第六章 企业项目开发--cookie

注：本章代码基于《第五章 企业项目开发--mybatis注解与xml并用》的代码

在实际项目中，我们会存储用户状态信息，基本使用两种手段：cookie和session

1、cookie：

1.1、流程：

- 服务端将cookie的属性值设置好之后，通过HttpServletResponse将cookie写入响应头；
- 服务端从请求头中通过HttpServletRequest将所有cookie（当前被访问页面的所有cookie）读取出来，遍历寻找指定name的cookie即可。

流程与下边的例子对应一下，立刻就清楚了。

1.2、包含选项：（当前使用最多的就是**version0**版本的cookie，这里仅说version0的几个最常用的属性）

- name：cookie名
- value：cookie值
- domain：cookie写在哪一个域下（例如：aaa.com）
- path：cookie写在哪一个uri下（例如：/admin）
- expires：cookie过期时间，超过某个指定时间点cookie就消失，当然设成-1的话，关闭浏览器cookie就会消失；指定了时间点的话，该cookie就会存在硬盘上

对于domain，一个例子解释清楚：

假设：www.baidu.com和e.baidu.com两个域，path="/"，

1）当domain="baidu.com"时，两个域都可以访问生成的cookie

2）当domain="www.baidu.com"时，只有前一个域有生成cookie

对于path，用一个例子解释清楚：

假设http://www.baidu.com/dev/zz/和http://www.baidu.com/dev/xx/，

1）当path="/dev"，生成的cookie上边的两个地址共享；

2）当path="/dev/zz"，生成的cookie只有第一个地址有，第二个地址没有。

根据上边的例子，对于path与domain是相互依赖的，共同决定某个页面地址是否可以生成指定的cookie

1.3、优点：

- 基本不需要使用服务器空间，用户会话信息存在了客户端（这个只是列出来，基本不能算优点，反而应该算是缺点，因为存在客户端不安全）
- 在分布式系统中，因为cookie存在了浏览器，所以不需要考虑同一个用户怎样定位到同一台服务器这个问题

1.4、缺点：

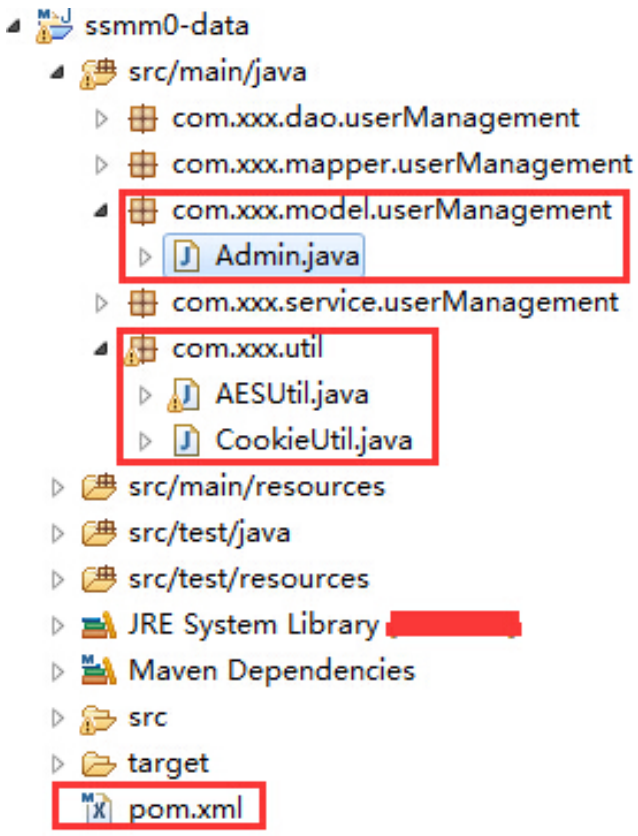
- cookie个数和总大小有限制，一般而言，每个域下可存放<=50个cookie，所有cookie的总大小<=4095个字节左右，chrome浏览器的cookie总大小会大很多（>80000个字节）。这些数据参考自**《深入分析Java Web技术内幕》**
- 如果cookie很多，客户端和服务端会浪费很对带宽，而且也会拖慢速度，一般而言，可以使用压缩算法做压缩，但是，可能压缩后的size也很大
- 不安全，如果被窃听者劫取到你的cookie，即使你对这些cookie做了加密，使其看不到cookie中的信息，但是其也可以通过该cookie模拟登录行为，然后获得一些权限，执行一些操作。（这是个疑问，记住我这个功能就是这样做的，怎样做到安全的呢？）**（答案见最下方）**

2、项目中使用cookie存储用户会话状态

代码实现：本章代码基于上一章代码实现，下面只列出修改或添加的一些类。

通常情况下，如果工具类很多，我们可以重开一个项目，例如：ssmm0-util，然后在这个子项目中添加类，当然，如果项目本身也不大，直接将工具类放在ssmm0-data子项目下即可。

2.1、ssmm0-data：



2.1.1、pom.xml：

```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/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <!-- 指定父模块 -->
    <parent>
        <groupId>com.xxx</groupId>
        <artifactId>ssmm0</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <groupId>com.xxx.ssmm0</groupId>
    <artifactId>ssmm0-data</artifactId>

    <name>ssmm0-data</name>
    <packaging>jar</packaging><!-- 只是作为其他模块使用的工具 -->

    <!-- 引入实际依赖 -->
    <dependencies>
        <!-- mysql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <!-- 数据源 -->
        <dependency>
            <groupId>org.apache.tomcat</groupId>
            <artifactId>tomcat-jdbc</artifactId>
        </dependency>
        <!-- mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
        </dependency>
        <!-- servlet --><!-- 为了会用cookie -->
```

```
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
        </dependency>
        <!-- bc-加密 -->
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcprov-jdk15on</artifactId>
        </dependency>
        <!-- cc加密 -->
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
        </dependency>
    </dependencies>
</project>
```

说明：

- 引入了servlet-jar包，主要是需要HttpServletResponse向响应头写cookie和使用HttpServletRequest从请求头读取cookie；
- 引入commons-codec和bouncy-castle，主要是为了实现AES加密与Base64编码。

注意：

- servlet-jar的scope是provided，不具有传递性，所以如果在ssmm0-userManagement中需要用到servlet-jar，则ssmm0-userManagement**需要**自己再引一遍
- commons-codec和bouncy-castle的scope都是采用了默认的compile，具有传递性，所以如果在ssmm0-userManagement中需要用到commons-codec和bouncy-castle，ssmm0-userManagement**不需要**自己再引一遍了

2.1.2、AESUtil：（AES加密）

```
package com.xxx.util;

import java.io.UnsupportedEncodingException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Security;
import java.security.spec.InvalidKeySpecException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

/**
 * 基于JDK或BC的AES算法，工作模式采用ECB
 */
public class AESUtil {
    private static final String ENCODING = "UTF-8";
    private static final String KEY_ALGORITHM = "AES";//产生密钥的算法
    private static final String CIPHER_ALGORITHM = "AES/ECB/PKCS5Padding";//加解密算法 格式：算法/工作模式/填充模式 注意：ECB不使用IV参数
    private static final String ENCRYPT_KEY = "WAUISIgpBh1R/+3f2Ze4csUU/tl/O8x56DbVb7mTs7w=";//这个是先运行一遍getKey()方法产生的，然后卸载了这里。
```

```
31
32      /**
33       * 产生密钥（线下产生好，然后配在项目中）
34       */
35      public static byte[] getKey() throws NoSuchAlgorithmException{
36          Security.addProvider(new BouncyCastleProvider());//在BC中用，JDK下去除
37          KeyGenerator keyGenerator = KeyGenerator.getInstance(KEY_ALGORITHM);
38          keyGenerator.init(256);//初始化密钥长度,128,192,256（选用192和256的时候需要配置无政策限制权限文件
--JDK6）
39          SecretKey key =keyGenerator.generateKey();//产生密钥
40          return key.getEncoded();
41      }
42
43      /**
44       * 还原密钥：二进制字节数组转换为Java对象
45       */
46      public static Key toKey(byte[] keyByte){
47          Security.addProvider(new BouncyCastleProvider());//在BC中用，JDK下去除
48          return new SecretKeySpec(keyByte, KEY_ALGORITHM);
49      }
50
51      /**
52       * AES加密，并转为16进制字符串或Base64编码字符串
53       */
54      public static String encrypt(String data, byte[] keyByte) throws InvalidKeyException,
55                                                                       NoSuchAlgorithmException,
56                                                                       InvalidKeySpecException,
57                                                                       NoSuchPaddingException,
58                                                                       IllegalBlockSizeException,
59                                                                       BadPaddingException,
60                                                                       UnsupportedEncodingException,
61                                                                       NoSuchProviderException,
62
InvalidAlgorithmParameterException {
63          Key key = toKey(keyByte);//还原密钥
64          //Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);//JDK下用
65          Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM,"BC");//BC下用
66          cipher.init(Cipher.ENCRYPT_MODE, key);//设置加密模式并且初始化key
67          byte[] encodedByte = cipher.doFinal(data.getBytes(ENCODING));
68          return Base64.encodeBase64String(encodedByte);//借助CC的Base64编码
69      }
70
71      /**
72       * AES解密
73       * @param data          待解密数据为字符串
74       * @param keyByte      密钥
75       */
76      public static String decrypt(String data, byte[] keyByte) throws InvalidKeyException,
77                                                                       NoSuchAlgorithmException,
78                                                                       InvalidKeySpecException,
79                                                                       NoSuchPaddingException,
80                                                                       IllegalBlockSizeException,
81                                                                       BadPaddingException,
82                                                                       UnsupportedEncodingException,
83                                                                       NoSuchProviderException,
84                                                                       InvalidAlgorithmParameterException {
85          Key key = toKey(keyByte);//还原密钥
86          //Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);//JDK下用
87          Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM,"BC");//BC下用
88          cipher.init(Cipher.DECRYPT_MODE, key);
89          byte[] decryptByte = cipher.doFinal(Base64.decodeBase64(data));//注意data不可以直接采用
data.getByte()方法转化为字节数组，否则会抛异常
90          return new String(decryptByte, ENCODING);//这里注意用new String()
91      }
92
```

```
 93     /**
 94      * 测试
 95      */
 96     public static void main(String[] args) throws NoSuchAlgorithmException,
 97                                                    InvalidKeyException,
 98                                                    InvalidKeySpecException,
 99                                                    NoSuchPaddingException,
100                                                    IllegalBlockSizeException,
101                                                    BadPaddingException,
102                                                    UnsupportedEncodingException,
103                                                    NoSuchProviderException,
104                                                    InvalidAlgorithmParameterException {
105         String data = "找一个好姑娘做老婆是我的梦 想!";
106         /*************测试encryptAESHex()、decrypt()**************/
107         System.out.println("原文-->"+data);
108         byte[] keyByte3 = Base64.decodeBase64(ENCRYPT_KEY);
109         System.out.println("密钥-->"+Base64.encodeBase64String(keyByte3));//这里将二进制的密钥使用base64加密保存，这也是在实际中使用的方式
110         String encodedStr = AESUtil.encrypt(data, keyByte3);
111         System.out.println("加密后-->"+encodedStr);
112         System.out.println("解密String后-->"+AESUtil.decrypt(encodedStr, keyByte3));
113     }
114 }
```

说明：

关于AES的具体内容，可以参照"Java加密与解密系列的第八章"

AESUtil这个类就是根据上边这篇博客的AESJDK这个类进行修改封装的。

注意：

AESUtil类的ENCRYPT_KEY变量的值是先运行了其中的getKey()方法，然后又通过Base64编码产生的，产生这个密钥后，我们将这个密钥写在类中。简单来说，就是线下先产生密钥，然后写在程序中，之后getKey()方法就再没有用了，可以直接删掉。

关于生成密钥的这段代码如下：

```
String key = Base64.encodeBase64String(AESUtil.getKey());
```

2.1.3、CookieUtil：（cookie的基本操作：增删查，注意没有改）

```
 1 package com.xxx.util;
 2
 3 import javax.servlet.http.Cookie;
 4 import javax.servlet.http.HttpServletRequest;
 5 import javax.servlet.http.HttpServletResponse;
 6
 7 /**
 8  * cookie操作相关类
 9  */
10 public class CookieUtil {
11
12     /**
13      * 增加cookie
14      * @param name        cookie名
15      * @param value        cookie值
16      * @param domain    指定cookie写在哪个域下
17      * @param path        指定cookie存在那个路径下(其实就是一个uri)
18      * @param expiry    指定cookie过期时间
19      */
20     public static void addCookie(String name,
21                                  String value,
22                                  String domain,
```

```
                                String path,
                                int expiry,
                                HttpServletResponse response){
        Cookie cookie = new Cookie(name, value);

        cookie.setDomain(domain);
        cookie.setPath(path);
        cookie.setMaxAge(expiry);

        response.addCookie(cookie);
    }

    /**
     * 获取cookie
     * @param request
     * @param key    将要查找的cookie的名
     * @return        返回cookie的值
     */
    public static String getCookie(HttpServletRequest request, String key){
        Cookie[] cookies = request.getCookies();
        /*
         * 注意在执行循环之前，不需要判断cookies是否为空，因为iterator会在循环前执行hasNext；
         * 但是，最好判断一下，这样如果cookies为null的话，我们就可以直接返回，不需要执行循环，
         * 这样就不需要平白的创建一个iterator对象并执行一次hasNext。
         *
         * 下边的判断也可以换成这样CollectionUtils.isEmpty(Arrays.asList(cookies));
         */
        if(cookies == null || cookies.length == 0){
            return null;
        }

        for(Cookie cookie : cookies){
            if(cookie.getName().equals(key)){
                return cookie.getValue();
            }
        }
        return null;
    }

    /**
     * 清空指定cookie
     * 值得注意的是，清空cookie不是只将相应的cookie的value置空即可，其他信息依旧设置，
     * 最后加在响应头中，去覆盖浏览器端的相应的cookie
     */
    public static void removeCookie(String name,
                                    String domain,
                                    String path,
                                    HttpServletResponse response){
        Cookie cookie = new Cookie(name, null);

        cookie.setMaxAge(0);
        cookie.setDomain(domain);
        cookie.setPath(path);

        response.addCookie(cookie);
    }
}
```

注意：

- 在使用response将cookie写入响应头后，我们可以在浏览器查看响应头中的Set-Cookie信息，每添加一条cookie，就会有一个Set-Cookie；
- 在使用request从请求头中获取cookie的时候，我们可以在浏览器查看请求头中的Cookie信息，所有cookie信息会以key=value的形式、多个key-value对之间以分号隔开的形式存成一条。
- 在删除cookie的时候，一定要注意我们不是只将相应的cookie的value置空即可，其他信息依旧设置，最后加在响应头中，去覆盖浏览器端的相应的

cookie

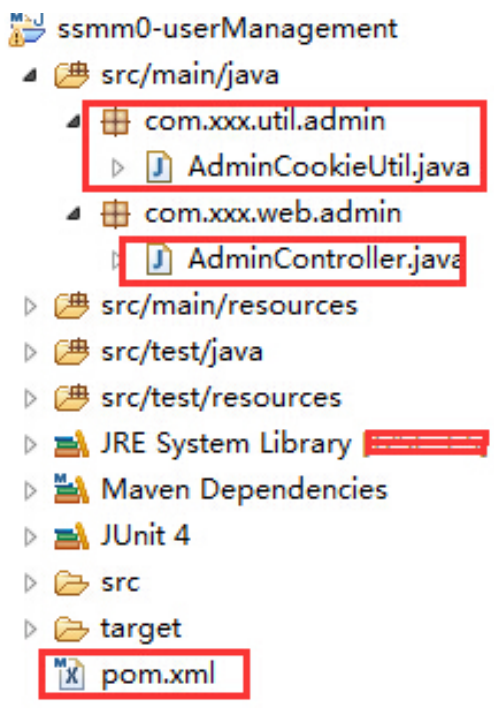
前两条注意点可以查看后边的截图来验证。

2.1.4、Admin：

```
package com.xxx.model.userManagement;

import com.alibaba.fastjson.JSON;

/**
 * 管理员
 */
public class Admin {
    private int id;
    private String username;
    private String password;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    //将json串转为Admin
    public static Admin parseJsonToAdmin(String jsonStr){
        try {
            return JSON.parseObject(jsonStr, Admin.class);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    //将当前实例转化为json串
    public String toJson(){
        return JSON.toJSONString(this);
    }
}
```

说明：在Admin中，增加了两个方法，一个是将当前实例转化为json串，一个是将json串转化为Admin；这是因为**cookie的传输只能传递字符串而不能传递对象。**

2.2、ssmm0-userManagement：



2.2.1、pom.xml

注意：这个类没有改动，之所以列出来，是要提醒去注意compile的传递性与provided的不可传递性

2.2.2、AdminCookieUtil

```
package com.xxx.util.admin;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.codec.binary.Base64;
import org.apache.commons.lang.StringUtils;

import com.xxx.model.userManagement.Admin;
import com.xxx.util.AESUtil;
import com.xxx.util.CookieUtil;

/**
 * Admin的cookie操作类
 */
public class AdminCookieUtil {
    private static final String COOKILE_NAME = "allinfo";
    private static final String USER_NAME = "username";
    private static final String DOMAIN = "";//when working on localhost the cookie-domain must be set to "" or NULL or FALSE
    private static final String PATH = "/";
    private static final int EXPIRY = -1;//关闭浏览器，则cookie消失

    private static final String ENCRYPT_KEY = "gEfcsJx8VUT406qI4r6/3104noOzI/YAaS98cToY+nI=";//加解密密钥

    public static void addLoginCookie(Admin admin, HttpServletResponse response){
        try{
            CookieUtil.addCookie(COOKILE_NAME, AESUtil.encrypt(admin.toJson(), Base64.decodeBase64(ENCRYPT_KEY)), DOMAIN, PATH, EXPIRY, response);
            CookieUtil.addCookie(USER_NAME, admin.getUsername(), DOMAIN, PATH, EXPIRY, response);
        }catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Admin getLoginCookie(HttpServletRequest request){
        String json = CookieUtil.getCookie(request, COOKILE_NAME);
        if(StringUtils.isNotBlank(json)){
            try{
```

```
38                 return Admin.parseJsonToAdmin(AESUtil.decrypt(json, Base64.decodeBase64(ENCRYPT_KEY)));
39             }catch (Exception e) {
40                 e.printStackTrace();
41                 return null;
42             }
43         }
44         return null;
45     }
46 }
```

注意点：

- 在localhost下设置的域，必须是""或null或false

步骤：（这就是cookie的使用流程）

- 登录成功后，写两个cookie，一个是username=value，一个是allinfo=一个加密串，这个加密串是先将admin实例转化为json串，然后通过AES进行加密。
- 在执行findAdmin方法时，假设前提是需要用户处于登录状态（即存在cookie），这个时候读cookie，先从request中获取所有cookie，然后遍历这些cookie，找出其中是否存在name为allinfo的cookie，如果没有，没登录，如果有，就登录了，将allinfo的cookie的value先解密再转化为Admin，之后使用该实例做一些事儿。

2.2.3、AdminController

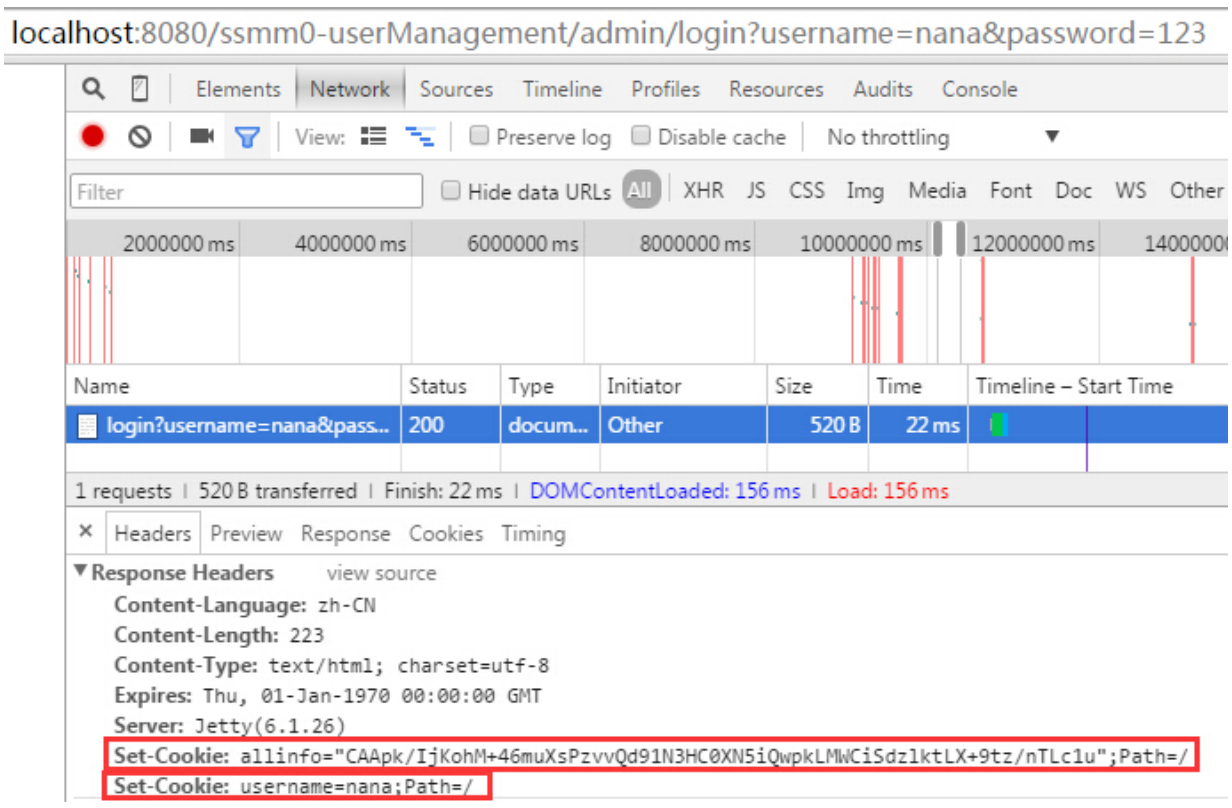
```
 1 package com.xxx.web.admin;
 2
 3 import java.util.List;
 4
 5 import javax.servlet.http.HttpServletRequest;
 6 import javax.servlet.http.HttpServletResponse;
 7
 8 import org.springframework.beans.factory.annotation.Autowired;
 9 import org.springframework.stereotype.Controller;
10 import org.springframework.web.bind.annotation.RequestMapping;
11 import org.springframework.web.bind.annotation.RequestParam;
12 import org.springframework.web.bind.annotation.ResponseBody;
13 import org.springframework.web.servlet.ModelAndView;
14
15 import com.xxx.model.userManagement.Admin;
16 import com.xxx.service.userManagement.AdminService;
17 import com.xxx.util.admin.AdminCookieUtil;
18
19 /**
20  * adminController
21  */
22 @Controller
23 @RequestMapping("/admin")
24 public class AdminController {
25
26     @Autowired
27     private AdminService adminService;
28
29     /**
30      * 管理员注册
31      */
32     @ResponseBody
33     @RequestMapping("/register")
34     public boolean register(@RequestParam("username") String username,
35                             @RequestParam("password") String password){
36         Admin admin = new Admin();
37         admin.setUsername(username);
38         admin.setPassword(password);
39
```

```
40          boolean isRegisterSuccess = adminService.register(admin);
41
42          return isRegisterSuccess;
43      }
44
45      /**
46       * 管理员登录
47       */
48      @RequestMapping("/login")
49      public ModelAndView login(@RequestParam("username") String username,
50                                @RequestParam("password") String password,
51                                HttpServletResponse response){
52
53
54          Admin admin = adminService.login(username, password);
55
56          ModelAndView modelAndView = new ModelAndView();
57          if(admin == null){
58              modelAndView.addObject("message", "用户不存在或者密码错误！请重新输入");
59              modelAndView.setViewName("error");
60          }else{
61              modelAndView.addObject("admin", admin);
62              modelAndView.setViewName("userinfo");
63              /*
64               * 这为什么不直接传一个username，而传了一个admin，
65               * 是因为在实际开发中，你传过去的信息可能不只是username，还有用户手机号、地址等等
66               */
67              AdminCookieUtil.addLoginCookie(admin, response);
68          }
69
70          return modelAndView;
71      }
72
73      /****************************mybatis xml方式解决的问题*****************************/
74      /**
75       * 根据username或password查找List<Admin>
76       */
77      @ResponseBody
78      @RequestMapping("/findAdmin")
79      public List<Admin> findAdmin(@RequestParam(value="username",required=false) String username,
80                                   @RequestParam(value="password",required=false) String password,
81                                   @RequestParam("start") int start,
82                                   @RequestParam("limit") int limit,
83                                   HttpServletRequest request){
84          Admin admin = AdminCookieUtil.getLoginCookie(request);
85          if(admin == null){//未登录
86              return null;
87          }
88          List<Admin> adminList = adminService.findAdmin(username, password, start, limit);
89          return adminList;
90      }
91
92      /**
93       * 插入一个用户并返回主键
94       * 注意：get请求也会自动装配（即将前台传入的username和password传入admin）
95       */
96      @ResponseBody
97      @RequestMapping("/insert")
98      public Admin insertAdminWithBackId(Admin admin){
99          return adminService.insertAdminWithBackId(admin);
100     }
101 }
```

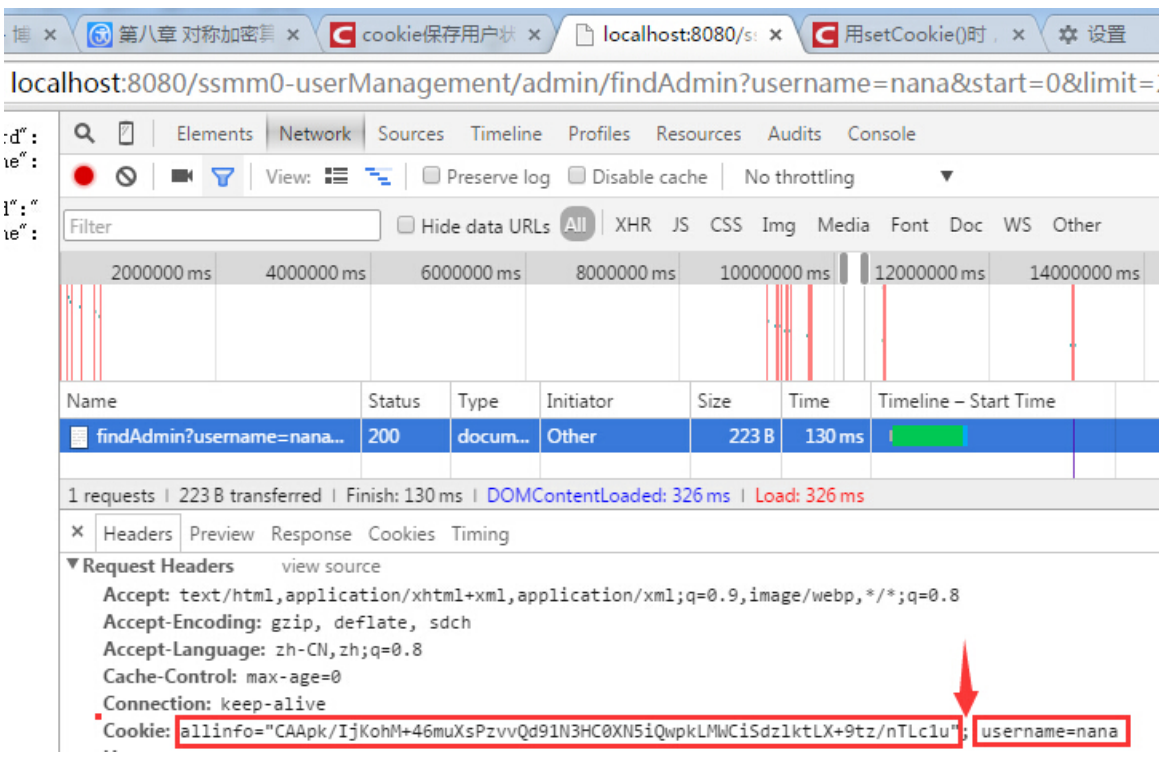
说明，这个类只修改了两个方法login()和findA dmin()。

测试：

- 向浏览器写入cookie



- 从浏览器读cookie



注意：

- 上述我们发了两个cookie，其中username的value没有加密，是因为这个value不是私密数据且我们在前台会直接使用，这样的话，省去了js解密的过程。
- 对于allinfo的加密是为了在浏览器端不让用户将cookie看的那么明显，保护一些私密数据；或者在整个传输过程中，即使被窃听者获取了，也难以知道其中的cookie值。但是，如果窃听者获取了allinfo这个cookie后，就可以使用这个cookie模拟登录然后做一些操作，这个不知道cookie机制会怎样解决？**（答案见最下方）**
- 对于allinfo的加密而言，仅使用AES加密事实上也不太合适，因为如果窃听者窃听到你的cookie后，对其进行篡改，当请求头再将这些信息传递过来的时候，可能经过json转化后就会是另一个Admin的信息了，如果需要防止这种情况发生，需要同时对cookie值进行消息摘要加密了（具体的消息摘要加密可以参照"**Java加密与解密**"系列博客），当然，在绝大多数情况下，如果窃听者不知道你的cookie值得话，如果他对这个值进行了随意的篡改，那么将来在将这个值进行解密后，对其转化成json的过程中就会抛出异常，因为解密后的json串很可能就不符合json的格式了，所以**绝大多数情况下，仅适用于AES是可行的。**
- 对于cookie的name实际上也应该加密，加密后浏览器端或窃听者截到的cookie他就不知道是什么意思了，这个加密非常简单，直接在线下使用SHA256出一个字符串或者就使用上边所讲的AES的生成key的方法生成一个就好，然后写在name中，代码再列一遍：

```
String key = Base64.encodeBase64String(AESUtil.getKey());
```

- cookie不可以在多种浏览器之间共享，因为每个浏览器存cookie路径不是一样的
- 同一种浏览器，多个标签页共享的话，需要再生成cookie的时候添加cookie的有效期；否则cookie为会话cookie，这种客户端是不会把cookie存到硬盘上的，其他标签也无法获取到cookie

总结：

- 如果安全措施搞得好+cookie数量不多+cookie的总大小对带宽的占用可以接受，使用cookie（对于安全措施这一块，cookie被劫持的可能性一般不大，如果不是非常敏感的权限，可以使用记住我这些功能，如果对于敏感权限，例如金钱操作，就强制重新登录，类似于shiro）
- 对于浏览器禁用cookie的事情，基本可以不考虑，绝大多数现代浏览器都不会禁用cookie。

疑问：

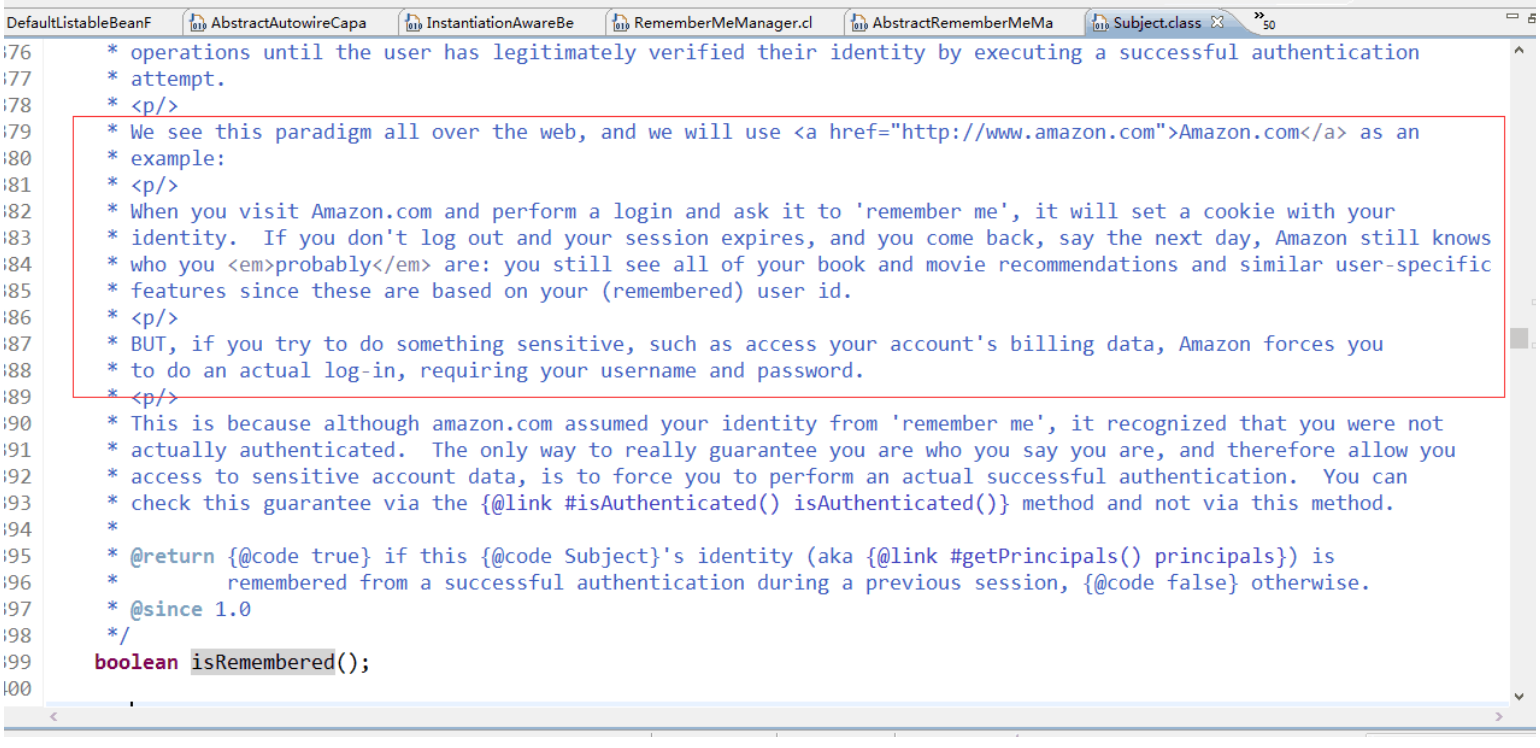
- 如果被窃听者劫取到你的cookie，即使你对这些cookie做了加密，使其看不到cookie中的信息，但是其也可以通过该cookie模拟登录行为，然后获得一些权限，执行一些操作。（这是个疑问，记住我这个功能就是这样做的，怎样做到安全的呢？）

解答：方案

1、使用权限框架shiro

注意：shiroshiro保存了你的账户名（编码）和可记住密码的有效时间到cookie里，下次会把账户名带过来，可以允许执行一些非敏感操作。根据你后台设定的权限；当然对于敏感权限的话，一定要重新登录才行。（当然，由于还是存在cookie里，可能还是存在会被劫持的情况）

具体看下边图片：（"记住我"的功能）



2、https（即使被劫取到，也不能重传）

这个是可以解决上边问题的，但是https没闹懂。

3、IP变动

1）客户端登陆的时候，记录客户端IP到数据库

2）下次登录从request中获取IP并与前一次记录的IP作对比，如果IP没有发生变化，则得直接进入已登录状态了；如果IP发生了变化，则重新登录，登录之后，修改IP

3）之后的过程如上

当然，对于以上这种方式，还有一种改进，就是将用户登录的IP放入缓存（当然可以指定缓存时间），这样每次就可以从缓存中查IP了。

但是这种方式有个问题，就是说同一台电脑换一个网络环境，这时候IP就变了，就得重新登陆了。

4、数字签名算法

数字签名的三个作用中的一个就是"认证数据来源"，感觉这会是一个思路，但是具体怎么实现还没想好。如果大家有想好的，帮指点一下！

数字签名算法的具体实现见"Java加密与解密"《第十四章 数字签名算法--RSA》

有关于session的细节与解决方案，以后再说！