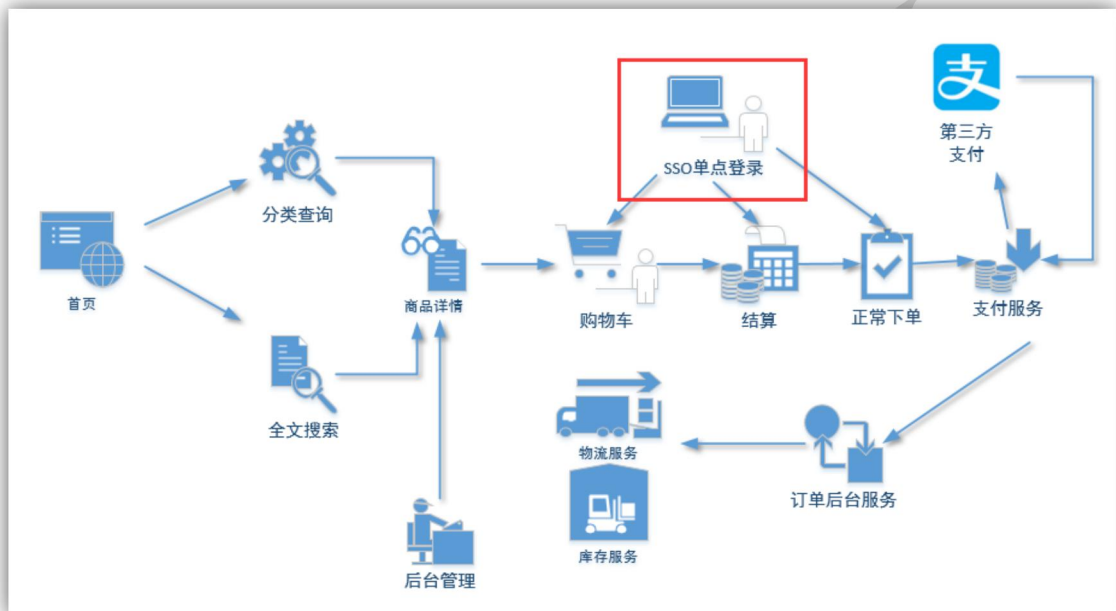


单点登录

版本：V 1.0
www.atguigu.com

所在位置：



一、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

WEB 应用集群，session 共享模式

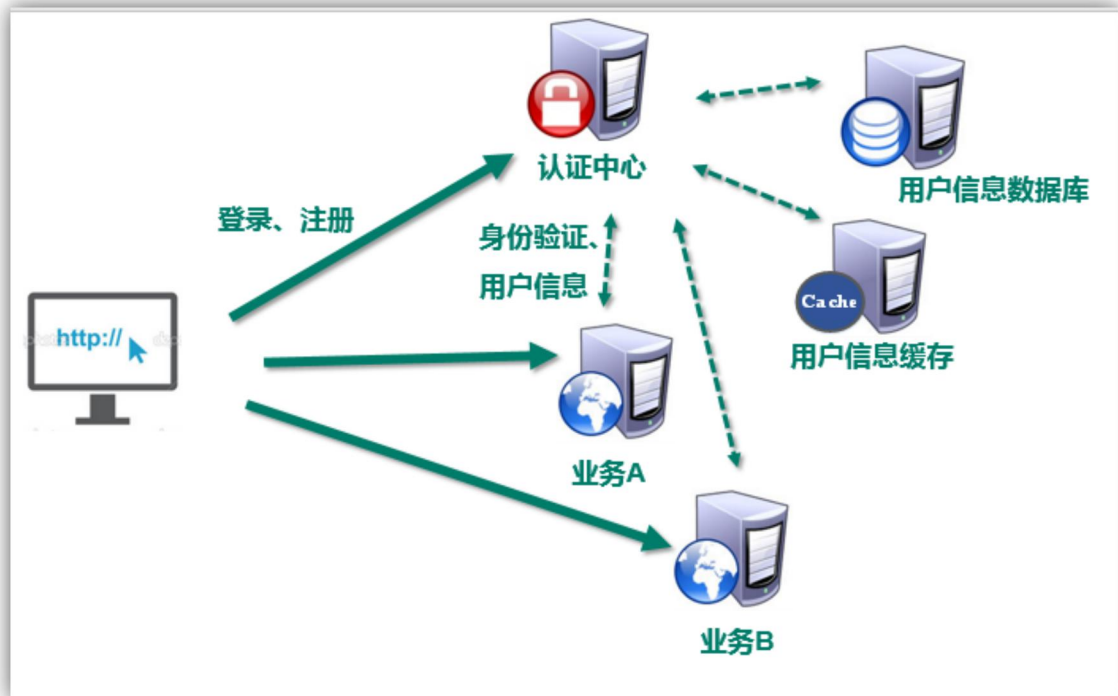


解决了单点性能瓶颈。

问题：

- 1、多业务分布式数据独立管理，不适合统一维护一份 session 数据。
- 2、分布式按业务功能切分，用户、认证解耦出来单独统一管理。
- 3、cookie 中使用 jsessionId 容易被篡改、盗取。
- 4、跨顶级域名无法访问。

分布式，SSO(single sign on)模式



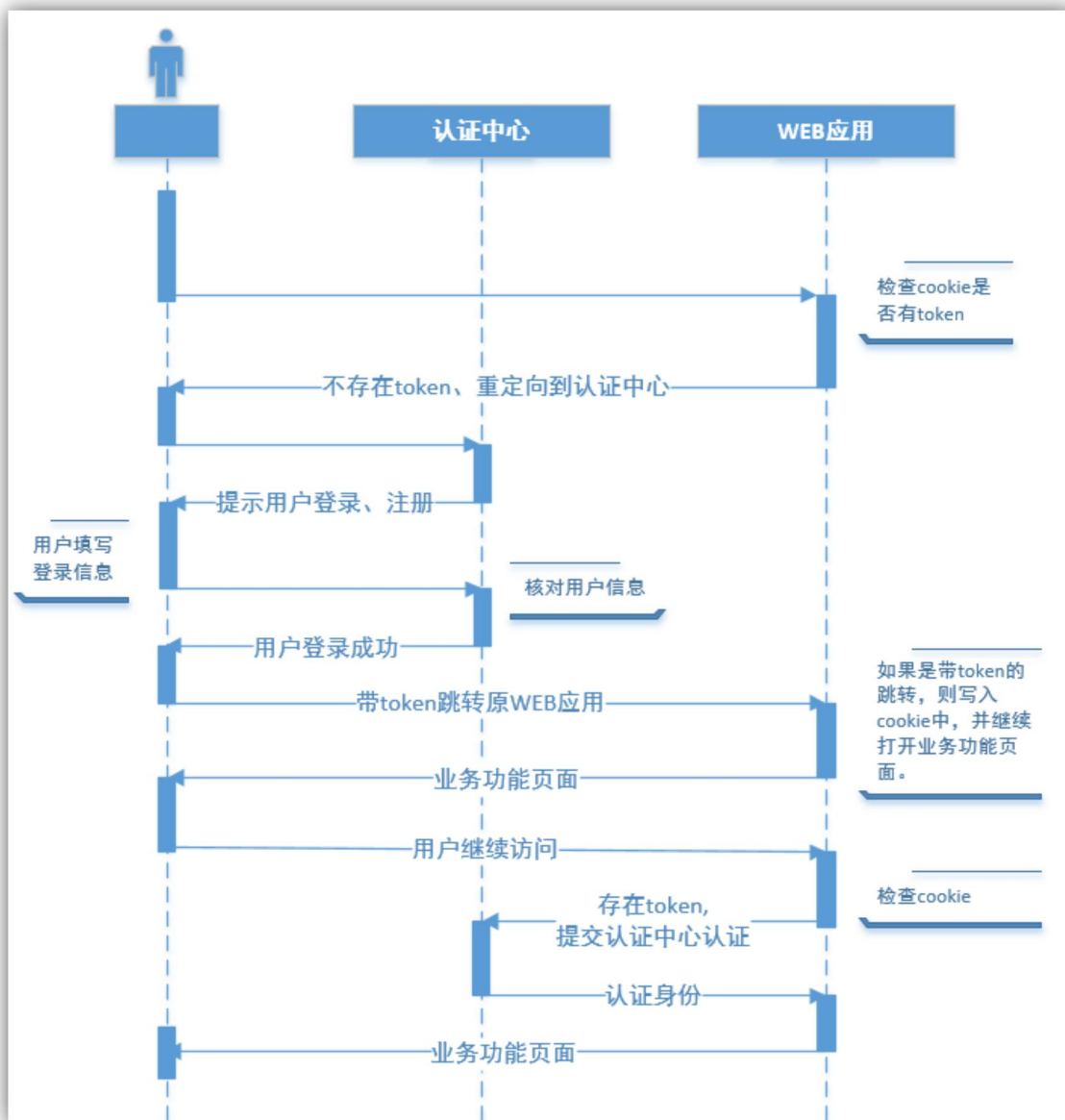
解决：

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
跨域不是问题

缺点：

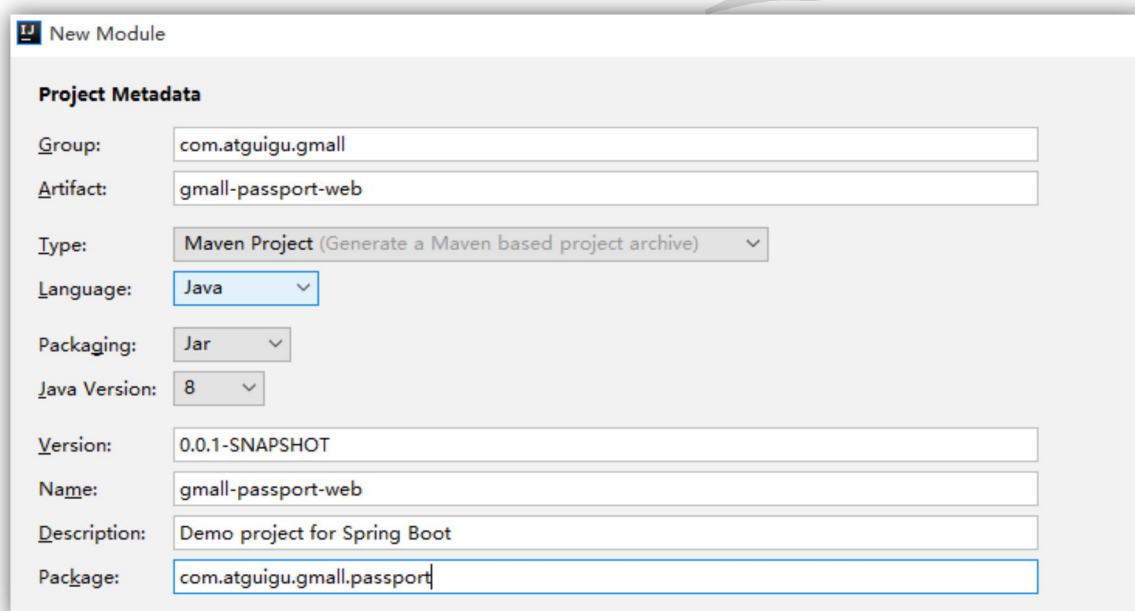
认证服务器访问压力较大。

业务流程图



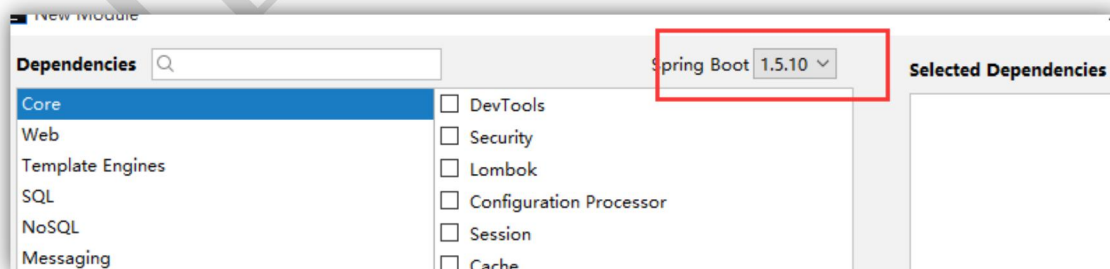
二、 认证中心模块

1 搭建认证中心模块



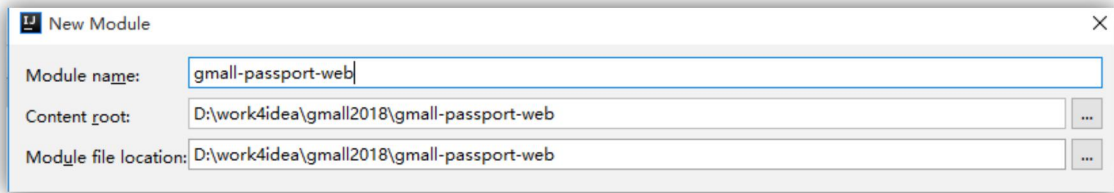
The 'New Module' dialog box is shown with the following fields:

- Group: com.atguigu.gmall
- Artifact: gmall-passport-web
- Type: Maven Project (Generate a Maven based project archive)
- Language: Java
- Packaging: Jar
- Java Version: 8
- Version: 0.0.1-SNAPSHOT
- Name: gmall-passport-web
- Description: Demo project for Spring Boot
- Package: com.atguigu.gmall.passport



The 'Dependencies' section of the 'New Module' dialog is shown. It includes a search bar, a list of dependency categories, and a list of selected dependencies.

Dependencies	Selected Dependencies
Core	Spring Boot 1.5.10
Web	
Template Engines	
SQL	
NoSQL	
Messaging	
DevTools	
Security	
Lombok	
Configuration Processor	
Session	
Cache	



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

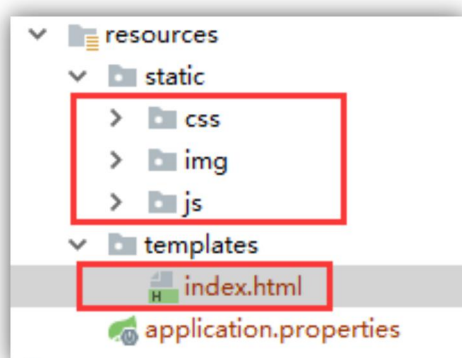
```
server.port=8086

spring.thymeleaf.cache=false

spring.thymeleaf.mode=LEGACYHTML5

spring.dubbo.application.name=passport-web
spring.dubbo.registry.protocol=zookeeper
spring.dubbo.registry.address=192.168.67.163:2181
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



把 index.html 中的路径从 ../static/ 换成 /

host 配置

```
192.168.67.163 passport.atguigu.com
```

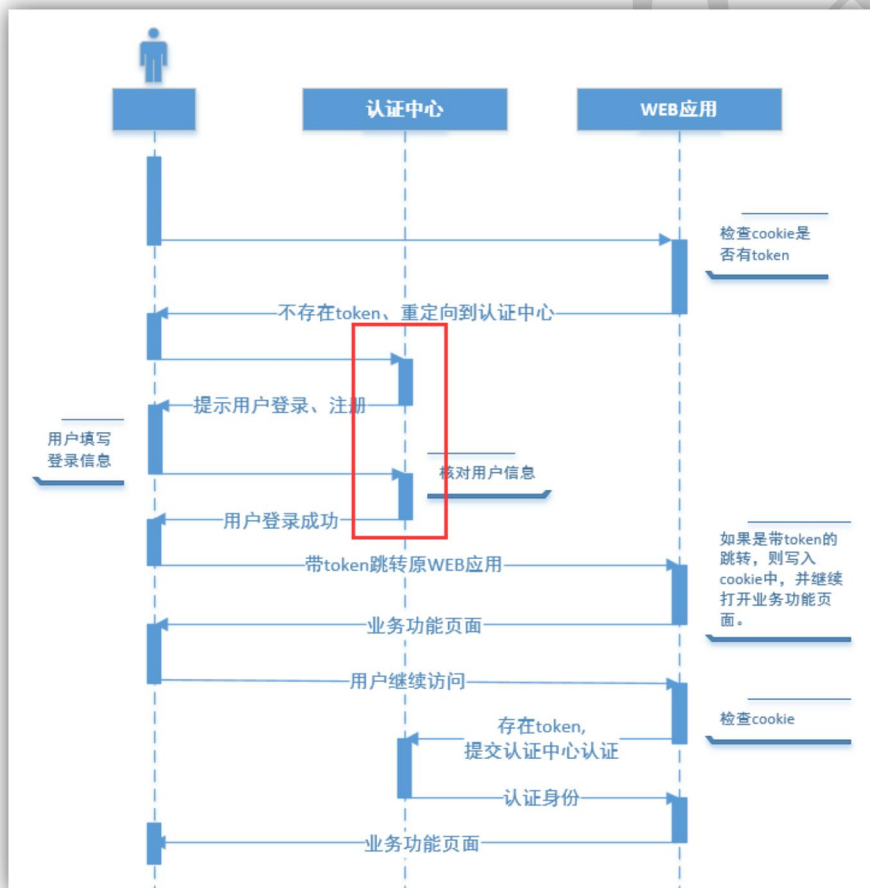
nginx 配置

```
upstream passport.atguigu.com{
    server 192.168.67.1:8086;
}
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;
    }
}
```



```
}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、用接受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 核对后台登录信息+用户登录信息载入缓存

UserManageServiceImpl

```
public UserInfo login(UserInfo userInfo){
    String passwd = DigestUtils.md5Hex(userInfo.getPasswd());
    userInfo.setPasswd(passwd);

    UserInfo userInfoResult = userInfoMapper.selectOne(userInfo);
    if(userInfoResult!=null){

        String userInfoKey="user:"+userInfoResult.getId()+"info";

        Jedis jedis = redisUtil.getJedis();
        String userInfoJson = JSON.toJSONString(userInfoResult);
        jedis.setex(userInfoKey,UserConst.sessionExpire,userInfoJson);

        return userInfoResult ;
    }else{
        return null;
    }
}
```

3.3 生成 token

JWT 工具

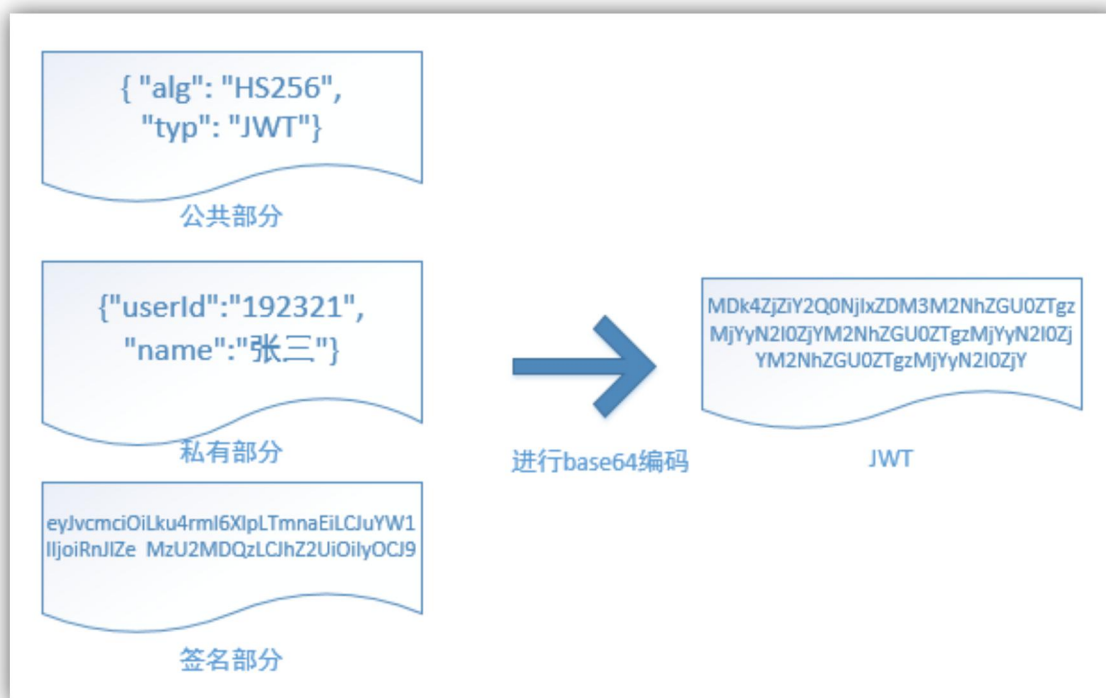
JWT（Json Web Token）是为了在网络应用环境间传递声明而执行的一种基于 JSON 的开放标准。

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。
- 2、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。
- 3、签名部分
根据用户信息+盐值+密钥生成的签名。如果想知道 JWT 是否是真实的只要把 JWT 的信息取出来，加上盐值和服务器中的密钥就可以验证真伪。所以不管由谁保存 JWT，只要没有密钥就无法伪造。
- 4、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吧 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：

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;
    }
}
```

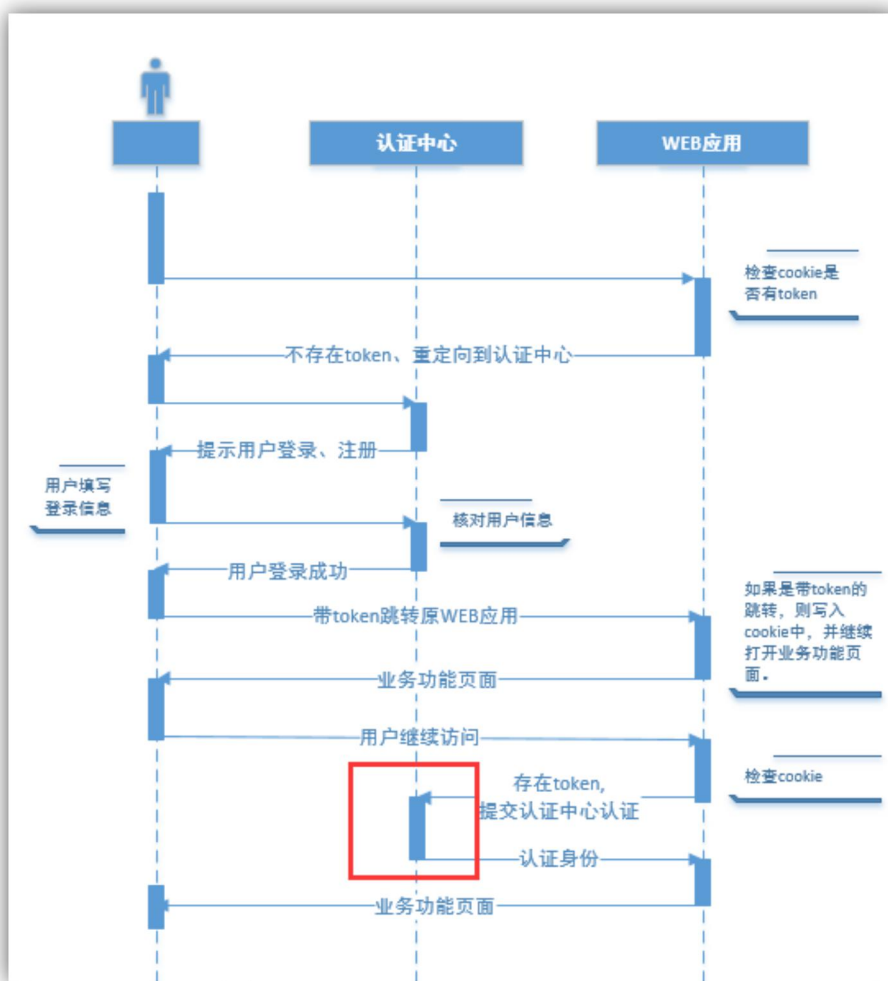
3.4 Controller

PassportController

```
@RequestMapping(value = "login", method = RequestMethod.POST)
@ResponseBody
public String login(UserInfo userInfo, HttpServletRequest httpRequest) {
    String remoteAddr = httpRequest.getHeader("x-forwarded-for");
    String userId = userManagerService.login(userInfo);
    if (userId == null) {
        return "fail";
    } else {
        Map map = new HashMap();
        map.put("userId", userId);
        map.put("nickName", userInfoResult.getNickName());

        String token = JwtUtil.encode(signKey, map, remoteAddr);
        return token;
    }
}
```

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

UserManageServiceImpl

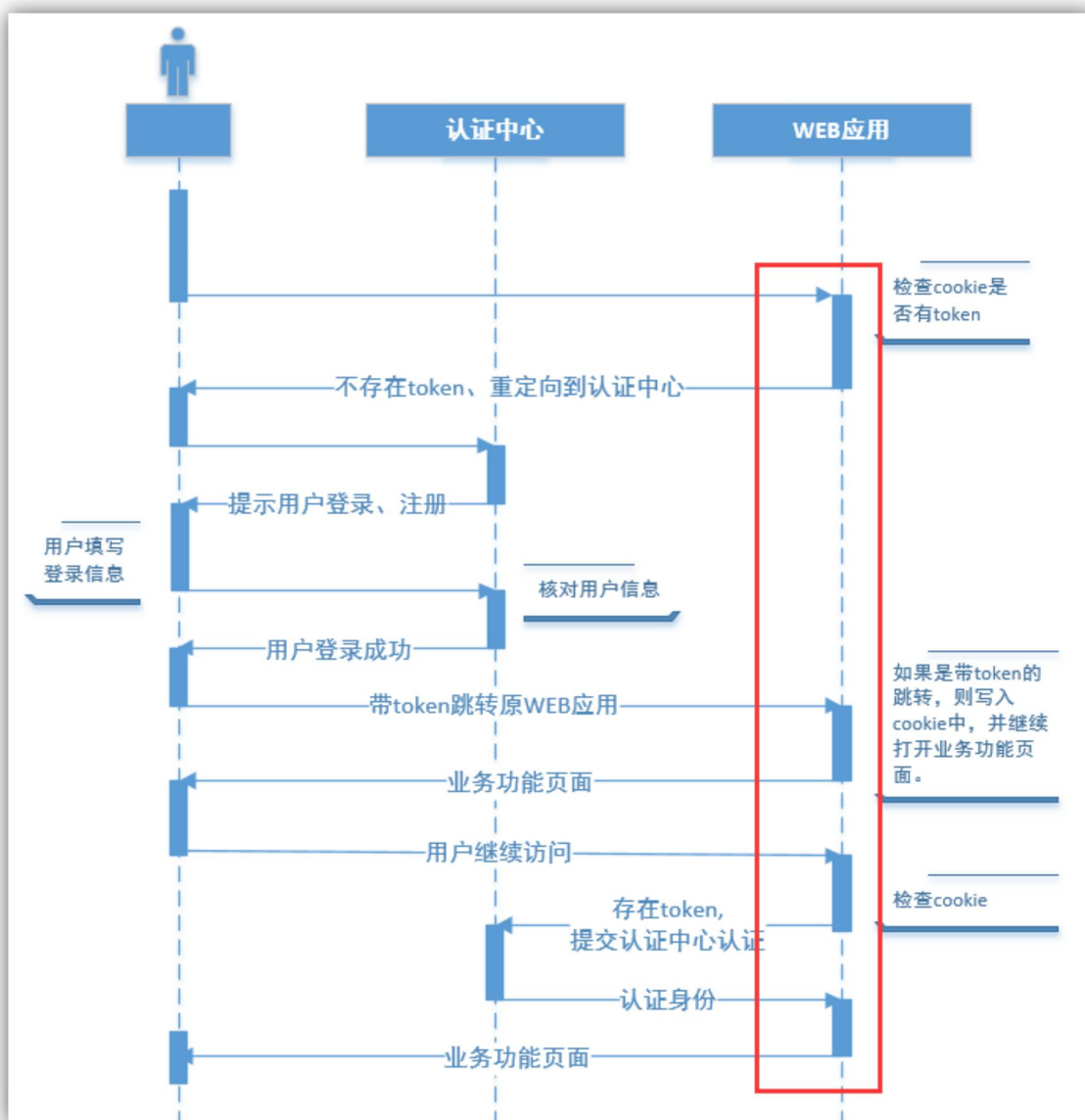
```
public boolean verify(String userId){
    Jedis jedis = redisUtil.getJedis();
    String userInfoKey="user:"+userId+"info";
    Boolean exists = jedis.exists(userInfoKey);
    if(exists) {
        jedis.expire(userInfoKey, UserConst.sessionExpire);
    }
    return exists;
}
```

UserController

```
@RequestMapping(value = "verify",method = RequestMethod.POST)
@ResponseBody
public String verify(HttpServletRequest httpServletRequest){
    String token = httpServletRequest.getParameter("token");
    String curIp=httpServletRequest.getParameter("currentIp");
    Map<String, Object> map = JwtUtil.decode(token, signKey, curIp);
    JSONObject jsonObject=new JSONObject();
    String userId =(String) map.get("userId");
    boolean verify = userManageService.verify(userId);

    return Boolean.toString(verify) ;
}
```

三、 业务模块页面登录情况检查



问题:

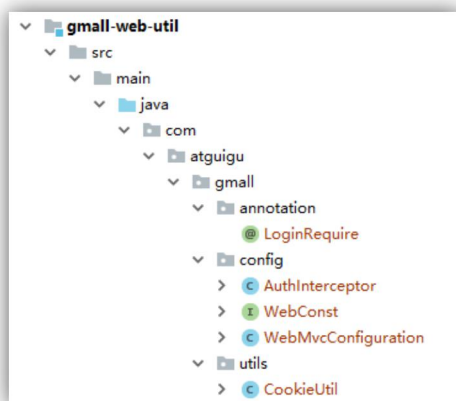
- 1、由认证中心签发的 token 如何保存？保存到浏览器的 cookie 中
- 2、难道每一个模块都要做一个 token 的保存功能？ 拦截器
- 3、如何区分请求是否一定要登录？自定义注解

1 加入拦截器

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
public class WebMvcConfiguration extends WebMvcConfigurerAdapter{
    @Autowired
    AuthInterceptor authInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry){
        registry.addInterceptor(authInterceptor).addPathPatterns("/**");
        super.addInterceptors(registry);
    }
}
```


2 登录成功后跳转回来的处理

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 newToken = request.getParameter("newToken");
    if(newToken!=null&&newToken.length()>0){
        CookieUtil.setCookie(request,response,"token",newToken,WebConst.cookieExpire,false);
    }

    return true;
}
}
```

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;
}
```

3 检查 cookie 中是否有 token

要做的工作：

- 4 检查 cookie 中是否有 token,如果有把 cookie 中的昵称取放入页面 request 属性中。
- 5 检查是否需要验证登录。

如果需要，调用认证模块接口

如果认证通过程序照常执行

如果认证不通过，跳转到登录页面。

6 检查是否是登陆页面跳转回来的，如果附带新的 token 则把 token 保存到 cookie 中。

思路：

。

AuthInterceptor

```
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws
Exception {

    String newToken = request.getParameter("newToken");
    if(newToken!=null&&newToken.length()>0){
        CookieUtil.setCookie(request,response,"token",newToken,WebConst.cookieExpire,false);
    }

    //1 进行如果能从 cookie 把 token 取出来，进行解析，显示页面上。
    String token = CookieUtil.getCookieValue(request, "token", false);
    String userId=null;
    if(token!=null){
        Base64UrlCodec base64UrlCodec=new Base64UrlCodec();
        // 两个“.”之间的部分是实际内容
        String tokenForDecode= StringUtils.substringBetween(token, ".");

        byte[] tokenByte = base64UrlCodec.decode(tokenForDecode);
        String tokenJson=new String(tokenByte,"UTF-8");
        System.out.println("tokenJson = " + tokenJson);

        JSONObject jsonObject = JSON.parseObject( tokenJson);

        userId = jsonObject.getString("userId");
        String nickName = jsonObject.getString("nickName");

        request.setAttribute("nickName",nickName);
    }

    return true;
}
```

4 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

```
@LoginRequire
@RequestMapping("/{skuId}.html")
public String getSkuInfo(@PathVariable("skuId") String skuId, Model model){

    SkuInfo skuInfo = manageService.getSkuInfo(skuId);
    model.addAttribute("skuInfo", skuInfo);

    // 取出该sku的所有属性和属性值
    /* List<SpuSaleAttr> spuSaleAttrList = manageService.getSpuSaleAttrList(skuInfo.
```

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。

```
//检查是否需要验证用户已经登录
HandlerMethod handlerMethod = (HandlerMethod) handler;
LoginRequire methodAnnotation = handlerMethod.getMethodAnnotation(LoginRequire.class);
if(methodAnnotation!=null){
    String currentIp = request.getHeader("x-forwarded-for");

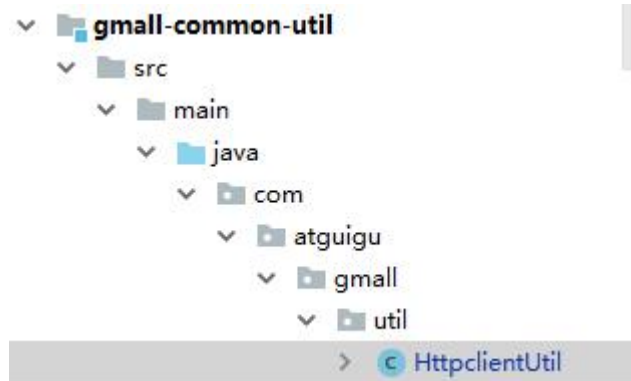
    Map map=new HashMap();
    map.put("currentIp",currentIp);
    map.put("token",token);
    String result=null;
    if(token !=null) {
        result = HttpClientUtil.doPost(WebConst.VERIFY_URL, map);
    }
    if(result!=null&&result.equals("true")){
        request.setAttribute("userId",userId); //只有验证过才能取到userId
        return true;
    }else{
        if(methodAnnotation.autoRedirect()) {
            String url = URLEncoder.encode(request.getRequestURL().toString(), "utf-8");

            response.sendRedirect(WebConst.LOGIN_URL + "?originUrl=" + url);
            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 String doPost(String url, Map<String,String> paramMap) {
        // 创建 HttpClient 对象
        CloseableHttpClient httpClient = HttpClients.createDefault();
        // 创建 http Post 请求
        HttpPost httpPost = new HttpPost(url);
        CloseableHttpResponse response = null;
        try {
            List<BasicNameValuePair> list=new ArrayList<>();
            for (Map.Entry<String, String> entry : paramMap.entrySet()) {
                list.add(new BasicNameValuePair(entry.getKey(),entry.getValue()));
            }
            HttpEntity httpEntity=new UrlEncodedFormEntity(list,"utf-8");
        }
    }
}
```

```
httpPost.setEntity(httpEntity);
// 执行请求
response = httpClient.execute(httpPost);

// 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;
}
```

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;
}
```

四、 测试效果