用 Jersey 2 和 Spring 4 构建 RESTful web service

原文
本文介绍了如何通过 Jersey 框架优美的在 Java 实现了 REST 的 API。CRUD 的 操作存储在 MySQL 中

1. 示例

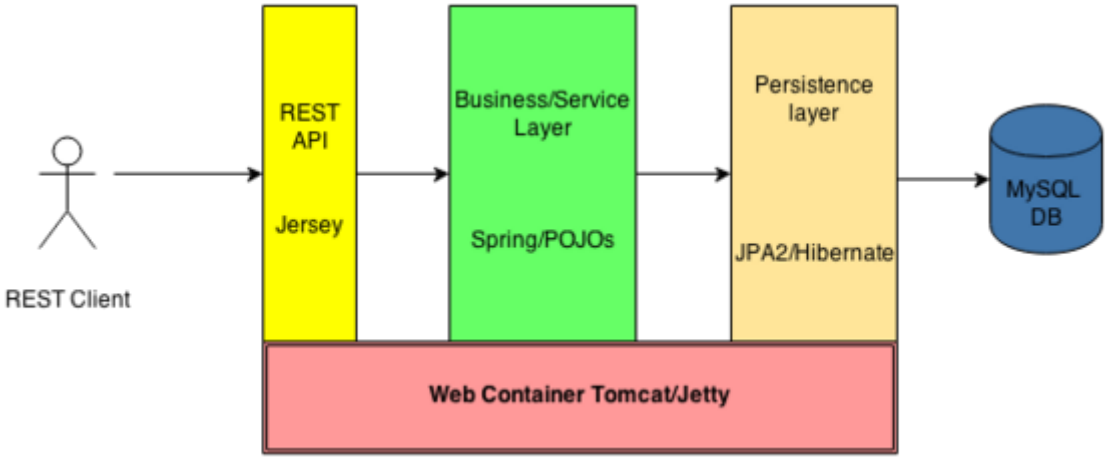
1.1 为什么

Spring 可以对于 REST 有自己的实现(见 https://spring.io/guides/tutorials/rest/)。 但本文展示的是用 “官方” 的 方法来实现 REST ，即使用 Jersey。

1.2 它是做什么的?

管理 资源。 REST API 将允许创建、检索、更新和删除这样的资源。

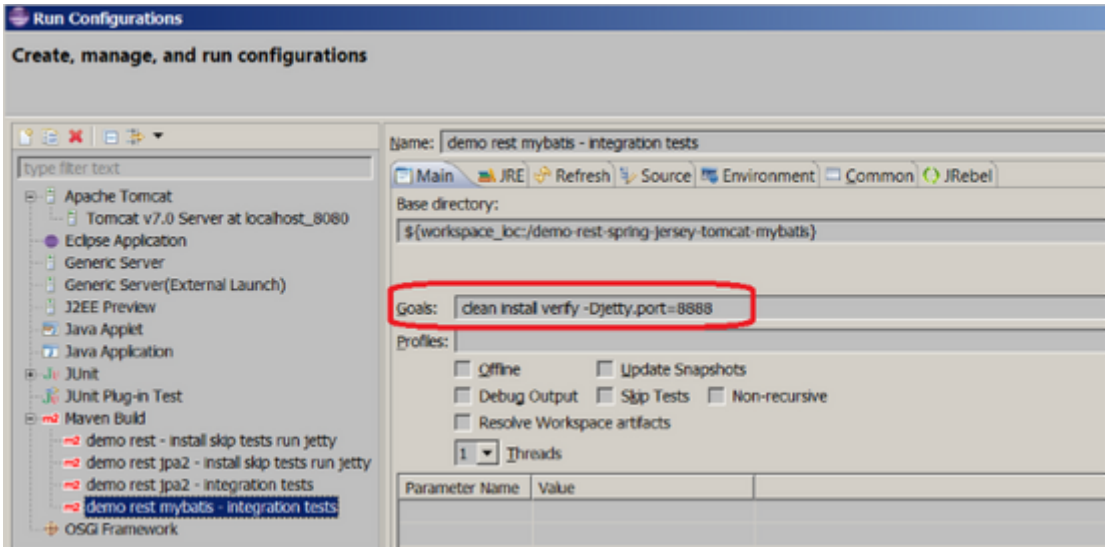
1.3 架构及技术



本示例项目使用多层结构，基于“Law of Demeter (LoD) or principle of least knowledge”（迪米特法则），是说一个软件实体要尽可能的只与和它最近的实体进行通讯。通常被表述为：talk only to your immediate friends (只和离你最近的朋友进行交互)。“talk”，其实就是对象间方法的调用。这条规则表明了对象间方法调用的原则：（1）调用对象本身的方法；（2）调用通过参数传入的对象的方法；（3）在方法中创建的对象的方法；（4）所包含对象的方法。

主要分为三层：

- 第一层：Jersey 实现对 REST 的支持，拥有 外观模式 的角色并代理到逻辑业务层
- 业务层: 发生逻辑的地方
- 数据访问层： 是与持久数据存储 (在我们的例子中是 MySql数据库)交互的地方



简述下技术框架：

1.3.1. Jersey (外观)

Jersey 是开源、拥有产品级别的质量，提供构建 RESTful Web Services,支持 JAX-RS APIs ，提供 JAX-RS (JSR 311 & JSR 339) 参考实现。

1.3.2. Spring (业务层)

在我看来没有什么 比 Spring 更好的办法让 pojo 具有不同的功能。 你会发现在本教程用 Jersey 2 和 Spring 4 构建 RESTful web service

1.3.3. JPA 2 / Hibernate (持久层)

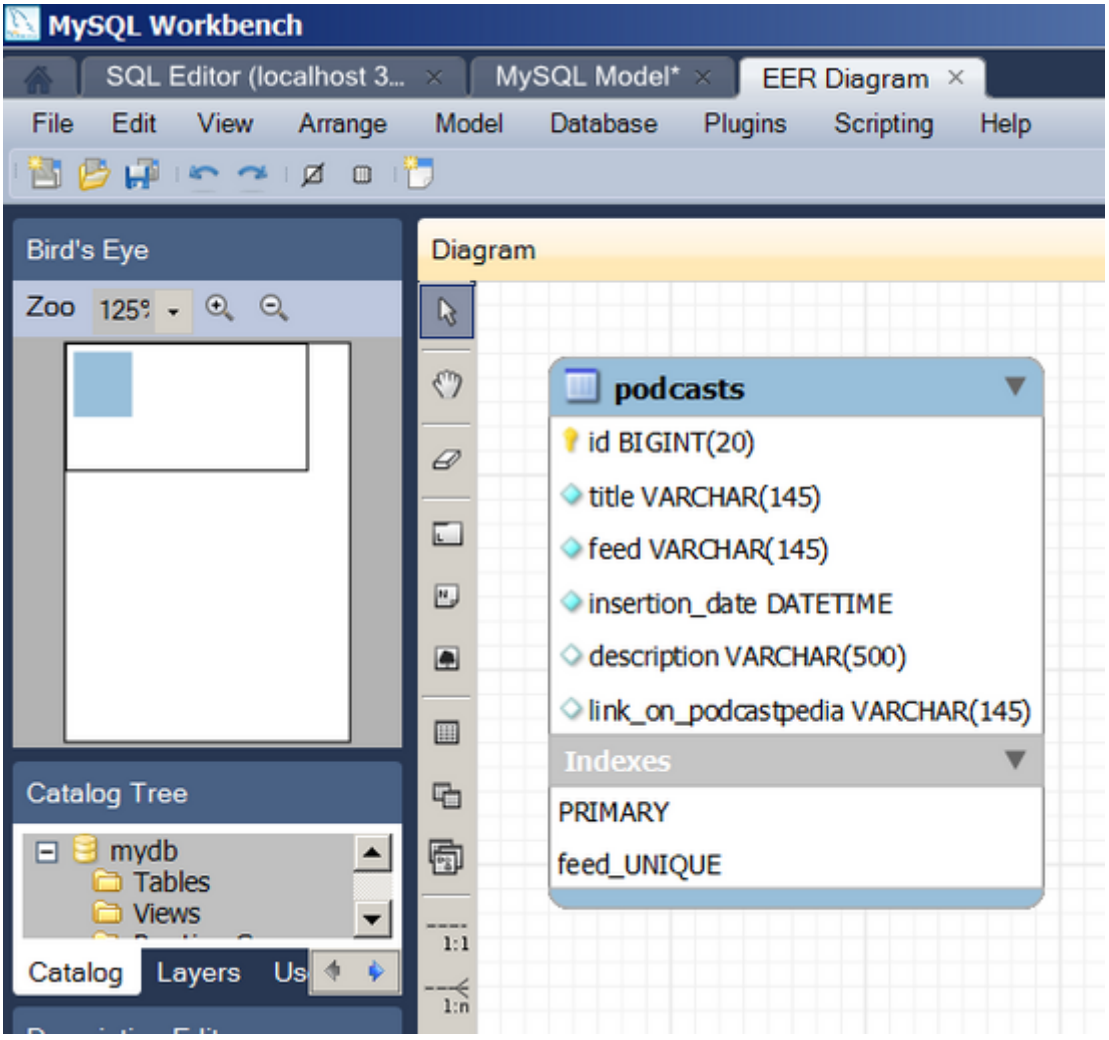
使用 Hibernate 实现 DAO 模式。

1.3.4. Web 容器

用 Maven 打包成 .war 文件开源部署在任意容器。一般用 Tomcat 和 Jetty ，也可以是 Glassfih, Weblogic, JBoss 或 WebSphere.

1.3.5. MySQL 数据库

示例数据存储在一个 MySQL 表:



1.3.6. 技术版本

Jersey 2.9

Spring 4.0.3

Hibernate 4

Maven 3

Tomcat 7

Jetty 9

MySql 5.6

1.4. 源码

见

2. 配置

开始呈现 REST API 的设计和实现之前,我们需要做一些配置。

2.1. 项目依赖

Jersey Spring 扩展包 是必须要放在 项目 classpath 中。在 pom.xml 中添加下面依赖：

```
<dependency>
  <groupId>org.glassfish.jersey.ext</groupId>
  <artifactId>jersey-spring3</artifactId>
  <version>${jersey.version}</version>
  <exclusions>
    <exclusion>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.media</groupId>
  <artifactId>jersey-media-json-jackson</artifactId>
  <version>2.4.1</version>
</dependency>
```

注意: jersey-spring3.jar 使用的是他自己的 Spring 库版本，所以如果你想使用自己的 (本例是使用 Spring 4.0.3.Release),你需要将这些库手动的移除。如果想看到其他 的库的依赖，请查看项目源码中的 pom.xml

2.2. web.xml

应用部署描述

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <display-name>Demo - Restful Web Application</display-name>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring/applicationContext.xml</param-
value>
  </context-param>

  <servlet>
    <servlet-name>jersey-serlvet</servlet-name>
    <servlet-class>
      org.glassfish.jersey.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-
value>org.codingpedia.demo.rest.RestDemoJaxRsApplication</param-
value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>jersey-serlvet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>

  <resource-ref>
    <description>Database resource rest demo web application
</description>
```

```
    <res-ref-name>jdbc/restDemoDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</web-app>
```

2.2.1. Jersey-servlet

注意 Jersey servlet 的配置， `javax.ws.rs.core.Application` 类定义了 JAX-RS 应用组件 (root 资源 和 提供者 类) .本例使用 `ResourceConfig` , 是 Jersey 自己实现的 `Application` 类，提供了简化 JAX-RS 组件的能力。详见 JAX-RS 应用模型

`org.codingpedia.demo.rest.RestDemoJaxRsApplication` 是自己实现的 `ResourceConfig` 类，注册应用的 resources, filters, exception mappers 和 feature :

```
package org.codingpedia.demo.rest.service;

//imports omitted for brevity

/**
 * Registers the components to be used by the JAX-RS application
 *
 * @author ama
 *
 */
public class RestDemoJaxRsApplication extends ResourceConfig {

    /**
     * Register JAX-RS application components.
     */
    public RestDemoJaxRsApplication() {
  // register application resources
  register(PodcastResource.class);
  register(PodcastLegacyResource.class);

  // register filters
  register(RequestContextFilter.class);
  register(LoggingResponseFilter.class);
  register(CORSResponseFilter.class);
```

```
  // register exception mappers
  register(GenericExceptionMapper.class);
  register(AppExceptionMapper.class);
  register(NotFoundExceptionMapper.class);

  // register features
  register(JacksonFeature.class);
  register(MultiPartFeature.class);
    }
}
```

注意：

- `org.glassfish.jersey.server.spring.scope.RequestContextFilter` 是 Spring filter 提供了 JAX-RS 和 Spring 请求属性之间的桥梁。
- `org.codingpedia.demo.rest.resource.PodcastsResource` 这是“外观”组件，通过注解 暴露了 REST 的API。稍后会描述
- `org.glassfish.jersey.jackson.JacksonFeature` ,是一个 feature ，用 Jackson JSON 的提供者来解释 JSON。

2.1.2. Spring 配置

配置文件在 classpath 目录下的 spring/applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd

    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd

    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-
context.xsd">

  <context:component-scan base-package="org.codingpedia.demo.rest.*"
```

```
/>

  <!-- ************ JPA configuration *********** -->
  <tx:annotation-driven transaction-manager="transactionManager" />
  <bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"
/>
  </bean>
  <bean id="transactionManagerLegacy"
class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory"
ref="entityManagerFactoryLegacy" />
  </bean>
  <bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactory
Bean">
    <property name="persistenceXmlLocation"
value="classpath:config/persistence-demo.xml" />
    <property name="persistenceUnitName" value="demoRestPersistence"
/>
    <property name="dataSource" ref="restDemoDS" />
    <property name="packagesToScan" value="org.codingpedia.demo.*" />
    <property name="jpaVendorAdapter">
      <bean
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="showSql" value="true" />
        <property name="databasePlatform"
value="org.hibernate.dialect.MySQLDialect" />
      </bean>
    </property>
  </bean>
  <bean id="entityManagerFactoryLegacy"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactory
Bean">
    <property name="persistenceXmlLocation"
value="classpath:config/persistence-demo.xml" />
    <property name="persistenceUnitName"
value="demoRestPersistenceLegacy" />
    <property name="dataSource" ref="restDemoLegacyDS" />
```

```
    <property name="packagesToScan" value="org.codingpedia.demo.*" />
    <property name="jpaVendorAdapter">
      <bean
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="showSql" value="true" />
        <property name="databasePlatform"
value="org.hibernate.dialect.MySQLDialect" />
      </bean>
    </property>
  </bean>

  <bean id="podcastDao"
class="org.codingpedia.demo.rest.dao.PodcastDaoJPA2Impl"/>
  <bean id="podcastService"
class="org.codingpedia.demo.rest.service.PodcastServiceDbAccessImpl"
/>
  <bean id="podcastsResource"
class="org.codingpedia.demo.rest.resource.PodcastsResource" />
  <bean id="podcastLegacyResource"
class="org.codingpedia.demo.rest.resource.PodcastLegacyResource" />

  <bean id="restDemoDS"
class="org.springframework.jndi.JndiObjectFactoryBean"
scope="singleton">
    <property name="jndiName" value="java:comp/env/jdbc/restDemoDB"
/>
    <property name="resourceRef" value="true" />
  </bean>
  <bean id="restDemoLegacyDS"
class="org.springframework.jndi.JndiObjectFactoryBean"
scope="singleton">
    <property name="jndiName"
value="java:comp/env/jdbc/restDemoLegacyDB" />
    <property name="resourceRef" value="true" />
  </bean>
</beans>
```

其中 `podcastsResource` 是指向 REST API 实体

3. REST API (设计与实现)

3.1. 资源

3.1.1. 设计

REST 中的资源主要包括下面两大思想：

- 每个都指向了全球标示符（如，HTTP 中的 URI ）
- 有一个或多个表示（我们将在本示例使用 JSON 格式）

REST 中的资源 一般是名词 (podcasts, customers, user, accounts 等) 而不是名词 (getPodcast, deleteUser 等)

本教程使用的端点有：

- `/podcasts` –（注意复数）URI标识的资源 podcasts 集合的播客
- `/podcasts/{id}` – 通过 podcasts 的ID, URI 标识一个podcasts 资源，

3.1.2. 实现

为求精简 ， podcast 只包含下列属性:

- id – podcast 的唯一标识
- feed – podcast 的 feed url
- title – 标题
- linkOnPodcastpedia – 链接
- description – 描述

我用了两种 Java 类来表示 podcast 代码，是为了避免 类及其属性/方法 被 JPA 和 XML/JAXB/JSON 的注释堆满了：

- PodcastEntity.java – JPA 注解类用在 DB 和业务层
- Podcast.java – JAXB/JSON 注解类用在外观和业务层

Podcast.java

```
package org.codingpedia.demo.rest.resource;

//imports omitted for brevity

/**
 * Podcast resource placeholder for json/xml representation
```

```
 *
 * @author ama
 *
 */
@SuppressWarnings("restriction")
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Podcast implements Serializable {

  private static final long serialVersionUID = -8039686696076337053L;

  /** id of the podcast */
  @XmlElement(name = "id")
  private Long id;

  /** title of the podcast */
  @XmlElement(name = "title")
  private String title;

  /** link of the podcast on Podcastpedia.org */
  @XmlElement(name = "linkOnPodcastpedia")
  private String linkOnPodcastpedia;

  /** url of the feed */
  @XmlElement(name = "feed")
  private String feed;

  /** description of the podcast */
  @XmlElement(name = "description")
  private String description;

  /** insertion date in the database */
  @XmlElement(name = "insertionDate")
  @XmlJavaTypeAdapter(DateISO8601Adapter.class)
  @PodcastDetailedView
  private Date insertionDate;

  public Podcast(PodcastEntity podcastEntity){
    try {
      BeanUtils.copyProperties(this, podcastEntity);
```

```
    } catch (IllegalAccessException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (InvocationTargetException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

  public Podcast(String title, String linkOnPodcastpedia, String
feed,
      String description) {

    this.title = title;
    this.linkOnPodcastpedia = linkOnPodcastpedia;
    this.feed = feed;
    this.description = description;

  }

  public Podcast(){}

//getters and setters now shown for brevity
}
```

转化成 JSON 输出如下

```
{
  "id":1,
  "title":"Quarks & Co - zum Mitnehmen-modified",

"linkOnPodcastpedia":"http://www.podcastpedia.org/podcasts/1/Quarks-
Co-zum-Mitnehmen",
  "feed":"http://podcast.wdr.de/quarks.xml",
  "description":"Quarks & Co: Das Wissenschaftsmagazin",
  "insertionDate":"2014-05-30T10:26:12.00+0200"
}
```

3.2. 方法

简单的说明

但不是一一映射，因为 PUT 也可以创建， POST也可以用在更新。

注：读取和删除它是很清楚的，他们确实是和 GET 、DELETE一对一的映射。无论如何，REST 是一种架构风格，不是一个规范，你应该适应你的架构需要，但如果你想让你的 API 让更多的公众愿意使用它，你应该遵循一定的“最佳实践”。

`PodcastRestResource` 类 是处理所有的请求

```
package org.codingpedia.demo.rest.resource;
//imports
.....................
@Component
@Path("/podcasts")
public class PodcastResource {
    @Autowired
    private PodcastService podcastService;
    ....................
}
```

注意 类定义前面的 `@Path("/podcasts")` ，所有与 podcast 关联的资源都会出现在 这个路径下。 @Path 注解值是关联 URI 的路径。

在上面的例子中，该 Java 类将托管在 `/podcasts` URI 路径。 `PodcastService` 接口公开的业务逻辑 到 REST 外观层。

3.2.1. 创建 podcast

3.2.1.1. 设计

常见的的方式利用 POST 创建资源，如前所述，创建一个新的资源，可以用 POST 和 PUT 的方法，我是这样做的：

Description	**URI**	HTTP method	**HTTP Status response**
增加新的 podcast	/podcasts/	POST	201 Created
增加新的 podcast (必须传所有的值)	/podcasts/{id}	PUT	201 Created

PUT POST 最大的区别是 ，PUT 就是把你应该事先知道资源将被创建的位置和发送所有可能值的实体。

3.2.1.2. 实现

3.2.1.2.1. POST 创建一个单资源

```
/**
 * Adds a new resource (podcast) from the given json format (at least
title
 * and feed elements are required at the DB level)
 *
 * @param podcast
 * @return
 * @throws AppException
 */
@POST
@Consumes({ MediaType.APPLICATION_JSON })
@Produces({ MediaType.TEXT_HTML })
public Response createPodcast(Podcast podcast) throws AppException {
    Long createPodcastId = podcastService.createPodcast(podcast);
    return Response.status(Response.Status.CREATED)// 201
            .entity("A new podcast has been created")
            .header("Location",
                    "http://localhost:8888/demo-rest-jersey-
spring/podcasts/"
                            +
String.valueOf(createPodcastId)).build();
}
```

注解

- @POST – 指示方法响应到 HTTP POST 请求
- @Consumes({MediaType.APPLICATION_JSON}) – 定义方法可以接受的媒体类型，本例为"application/json"
- @Produces({MediaType.TEXT_HTML}) – 定义方法产生的媒体类型本例为 "text/html"

响应

- 错误:

- 400：没有足够的数据提供
- 409：冲突了。如果在服务器端被确定 具有相同的 podcast 的存在

3.2.1.2.2. 通过 PUT 创建单资源 (“podcast”)

这将执行 更新 Podcast 处理。

3.2.1.2.3. 附加 – 通过表单创建 (“podcast”)资源

```
/**
 * Adds a new podcast (resource) from "form" (at least title and feed
 * elements are required at the DB level)
 *
 * @param title
 * @param linkOnPodcastpedia
 * @param feed
 * @param description
 * @return
 * @throws AppException
 */
@POST
@Consumes({ MediaType.APPLICATION_FORM_URLENCODED })
@Produces({ MediaType.TEXT_HTML })
@Transactional
public Response createPodcastFromApplicationFormURLencoded(
        @FormParam("title") String title,
        @FormParam("linkOnPodcastpedia") String linkOnPodcastpedia,
        @FormParam("feed") String feed,
        @FormParam("description") String description) throws
AppException {

    Podcast podcast = new Podcast(title, linkOnPodcastpedia, feed,
            description);
    Long createPodcastid = podcastService.createPodcast(podcast);

    return Response
            .status(Response.Status.CREATED)// 201
            .entity("A new podcast/resource has been created at
/demo-rest-jersey-spring/podcasts/"
                    + createPodcastid)
            .header("Location",
```

```
                    "http://localhost:8888/demo-rest-jersey-
spring/podcasts/"
                            +
String.valueOf(createPodcastid)).build();
}
```

注解