous. It also contains some other prominent features, such as listing of modules and dependencies that are used for implementing JAX-RS 2.0. Besides the previously mentioned features, it also provides a demonstration of JAX-RS 2.0 with examples.

What is JAX-RS 2.0?

JAX-RS 2.0 is a framework that helps you in writing the RESTful web services on the client side as well as on the server side. Jersey 2.0 is the reference implementation of the JAX-RS 2.0 (JSR 339 specification). Along with the enhancements in Java EE 7, JAX-RS 2.0 has also been revised dramatically.

The following is a list of the RESTful principles that must be followed:

- An ID should be assigned to everything
- Things should be linked together
- A common set of methods must be used
- Multiple representations should be allowed
- Stateless communication must be kept

For More Information:
Before moving ahead, let's look at the existing features of JAX-RS 1.0:

- POJO-based Resource API
- Provides access to the resource classes using HTTP
- Format (content types) independence
- Container (web server) independence
- Inclusion in Java EE

**Features of JAX-RS 2.0**

JAX-RS 2.0 remains consistent with the central theme of Java EE 7, but it contains other long-awaited APIs too. These APIs are mostly focused around what is referred to as **Simplified API**. They can be categorized as follows:

| Client API | The specifications in the earlier versions of JAX-RS were only accountable for a server-side API for the RESTful calls. However, the same resulted into the development of different implementers of the server API, that is, **Jersey** and **RESTEasy**. This led to the independent development of a client-side API. Now, JAX-RS 2.0 has a Client API that provides support to the server-side specification. This feature is a major incorporation of the previous version of the Jersey Client API. This can be summarized as:
| Filters and interceptors | This JAX-RS 2.0 specification is accountable for the client and server filters. In the case of the Client API, implementers provided their own versions of these filters for the client and server. This JAX-RS 2.0 specification, however, absorbs the same into its API. Things that can be done through this specification are:

- Low-level HTTP Client APIs
- Shared APIs with Server APIs
- Compatible with some JAX-RS 1.0 implementations
- Customization of JAX-RS implementations through well-defined extension points
- Logging, Compression, Security, and so on can be achieved
- Shared APIs with Server APIs
- Compatibility with some JAX-RS 1.0 implementations using some different semantics can be achieved

For More Information:

Client-side and Server-side Asynchronous

This feature allows a request to be dispatched in a nonblocking manner, while the results are made available asynchronously. Also, long-running requests on the server side that are I/O bound can be dispatched. This releases the application server thread and enables it to service other requests. It helps in accomplishing the following tasks:

- Allows free running of different threads at the server side
- Request threads can be suspended and resumed as per the need
- Servlet 3 async support
- Provides Client API support

Improved Connection Negotiation

This feature helps in automatically determining the response type even if client specification is already present.

Validation

This feature consists of:

- Services-enabled data validation
- Bean Validation
- Inline-constraint annotations on:
  - Fields and properties
  - Request parameters
  - Methods based on response entity
  - Resource classes, that is, path validation using regular expressions

HyperMedia as the Engine of Application State (HATEOAS)

This feature is an important aspect of the RESTful architecture. HATEOAS allows us to provide hyperlinks/URIs in the request/response of the web services. This can be compared with the hyperlinks in an HTML form.

However, Jersey 2.0 can be deployed on several web containers that support Servlet 2.5 or higher, Grizzly 2 HTTP server (which is also the default server for testing), and OSGi containers. For the new async feature of JAX-RS 2.0, Server-Side Events (SSE), we need containers supporting Servlet 3.0.
Ease of using and reusing JAX-RS artifacts

As a part of Java EE 7, it is easy to create RESTful web services using JAX-RS 2.0. Using JAX-RS annotations, we can map the POJO resource class to the URIs and the URI templates. Using annotations, it becomes easier to develop web services. Different @*Param* annotations are available to access the values of the user request:

- @PathParam
- @QueryParam
- @MatrixParam
- @HeaderParam
- @CookieParam
- @FormParam
- @DefaultValue
- @Context

JAX-RS also supports all the Java data types to be supplied in the @*Param* annotations. There are other APIs available to create the RESTful web services, but mostly they require more coding. However, in the Jersey implementation of JAX-RS, it is simpler and easier to create the RESTful web services. JAX-RS manages to encode and decode the request/response content according to the media type it mentioned.

Modules and dependencies

To provide backward compatibility, all Jersey applications are compiled with Java SE 6. Thus, we can run our Jersey application on Java SE 6 easily though JAX-RS 2.0 is part of Java EE 7.

To create Jersey-based applications, we require several dependencies and for other modules, we require third-party dependencies. Jersey dependencies are loosely-coupled and separated according to the module dependencies. Jersey dependencies are lightweight, so our application has much complexity for dependencies and creation of the RESTful web services.

The following are the core Jersey modules:

- Jersey Core
- jersey-client
- jersey-common
- jersey-server

For More Information:
You can download the Jersey Core JAX-RS 2.0 Bundle from:
http://repo1.maven.org/maven2/org/glassfish/jersey/bundles/jaxrs-ri/2.0/jaxrs-ri-2.0.zip

Apart from these dependencies, there are Jersey Containers, Connectors, Media, Extensions, Test Framework, and Glassfish Bundles. These dependencies can be easily plugged. You can find Jersey 2.0 binaries on the Jersey 2.0 Maven repository,
http://repo1.maven.org/maven2/org/glassfish/jersey/,
and on
https://maven.java.net/content/repositories/releases/org/glassfish/jersey/.

**Creating a new project**

The first step for creating a new project is to accumulate the required tools:

- Java JDK (Version 6 or higher)
- Eclipse IDE (Juno)
- Apache Tomcat Server (Version 7) or Glassfish Server (Version 4.0)
- Jersey Framework

Firstly, we will create a web project using Eclipse IDE. Go to File | New | Others | Dynamic Web Project. Follow the steps, and after the project is created, add the following libraries into the classpath:

- asm-all-repackaged-2.2.0-b14.jar
- cglib-2.2.0-b14.jar
- guava-14.0.1.jar
- hk2-api-2.2.0-b14.jar
- hk2-locator-2.2.0-b14.jar
- hk2-utils-2.2.0-b14.jar
- javax.annotation-api-1.2.jar
- javax.inject-2.2.0-b14.jar
- javax.ws.rs-api-2.0.jar

For More Information:
Getting Started

- jersey-client-2.2.jar
- jersey-common-2.2.jar
- jersey-container-servlet-core-2.2.jar
- jersey-server-2.2.jar
- osgi-resource-locator-1.0.1.jar
- validation-api-1.1.0.Final.jar

Now, we need to configure web.xml to the bound Jersey Container with the resources packages:

............
<web-app>
  <servlet>
    <servlet-name>simpleJerseyExample</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>com.demo</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>simpleJerseyExample</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
............

A servlet container is treated as a controller to redirect the specified resource that is being called. jersey.config.server.provider.packages maps the resources that are available in the com.demo package. So, whenever any resource is being requested, ServletContainer checks into the com.demo package for the resource URI and serves the request accordingly.

For More Information:
The next step is to create the Resource class that contains the business logic:

```java
package com.demo;

import javax.ws.rs.Path;
import javax.ws.rs.Get;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.Produces;

/**
 *    helloWorld Root Resource
 */
@Path("helloWorld")
public class HelloWorldResource{

    @GET
    @PRODUCES(MediaType.TEXT_PLAIN)
    public String greet(){
        return "Hello World!!!";
    }
}
```

A JAX-RS resource is an annotated POJO, which provides the so-called resource methods that are able to handle the HTTP requests for the URI paths bound to the resource. In the previous code, "helloWorld" is the resource URI:

```java
@Path("helloWorld")
```

To run the application, create a WAR file and deploy it on the Apache Tomcat Server. Once the project is deployed on the Tomcat server, we are ready to consume the "helloWorld" web service. We can type the resource URL in the browser or we can use curl:

```
$curl http://localhost:8080/projectName/services/helloWorld
Hello World!!!
```

For More Information:
Summary
We have covered a brief introduction about the JAX-RS 2.0 and Jersey 2.0 features that have enriched the JAX-RS web services. We've analyzed a simple RESTful web service example that shows the basic implementation of the JAX-RS 2.0. In the next chapter, we will see how to use the Server API with different examples of complete sets of code to implement the Server-side web services.
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.