## JAX-RS

JAX-RS是Java 6引入的新技术，全称是：Java Api for Restful WebService，是一个Java编程语言的应用程序接口，支持Rest架构风格创建Web服务。JAX-RS使用了Java 5引入的注解来简化web服务的客户端和服务端的开发和部署。

JAX-RS提供了一些注解将一个资源类，一个POJO类封装成Web资源，注解包括：

@Path：标注资源类或者方法的相对路径

@GET、@PUT、@POST、@DELETE：标注方法的Http请求类型

@Produces：标注返回的MIME媒体类型

@Consumes：标注可接受的MIME媒体类型

@PathParam：标注参数来自Http请求的URL路径

@QueryParam：标注参数来自Http请求的URL查询参数

@HeaderParam：标注参数来自Http请求的头信息

@CookieParam：标注参数来自Http请求的Cookie信息

@MatrixParam：标注参数来自Http请求的矩阵信息

@FormParam：标注参数来自Http请求的表单信息

基于JAX-RS实现的框架有Jersey、RestEasy等。这两个框架创建的应用可以很方便地部署到Servlet 容器中，比如Tomcat，JBoss等。值得一提的是RESTEasy是由JBoss公司开发的，所以将用RESTEasy框架实现的应用部署到JBoss服务器上，可以实现很多额外的功能。

## Jersey开始

Jersey提供了两种maven archetype。一种是基于轻量级的Grizzly HTTP服务器，使用的maven命令如下：

mvn archetype:generate -DarchetypeArtifactId=jersey-quickstart-grizzly2 -DarchetypeGroupId=org.glassfish.jersey.archetypes -DinteractiveMode=false -DgroupId=com.example -DartifactId=simple-service -Dpackage=com.example -DarchetypeVersion=2.26-b09

可以使用mvn clean test 、mvn exec:java启动项目；

一种是基于Java Web项目，命令如下：

mvn archetype:generate -DarchetypeArtifactId=jersey-quickstart-webapp -DarchetypeGroupId=org.glassfish.jersey.archetypes -DinteractiveMode=false -DgroupId=com.example -DartifactId=simple-service-webapp -Dpackage=com.example -DarchetypeVersion=2.16

使用mvn clean package并将生成的war包部署到Tomcat等Web容器即可。

## 模块和依赖

1. 兼容性

2.6以前的版本，Jersey由Java SE 6编译；2.7版本后几乎所有的Jersey组件用Java SE 7编译。

只有core-common和core-client模块仍然需要Java SE 6编译

1. Jersey的依赖

一个基于Jersey的应用，需要依赖于Jersey模块，可以使用maven或ant来管理项目的依赖。

<!—jersey客户端 仅使用 JAX-RS Client 时添加 -->

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-client</artifactId>

<version>2.25.1</version>

</dependency>

<!—jersey服务端 -->

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-server</artifactId>

<version>2.25.1</version>

</dependency>

<!—Servlet容器3.0时使用 -->

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet-core</artifactId>

<version>2.25.1</version>

</dependency>

<!—jsr311-api -->

<dependency>

<groupId>javax.ws.rs</groupId>

<artifactId>jsr311-api</artifactId>

<version>1.1.1</version>

</dependency>

## Spring集成

### 1.Maven依赖

<properties>

<servlei-api.version>3.0.1</servlei-api.version>

<jackson.version>2.8.9</jackson.version>

<slf4j.version>1.7.25</slf4j.version>

<commmon.io.version>2.5</commmon.io.version>

<commmon.fileupload.version>1.3.1</commmon.fileupload.version>

<commmon.langs.version>3.5</commmon.langs.version>

<jersey.version>2.26-b09</jersey.version>

</properties>

<dependencies>

<!-- spring Mvc -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

<exclusions>

<exclusion>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- servlet-api -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>${servlei-api.version}</version>

<scope>provided</scope>

</dependency>

<!-- jackson -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>${jackson.version}</version>

</dependency>

<!-- logger -->

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.8.2</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-slf4j-impl</artifactId>

<version>2.8.2</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jcl-over-slf4j</artifactId>

<version>1.7.25</version>

</dependency>

<!-- 文件上传 -->

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>${commmon.io.version}</version>

</dependency>

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>${commmon.fileupload.version}</version>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

<version>${commmon.langs.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

<exclusions>

<exclusion>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- jsp标注函数库jstl -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- jersey 开始 -->

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet</artifactId>

<version>${jersey.version}</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.inject</groupId>

<artifactId>jersey-hk2</artifactId>

<version>${jersey.version}</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.ext</groupId>

<artifactId>jersey-spring4</artifactId>

<version>${jersey.version}</version>

</dependency>

<!-- json支持 -->

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson</artifactId>

<version>${jersey.version}</version>

</dependency>

<!-- jersey 结束 -->

</dependencies>

### 2.web.xml配置

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

<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance

http://www.springmodules.org/schema/cache/springmodules-cache.xsd

http://www.springmodules.org/schema/cache/springmodules-ehcache.xsd"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_3\_0.xsd">

<display-name>Spring MVC</display-name>

<!-- 配置业务层的Spring容器 开始 -->

<listener>

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

</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<!-- 配置业务层的Spring容器 结束 -->

<!-- spring MVC 开始 -->

<servlet>

<servlet-name>simpleMvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:simpleMvc-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>simpleMvc</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<!-- spring MVC 结束 -->

<!-- jersey开始 -->

<servlet>

<servlet-name>SpringApplication</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer

</servlet-class>

<init-param>

<param-name>javax.ws.rs.Application</param-name>

<param-value>com.wangdh.spring.jersey.config.JerseyApplication

</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>SpringApplication</servlet-name>

<url-pattern>/api/\*</url-pattern>

</servlet-mapping>

<!-- jersey结束 -->

<!-- 首页列表 -->

<welcome-file-list>

<welcome-file>/index</welcome-file>

</welcome-file-list>

</web-app>

### 3.Spring业务容器配置applicationContext.xml

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

<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:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/beans

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

http://www.springframework.org/schema/context

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

http://www.springframework.org/schema/mvc

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

<context:component-scan base-package="com.wangdh.spring.mvc.service"/>

<bean class="com.wangdh.spring.jersey.config.DateTimeService" scope="request"/>

</beans>

### 4.Spring MVC配置

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

<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:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/beans

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

http://www.springframework.org/schema/context

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

http://www.springframework.org/schema/mvc

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

<!-- <mvc:annotation-driven/>是Spring MVC提供的一键式配置方法，配置此标签后Spring MVC会自动做一些注册组件的事 -->

<mvc:annotation-driven />

<context:component-scan base-package="com.wangdh.spring.mvc.controller"

use-default-filters="false">

<context:include-filter type="annotation"

expression="org.springframework.stereotype.Controller" />

</context:component-scan>

<!-- 如果没有配置ViewResolver，默认使用org.springframework.web.servlet.view.InternalResourceViewResolver

作为ViewResolver，并且prefix和suffix都为空 -->

</beans>

### 5.Jersey配置JerseyApplication

package com.wangdh.spring.jersey.config;

import org.glassfish.jersey.server.ResourceConfig;

import org.glassfish.jersey.server.spring.scope.RequestContextFilter;

/\*\*

\* Jersey相关配置

\* @author wdhcxx

\*

\*/

public class JerseyApplication extends ResourceConfig{

/\*\*

\* 注册JAX-RS相关组件

\*/

public JerseyApplication(){

register(RequestContextFilter.class);

register(CustomExceptionMapper.class);

register(JerseyResource.class);

}

}

## 注解

### 1.参数注解

@PathParam：获取URL请求的路径参数

@QueryParam：获取URL的查询参数

@FormParam：请求MIME媒体类型为application/x-www-form-urlencoded

@BeanParam:组合多个请求参数为一个bean，该bean属性可以使用不同参数注解；

@DefaultValue：获取默认值

### 2.资源

根资源类：Root Resource Classes，带有@Path注解的类，包含至少一个@Path注解的方法或者方法带有@GET、@PUT、@POST、@DELETE资源方法指示器的POJO。

资源方法：带有资源方法指示器注解的方法；

@Path用在资源方法上，这类方法被称为子资源；

@Path用在没有资源指示器的方法上，这种方法被称为子资源定位器；

### 3.资源类生命周期

默认情况下，根资源类的生命周期是Request，每次匹配到请求都会创建一个新实例，Jersey支持下面类型的生命周期：

@RequestScoped：默认，每个请求创建一个实例；

@PerLookup：同一个请求只创建一次；

@Singleton：单例，应用程序中只有一个实例；

### 4.注入规则

## Jersey配置

JAX-RS提供了部署无关的抽象类javax.ws.rs.core.Application来声明资源类，Web服务可以扩展这个类来声明Jersey应用程序。

public class MyJerseyApplication extends Application{

@Override

public Set<Class<?>> getClasses() {

Set<Class<?>> set = new HashSet<>();

set.add(JerseyResource.class);

// Spring集成Jersey

set.add(RequestContextFilter.class);

return set;

}

}

或者直接继承自ResourceConfig(Jersey实现的Application类)，配置代码或扩展代码放置在构造函数中，如：

public class JerseyApplication extends ResourceConfig{

/\*\*

\* 注册JAX-RS相关组件

\*/

public JerseyApplication(){

// Spring集成Jersey

register(RequestContextFilter.class);

register(CustomExceptionMapper.class);

register(JerseyResource.class);

}

}

与Application相比，ResourceConfig封装了一些方法来简化JAX-RS组件的注册，如：包名扫描等。

### 1.依赖jetty

dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-jetty-http</artifactId>

<version>2.21</version>

</dependency>

### 2.基于Servlet的部署

Jersey集成Servlet容器支持至少是Servlet2.5规范，Servlet Api 3.0及更改版本的容器支持更多的特性，如：异步请求等。

Servlet2.5环境需要在web.xml配置文件中声明Jersey容器Servlet：

<!-- jersey开始 -->

<servlet>

<servlet-name>SpringApplication</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer

</servlet-class>

<init-param>

<param-name>javax.ws.rs.Application</param-name>

<param-value>com.wangdh.spring.jersey.config.JerseyApplication

</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>SpringApplication</servlet-name>

<url-pattern>/api/\*</url-pattern>

</servlet-mapping>

<!-- jersey结束 -->

也可以将该Servlet注册成过滤器：

<!-- jersey开始 -->

< filter >

< filter -name>SpringApplication</servlet-name>

< filter -class>org.glassfish.jersey.servlet.ServletContainer

</ filter -class>

<init-param>

<param-name>javax.ws.rs.Application</param-name>

<param-value>com.wangdh.spring.jersey.config.JerseyApplication

</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</ filter >

< filter -mapping>

< filter -name>SpringApplication</servlet-name>

<url-pattern>/api/\*</url-pattern>

</ filter -mapping>

<!-- jersey结束 -->

### 3.模块依赖

Jersey 提供了两个 Servlet 模块。第一个模块是 Jersey 核心 Servlet 模块,提供核心 Servlet 需要集成的支持,需要 Servlet 2.5 或更高的容器:

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet-core</artifactId>

</dependency>

为了支持额外的 Servlet 3.x 部署模式和异步 JAX-RS 资源的编程模型,另外一个 Jersey 模块为:

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet</artifactId>

</dependency>

jersey-container-servlet 模块取决于 jersey-container-servlet-core 模块,因此当使用它时,它不需要显式地声明 jersey-container-servlet-core 依赖性。

## 媒体类型与对应的Java类型

JAX-RS支持媒体类型与Java类型的转换，下面是映射关系：

### 所有媒体类型(\*)

byte[]

java.lang.String

java.io.Reader

java.io.File

javax.activation.DataSource

javax.ws.rs.core.StreamingOutput

### Xml媒体类型(text/xml、application/xml、application/+xml)

javax.xml.transform.Source

javax.xml.bind.JAXBElement

应用了 JAXB 类的应用 (使用了 @XmlRootElement 或者 @XmlType 的类型)

### Form表单(application/x-www-form-urlencoded)

MultivaluedMap

### 纯文本(text/plain)

java.lang.Boolean

java.lang.Character

java.lang.Number

## 常用媒体类型的支持

### 1.JSON

MOXy：自动发现并注册，需要添加相应jar包，maven依赖如下：

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-moxy</artifactId>

<version>2.16</version>

</dependency>

Jackson：有1.x和2.x版本，maven依赖如下：

<!—2.x版本 -->

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson</artifactId>

<version>2.16</version>

</dependency>

<!—1.x版本 -->

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson1</artifactId>

<version>2.16</version>

</dependency>

### 2.Multipart：文件上传

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-multipart</artifactId>

<version>2.16</version>

</dependency>