Deamon 权限系统使用手册

提示：可以直接跳到第三，查看设置 sdk

一、 系统说明

帝门权限系统，简称帝门系统，全称帝门(deamon)分布式权限系统。
目前有 21 个运营系统接入，1267 个用户，111 个角色，1975 个资源。

通过帝门系统的配置中心，可以将运营系统的所有资源录入进来，分配多个角色，每个角色关联自己的资源。运营系统管理员，在帝门配置中心，快速给用户分配角色。
运营系统，引用帝门系统的 sdk，得到 4W 解决方案，即 WHO 看到 WHICH、对 WHAT 可以 HOW。
帝门系统，提供权限配置中心系统和 SDK 程序包，下面从使用配置中心和引入 sdk 两方面讲解。

二、 配置中心系统

1. 原理

提供录入资源，通过角色将不同资源关联在一起，然后对用户分配一个或多个角色。
通过配置中心(测试域名)快速浏览下系统 http://auth.jr.jd.com/
Host:192.168.146.62 auth.jr.jd.com 测试账号:bjyangkuan xinxibu456
线上 http://auth.lc.jd.com/

2. 名词讲解

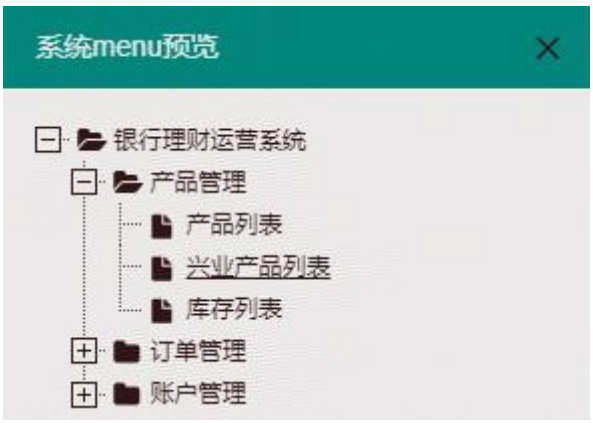
中心系统有资源管理、角色管理和用户管理。
角色管理和用户管理相对容易操作，资源管理较难，这里主要讲资源管理操作。

资源的定义。
从 web 页面来说，所见到的每一个页签、每一个按钮，及用户动作产生的动态请求都是资源(一般静态资源不关注)。
资源的类型，本系统定义为系统节点、枝节点、叶子节点、按钮节点，这样可以关联出一个树形结构。即一个运营系统，在帝门配置中心里算一个系统节点；系统节点下有很多枝节点，枝节点下有多个下级枝节点和叶子节点，这样可以推导出 menu 菜单，点击叶子节点可以显示出一个页面。页面有很多按钮，如果按钮会请求动态资源，即按钮节点。
叶子节点和按钮节点的 URL 属性必须有值，通常使用请求路径(例如/user/add)。
展示如下图:

Menu 含义：菜单、页签。即展示形式，即上图。

3. 其他作用

1. 资源排序。选择一个上级编码，可以过滤出子节点，对子节点进行排序，即在对应的运营平台，menu 的标签可以按顺序展示。
2. 资源批量导入。需要结合 sdk 的扫描程序，将资源扫描组装，然后批量导入。
3. menu 预览作用。在资源管理、角色管理、用户管理，都可以预览所属菜单。可以让操作员快速操作。如下图



4. 提供 jsf 接口：创建用户、给用户增加或者更新角色。
5. 其他待提供：审批流程、用户批量导入功能、系统资源用户等下载功能等。

4. 运营

目前提供了配置系统的管理模块，可以放开给运营人员自己配置角色和添加用户。
后期会更具需求添加审批流，对新加的用户的角色进行流程审批。

三、 SDK 使用说明

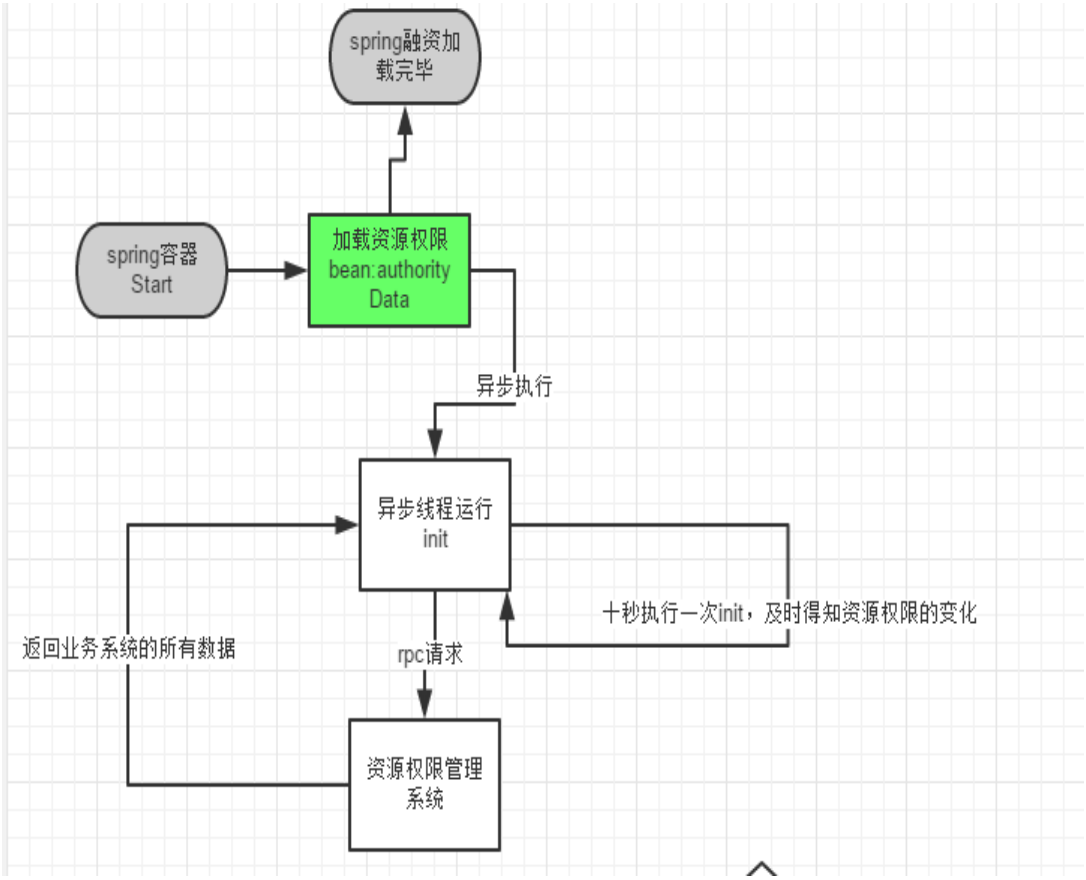
a) 配置清单

仅仅提供 java 的 sdk，客户端配置清单。

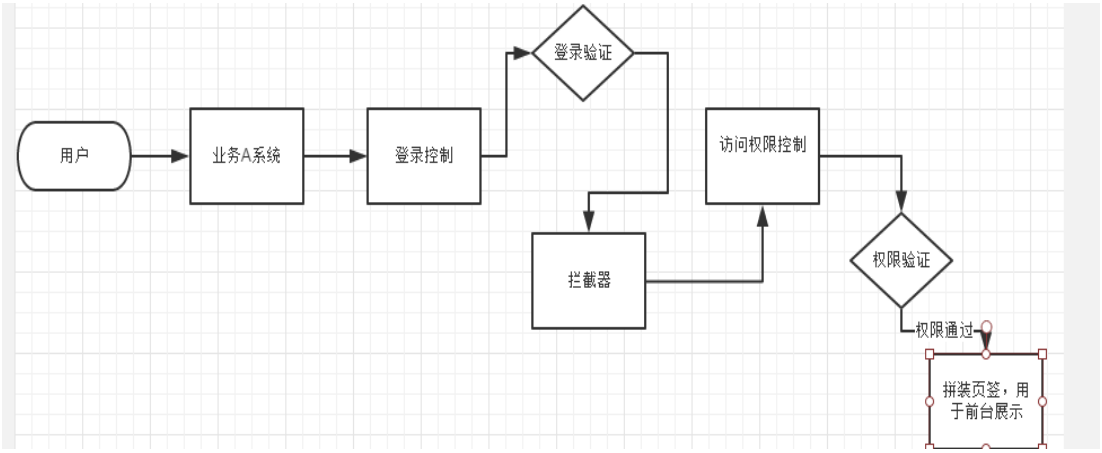
1. 一个依赖 jar 包。通过 spring 配置和注入，通过 jar 包里的程序，会和配置中心通信，将指定系统编码的所有资源、角色、用户，加载在到本地缓存。有多种数据模型(见文末)。数据模型，用于后续拼装 menu，和资源的请求验证。
2. 拦截器。用于拦截资源请求，进行验证。
3. 用于请求封装 menu 的 js，同时可以控制页面按钮的展示。后文详解使用。
4. 用于自定义方法的 service。拦截器里调用 service 的方法 loadResourceUrl 进行验证。Js 请求的 menu，在 service 的 setUpMenuHtml 方法里进行拼装。
   以上文件在提供的 demo 程序里已经使用起来。Demo 程序里还有其他文件。
5. 扫描 springmvc restful 资源的工具类：ScanRestUrlUtil.java。
   类似@RequestMapping(value = "/user/add",name="增加用户")，可以得出资源编码(user-add)、资源名字、资源的 url(/user/add)。集中起来，可以批量导入资源。
   一般一个 controller 类对应目录资源，controller 类里的方法对应叶子节点和按钮节点资源。

b) SDK 和配置中心交互原理如下图：

引入 sdk，配置 springbean，会随 web 容器启动一个线程，暴力 pull 服务器的数据，组装放在 local 内存中。



拦截器和封装原理如下图：



c) 详细使用

1. Jar 配置。引入

```
<groupId>com.jd.jr.authority</groupId>
<artifactId>console-authority-export</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

2. 和配置中心通信的配置清单。详见 demo 里的 spring-deamonauth.xml 文件

```
<bean id="httpClient"
   class="com.jd.jr.authority.export.proxy.PoolingHttpClient">
        <property name="connectTimeout" value="10000"></property>
        <property name="readTimeout" value="50000"></property>
        <property name="waitTimeout" value="1000"></property>
        <property name="totalMaxConnection" value="2000"></property>
</bean>
```

```
<bean id="authorityResource"
  class="com.jd.jr.authority.export.proxy.RestResourceClientProxyFactoryBean">
       <property name="serviceInterface"
         value="com.jd.jr.authority.export.rpc.AuthorityResourceService">
       </property>
       <!--192.168.146.62 auth.jr.jd.com线上使用http://auth.lc.jd.com/-->
       <property name="baseUri" value="http://auth.jr.jd.com/"></property>
       <property name="httpClient" ref="httpClient"></property>
</bean>
```

3. 注入数据模型的 bean-authorityData。详见 demo 里的 spring-deamonauth.xml 文件。模型清单见文尾。

```
<bean id="authorityData"
   class="com.jd.jr.authority.export.AuthorityResourcesLoadV2">
        <!--单位毫秒。超过1分钟，系统自动设置为1分钟-->
        <property name="refreshTime" value="6000"></property>
        <property name="systemCode" value="bigoneyy"/>
</bean>
```

4. 配置拦截器。详见 demo 里的 spring-deamonauth.xml 文件。

```
<mvc:interceptors>
  <bean class="com.jd.jr.authority.demo.AuthorityInterceptor" >
     <!--指定排除url,一般为静态资源-->
      <property name="excludePath"
    value="/static,/index.html"/>
  </bean>
</mvc:interceptors>
```

5. Js 配置。

需要引入 DaemonAuthority.js 和 map.js(js 的 HaspMap 实现)。当然也可以实现自己的 js 和不同的加载方式。

✧ DaemonAuthority.js。

是一个闭包，在当前页面加载完毕，声明一个对象，然后调用其 auth 方法即可。但 auth 方法的参数是关键。现在详细介绍 auth 的参数，auth 的参数是一个数组对象，对象里有以下属性。

属性	作用	备注
menuLoadUrl	查询页签的 url	可以参考 demo 的 PageController 类 menu 方法

menuId	需要被页签 append 的区域 id	此 id 在 html 中。
uriLoadUrl	查询用户拥有资源的 url	可以参考 demo 的 PageController 类 uriResourceList 方法
loadAfterMenuMethod	加载 menuLoadUrl 完毕后，需要执行的一系列自定义方法。	可以为空。也可以自定义多个方法，比如页签加载完毕，需要对页签绑定点击展开事件等。

引用加载权限配置清单，见 demo 的 index.html 底部。

```
<script>
    jQuery(document).ready(function () {
        loadAuth();
    });
    function loadAuth() {//加载权限相关
        var authority = new DaemonAuthority.getInstance();
        var authOptions = {//自定义属性
            'menuLoadUrl': "/page/menu",//查询页签
            'menuId': "menuDiv",//append页签的id
            'uriLoadUrl': "/page/uriResourceList",//查询用户拥有url的请求地址
            'loadAfterMenuMethod':{//页签加载完毕，执行的方法集合
                '0':menu_resourceClick,
                '1':menu_resourceClick
            }
        };
        authority.auth(authOptions);
    }
    function menu_resourceClick(){
        //绑定menu页签监听事件
    }
</script>
```

- ✧ 现在详细说说按钮粒度的展示。
  如果按钮被点击，需要发出 url 请求，那么可以在按钮里添加一个自定义属性:authurl。比如 authurl=”/user/add”。可以在页面装载完毕，显示调用 DaemonAuthority.js 的 checkElementAuth 方法，进行页面按钮是否可以展示的检查。已经兼容 input、a、radio 等主要控件展示。
  目前页面主流加载方式：先加载整体网站页面，如果请求加载不同的页面，是通过 ajax 异步加载，替换主要区域，那么 dom 变动可以被监听到。所以 DaemonAuthority.js 的 ObserverDomChange 方法监听 dom 变动，执行 checkElementAuth 方法。在异步加载页面的情况下，可以不用显示调用 checkElementAuth 方法。
  另外讲讲按钮的显示原理：当 DaemonAuthority 的 auth 执行完毕，已经通过 uriLoadUrl 属性设置，将用户可以访问的 url 加载到 map 里。执行 checkElementAuth，实际上是将解析到的 authurl 去 map 里取值，如果为空，则不展示。

6. 自定义方法的 service 类—AuthorityCheckService.java。

菜单的制作流程，应该是现有静态展示，当系统开发尾声，再通过加载权限模块，动态组成。自定义组装 menu，在 AuthorityCheckService 类的 setUpMenuHtml 方法。

如果 demo 工程可以运行起来，http://localhost/ (tomcat)，如果是其他容器，使用 http://localhost/index.html。

画面如下，这里只是最简洁的 demo。实际菜单的样式，需要自己设计，还需要操作菜单的折叠、展开、搜索等。稍美观的 demo 页面参考：http://localhost/indexV2.html

同时在 AuthorityInterceptor 拦截器里，调用 AuthorityCheckService 的方法 loadResourceUrl 进行资源验证。

d) 数据模型

可以根据自己需要使用

数据变量名字	类型	解释
rolesResources_resourceCodeMultimap	Multimap<String, RolesResources>	角色资源表,根据资源码维度存放。一对多
rolesResources_roleCodeMultimap	Multimap<String, RolesResources>	角色资源表,根据角色码维度存放。一对多
resources_resourceCodeMap	Map<String, Resources>	资源表，根据资源编码存放。一对一
resources_parentCodeMultimap	Multimap<String, RolesResources>	部分角色资源。根据资源父编码维度存放

roles_roleCodeMap	Map<String, Roles>	角色表。根据角色表编码维度存放。一对一
roles_systemCodeMultimap	Multimap<String, UsersRoles>	存放指定系统编码下的用户角色
usersRoles_roleCodeMultimap	Multimap<String, UsersRoles>	用户角色表。根据角色编码维度存放。一对多
usersRoles_userPinMultimap	Multimap<String, UsersRoles>	用户角色表。根据用户 id 维度存放。一对多
users_userPinMap	Map<String, Users>	全量用户表。
resources_userIdAndUrlHBase	HashBasedTable<String, String, Resources>	资源的经纬度表。以用户 id 为经，以 url 为纬度。用来拦截判断登录用户是否有方位此 url 的权限
resources_userIdAndSourceCodeHBase1	HashBasedTable<String, String, Resources>	资源的经纬度表。以用户 id 为经，以资源编码为纬度。
resourcesLeaf_userIdMultimap	Multimap<String, Resources>	存放具有 url 属性的资源，按照用户 id 纬度存放。可以验证具体按钮是否可以展示，也可以用来自下而上推算页签展示。

e) **Demo 的 git 地址**

http://source.jd.com/app/deamonAuth.git