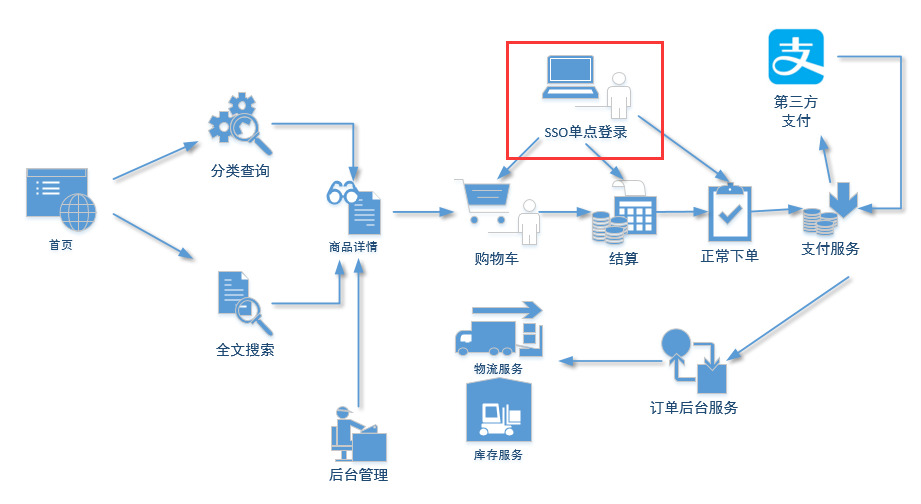
**谷粒商城**

版本：V 1.0

**单点登录**

所在位置：



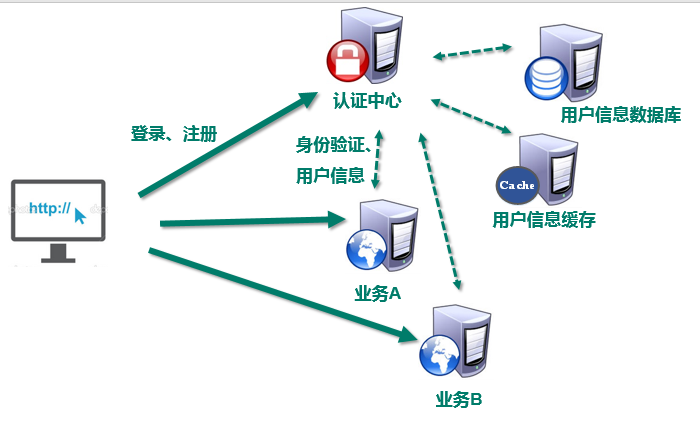
# 单点登录业务介绍

早期单一服务器，用户认证。



缺点：单点性能压力，无法扩展

分布式，SSO(single sign on)模式



解决 ：

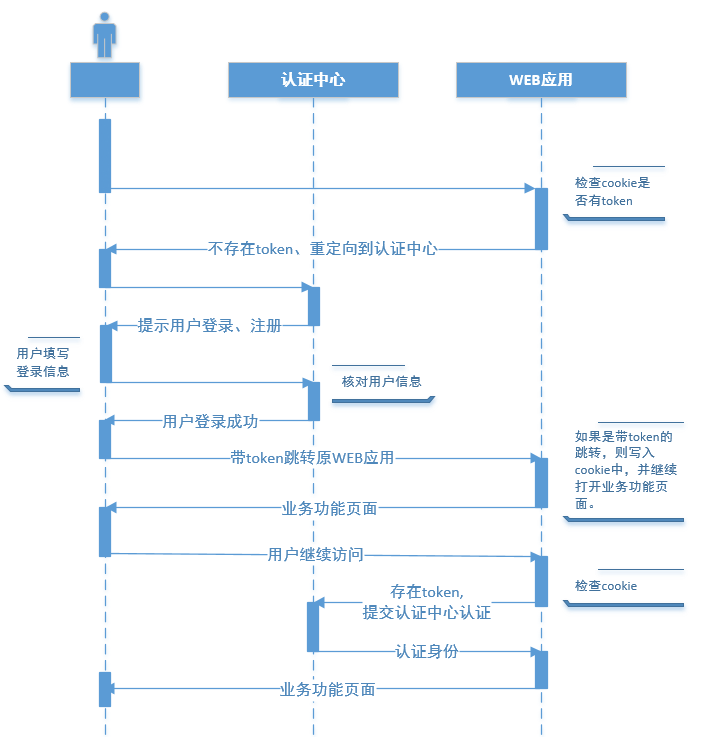
用户身份信息独立管理，更好的分布式管理。

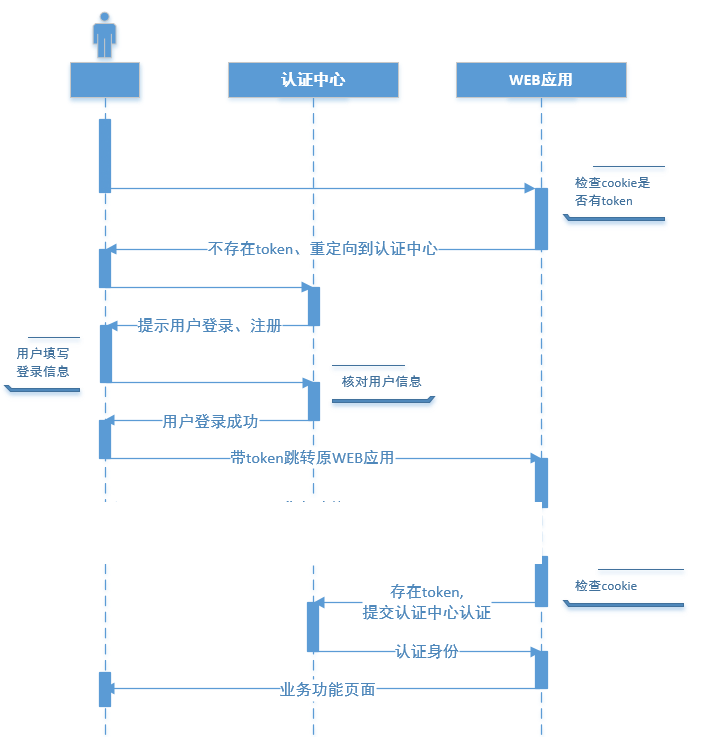
可以自己扩展安全策略

缺点：

认证服务器访问压力较大。

业务流程图{用户访问业务时，必须登录的流程}{单点登录的过程}

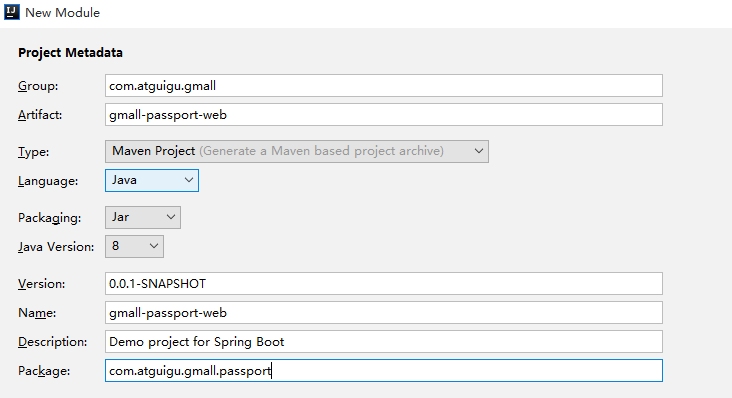


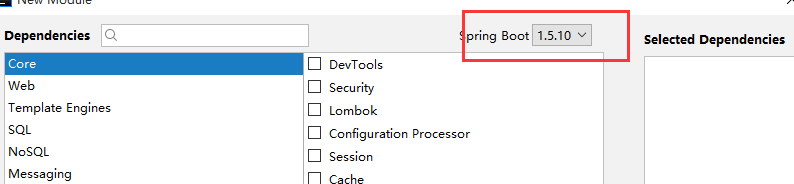
下面那張圖才正確 

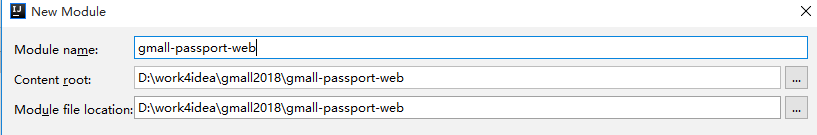
# 认证中心模块

数据库表：user\_info，并添加一条数据！**密码应该是加密的！**

## 1 搭建认证中心模块







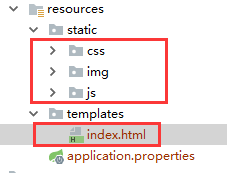
pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-passport-web</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>gmall-passport-web</name>  <description>Demo project for Spring Boot</description>   <parent>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-parent</artifactId>  <version>1.0-SNAPSHOT</version>  </parent>  <dependencies>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-interface</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-web-util</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>    </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>   </project> |

application.properties

|  |
| --- |
| server.port=8087  spring.thymeleaf.cache=false  spring.thymeleaf.mode=LEGACYHTML5  spring.dubbo.application.name=passport-web spring.dubbo.registry.protocol=zookeeper spring.dubbo.registry.address=192.168.67.202:2181 spring.dubbo.base-package=com.atguigu.gmall spring.dubbo.protocol.name=dubbo spring.dubbo.consumer.timeout=100000 spring.dubbo.consumer.check=false |

导入静态资源和登录页面 01资料中



建立controller控制器，测试页面

|  |
| --- |
| **package** com.atguigu.gmall.passport.controller;  **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.RequestMapping;  @RequestMapping(**"index"**) **public** String index(HttpServletRequest request){  String originUrl = request.getParameter(**"originUrl"**);  *// 保存上* request.setAttribute(**"originUrl"**,originUrl);  **return "index"**; } |
|  |

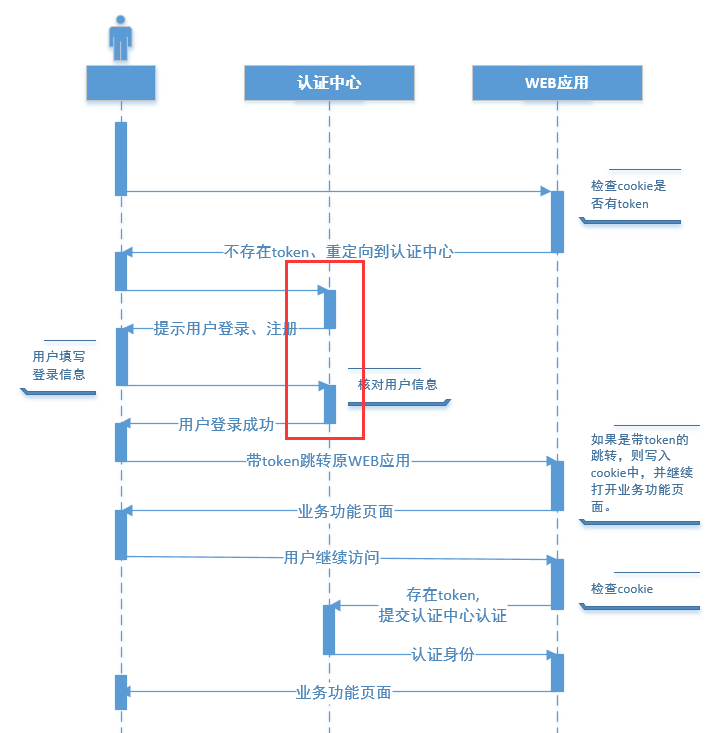
## 2 登录页面html

|  |
| --- |
| <div class="si\_bom1 tab" style="display: block;">  <div class="error">  请输入账户名和密码  </div>   <form id="loginForm" action="/login" method="post">  <ul>  <li class="top\_1">  <img src="/img/user\_03.png" class="err\_img1"/>   <input type="text" name="loginName" placeholder=" 邮箱/用户名/已验证手机" class="user"/>  </li>  <li>  <img src="/img/user\_06.png" class="err\_img2"/>  <input type="password" name="passwd" placeholder=" 密码" class="password"/>   </li>  <li class="bri">  <a href="">忘记密码</a>  </li>  <li class="ent"><button id="btn2" class="btn2"><a class="a">登 &nbsp; &nbsp;录</a></button></li>   </ul>  <input type="hidden" id="originUrl" name="originUrl" th:value="${originUrl}"/>  </form> </div> |

页面中的js

|  |
| --- |
| *btn2*.onclick=function(){  if(*user*.value == '' || *pass*.value ==''){  *err*.style.display='block';  *user*.style.border='1px solid red';  *pass*.style.border='1px solid red';    *err\_img1*.src='img/img11.png';  *err\_img2*.src='img/img22.png';  }else{  $.post("/login",$("#loginForm").serialize(),function (data) {  if(data&&data!='fail'){  var originUrl = $("#originUrl").val();  originUrl = decodeURIComponent(originUrl);  var idx=originUrl.indexOf('?');   if(idx<0){  originUrl+='?'  }else{  originUrl+='&'  }  window.location.href=originUrl+"newToken="+data;  }else{  $(".error").text("用户名密码错误！");  $(".error").show();  }  } );    }*user*.onfocus=function(){  *err\_img1*.src='img/grow1.png';  *user*.style.border='1px solid #999'; }  } |

## 3 登录功能



### 3.1 思路：

1. 用接收的用户名密码核对后台数据库
2. 将用户信息加载到写入redis，redis中有该用户视为登录状态。
3. 用userId+当前用户登录ip地址+密钥生成token
4. 重定向用户到之前的来源地址，同时把token作为参数附上。

### 3.2 UserInfoServiceImpl核对后台登录信息+用户登录信息载入缓存

UserInfoServiceImpl

|  |
| --- |
| **public** String **userKey\_prefix**=**"user:"**; **public** String **userinfoKey\_suffix**=**":info"**; **public int userKey\_timeOut**=60\*60\*24;  @Override **public** UserInfo login(UserInfo userInfo) {  String password = DigestUtils.*md5DigestAsHex*(userInfo.getPasswd().getBytes());  userInfo.setPasswd(password);  UserInfo info = **userInfoMapper**.selectOne(userInfo);   **if** (info!=**null**){  *// 获得到redis ,将用户存储到redis中* Jedis jedis = **redisUtil**.getJedis();  jedis.setex(**userKey\_prefix**+info.getId()+**userinfoKey\_suffix**,**userKey\_timeOut**, JSON.*toJSONString*(info));  jedis.close();  **return** info;  }  **return null**; } |
| **在启动类的时候，不要忘记加，扫描RedisUtils 工具类** |
| **在application.properties 配置文件中要添加**  **spring.redis.host**=**192.168.67.201 spring.redis.port**=**6379 spring.redis.database**=**0** |

### 3.3 生成token

京东：使用

JWT工具

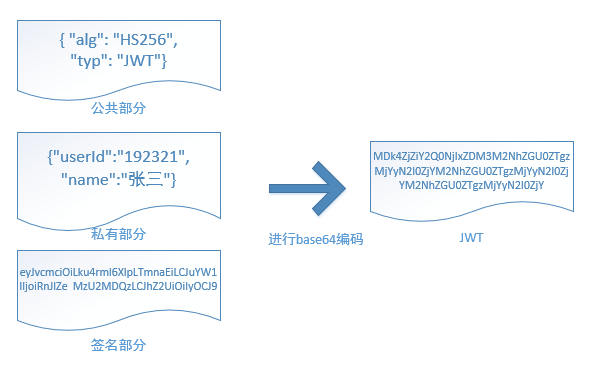
JWT（Json Web Token） 是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准。

JWT的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源。比如用在用户登录上

JWT 最重要的作用就是对 token信息的**防伪**作用。

JWT的原理，

一个JWT由三个部分组成：公共部分、私有部分、签名部分。最后由这三者组合进行base64编码得到JWT。



1. 公共部分

主要是该JWT的相关配置参数，比如签名的加密算法、格式类型、过期时间等等。

Key=ATGUIGU

1. 私有部分

用户自定义的内容，根据实际需要真正要封装的信息。

userInfo {用户的Id，用户的昵称nickName}

1. 签名部分

Salt iP: 当前服务器的Ip地址!{linux 中配置代理服务器的ip}

主要用户对JWT生成字符串的时候，进行加密{盐值}

最终组成 key+salt+userInfo 🡺 token!

1. base64编码，并不是加密，只是把明文信息变成了不可见的字符串。但是其实只要用一些工具就可以把base64编码解成明文，所以不要在JWT中放入涉及私密的信息，因为实际上JWT并不是加密信息。

pom依赖 放到gmall-web-util中

|  |
| --- |
| <**dependency**>  <**groupId**>io.jsonwebtoken</**groupId**>  <**artifactId**>jjwt</**artifactId**>  <**version**>0.9.0</**version**> </**dependency**> |

使用：放入到gmall-passport-web 项目中！

制作 JWT的工具类

|  |
| --- |
| **public class** JwtUtil {   **public static** String encode(String key,Map<String,Object> param,String salt){  **if**(salt!=**null**){  key+=salt;  }  JwtBuilder jwtBuilder = Jwts.*builder*().signWith(SignatureAlgorithm.***HS256***,key);   jwtBuilder = jwtBuilder.setClaims(param);   String token = jwtBuilder.compact();  **return** token;   }    **public static** Map<String,Object> decode(String token ,String key,String salt){  Claims claims=**null**;  **if** (salt!=**null**){  key+=salt;  }  **try** {  claims= Jwts.*parser*().setSigningKey(key).parseClaimsJws(token).getBody();  } **catch** ( JwtException e) {  **return null**;  }  **return** claims;  }  } |

在controller控制器中测试工具类

|  |
| --- |
| @Test **public void** test01(){  String key = **"atguigu"**;  String ip=**"192.168.67.201"**;  Map map = **new** HashMap();  map.put(**"userId"**,**"1001"**);  map.put(**"nickName"**,**"marry"**);  String token = JwtUtil.*encode*(key, map, ip);  Map<String, Object> decode = JwtUtil.*decode*(token, key, **"192.168.67.102"**); } |

### 3.4 Controller

在application.properties 添加一个signKey

|  |
| --- |
| **token.key**=**ATGUIGU\_GMALL\_KEY** |

PassportControler

|  |
| --- |
| @Value(**"${token.key}"**) String **signKey**;  @RequestMapping(**"login"**) @ResponseBody **public** String login(HttpServletRequest request, UserInfo userInfo){  *// 取得ip地址* String remoteAddr = request.getHeader(**"X-forwarded-for"**);  **if** (userInfo!=**null**) {  UserInfo loginUser = **userInfoService**.login(userInfo);  **if** (loginUser == **null**) {  **return "fail"**;  } **else** {  *// 生成token* Map map = **new** HashMap();  map.put(**"userId"**, loginUser.getId());  map.put(**"nickName"**, loginUser.getNickName());  String token = JwtUtil.*encode*(**signKey**, map, remoteAddr);  **return** token;  }  }  **return "fail"**; } |

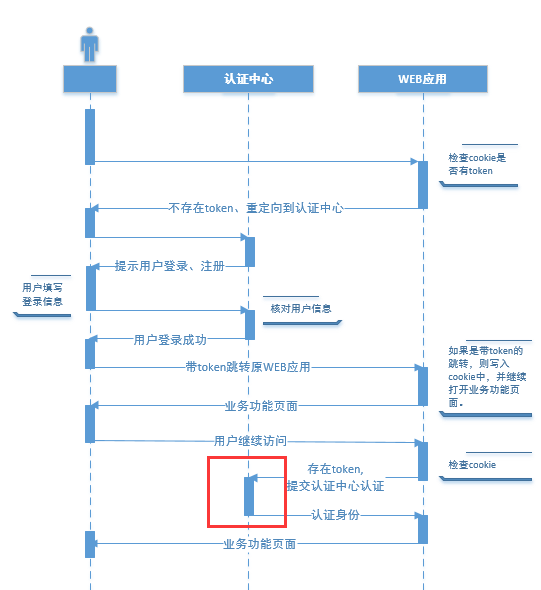
host配置

|  |
| --- |
| 192.168.67.201 passport.atguigu.com |

nginx配置

|  |
| --- |
| upstream passport.atguigu.com{  server 192.168.67.1:8087;  }  server {  listen 80;  server\_name passport.atguigu.com;  location / {  proxy\_pass http://passport.atguigu.com;  proxy\_set\_header X-forwarded-for $proxy\_add\_x\_forwarded\_for;  }  } |

## 4 验证功能



功能：当业务模块某个页面要检查当前用户是否登录时，提交到认证中心，认证中心进行检查校验，返回登录状态、用户Id和用户名称。

### 4.1 思路：

1. 利用密钥和IP检验token是否正确，并获得里面的userId
2. 用userId检查Redis中是否有用户信息,如果有延长它的过期时间。
3. 登录成功状态返回。

### 4.2 代码

UserInfoServiceImpl

|  |
| --- |
| **public** UserInfo verify(String userId){  *// 去缓存中查询是否有redis* Jedis jedis = **redisUtil**.getJedis();  String key = **userKey\_prefix**+userId+**userinfoKey\_suffix**;  String userJson = jedis.get(key);  *// 延长时效* jedis.expire(key,**userKey\_timeOut**);  **if** (userJson!=**null**){  UserInfo userInfo = JSON.*parseObject*(userJson, UserInfo.**class**);  **return** userInfo;  }  **return null**; } |

UserController

|  |
| --- |
| @RequestMapping(**"verify"**) @ResponseBody **public** String verify(HttpServletRequest request){  String token = request.getParameter(**"token"**);  String currentIp = request.getParameter(**"currentIp"**);  *// 检查token  // Map<String, Object> map = JwtUtil.decode(token, signKey, currentIp);* Map<String, Object> map = JwtUtil.*decode*(token, **signKey**, currentIp);  **if** (map!=**null**){  *// 检查redis信息* String userId = (String) map.get(**"userId"**);  UserInfo userInfo = **userInfoService**.verify(userId);  **if** (userInfo!=**null**){  **return "success"**;  }  }  **return "fail"**; } |

测试：

<http://passport.atguigu.com/index?originUrl=https%3A%2F%2Fwww.jd.com%2F>

登陆成功之后返回数据为：在index.html

**window**.**location**.**href**=originUrl+**"newToken="**+data;

输入用户名，密码则会返回一个字符串

https://www.jd.com/?newToken=eyJhbGciOiJIUzI1NiJ9.eyJuaWNrTmFtZSI6IkF0Z3VpZ3UiLCJ1c2VySWQiOiIxIn0.XzRrXwDhYywUAFn-ICLJ9t3Xwz7RHo1VVwZZGNdKaaQ

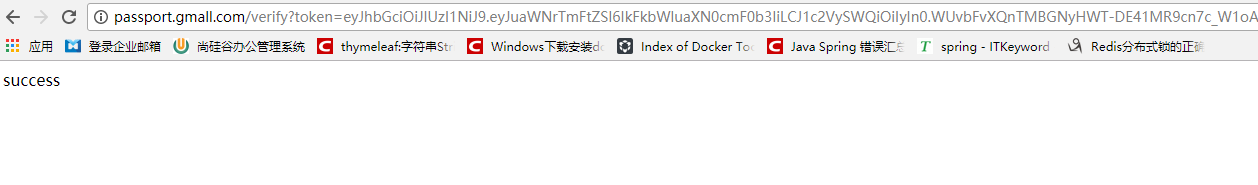
index.html;

字符串中有token值

测试验证的控制器

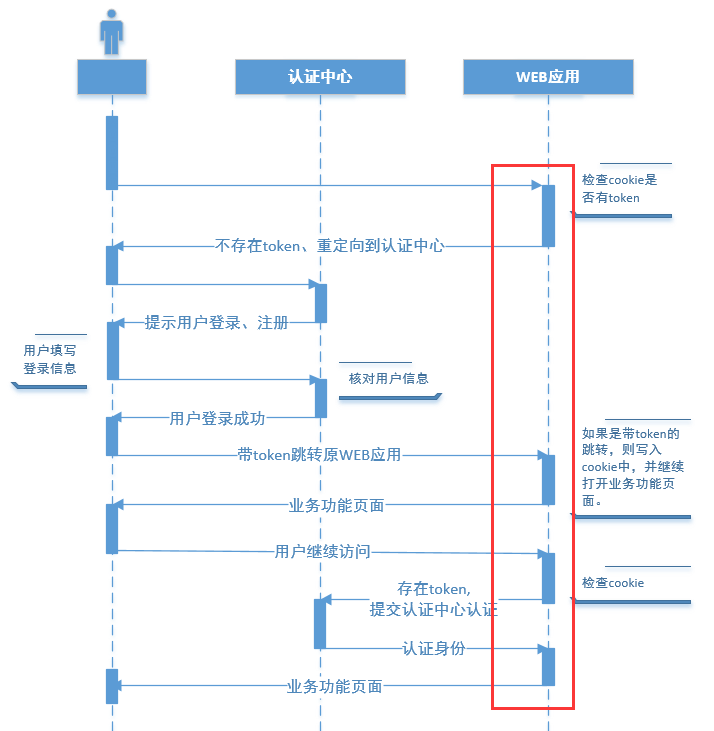
<http://passport.atguigu.com/verify?token=eyJhbGciOiJIUzI1NiJ9.eyJuaWNrTmFtZSI6IkF0Z3VpZ3UiLCJ1c2VySWQiOiIxIn0.XzRrXwDhYywUAFn-ICLJ9t3Xwz7RHo1VVwZZGNdKaaQ&currentIp=192.168.67.1>

页面返回



# 业务模块页面登录情况检查

1. 将登录之后的token 放入cookie 中
2. 使用拦截器来获取token！{token 中已经有用户的昵称}
3. 系统中哪些模块需要用户登录 {在需要登录的控制器上添加一个注解} 自定义注解



### 1 登录成功后将token写道cookie中

在gmall-web-util项目登录成功后写入cookie。

|  |
| --- |
| @Component **public class** AuthInterceptor **extends** HandlerInterceptorAdapter {   **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {  String token = request.getParameter(**"newToken"**);  *//把token保存到cookie* **if**(token!=**null**){  CookieUtil.*setCookie*(request,response,**"token"**,token,WebConst.***COOKIE\_MAXAGE***,**false**);  }  **if**(token==**null**){  token = CookieUtil.*getCookieValue*(request, **"token"**, **false**);  }   **if**(token!=**null**) {  *//读取token* Map map = getUserMapByToken(token);  String nickName = (String) map.get(**"nickName"**);  request.setAttribute(**"nickName"**, nickName);  }  **return true**;  }  **private** Map getUserMapByToken(String token){  String tokenUserInfo = StringUtils.*substringBetween*(token, **"."**);  Base64UrlCodec base64UrlCodec = **new** Base64UrlCodec();  **byte**[] tokenBytes = base64UrlCodec.decode(tokenUserInfo);  String tokenJson = **null**;  **try** {  tokenJson = **new** String(tokenBytes, **"UTF-8"**);  } **catch** (UnsupportedEncodingException e) {  e.printStackTrace();  }  Map map = JSON.*parseObject*(tokenJson, Map.**class**);  **return** map;  }  } |

其中用到了CookieUtil的工具。代码如下：

主要三个方法：从cookie中获得值，把值存入cookie, 设定cookie的作用域。

|  |
| --- |
| **public class** CookieUtil {   **public static** String getCookieValue(HttpServletRequest request, String cookieName, **boolean** isDecoder) {  Cookie[] cookies = request.getCookies();  **if** (cookies == **null** || cookieName == **null**){  **return null**;  }  String retValue = **null**;  **try** {  **for** (**int** i = 0; i < cookies.**length**; i++) {  **if** (cookies[i].getName().equals(cookieName)) {  **if** (isDecoder) {*//如果涉及中文* retValue = URLDecoder.*decode*(cookies[i].getValue(), **"UTF-8"**);  } **else** {  retValue = cookies[i].getValue();  }  **break**;  }  }  } **catch** (UnsupportedEncodingException e) {  e.printStackTrace();  }  **return** retValue;  }    **public static void** setCookie(HttpServletRequest request, HttpServletResponse response, String cookieName, String cookieValue, **int** cookieMaxage, **boolean** isEncode) {  **try** {  **if** (cookieValue == **null**) {  cookieValue = **""**;  } **else if** (isEncode) {  cookieValue = URLEncoder.*encode*(cookieValue, **"utf-8"**);  }  Cookie cookie = **new** Cookie(cookieName, cookieValue);  **if** (cookieMaxage >= 0)  cookie.setMaxAge(cookieMaxage);  **if** (**null** != request)*// 设置域名的cookie* cookie.setDomain(*getDomainName*(request));  cookie.setPath(**"/"**);  response.addCookie(cookie);  } **catch** (Exception e) {  e.printStackTrace();  }  }     */\*\*  \* 得到cookie的域名  \*/* **private static final** String getDomainName(HttpServletRequest request) {  String domainName = **null**;   String serverName = request.getRequestURL().toString();  **if** (serverName == **null** || serverName.equals(**""**)) {  domainName = **""**;  } **else** {  serverName = serverName.toLowerCase();  serverName = serverName.substring(7);  **final int** end = serverName.indexOf(**"/"**);  serverName = serverName.substring(0, end);  **final** String[] domains = serverName.split(**"\\."**);  **int** len = domains.**length**;  **if** (len > 3) {  *// www.xxx.com.cn* domainName = domains[len - 3] + **"."** + domains[len - 2] + **"."** + domains[len - 1];  } **else if** (len <= 3 && len > 1) {  *// xxx.com or xxx.cn* domainName = domains[len - 2] + **"."** + domains[len - 1];  } **else** {  domainName = serverName;  }  }   **if** (domainName != **null** && domainName.indexOf(**":"**) > 0) {  String[] ary = domainName.split(**"\\:"**);  domainName = ary[0];  }  System.***out***.println(**"domainName = "** + domainName);  **return** domainName;  } } |

WebConst是常量类，

|  |
| --- |
| public interface WebConst {  *//登录页面* public final static String *LOGIN\_URL*="http://passport.atguigu.com/index";  *//认证接口* public final static String *VERIFY\_URL*="http://passport.atguigu.com/verify";  *//cookie的有效时间：默认给7天* public final static int *cookieMaxAge*=7\*24\*3600; } |

Gmall-item-web 项目中item.html页面中



### 2 加入拦截器的配置

首先这个验证功能是每个模块都要有的，也就是所有web模块都需要的。在每个controller方法进入前都需要进行检查。可以利用在springmvc中的拦截器功能。

因为咱们是多个web模块分布式部署的，所以不能写在某一个web模块中，可以一个公共的web模块，就是gmall-web-util中。

位置：

首先自定义一个拦截器，继承成springmvc的HandlerInterceptorAdapter

，通过重新它的preHandle方法实现，业务代码前的校验工作

@Configuration == xml

|  |
| --- |
| @Configuration **public class** WebMvcConfiguration **extends** WebMvcConfigurerAdapter{  @Autowired  AuthInterceptor **authInterceptor**;   @Override  **public void** addInterceptors(InterceptorRegistry registry){  registry.addInterceptor(**authInterceptor**).addPathPatterns(**"/\*\*"**);  **super**.addInterceptors(registry);  } } |

配置完拦截器之后，需要在web 项目中添加包扫描

@ComponentScan(basePackages = **"com.atguigu.gmall1128"**)

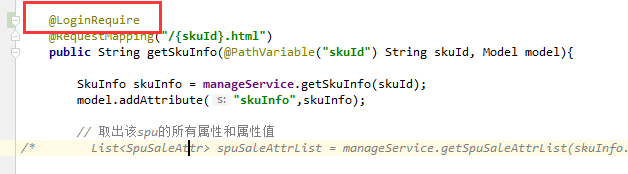
### 3 检验方法是否需要验证用户登录状态

为了方便程序员在controller方法上标记，可以借助自定义注解的方式。

比如某个controller方法需要验证用户登录，在方法上加入自定义的@LoginRequie。

像这样

|  |
| --- |
| 1、RetentionPolicy.SOURCE：注解只保留在源文件，当Java文件编译成class文件的时候，注解被遗弃； 2、RetentionPolicy.CLASS：注解被保留到class文件，但jvm加载class文件时候被遗弃，这是默认的生命周期； 3、RetentionPolicy.RUNTIME：注解不仅被保存到class文件中，jvm加载class文件之后，仍然存在；  这3个生命周期分别对应于：Java源文件(.java文件) ---> .class文件 ---> 内存中的字节码。 |



如果方法被加了注解，在拦截器中就可以捕捉到。

添加自定义注解

|  |
| --- |
| @Target(ElementType.***METHOD***) @Retention(RetentionPolicy.***RUNTIME***) **public** @**interface** LoginRequire {   **boolean** autoRedirect() **default true**; } |

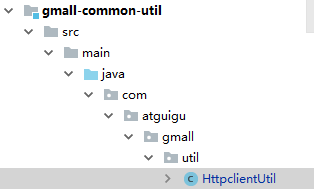
在拦截方法preHandle方法中继续添加，检验登录的代码。

|  |
| --- |
| *If 代码块后面*  HandlerMethod handlerMethod =(HandlerMethod) handler; LoginRequire loginRequireAnnotation = handlerMethod.getMethodAnnotation(LoginRequire.**class**); **if**(loginRequireAnnotation!=**null**){  String remoteAddr = request.getHeader(**"x-forwarded-for"**);  String result = HttpClientUtil.*doGet*(WebConst.***VERIFY\_ADDRESS*** + **"?token="** + token + **"&currentIp="** + remoteAddr);  **if**(**"success"**.equals(result)){  Map map = getUserMapByToken(token);  String userId =(String) map.get(**"userId"**);  request.setAttribute(**"userId"**,userId);  **return true**;  }**else**{  **if**(loginRequireAnnotation.autoRedirect()){  String requestURL = request.getRequestURL().toString();  String encodeURL = URLEncoder.*encode*(requestURL, **"UTF-8"**);  response.sendRedirect(WebConst.***LOGIN\_ADDRESS***+**"?originUrl="**+encodeURL);  **return false**;  }  } } |

以上方法，检查业务方法是否需要用户登录，如果需要就把cookie中的token和当前登录人的ip地址发给远程服务器进行登录验证，返回的result是验证结果true或者false。如果验证未登录，直接重定向到登录页面。

以上使用到了一个自定义的HttpclientUtil工具类，放在gmall-common-util模块中。专门负责通过restful风格调用接口。

位置：



代码如下。

|  |
| --- |
| **public class** HttpClientUtil {  **public static** String doGet(String url) {  *// 创建Httpclient对象* CloseableHttpClient httpclient = HttpClients.*createDefault*();  *// 创建http GET请求* HttpGet httpGet = **new** HttpGet(url);  CloseableHttpResponse response = **null**;  **try** {  *// 执行请求* response = httpclient.execute(httpGet);  *// 判断返回状态是否为200* **if** (response.getStatusLine().getStatusCode() == HttpStatus.***SC\_OK***) {  HttpEntity entity = response.getEntity();  String result = EntityUtils.*toString*(entity, **"UTF-8"**);  EntityUtils.*consume*(entity);  httpclient.close();  **return** result;  }  httpclient.close();  }**catch** (IOException e){  e.printStackTrace();  **return null**;  }   **return null**;  }    **public static void** download(String url,String fileName) {   *// 创建Httpclient对象* CloseableHttpClient httpclient = HttpClients.*createDefault*();  *// 创建http GET请求* HttpGet httpGet = **new** HttpGet(url);  CloseableHttpResponse response = **null**;  **try** {  *// 执行请求* response = httpclient.execute(httpGet);  *// 判断返回状态是否为200* **if** (response.getStatusLine().getStatusCode() == HttpStatus.***SC\_OK***) {  HttpEntity entity = response.getEntity();   *// String result = EntityUtils.toString(entity, "UTF-8");* **byte**[] bytes = EntityUtils.*toByteArray*(entity);  File file =**new** File(fileName);  *// InputStream in = entity.getContent();* FileOutputStream fout = **new** FileOutputStream(file);  fout.write(bytes);   EntityUtils.*consume*(entity);   httpclient.close();  fout.flush();  fout.close();  **return** ;  }  httpclient.close();  }**catch** (IOException e){  e.printStackTrace();  **return** ;  }  **return** ;  } } |

# 测试效果

Bug：测试会发现取不到IP地址！

原因是：ip地址是通过nginx来配置出来的,所有测试item的时候，需要在item的nginx 中添加

proxy\_set\_header X-forwarded-for $proxy\_add\_x\_forwarded\_for;

重新测试即可！