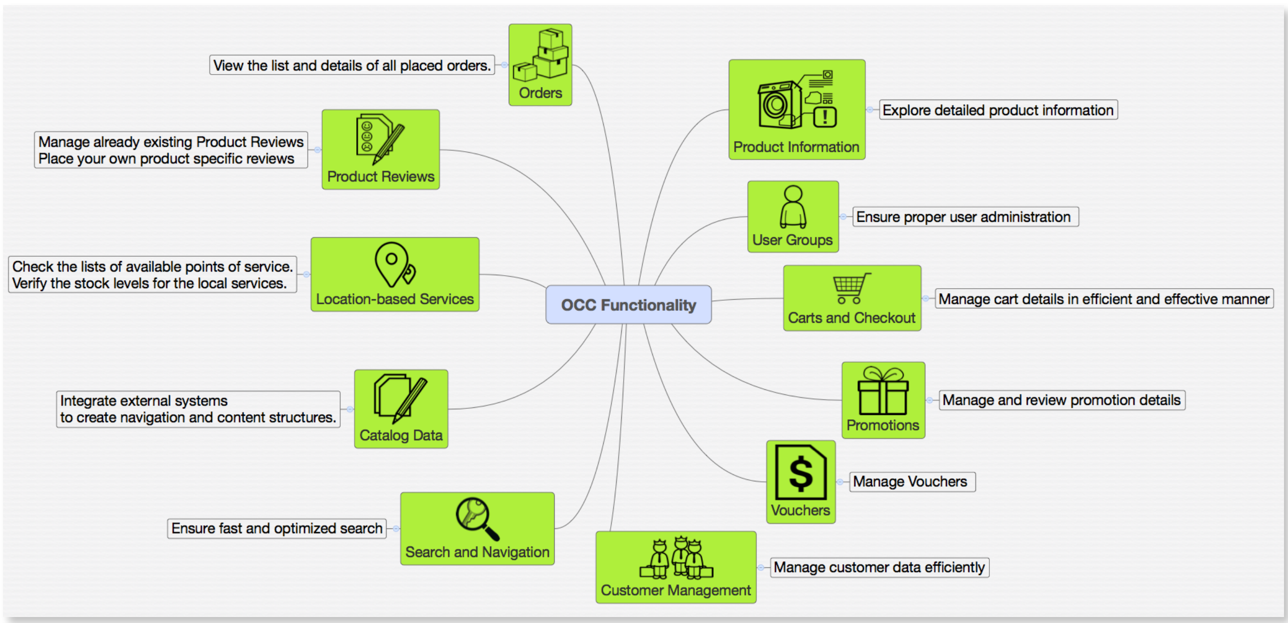
Hybris OCC

环境搭建与自定义操作

鲍传琦 - 2015年5月4日


OCC Functionality
Orders
View the list and details of all placed orders.
Product Reviews
Manage already existing Product Reviews
Place your own product specific reviews
Location-based Services
Check the lists of available points of service.
Verify the stock levels for the local services.
Catalog Data
Integrate external systems
to create navigation and content structures.
Search and Navigation
Ensure fast and optimized search
Customer Management
Manage customer data efficiently
Product Information
Explore detailed product information
User Groups
Ensure proper user administration
Carts and Checkout
Manage cart details in efficient and effective manner
Promotions
Manage and review promotion details
Vouchers
Manage Vouchers

The Omni Commerce Connect (OCC) is a next-generation commerce API that offers a broad set of commerce and data services which enable you to use and leverage the complete hybris Commerce Suite functionality anywhere in your existing application landscape. Its main benefits are as follows:

1. It allows hybris customers to quickly commerce-enable new touch points and channels without lengthy and costly IT cycles.

2. It makes it easy to reuse commerce processes and data across all touch points, increasing the speed and lowering the costs of providing new transactional interfaces.

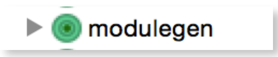
3. It is not restricted to human user interfaces; you can easily integrate with other systems and even provide interfaces to partners and other organizations.

The Omni Commerce Connect (OCC) is based on RESTful web services. To maximize the list of potential API clients, the OCC supports both XML and JSON representations. Additional built-in features include ETag-based caching and easy authentication via HTTP Basic Authentication secured via HTTPS.

参考网址：https://wiki.hybris.com/display/release5/Omni+Commerce+Connect+Module

部署

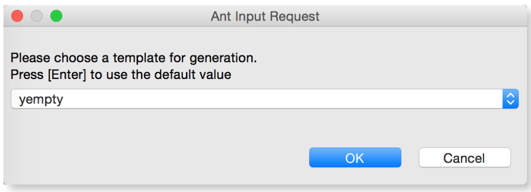
1. 相关准备
   环境：JDK_1.7版本、Hybris5.5版本
   编译：Ant_1.9.1
   工具：STS_3.6.4 (Spring Tool Suite)
具体怎么对Hybris进行初始化安装配置，在此不做多描述，可见trail：
https://wiki.hybris.com/display/R5T/Trail+~+Preparation

2. 在bin文件夹中新建custom文件夹，在eclipse中利用ant命令执行 

命令，利用模板创建一系列自定义模板工程文件。

3.在eclipse中利用ant命令

执行命令，利用yempty模板建立service工程。

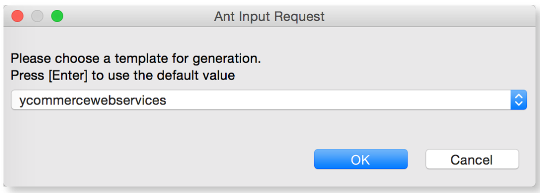

Ant Input Request

Please choose a template for generation.
Press [Enter] to use the default value

yempty

4. 在eclipse中利用ant命令

执行命令，利用ycommercewebservice模板建立webservice工程。


Ant Input Request

Please choose a template for generation.
Press [Enter] to use the default value

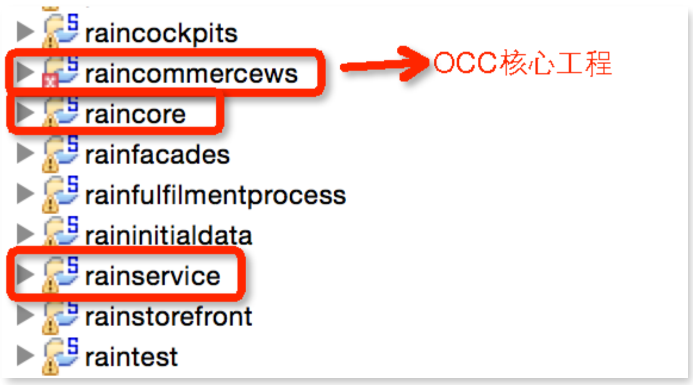
ycommercewebservices

5. 以上，所有工程建立完毕。OCC的核心工程是以ycommercewebservice模板建立的，整个项目工程的层次结构主要是：

core工程：核心数据定义(model层面)

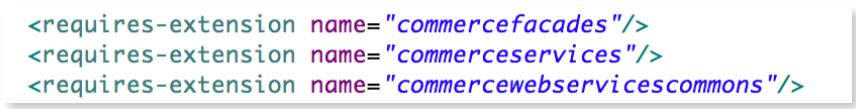
service工程：基于model数据逻辑处理(DAO)

occ核心工程：facades层面对基于data(Spring Bean)数据进行处理，web层面提供调用API，对获取的数据进行解析转换



依赖关系：
service工程依赖于core工程，
occ核心工程依赖于service工程。

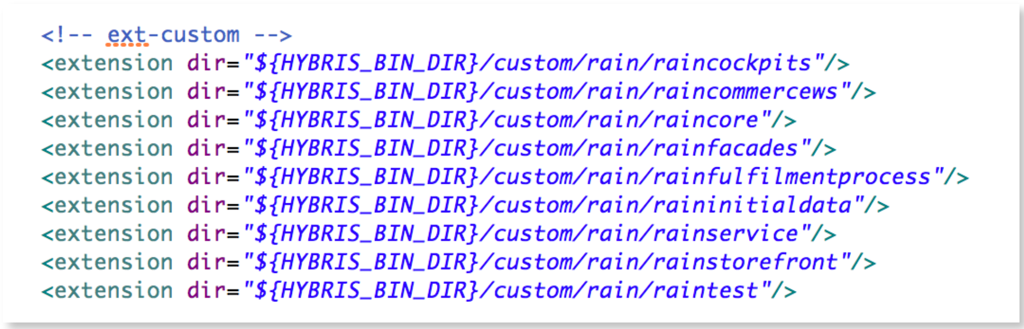
6. 根据occ核心工程中的extensioninfo.xml中，此extension主要依赖的工程：



```xml
<requires-extension name="commercefacades"/>
<requires-extension name="commerceservices"/>
<requires-extension name="commercewebservicescommons"/>
```

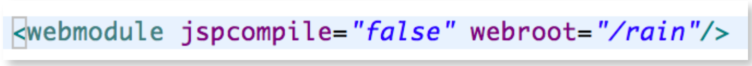
在对应的config/localextension.xml中，加入这些工程的extension。

7. 在config/localextension.xml中加入自定义工程的extension：



```xml
<!-- ext-custom -->
<extension dir="${HYBRIS_BIN_DIR}/custom/rain/raincockpits"/>
<extension dir="${HYBRIS_BIN_DIR}/custom/rain/raincommercews"/>
<extension dir="${HYBRIS_BIN_DIR}/custom/rain/raincore"/>
<extension dir="${HYBRIS_BIN_DIR}/custom/rain/rainfacades"/>
<extension dir="${HYBRIS_BIN_DIR}/custom/rain/rainfulfilmentprocess"/>
<extension dir="${HYBRIS_BIN_DIR}/custom/rain/raininitialdata"/>
<extension dir="${HYBRIS_BIN_DIR}/custom/rain/rainservice"/>
<extension dir="${HYBRIS_BIN_DIR}/custom/rain/rainstorefront"/>
<extension dir="${HYBRIS_BIN_DIR}/custom/rain/raintest"/>
```

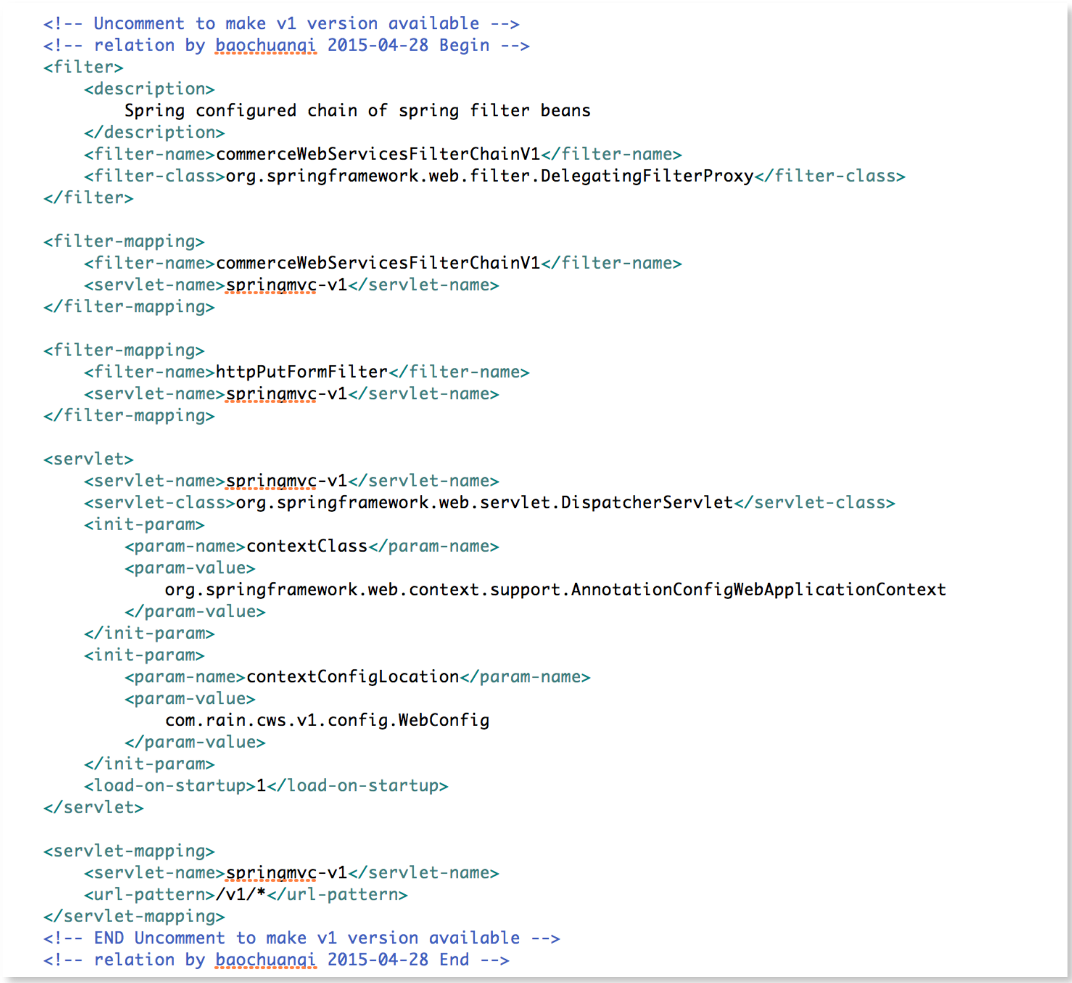
8. 更改occ核心工程中的extensioninfo.xml中webroot，定义webservice的访问路径。



```xml
<webmodule jspcompile="false" webroot="/rain"/>
```

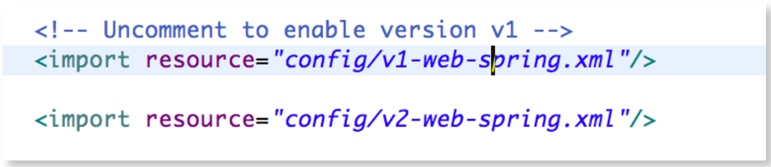
基本配置

1. Hybris OCC提供的Webservice版本有两套，V1与V2版本。Hybris5.5默认开启版本V2。具体的配置信息在occ核心工程中web/webroot/WEB_INF/web.xml中：



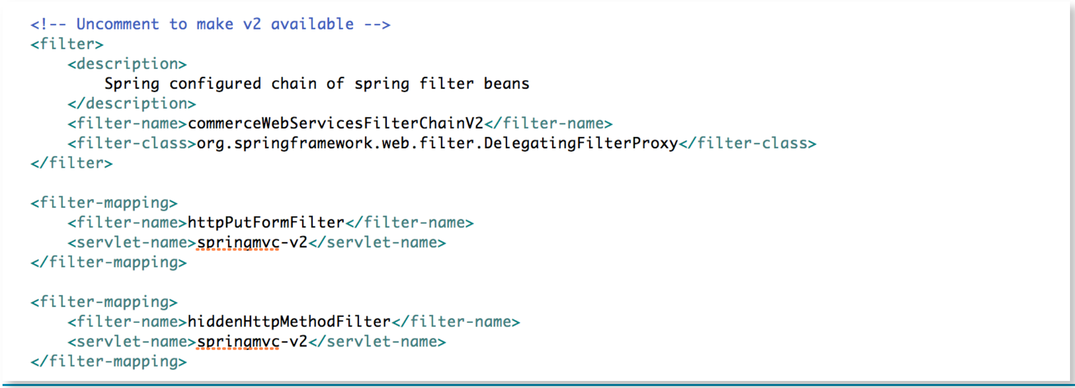
```xml
<!-- Uncomment to make v1 version available -->
<!-- relation by baochuangi 2015-04-28 Begin -->
<filter>
    <description>
        Spring configured chain of spring filter beans
    </description>
    <filter-name>commerceWebServicesFilterChainV1</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>commerceWebServicesFilterChainV1</filter-name>
    <servlet-name>springmvc-v1</servlet-name>
</filter-mapping>

<filter-mapping>
    <filter-name>httpPutFormFilter</filter-name>
    <servlet-name>springmvc-v1</servlet-name>
</filter-mapping>

<servlet>
    <servlet-name>springmvc-v1</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextClass</param-name>
        <param-value>
            org.springframework.web.context.support.AnnotationConfigWebApplicationContext
        </param-value>
    </init-param>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            com.rain.cws.v1.config.WebConfig
        </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>springmvc-v1</servlet-name>
    <url-pattern>/v1/*</url-pattern>
</servlet-mapping>
<!-- END Uncomment to make v1 version available -->
<!-- relation by baochuangi 2015-04-28 End -->
```

V1配置信息(在标准工程中，此段配置是被注释掉的，所以通过V1版本去访问，是要把这段代码释放出来，保证V1版本生效！)



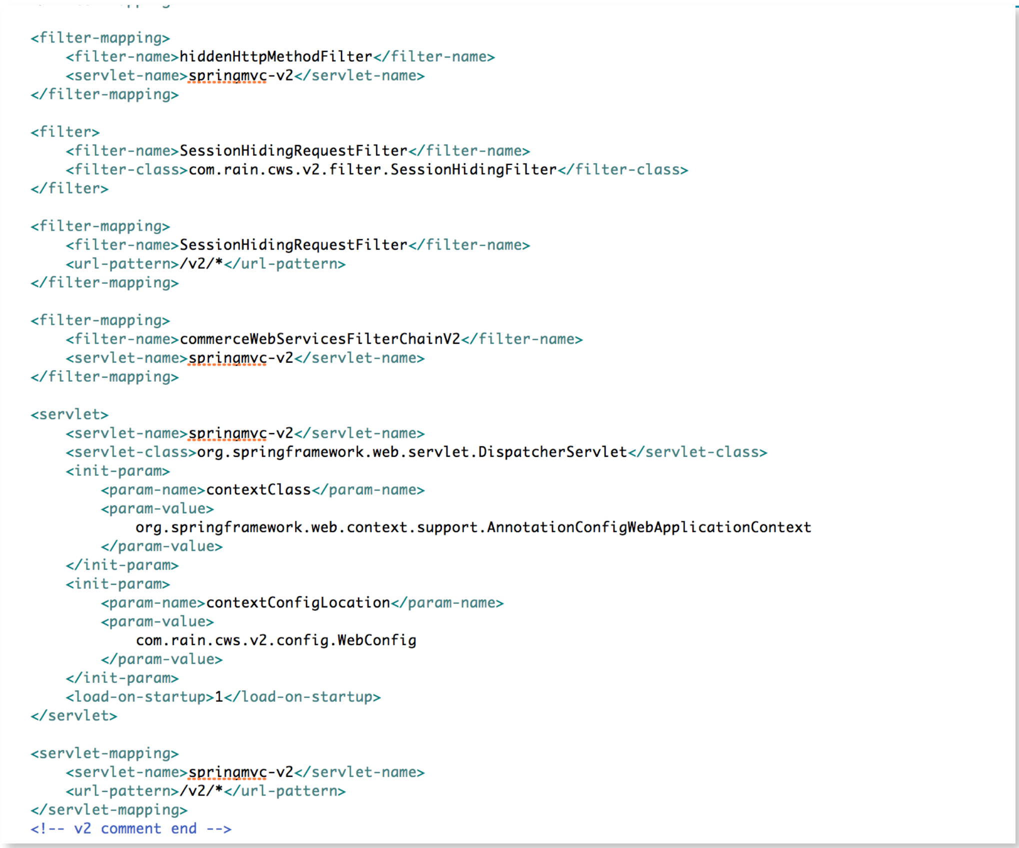
```xml
<!-- Uncomment to enable version v1 -->
<import resource="config/v1-web-spring.xml"/>

<import resource="config/v2-web-spring.xml"/>
```

如若使V1生效，还需修改web/webroot/WEB_INF/*******-web-spring.xml文件，具体修改如图。



```xml
<!-- Uncomment to make v2 available -->
<filter>
    <description>
        Spring configured chain of spring filter beans
    </description>
    <filter-name>commerceWebServicesFilterChainV2</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>httpPutFormFilter</filter-name>
    <servlet-name>springmvc-v2</servlet-name>
</filter-mapping>

<filter-mapping>
    <filter-name>hiddenHttpMethodFilter</filter-name>
    <servlet-name>springmvc-v2</servlet-name>
</filter-mapping>
```

V2配置信息

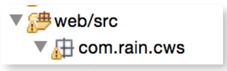


```
<filter-mapping>
    <filter-name>hiddenHttpMethodFilter</filter-name>
    <servlet-name>springmvc-v2</servlet-name>
</filter-mapping>

<filter>
    <filter-name>SessionHidingRequestFilter</filter-name>
    <filter-class>com.rain.cws.v2.filter.SessionHidingFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>SessionHidingRequestFilter</filter-name>
    <url-pattern>/v2/*</url-pattern>
</filter-mapping>

<filter-mapping>
    <filter-name>commerceWebServicesFilterChainV2</filter-name>
    <servlet-name>springmvc-v2</servlet-name>
</filter-mapping>

<servlet>
    <servlet-name>springmvc-v2</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextClass</param-name>
        <param-value>
            org.springframework.web.context.support.AnnotationConfigWebApplicationContext
        </param-value>
    </init-param>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            com.rain.cws.v2.config.WebConfig
        </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>springmvc-v2</servlet-name>
    <url-pattern>/v2/*</url-pattern>
</servlet-mapping>
<!-- v2 comment end -->
```

V2配置信息

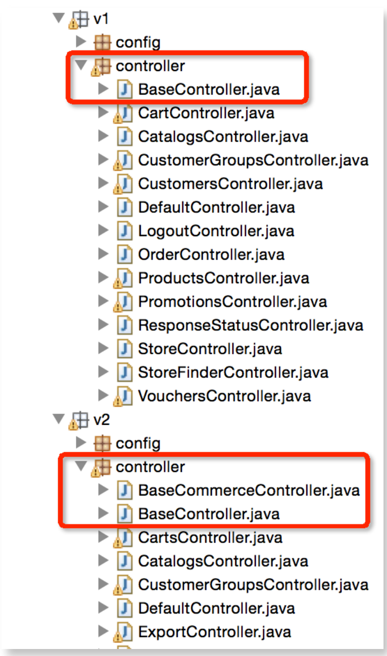
2. 在做完这些配置后，可以对Hybris进行ant clean all编译，并启动，注意Update(具体Update哪一项，可自行判断，根据Type，Localization等)。

3. 基本上，我们在web/src文件夹中对应的包package

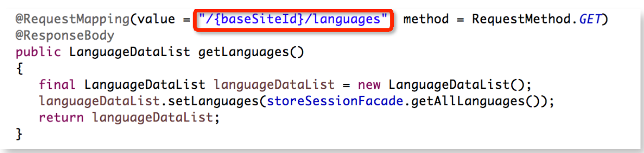


```
▼ web/src
  ▼ com.rain.cws
```

中可以看到v1，v2两个包，occ提供的webservice的入口程序，主要在这两个包中的controller包中。



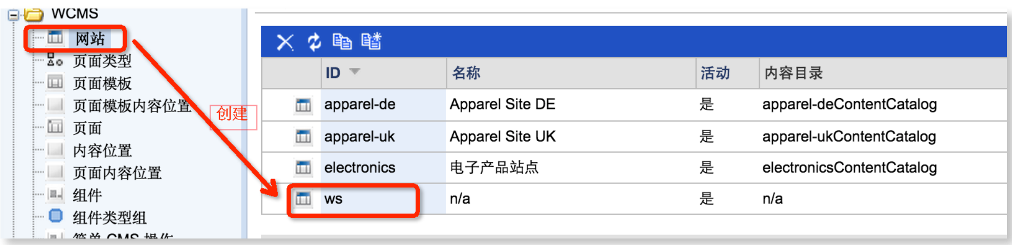
```
▼ v1
  ▶ config
  ▼ controller
    ▶ BaseController.java
    ▶ CartController.java
    ▶ CatalogsController.java
    ▶ CustomerGroupsController.java
    ▶ CustomersController.java
    ▶ DefaultController.java
    ▶ LogoutController.java
    ▶ OrderController.java
    ▶ ProductsController.java
    ▶ PromotionsController.java
    ▶ ResponseStatusController.java
    ▶ StoreController.java
    ▶ StoreFinderController.java
    ▶ VouchersController.java
▼ v2
  ▶ config
  ▼ controller
    ▶ BaseCommerceController.java
    ▶ BaseController.java
    ▶ CartsController.java
    ▶ CatalogsController.java
    ▶ CustomerGroupsController.java
    ▶ DefaultController.java
    ▶ ExportController.java
```



```
@RequestMapping(value = "/{baseSiteId}/languages" method = RequestMethod.GET)
@ResponseBody
public LanguageDataList getLanguages()
{
   final LanguageDataList languageDataList = new LanguageDataList();
   languageDataList.setLanguages(storeSessionFacade.getAllLanguages());
   return languageDataList;
}
```

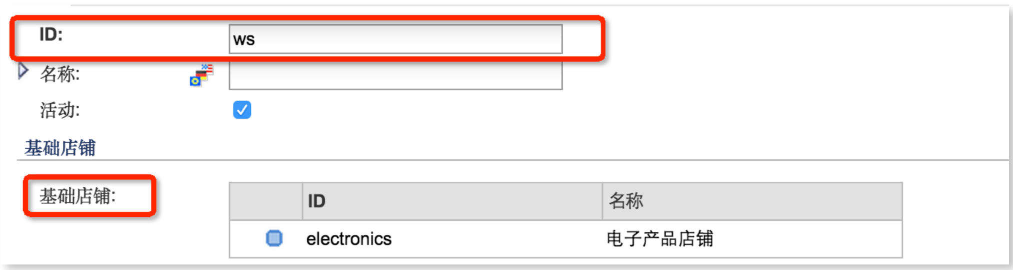
由此两张图可见，occ的入口就是通过一个个网址去访问到具体的方法中，通过Http请求，设置访问的路径和参数，最后返回一个结果。{baseSiteId}就是我们需要配置的站点CMSite信息，基本V1和V2版本都是需要在HMC中定义站点来访问的！(核心的实现逻辑可以了解一下SpringMVC)

4. 在HMC中配置站点。



ID	名称	活动	内容目录
apparel-de	Apparel Site DE	是	apparel-deContentCatalog
apparel-uk	Apparel Site UK	是	apparel-ukContentCatalog
electronics	电子产品站点	是	electronicsContentCatalog
ws	n/a	是	n/a

创建站点，名字跟访问路径有关。



ID:	ws
名称:	
活动:	✓

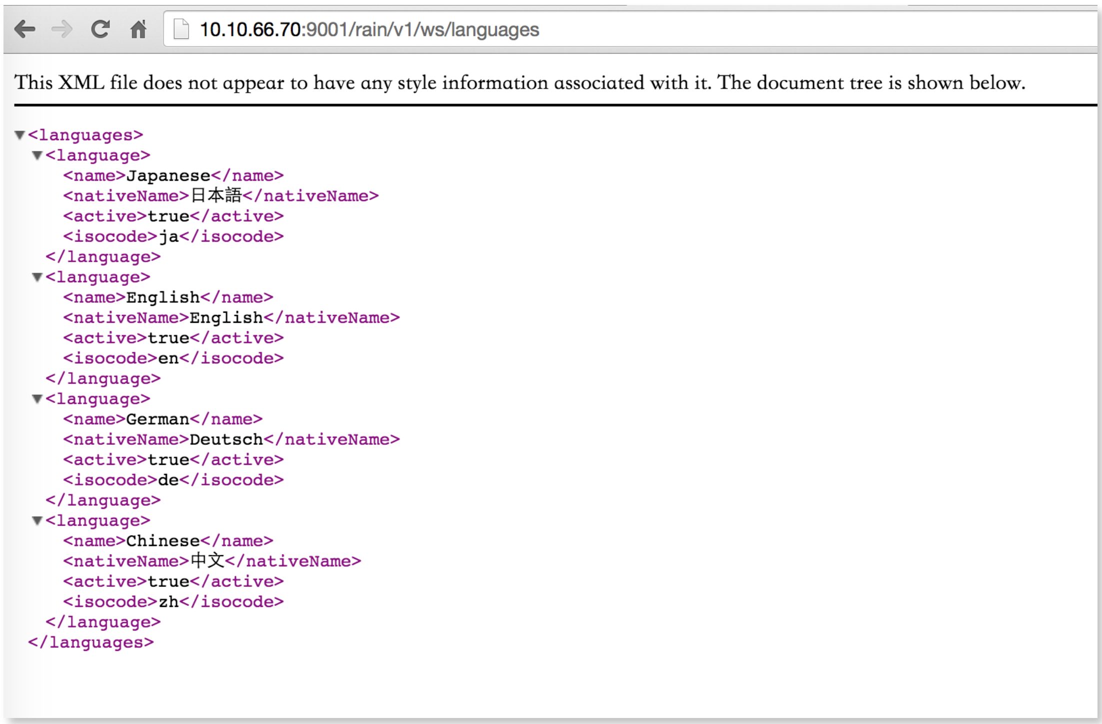
基础店铺

基础店铺:

ID	名称
electronics	电子产品店铺

主要是ID(字段名uid)，还有关联的店铺。

5. 配置信息完成，测试get方法，获取语言。
http://{ServerIP}:9001/rain/v1/ws/languages

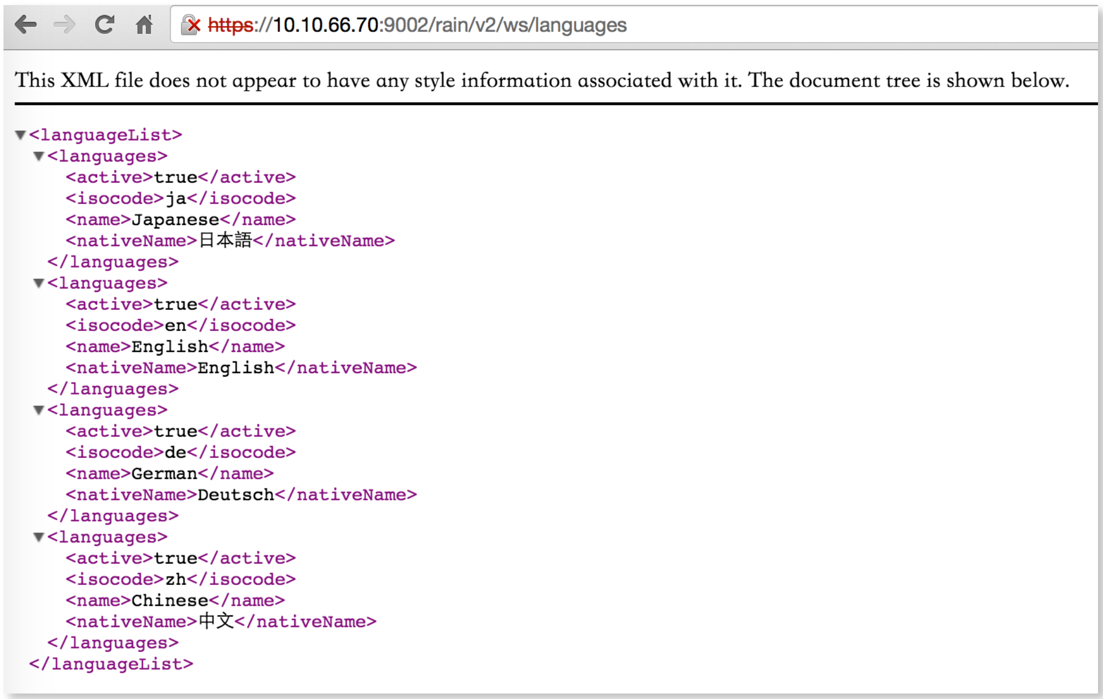


10.10.66.70:9001/rain/v1/ws/languages

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<languages>
  ▼<language>
     <name>Japanese</name>
     <nativeName>日本語</nativeName>
     <active>true</active>
     <isocode>ja</isocode>
   </language>
  ▼<language>
     <name>English</name>
     <nativeName>English</nativeName>
     <active>true</active>
     <isocode>en</isocode>
   </language>
  ▼<language>
     <name>German</name>
     <nativeName>Deutsch</nativeName>
     <active>true</active>
     <isocode>de</isocode>
   </language>
  ▼<language>
     <name>Chinese</name>
     <nativeName>中文</nativeName>
     <active>true</active>
     <isocode>zh</isocode>
   </language>
 </languages>
```

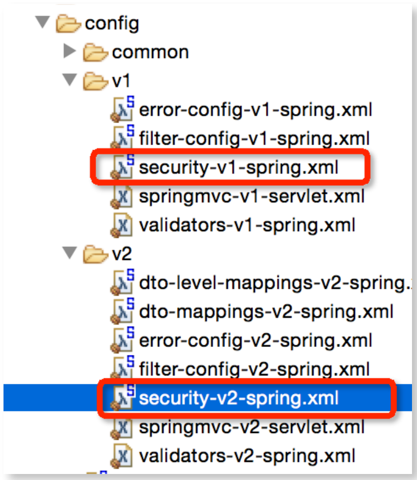
https://{ServerIP}:9002/rain/v2/ws/languages


https://10.10.66.70:9002/rain/v2/ws/languages

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<languageList>
  ▼<languages>
     <active>true</active>
     <isocode>ja</isocode>
     <name>Japanese</name>
     <nativeName>日本語</nativeName>
   </languages>
  ▼<languages>
     <active>true</active>
     <isocode>en</isocode>
     <name>English</name>
     <nativeName>English</nativeName>
   </languages>
  ▼<languages>
     <active>true</active>
     <isocode>de</isocode>
     <name>German</name>
     <nativeName>Deutsch</nativeName>
   </languages>
  ▼<languages>
     <active>true</active>
     <isocode>zh</isocode>
     <name>Chinese</name>
     <nativeName>中文</nativeName>
   </languages>
 </languageList>
```

6. v1与v2访问的链接(http/https)，在对应文件夹的配置文件中。


config
common
v1
error-config-v1-spring.xml
filter-config-v1-spring.xml
security-v1-spring.xml
springmvc-v1-servlet.xml
validators-v1-spring.xml
v2
dto-level-mappings-v2-spring.
dto-mappings-v2-spring.xml
error-config-v2-spring.xml
filter-config-v2-spring.xml
security-v2-spring.xml
springmvc-v2-servlet.xml
validators-v2-spring.xml

例:

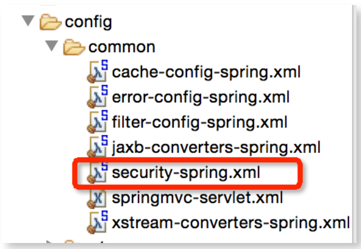
```
<intercept-url pattern="/**/cart/address/**"
requires-channel="https" />

<intercept-url pattern="/**" requires-
channel=“https"/>
```

7. 提供v1与v2版本API访问的对应:
https://wiki.hybris.com/display/release5/Migrating+to+OCC+v2

权限配置

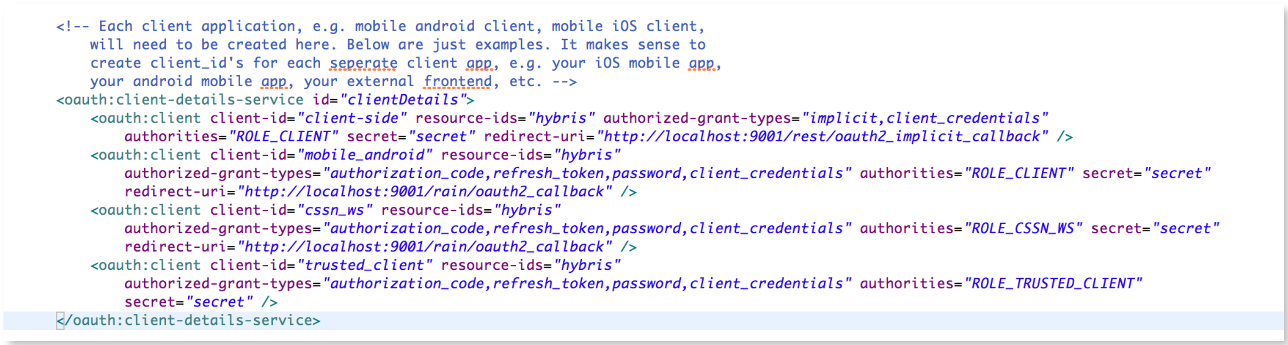
The OAuth 2.0 used in the hybris OCC web services.

参考网址：https://wiki.hybris.com/display/release5/OAuth+2.0
https://wiki.hybris.com/display/release5/OCC+Calls+Security



具体的关于配置权限信息，都在在occ核心工程中web/webroot/WEB_INF/config/security-spring.xml文件中。

自定义oauth_Client配置：

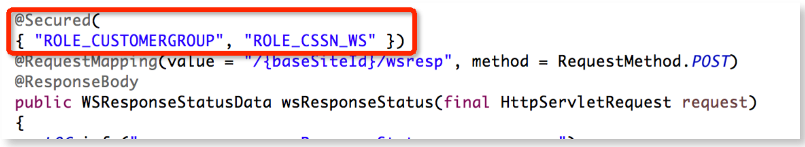


```
<!-- Each client application, e.g. mobile android client, mobile iOS client,
    will need to be created here. Below are just examples. It makes sense to
    create client_id's for each seperate client app, e.g. your iOS mobile app,
    your android mobile app, your external frontend, etc. -->
<oauth:client-details-service id="clientDetails">
    <oauth:client client-id="client-side" resource-ids="hybris" authorized-grant-types="implicit,client_credentials"
        authorities="ROLE_CLIENT" secret="secret" redirect-uri="http://localhost:9001/rest/oauth2_implicit_callback" />
    <oauth:client client-id="mobile_android" resource-ids="hybris"
        authorized-grant-types="authorization_code,refresh_token,password,client_credentials" authorities="ROLE_CLIENT" secret="secret"
        redirect-uri="http://localhost:9001/rain/oauth2_callback" />
    <oauth:client client-id="cssn_ws" resource-ids="hybris"
        authorized-grant-types="authorization_code,refresh_token,password,client_credentials" authorities="ROLE_CSSN_WS" secret="secret"
        redirect-uri="http://localhost:9001/rain/oauth2_callback" />
    <oauth:client client-id="trusted_client" resource-ids="hybris"
        authorized-grant-types="authorization_code,refresh_token,password,client_credentials" authorities="ROLE_TRUSTED_CLIENT"
        secret="secret" />
</oauth:client-details-service>
```

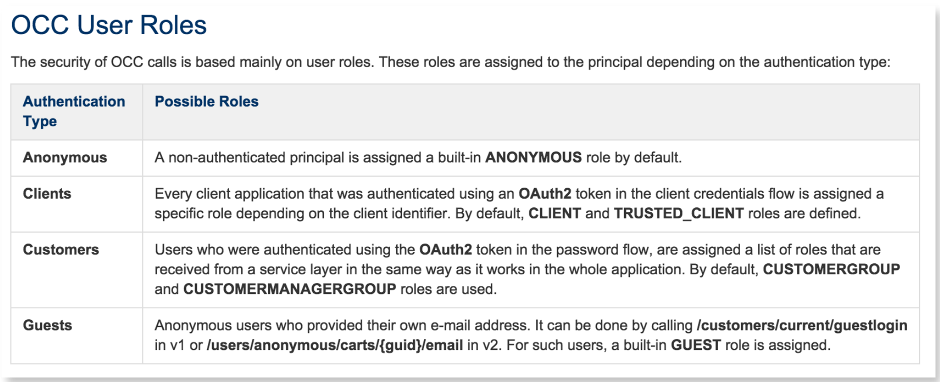
在此配置信息中，添加一条记录：

```
<oauth:client client-id="cssn_ws" resource-ids="hybris"
      authorized-grant-types="authorization_code,refresh_token,password,client_credentials"
authorities="ROLE_CSSN_WS" secret="secret" redirect-uri="http://localhost:9001/rain/
oauth2_callback" />
```

在此记录中，client_id是需要提供给客户端使用(调用OCC的WebService时，要根据client_id生成对应的access_token)。authorities主要用于标注此客户端可以访问到资源。



```
@Secured(
{ "ROLE_CUSTOMERGROUP", "ROLE_CSSN_WS" })
@RequestMapping(value = "/{baseSiteId}/wsresp", method = RequestMethod.POST)
@ResponseBody
public WSResponseStatusData wsResponseStatus(final HttpServletRequest request)
{
```

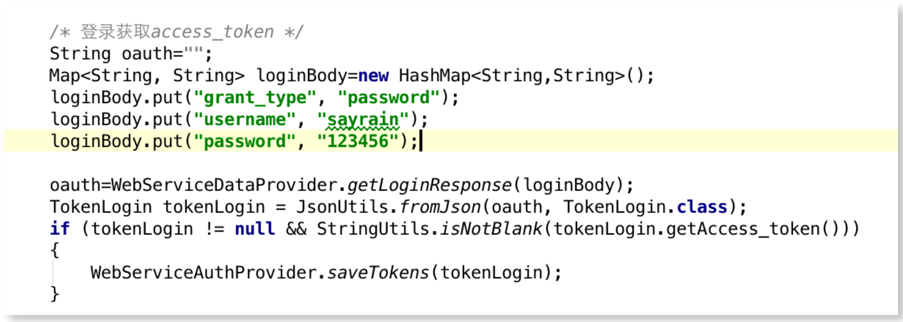


OCC User Roles

The security of OCC calls is based mainly on user roles. These roles are assigned to the principal depending on the authentication type:

Authentication Type	Possible Roles
Anonymous	A non-authenticated principal is assigned a built-in **ANONYMOUS** role by default.
Clients	Every client application that was authenticated using an **OAuth2** token in the client credentials flow is assigned a specific role depending on the client identifier. By default, **CLIENT** and **TRUSTED_CLIENT** roles are defined.
Customers	Users who were authenticated using the **OAuth2** token in the password flow, are assigned a list of roles that are received from a service layer in the same way as it works in the whole application. By default, **CUSTOMERGROUP** and **CUSTOMERMANAGERGROUP** roles are used.
Guests	Anonymous users who provided their own e-mail address. It can be done by calling **/customers/current/guestlogin** in v1 or **/users/anonymous/carts/{guid}/email** in v2. For such users, a built-in **GUEST** role is assigned.

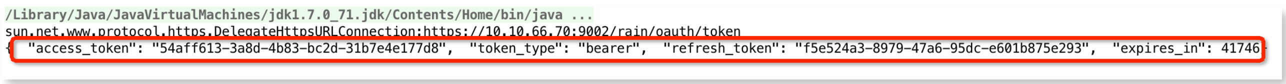
权限测试:
(Hybris OCC访问的测试工程　参考Android APP访问数据的核心逻辑编写 语言:Java)



```java
/* 登录获取access_token */
String oauth="";
Map<String, String> loginBody=new HashMap<String,String>();
loginBody.put("grant_type", "password");
loginBody.put("username", "sayrain");
loginBody.put("password", "123456");

oauth=WebServiceDataProvider.getLoginResponse(loginBody);
TokenLogin tokenLogin = JsonUtils.fromJson(oauth, TokenLogin.class);
if (tokenLogin != null && StringUtils.isNotBlank(tokenLogin.getAccess_token()))
{
    WebServiceAuthProvider.saveTokens(tokenLogin);
}
```

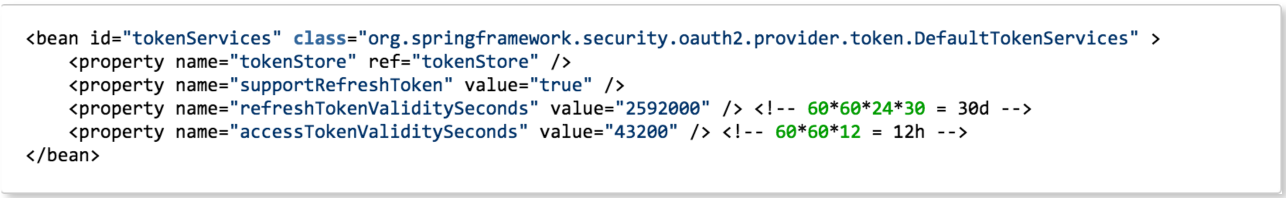
username和password都是hybris中客户的用户名和密码。(可维护)

grant_type授权类型，由上述的xml文件控制：authorized-grant-types

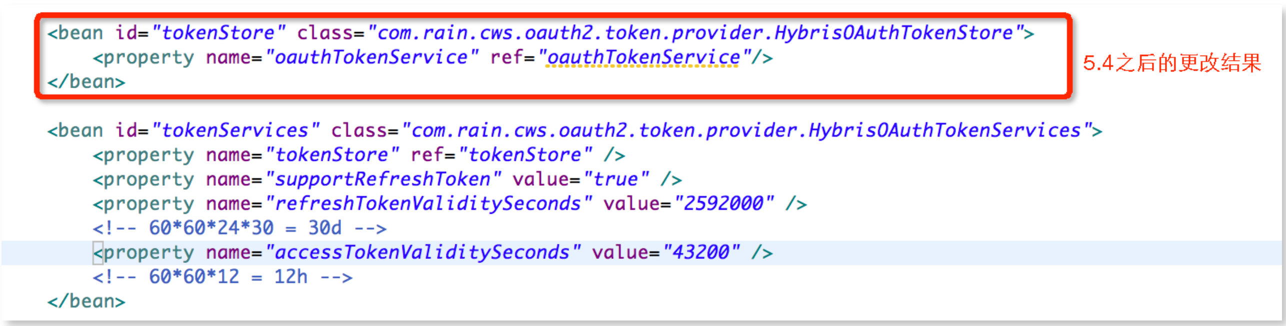


```
/Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/bin/java ...
sun.net.www.protocol.https.DelegateHttpsURLConnection:https://10.10.66.70:9002/rain/oauth/token
{  "access_token": "54aff613-3a8d-4b83-bc2d-31b7e4e177d8",  "token_type": "bearer",  "refresh_token": "f5e524a3-8979-47a6-95dc-e601b875e293",  "expires_in": 41746
```

访问一些有权限的站点，只有先进行登录操作，获取token之后，才能进行相关的访问。



```xml
<bean id="tokenServices" class="org.springframework.security.oauth2.provider.token.DefaultTokenServices" >
    <property name="tokenStore" ref="tokenStore" />
    <property name="supportRefreshToken" value="true" />
    <property name="refreshTokenValiditySeconds" value="2592000" /> <!-- 60*60*24*30 = 30d -->
    <property name="accessTokenValiditySeconds" value="43200" /> <!-- 60*60*12 = 12h -->
</bean>
```



```xml
<bean id="tokenStore" class="com.rain.cws.oauth2.token.provider.HybrisOAuthTokenStore">
    <property name="oauthTokenService" ref="oauthTokenService"/>
</bean>

<bean id="tokenServices" class="com.rain.cws.oauth2.token.provider.HybrisOAuthTokenServices">
    <property name="tokenStore" ref="tokenStore" />
    <property name="supportRefreshToken" value="true" />
    <property name="refreshTokenValiditySeconds" value="2592000" />
    <!-- 60*60*24*30 = 30d -->
    <property name="accessTokenValiditySeconds" value="43200" />
    <!-- 60*60*12 = 12h -->
</bean>
```

5.4之后的更改结果

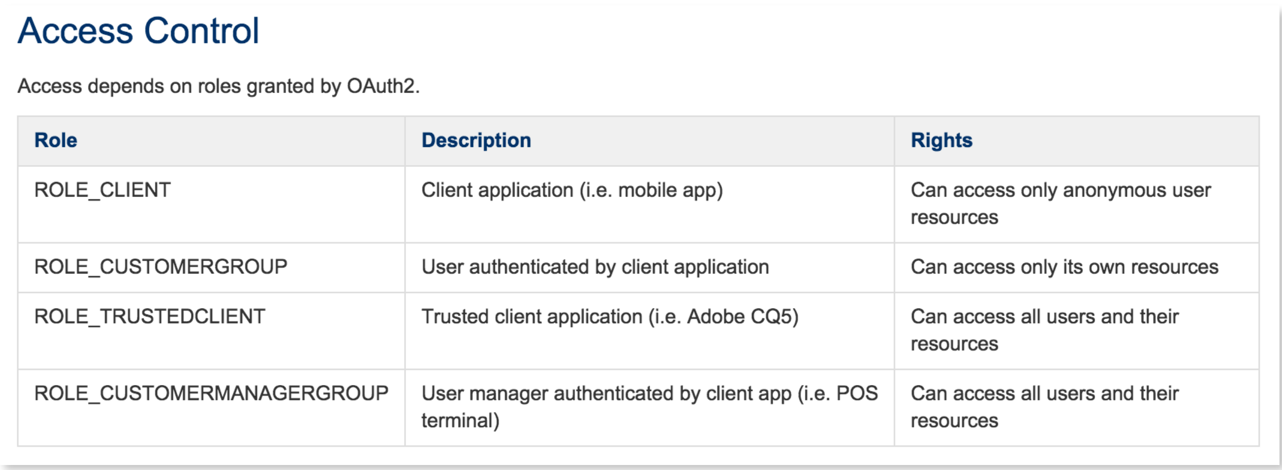
以上是token的生效时间的定义部分。

客户端访问逻辑:
1. 登录(获取相关的token)
2. 访问有权限资源，首先应确定当前的token是否在有效期内。
3. 利用refresh_token去刷新access_token。
4. 把access_token放在请求头(head)中，发送请求。

访问数据配置

OCC v2 became more RESTful and fully stateless from the end user perspective. This document describes some of the new ideas and approaches regarding URLs and access control in v2.

参考网址：https://wiki.hybris.com/display/release5/RESTful+Implementation+in+v2



Access Control

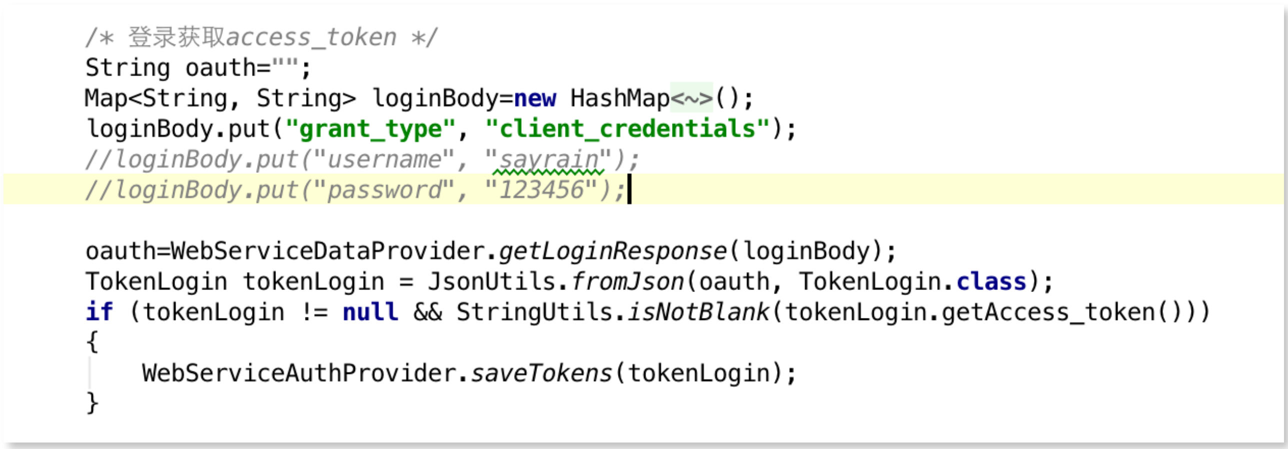
Access depends on roles granted by OAuth2.

Role	Description	Rights
ROLE_CLIENT	Client application (i.e. mobile app)	Can access only anonymous user resources
ROLE_CUSTOMERGROUP	User authenticated by client application	Can access only its own resources
ROLE_TRUSTEDCLIENT	Trusted client application (i.e. Adobe CQ5)	Can access all users and their resources
ROLE_CUSTOMERMANAGERGROUP	User manager authenticated by client app (i.e. POS terminal)	Can access all users and their resources

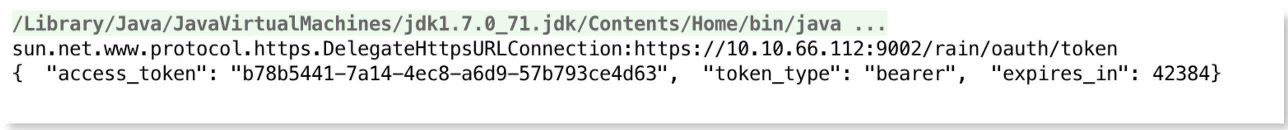
以上是对访问v2时的一些说明，介绍post、put、get等方法以及一些角色权限说明。

在上文的 ROLE_CSSN_WS 可以和 ROLE_CUSTOMERGROUP 结合使用，代表此客户端发送过来的请求信息，必须是要有相应用户登录才能访问此权限下的资源！（如果不用用户名和密码，可以根据 authorized-grant-types 里面的授权类型，做相应的调整去访问一些只有此客户端权限下的资源！具体配置信息见上文 security-spring.xml）

例：



```
/* 登录获取access_token */
String oauth="";
Map<String, String> loginBody=new HashMap<~>();
loginBody.put("grant_type", "client_credentials");
//loginBody.put("username", "sayrain");
//loginBody.put("password", "123456");

oauth=WebServiceDataProvider.getLoginResponse(loginBody);
TokenLogin tokenLogin = JsonUtils.fromJson(oauth, TokenLogin.class);
if (tokenLogin != null && StringUtils.isNotBlank(tokenLogin.getAccess_token()))
{
    WebServiceAuthProvider.saveTokens(tokenLogin);
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/bin/java ...
sun.net.www.protocol.https.DelegateHttpsURLConnection:https://10.10.66.112:9002/rain/oauth/token
{  "access_token": "b78b5441-7a14-4ec8-a6d9-57b793ce4d63",  "token_type": "bearer",  "expires_in": 42384}
```

在上文所说的Hybris OCC访问的测试工程中：
WebServiceDataProvider中有此方法访问只基于 grant_type=client_credentials (**ROLE_CSSN_WS**) 的资源。

```
public static String getClientCredentialsResponse(String url,
                                                  String clientCredentialsToken,
                                                  String httpMethod,
                                                  Map<String,String> httpBody)
throws MalformedURLException, IOException, ProtocolException, JSONException;
```

1. 定义交互数据Data
在Hybris与外围系统做交互的时候，一般都是基于Data数据。如：CustomerModel对应的CustomerData，还有CountryData，List类型的CountryDataList等。此数据，可以在对应的occ核心工程中resources/raincommercews-beans.xml中定义。
测试数据：(定义访问数据的返回状态)

```
<bean class="com.rain.cws.data.WSResponseStatusData">
      <property name="code" type="String"/>
      <property name="status" type="String"/>
</bean>
```

执行ant all后，此配置会生成对应的Spring Bean，还需要一些配置来规范返回的数据格式(json/xml)**[如果此数据配置有修改，需要先ant clean再ant all]**
此配置文件在webroot/WEB-INF/config/common/xstream-converters-spring.xml中。

参考网址：https://wiki.hybris.com/display/release5/Customizing+the+Commerce+Web+Services#CustomizingtheCommerceWebServices-XML/JSONSerializationofReturnedObjects

对应，在此文件中加上如下配置信息：

```
<bean
class="de.hybris.platform.commercefacades.xstream.alias.TypeAliasMapping">
      <property name="alias" value="wsResponseStatus" />
      <property name="aliasedClass"
value="com.rain.cws.data.WSResponseStatusData" />
</bean>
```

加上此配置信息，就可以根据请求的条件，返回相应类型的数据(json/xml)。

注意：如若有List类型的数据，除了上述的配置，还需要加上下列配置：

```
<bean
class="de.hybris.platform.commercefacades.xstream.alias.ImplicitCollection">
            <property name="ownerType"
value="com.rain.cws.product.data.PromotionDataList" />
            <property name="fieldName" value="promotions" />
</bean>
```

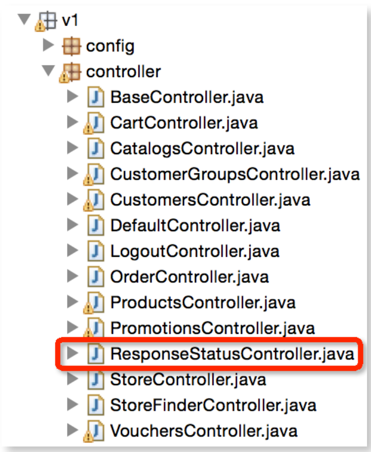
这样可以保证返回的List信息(json/xml)，显示正常。

以上，是我们需要展现给外围系统的数据，需要在返回数据格式做配置，如果只是单纯的外围系统给我们发送数据，是可以不做返回数据格式的配置!

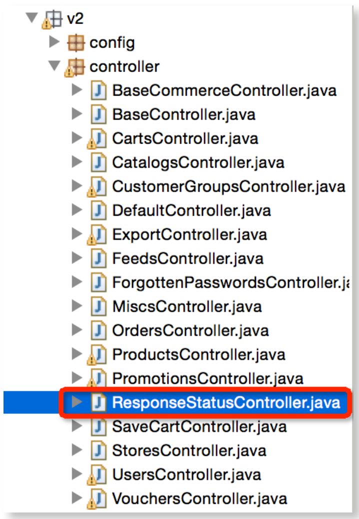
再加上一些，交互数据(resources/raincommercews-beans.xml):

```
<bean class="com.rain.cws.data.CSTestData">
      <property name="code" type="String"/>
      <property name="value" type="String"/>
</bean>

<bean class="com.rain.cws.data.CSTestDataList">
      <property name="CSTestDatas"
type="java.util.List&lt;com.rain.cws.data.CSTestData>"/>
</bean>
```

2. 分别在v1和v2定义访问入口。(ResponseStatusController)



```
▼ v1
  ▶ config
  ▼ controller
    ▶ BaseController.java
    ▶ CartController.java
    ▶ CatalogsController.java
    ▶ CustomerGroupsController.java
    ▶ CustomersController.java
    ▶ DefaultController.java
    ▶ LogoutController.java
    ▶ OrderController.java
    ▶ ProductsController.java
    ▶ PromotionsController.java
    ▶ ResponseStatusController.java
    ▶ StoreController.java
    ▶ StoreFinderController.java
    ▶ VouchersController.java
```



```
▼ v2
  ▶ config
  ▼ controller
    ▶ BaseCommerceController.java
    ▶ BaseController.java
    ▶ CartsController.java
    ▶ CatalogsController.java
    ▶ CustomerGroupsController.java
    ▶ DefaultController.java
    ▶ ExportController.java
    ▶ FeedsController.java
    ▶ ForgottenPasswordsController.ja
    ▶ MiscsController.java
    ▶ OrdersController.java
    ▶ ProductsController.java
    ▶ PromotionsController.java
    ▶ ResponseStatusController.java
    ▶ SaveCartController.java
    ▶ StoresController.java
    ▶ UsersController.java
    ▶ VouchersController.java
```

3. 测试基于 grant_type=client_credentials的访问，v1。

在webroot/WEB-INF/config/v1/security-v1-spring.xml文件中，需要加上相关的https链接的配置。

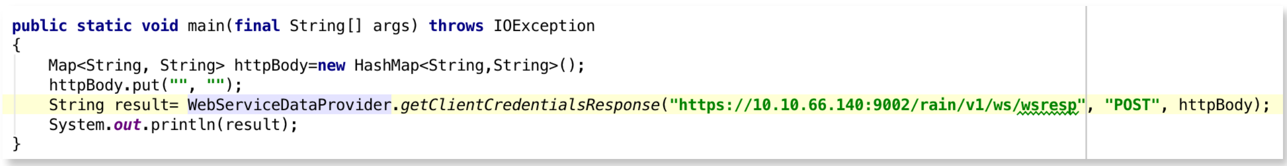
```
<intercept-url pattern="/**/wsresp/**" access="ROLE_CSSN_WS" requires-
channel="https" />
```

表示，在/wsresp/**访问路径中，只能是client_id是在 ROLE_CSSN_WS 中，对此进行安全访问。

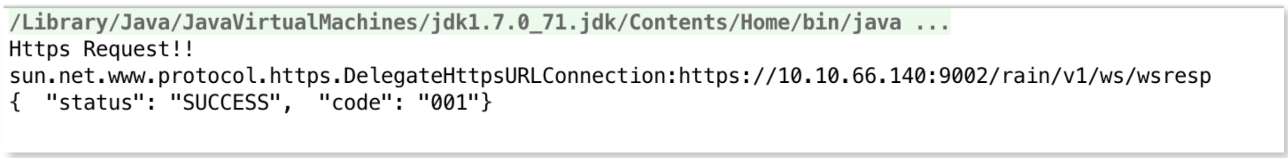
Hybris v1 定义访问资源的java代码：

```
@Secured("ROLE_CSSN_WS")
@RequestMapping(value = "/{baseSiteId}/wsresp", method = RequestMethod.POST)
@ResponseBody
public WSResponseStatusData wsResponseStatus()
{
    LOG.info("---------------wsResponseStatus-------------");
    final WSResponseStatusData wsResponseStatus = new WSResponseStatusData();
    wsResponseStatus.setCode("001");
    wsResponseStatus.setStatus("SUCCESS");
    return wsResponseStatus;
}
```

先根据上文所描述，获取access_token，然后进行资源访问。



```
public static void main(final String[] args) throws IOException
{
    Map<String, String> httpBody=new HashMap<String,String>();
    httpBody.put("", "");
    String result= WebServiceDataProvider.getClientCredentialsResponse("https://10.10.66.140:9002/rain/v1/ws/wsresp", "POST", httpBody);
    System.out.println(result);
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/bin/java ...
Https Request!!
sun.net.www.protocol.https.DelegateHttpsURLConnection:https://10.10.66.140:9002/rain/v1/ws/wsresp
{  "status": "SUCCESS",  "code": "001"}
```

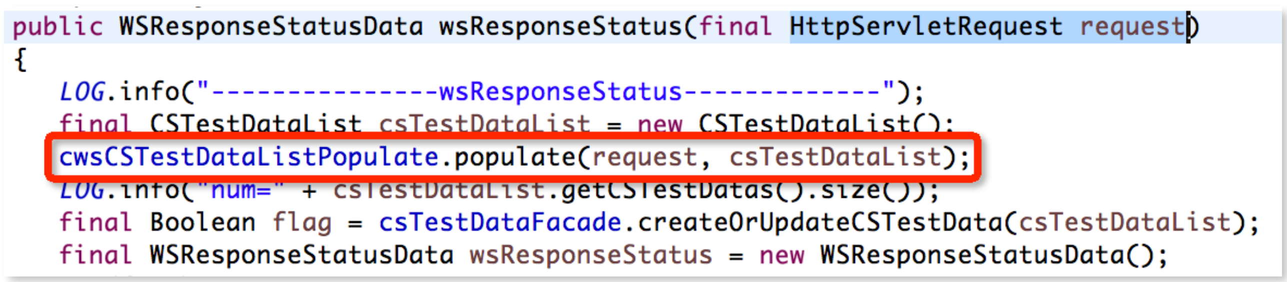
4. 测基于 grant_type=password的访问，进行的数据交互，v2。

Hybris v1 定义访问资源的java代码中的权限和路径设置：

```
@Secured(
{ "ROLE_CUSTOMERGROUP", "ROLE_CSSN_WS" })
@RequestMapping(value = "/{baseSiteId}/wsresp", method = RequestMethod.POST)
@ResponseBody
public WSResponseStatusData wsResponseStatus(final HttpServletRequest request)
```

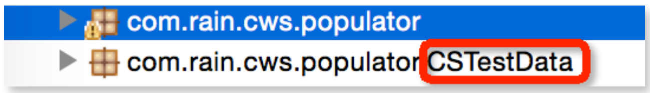
所有，传过来的参数，都在HttpServletRequest request中，所以，在接受这部分数据时，需要对这部分数据进行解析和转化，保存到交互的数据Data中。

那么对应此测试部分的数据转换，采用时Gson的方式，根据对应的json数据串和相应的实体类，解析数据到Data中。



```
public WSResponseStatusData wsResponseStatus(final HttpServletRequest request)
{
    LOG.info("---------------wsResponseStatus-------------");
    final CSTestDataList csTestDataList = new CSTestDataList();
    cwsCSTestDataListPopulate.populate(request, csTestDataList);
    LOG.info("num=" + csTestDataList.getCSTestDatas().size());
    final Boolean flag = csTestDataFacade.createOrUpdateCSTestData(csTestDataList);
    final WSResponseStatusData wsResponseStatus = new WSResponseStatusData();
```

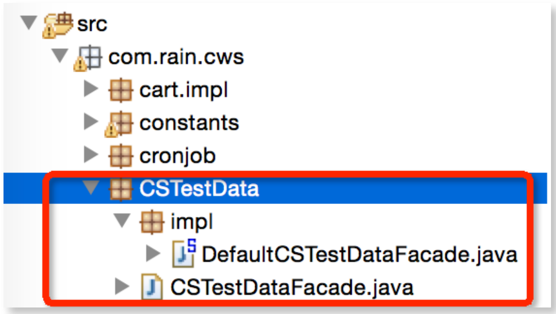
相关的数据解析，都会写在occ核心工程的web/src/com/rain/cws/populator包中。



```
▶ com.rain.cws.populator
▶ com.rain.cws.populatorCSTestData
```

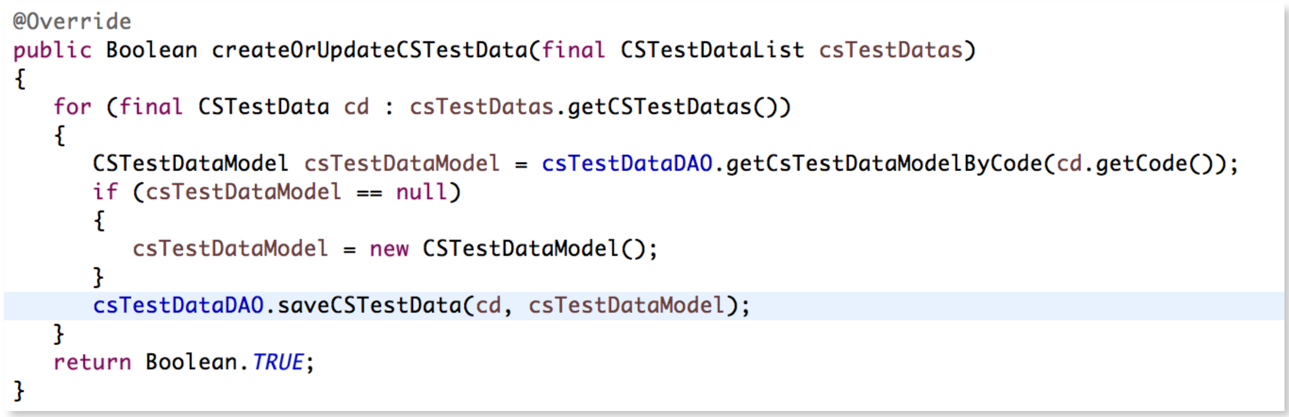
core工程已经定义好相关的数据Type，将数据解析完毕，在service工程中，写好Data保存到Model的方法。

在occ核心工程中(src/com/rain/cws/CSTestData包)写好Facades层(逻辑层)方法，调用service的保存数据方法。



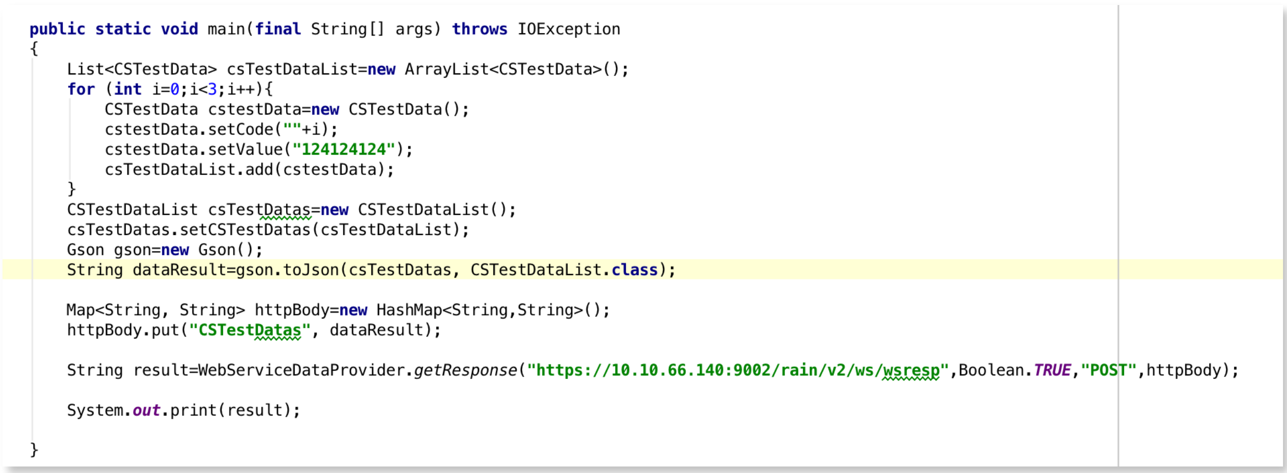
```
▼ src
  ▼ com.rain.cws
    ▶ cart.impl
    ▶ constants
    ▶ cronjob
    ▼ CSTestData
      ▼ impl
        ▶ DefaultCSTestDataFacade.java
      ▶ CSTestDataFacade.java
```

通过occ的post方法将数据保存到hybris的全过程，就是这么一个逻辑。

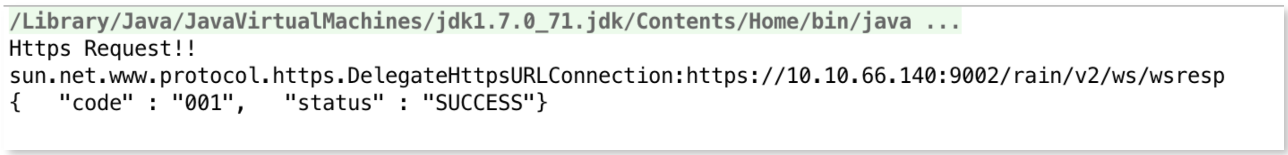


```
@Override
public Boolean createOrUpdateCSTestData(final CSTestDataList csTestDatas)
{
    for (final CSTestData cd : csTestDatas.getCSTestDatas())
    {
        CSTestDataModel csTestDataModel = csTestDataDAO.getCsTestDataModelByCode(cd.getCode());
        if (csTestDataModel == null)
        {
            csTestDataModel = new CSTestDataModel();
        }
        csTestDataDAO.saveCSTestData(cd, csTestDataModel);
    }
    return Boolean.TRUE;
}
```

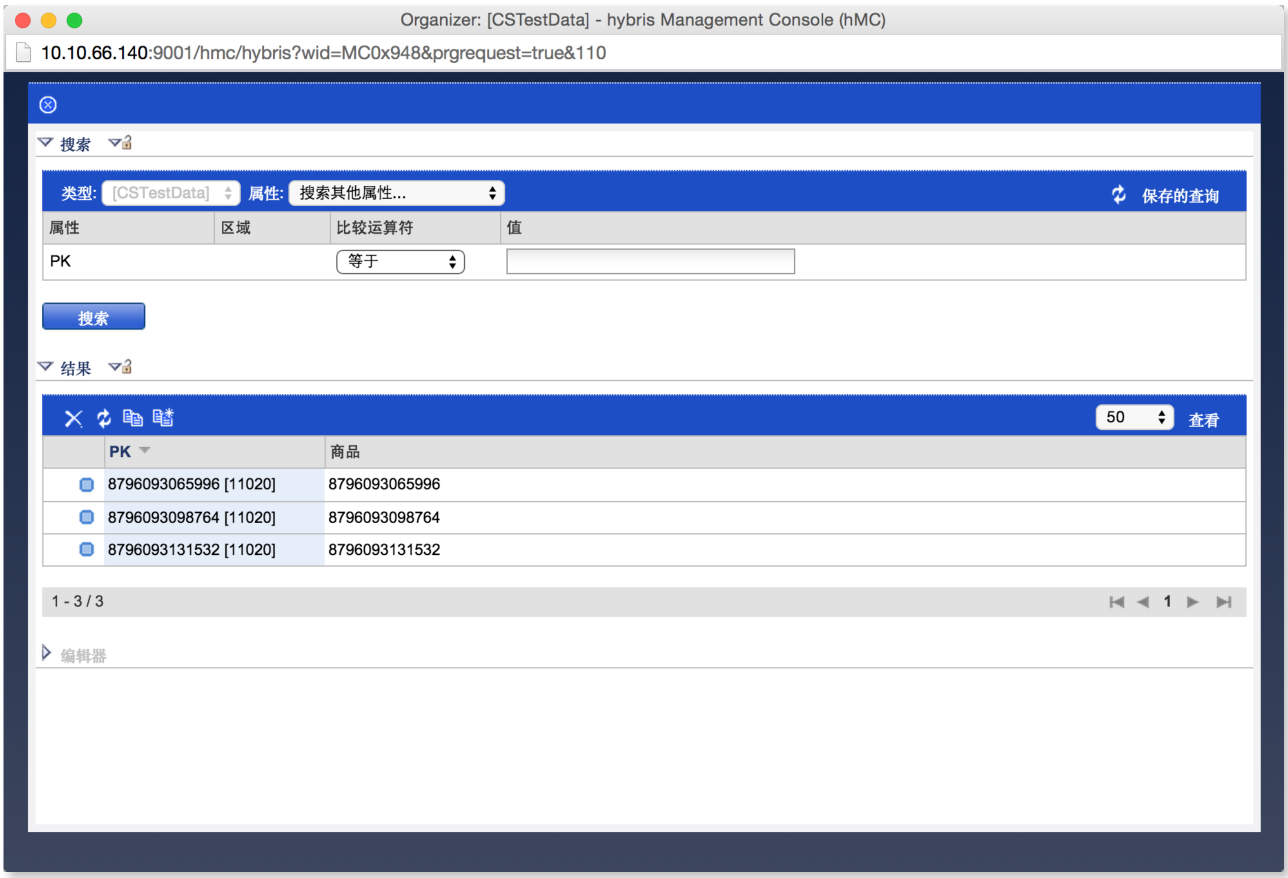
测试，先需要通过grant_type=password的方式，获取相关的token。(上文权限访问有说明)
数据Post请求：



```
public static void main(final String[] args) throws IOException
{
    List<CSTestData> csTestDataList=new ArrayList<CSTestData>();
    for (int i=0;i<3;i++){
        CSTestData cstestData=new CSTestData();
        cstestData.setCode(""+i);
        cstestData.setValue("124124124");
        csTestDataList.add(cstestData);
    }
    CSTestDataList csTestDatas=new CSTestDataList();
    csTestDatas.setCSTestDatas(csTestDataList);
    Gson gson=new Gson();
    String dataResult=gson.toJson(csTestDatas, CSTestDataList.class);

    Map<String, String> httpBody=new HashMap<String,String>();
    httpBody.put("CSTestDatas", dataResult);

    String result=WebServiceDataProvider.getResponse("https://10.10.66.140:9002/rain/v2/ws/wsresp",Boolean.TRUE,"POST",httpBody);

    System.out.print(result);

}
```



```
/Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/bin/java ...
Https Request!!
sun.net.www.protocol.https.DelegateHttpsURLConnection:https://10.10.66.140:9002/rain/v2/ws/wsresp
{   "code" : "001",   "status" : "SUCCESS"}
```



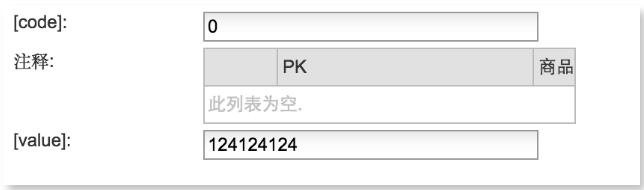
Organizer: [CSTestData] - hybris Management Console (hMC)

10.10.66.140:9001/hmc/hybris?wid=MC0x948&prgrequest=true&110

属性	区域	比较运算符	值
PK		等于	

	PK	商品
●	8796093065996 [11020]	8796093065996
●	8796093098764 [11020]	8796093098764
●	8796093131532 [11020]	8796093131532

1 - 3 / 3



[code]:	0
注释:	PK 商品 — 此列表为空.
[value]:	124124124