## **一.REST内容协商介绍**

RESTful服务中很重要的一个特性即是同一资源,多种表述.也即如下面描述的三种方式:

1.使用http request header: Accept

http://www.blogjava.net/Images/OutliningIndicators/None.gifGET /user/123 HTTP/1.1  
http://www.blogjava.net/Images/OutliningIndicators/None.gifAccept: application/xml                 //将返回xml格式数据  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gifGET /user/123 HTTP/1.1  
http://www.blogjava.net/Images/OutliningIndicators/None.gifAccept: application/json               //将返回json格式数据

2.使用扩展名

http://www.blogjava.net/Images/OutliningIndicators/None.gif/user/123.xml  将返回xml格式数据  
http://www.blogjava.net/Images/OutliningIndicators/None.gif/user/123.json 将返回json格式数据  
http://www.blogjava.net/Images/OutliningIndicators/None.gif/user/123.html 将返回html格式数据

3.使用参数

/user/123?format=xml          //将返回xml数据  
/user/123?format=json          //将返回json数据

而以上三种各有优缺点:  
1.使用Accept header:  
   这一种为教科书中通常描述的一种,理想中这种方式也是最好的,但如果你的资源要给用户直接通过浏览器访问(即html展现),那么由于浏览器的差异,发送上来的Accept Header头将是不一样的. 将导致服务器不知要返回什么格式的数据给你. 下面是浏览器的Accept Header

http://www.blogjava.net/Images/OutliningIndicators/None.gifchrome:     
http://www.blogjava.net/Images/OutliningIndicators/None.gifAccept:application/xml,application/xhtml+xml,textml;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5     
http://www.blogjava.net/Images/OutliningIndicators/None.gif    
http://www.blogjava.net/Images/OutliningIndicators/None.giffirefox:     
http://www.blogjava.net/Images/OutliningIndicators/None.gifAccept:text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8     
http://www.blogjava.net/Images/OutliningIndicators/None.gif    
http://www.blogjava.net/Images/OutliningIndicators/None.gifIE8:     
http://www.blogjava.net/Images/OutliningIndicators/None.gifAccept:image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash, application/x-silverlight, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, \*/\*

2.使用扩展名  
  丧失了同一url多种展现的方式,但现在这种在实际环境中是使用最多的.因为更加符合程序员的审美观.

3.使用参数  
  可能由于要编写的字符较多,所以较少使用.

带着上面的选择: 使用扩展名,我们来看一下spring中如何配置这部分.

## **二.spring rest配置**

现spring完成内容协商(content negotiation)的工作是由ContentNegotiatingViewResolver来完成的.它的工作模式支持我上面讲的三种,   
ContentNegotiatingViewResolver是根据客户提交的MimeType(如 text/html,application/xml)来跟服务端的一组viewResover的MimeType相比较,如果符合,即返回viewResover的数据.  
而 /user/123.xml, ContentNegotiatingViewResolver会首先将 .xml 根据mediaTypes属性将其转换成 application/xml,然后完成前面所说的比较.

下面是ContentNegotiatingViewResolver的完全配置.

    <!-- 根据客户端的不同的请求决定不同的view进行响应, 如 /blog/1.json /blog/1.xml -->  
    <bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">  
                <!-- 设置为true以忽略对Accept Header的支持-->  
                <property name="ignoreAcceptHeader" value="true"/>  
                <!-- 在没有扩展名时即: "/user/1" 时的默认展现形式 -->  
        <property name="defaultContentType" value="text/html"/>  
          
                <!-- 扩展名至mimeType的映射,即 /user.json => application/json -->  
        <property name="mediaTypes">  
            <map>  
                <entry key="json" value="application/json" />  
                <entry key="xml" value="application/xml" />  
            </map>  
        </property>  
        <!-- 用于开启 /userinfo/123?format=json 的支持 -->  
        <property name="favorParameter" value="false"/>  
        <property name="viewResolvers">  
            <list>  
                <bean class="org.springframework.web.servlet.view.BeanNameViewResolver" />  
                <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
                    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>  
                    <property name="prefix" value="/pages"/>  
                    <property name="suffix" value=".jsp"></property>  
                </bean>  
            </list>  
        </property>  
        <property name="defaultViews">  
            <list>  
                <!-- for application/json -->  
                <bean class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />  
                <!-- for application/xml -->  
                <!--   
                <bean class="org.springframework.web.servlet.view.xml.MarshallingView" >  
                    <property name="marshaller">  
                        <bean class="org.springframework.oxm.xstream.XStreamMarshaller"/>  
                    </property>  
                </bean>  
                 -->  
            </list>  
        </property>  
    </bean>

查看demo:  <http://demo.rapid-framework.org.cn:8080/springmvc_rest_demo/userinfo>  
demo下载:  <http://rapid-framework.googlecode.com/files/springmvc_rest_demo.zip>

一

RESTful服务中很重要的一个特性即是同一资源,多种表述我们使用ContentNegotiatingViewResolver就可以做到，这个视图解析器允许你用同样的内容数据来呈现不同的view

如下面描述的三种方式:

方式1  使用扩展名

<http://www.test.com/user.xml>    呈现xml文件

<http://www.test.com/user.json>    呈现json格式

<http://www.test.com/user>       使用默认view呈现，比如jsp等

-----------------------------------------------------------------------------------------------------

方式2  使用http request header的Accept

GET /user HTTP/1.1

Accept:application/xml

GET /user HTTP/1.1

Accept:application/json

….

-----------------------------------------------------------------------------------------------------

方式3  使用参数

<http://www.test.com/user?format=xml>

<http://www.test.com/user?format=json>

这三种方式各自的优缺点这里就不再介绍了

如何使用ContentNegotiatingViewResolver？

假设我们有这么一个目标：

/user/{userid}.json    用于返回一个描述User的JSON

/user/{userid}        用于返回一个展示User的JSP页面

/user/{userid}.xml     用于返回一个展示User的XML文件

配置文件说明   （具体例子下篇文章放上）

我们知道有accept header,扩展名以及参数这三种方式，配置文件中

这里是解析器的执行顺序，如果有多个的话（前面多次解释过）

<property name=*"order"* value=*"1"*></property>

--------------------------------------------------------------------------------------------------------------

如果所有的mediaType都没匹配上，就会使用defaultContentType

<property name=*"defaultContentType"* value=*"text/html"* />

这里是是否启用扩展名支持，默认就是true

例如  /user/{userid}.json

<property name=*"favorPathExtension"* value=*"true"*></property>

这里是是否启用参数支持，默认就是true

例如  /user/{userid}?format=json

<property name=*"favorParameter"*value=*"false"*></property>

这里是否忽略掉accept header，默认就是false

例如     GET /user HTTP/1.1

Accept:application/json

<property name=*"ignoreAcceptHeader"* value=*"true"*></property>

我们的例子是采用.json , .xml结尾的,所以关掉两个

--------------------------------------------------------------------------------------------------------------

这里是扩展名到mimeType的映射,

例如 /user/{userid}.json  中的   .json  就会映射到   application/json

<property name=*"mediaTypes"*>

           <map>

              <entry key=*"json"*value=*"*application/json*"* />

              <entry key=*"xml"*value=*"*application/xml*"*/>                     </map>

</property>

注:

ContentNegotiatingViewResolver是根据客户提交的MimeType(如 text/html,application/xml)来跟服务端的一组viewResover的MimeType相比较,如果符合,即返回viewResover的数据.  
而 /user/123.xml, ContentNegotiatingViewResolver会首先将 .xml 根据mediaTypes属性将其转换成 application/xml,然后完成前面所说的比较.

使用内容协商实现多视图例

根据前篇文件的介绍，这里直接给出例子

配置xml

<context:component-scan base-package=*"com.controls"* />

    <context:annotation-config />

    <bean class=*"org.springframework.web.servlet.view.ContentNegotiatingViewResolver"*>

        <property name=*"order"* value=*"1"* />

        <property name=*"favorParameter"*value=*"false"* />

        <property name=*"ignoreAcceptHeader"*value=*"true"* />

        <property name=*"mediaTypes"*>

            <map>

                <entry key=*"json"* value=*"application/json"* />

                <entry key=*"xml"* value=*"application/xml"* />

            </map>

        </property>

        <property name=*"defaultViews"*>

            <list>

                <bean class=*"org.springframework.web.servlet.view.json.MappingJacksonJsonView"*></bean>

                <bean class=*"org.springframework.web.servlet.view.xml.MarshallingView"*>

                    <constructor-arg>

                        <bean class=*"org.springframework.oxm.jaxb.Jaxb2Marshaller"*>

                             <property name=*"classesToBeBound"*>

                                <list>

                                   <value>com.model.User</value>

                                </list>

                             </property>

                        </bean>

                    </constructor-arg>

                </bean>

            </list>

        </property>

    </bean>

    <!-- 上面没匹配到则会使用这个视图解析器 -->

    <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

        <property name=*"order"* value=*"2"* />

        <property name=*"prefix"* value=*"/WEB-INF/views/"* />

        <property name=*"suffix"* value=*".jsp"* />

        <property name=*"viewClass"* value=*"org.springframework.web.servlet.view.JstlView"* />

    </bean>

Model

@XmlRootElement

**public class** User {

**private long** userID;

**private** String userName;

**private** Date birth;

**public** String getUserName() {

**return** userName;

    }

**public void** setUserName(String userName) {

**this**.userName = userName;

    }

**public** Date getBirth() {

**return** birth;

    }

**public void** setBirth(Date birth) {

**this**.birth = birth;

    }

**public long** getUserID() {

**return** userID;

    }

**public void** setUserID(**long** userID) {

**this**.userID = userID;

    }

}

Contoller

@RequestMapping(value="/user/{userid}")

**public** String queryUser(@PathVariable("userid") **long** userID, ModelMap model)

{

       User u = **new** User();

       u.setUserID(userID);

       u.setUserName("zhaoyang");

       model.addAttribute("User", u);

**return** "User";

}

如果是返回text/html，还需要建立个jsp文件

<body>

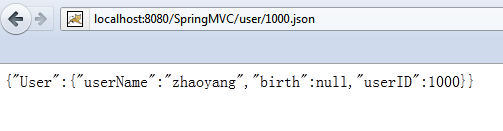
    UserName: ${requestScope.User.userID } <br />

    Age: ${requestScope.User.userName }

  </body>

测试结果

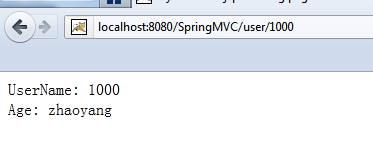
json



xml



jsp



二

参考文章：http://blog.csdn.net/z69183787/article/details/41658489

http://blog.csdn.net/z69183787/article/details/41693015

http://blog.csdn.net/z69183787/article/details/41693035

http://blog.csdn.net/z69183787/article/details/41654603

参考了上面几篇文章后，发现还不是了解的很透彻，比如defaultViews、contentType与匹配规则的关系、视图匹配原则，InternalResourceViewResolver为何无法匹配等等，之后将spring-webmvc的java代码本地编译，并设了断点后一步步调试，才终于搞明白。

附上关键xml配置：

**[html]** [view plaincopy](http://blog.csdn.net/z69183787/article/details/41696709)[在CODE上查看代码片](https://code.csdn.net/snippets/540808)

1. **<bean** id="cnManager" class="org.springframework.web.accept.ContentNegotiationManagerFactoryBean"**>**
2. **<property** name="ignoreAcceptHeader" value="true"**/>**
3. **<property** name="favorPathExtension" value="true"**/>**
4. **<property** name="defaultContentType" value="text/html"**/>**
5. **<property** name="favorParameter" value="true"**/>**
6. **<property** name="mediaTypes"**>**
7. **<map>**
8. **<entry** key="xml" value="application/xml"**/>**
9. **<entry** key="json" value="text/plain"**/>**
10. **<entry** key="xls" value="application/vnd.ms-excel"**/>**
11. **</map>**
12. **</property>**
13. **</bean>**

**[html]** [view plaincopy](http://blog.csdn.net/z69183787/article/details/41696709)[在CODE上查看代码片](https://code.csdn.net/snippets/540808)

1. **<bean** id="viewResolver"
2. class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver"**>**
3. **<property** name="contentNegotiationManager" ref="cnManager"**/>**
4. **<property** name="viewResolvers"**>**<!-- 针对freemarker的视图配置 -->
5. **<list>**
6. **<bean** class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver"**>**
7. **<property** name="viewClass" value="org.springframework.web.servlet.view.freemarker.FreeMarkerView"**/>**
8. **<property** name="viewNames" value=".ftl"**/>**
9. **<property** name="contentType" value="text/html; charset=utf-8"**/>**
10. **<property** name="cache" value="true" **/>**
11. **<property** name="prefix" value="" **/>**
12. **<property** name="suffix" value="" **/>**
13. **<property** name="order" value="2"**/>**
14. **</bean>**
15. **<bean** class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
16. **<property** name="prefix" value="/WEB-INF/view/jsp/"**></property>**<!-- 配置页面路径 -->
17. **<property** name="suffix" value=".jsp"**></property>**<!-- 文件以value值结尾 -->
18. **</bean>**
19. **</list>**
20. **</property>**
21. **<property** name="defaultViews"**>**
22. **<list>**
23. **<bean** class="org.springframework.web.servlet.view.json.MappingJackson2JsonView"**>**
24. <!-- <property name="extractValueFromSingleKeyModel" value="true"/> -->
25. **<property** name="prettyPrint" value="true"**/>**
26. **<property** name="contentType" value="text/plain"**/>**
27. **</bean>**
28. **<bean** class="org.springframework.web.servlet.view.xml.MarshallingView"**>**
29. **<constructor-arg** ref="jaxb2Marshaller"**>**
31. **</constructor-arg>**
32. **</bean>**
33. **<bean** class="com.wonders.stpt.bid.controller.JXLExcelView"**/>**
34. **</list>**
35. **</property>**
36. **</bean>**

步骤概要：

方法resolveViewName中根据方法getMediaTypes获取mediaType（由accept-header、请求参数或后缀名中获取，springmvc每个view的默认contenttype为text/html），根据访问的viewname与mediaType查找候选视图集合candidateViews，再根据方法getBestView获取最优视图并返回。

其中，ContentNegotiatingViewResolver 中包含2个属性：

ContentNegotiationManagerFactoryBean（**XML中viewResolver**）：使用ContentNegotiationStrateg解析策略构造ContentNegotiationManager（afterPropertiesSet方法中初始化）

ContentNegotiationManager（**XML中cnManager**）：包含List<ContentNegotiationStrategy> contentNegotiationStrategies属性记录匹配请求的解析策略，可能涉及的解析策略如下

1、org.springframework.web.accept.ServletPathExtensionContentNegotiationStrategy

后缀名，例：请求为 aaa.json

2、org.springframework.web.accept.ParameterContentNegotiationStrategy

参数，默认参数名称为format，例：请求为format=json

3、org.springframework.web.accept.HeaderContentNegotiationStrategy

通过accept-header匹配 content-type

4、org.springframework.web.accept.FixedContentNegotiationStrategy

通过设定defaultContentType content-type

**[html]** [view plaincopy](http://blog.csdn.net/z69183787/article/details/41696709)[在CODE上查看代码片](https://code.csdn.net/snippets/540808)

1. **<span** style="font-size:12px;"**>** **<property** name="ignoreAcceptHeader" value="true"**/>**
2. **<property** name="favorPathExtension" value="true"**/>**
3. **<property** name="defaultContentType" value="text/html"**/>**
4. **<property** name="favorParameter" value="true"**/>**
5. **<property** name="mediaTypes"**>**
6. **<map>**
7. **<entry** key="xml" value="application/xml"**/>**
8. **<entry** key="json" value="text/plain"**/>**
9. **<entry** key="xls" value="application/vnd.ms-excel"**/>**
10. **</map>**
11. **</property></span>**

favorPathExtension表示支持后缀匹配，favorParameter表示支持参数匹配，属性ignoreAcceptHeader默认为fasle，表示accept-header匹配，defaultContentType开启默认匹配。

其中cnManager中的mediaTypes预设key对应后缀名及参数format的值， 与匹配策略1、2相关。

例如：请求aaa.xx，若设置<entry key="xx" value="application/xml"/> 也能匹配以xml返回。

根据以上条件进行一一匹配最终，得到相关并符合的策略初始化ContentNegotiationManager

1、getMediaTypes方法使用ContentNegotiationManager的resolveMediaTypes方法（使用上述匹配策略）得到需要的contenttype类型例：请求aaa.xx，若设置<entry key="xx" value="application/xml"/> ，返回application/xml的MediaType

2、getCandidateViews方法根据XML中预设的viewResolvers匹配1中的MediaType获取候选View，同时增加XML中defaultViews中的view。每个view都有对应的content-type，大部分的视图继承于AbstractView，默认值都是text/html;charset=ISO-8859-1。有些特定视图，比如MappingJackson2JsonView会重设contentType为application/json。每个view中的contentType在最后匹配中都会实际用到，所以很重要。

在此方法中，会调用每一个viewResolvers的resolveViewName方法创建view，需要传入的参数为viewname与locale，创建过程中，

会先从cache中查看是否已经匹配过，若存在，直接返回view；若不存在，继续创建view。

第一步：检测viewResolvers中的viewNames属性，<property name="viewNames" value=".ftl"/>（支持正则表达式 如\*.ftl,），

表达式：return (viewNames == null || PatternMatchUtils.simpleMatch(viewNames, viewName));

若未设置viewNames属性，则默认为null，允许创建，进入第二步；若设置，会根据viewNames进行正则表达式的匹配，若匹配继续第二步，若不匹配，返回null view，跳出此viewResolver；继续与下一个viewresolvers进行匹配并获得view。

第二步：进行每个viewResolve都有的view的创建过程（初始化content-type,记录最终跳转页面prefix，postfix等），但诸如Freemarker，velocity 的 viewResolvers都会在view创建过程中的最后一步调用重写checkResource方法，查看最终跳转的ftl或vm文件是否存在？若不存在，则将创建的视图View重设成null并返回，故无法匹配并加入候选视图。

而InternalResourceView的checkResource没有重写，调用父类AbstractUrlBasedView的checkResource，默认返回true，为所有请求创建view，无论是否有最终文件。所以，InternalResourceView一般都作为springmvc最后的视图解析器，来处理一切请求。

**[java]** [view plaincopy](http://blog.csdn.net/z69183787/article/details/41696709)[在CODE上查看代码片](https://code.csdn.net/snippets/540808)

1. **private** List<View> getCandidateViews(String viewName, Locale locale, List<MediaType> requestedMediaTypes)
2. **throws** Exception {
4. List<View> candidateViews = **new** ArrayList<View>();
5. **for** (ViewResolver viewResolver : **this**.viewResolvers) {
6. View view = viewResolver.resolveViewName(viewName, locale);
7. **if** (view != **null**) {
8. candidateViews.add(view);
9. }
10. **for** (MediaType requestedMediaType : requestedMediaTypes) {
11. List<String> extensions = **this**.contentNegotiationManager.resolveFileExtensions(requestedMediaType);
12. **for** (String extension : extensions) {
13. String viewNameWithExtension = viewName + "." + extension;
14. view = viewResolver.resolveViewName(viewNameWithExtension, locale);
15. **if** (view != **null**) {
16. candidateViews.add(view);
17. }
18. }
19. }
20. }
21. **if** (!CollectionUtils.isEmpty(**this**.defaultViews)) {
22. candidateViews.addAll(**this**.defaultViews);
23. }
24. **return** candidateViews;
25. }

3、getBestView方法获取最符合条件的匹配视图

首先，如果是smartView，即redirectView则return

否则：筛选getMediaTypes返回的content-type类型，与candidateViews中的候选视图匹配，其中：

下面的contentType重写了MappingJackson2JsonView的contentType，用于匹配cnManager中的mediaTypes，若该视图可以解析此contenttype，则返回

例子：若访问localhost/xxx/a.json，匹配策略为后缀匹配（favorExtension开启），由于后缀为json，则匹配cnManager中的text/plain筛选view后发现匹配视图为MappingJackson2JsonView，返回json格式的数据；若访问localhost/xxx/a,由于无后缀及参数匹配，则匹配策略为accept-header匹配，由于content-type为text/html，则匹配InternalResourceView中的视图，返回jsp页面。

你可能注意到，上述viewResolvers中配置了两个视图，为什么匹配InternalResourceView,而不是freeMarker？这是因为进入候选试图的只有InternalResourceView视图解析器，具体原因参见getCandidateViews。

**[html]** [view plaincopy](http://blog.csdn.net/z69183787/article/details/41696709)[在CODE上查看代码片](https://code.csdn.net/snippets/540808)

1. **<bean** class="org.springframework.web.servlet.view.json.MappingJackson2JsonView"**>**
2. <!-- <property name="extractValueFromSingleKeyModel" value="true"/> -->
3. **<property** name="prettyPrint" value="true"**/>**
4. **<property** name="contentType" value="text/plain"**/>**
5. **</bean>**

**[java]** [view plaincopy](http://blog.csdn.net/z69183787/article/details/41696709)[在CODE上查看代码片](https://code.csdn.net/snippets/540808)

1. **private** View getBestView(List<View> candidateViews, List<MediaType> requestedMediaTypes, RequestAttributes attrs) {
2. **for** (View candidateView : candidateViews) {
3. **if** (candidateView **instanceof** SmartView) {
4. SmartView smartView = (SmartView) candidateView;
5. **if** (smartView.isRedirectView()) {
6. **if** (logger.isDebugEnabled()) {
7. logger.debug("Returning redirect view [" + candidateView + "]");
8. }
9. **return** candidateView;
10. }
11. }
12. }
13. **for** (MediaType mediaType : requestedMediaTypes) {
14. **for** (View candidateView : candidateViews) {
15. **if** (StringUtils.hasText(candidateView.getContentType())) {
16. MediaType candidateContentType = MediaType.parseMediaType(candidateView.getContentType());
17. **if** (mediaType.isCompatibleWith(candidateContentType)) {
18. **if** (logger.isDebugEnabled()) {
19. logger.debug("Returning [" + candidateView + "] based on requested media type '"
20. + mediaType + "'");
21. }
22. attrs.setAttribute(View.SELECTED\_CONTENT\_TYPE, mediaType, RequestAttributes.SCOPE\_REQUEST);
23. **return** candidateView;
24. }
25. }
26. }
27. }
28. **return** **null**;
29. }