一、Shiro

> 为什么要用shiro：
>
> 1.项目中的密码是否可以明文存储?
>
> 2.是否任意访客，无论是否登录都可以访问任何功能?
>
> 3.项目中的各种功能操作，是否是所有用户都可以随意使用?
>
> 综上，当项目中的某些功能被使用时，需要进行安全校验，进而保证整个系统的运行秩序。

1.1 Shiro是什么

- Apache Shiro 是 Java 的一个安全（权限）框架。
  Shiro 可以轻松的完成：身份认证、授权、加密、会话管理等功能
- Shiro 可以非常容易的开发出足够好的应用，其不仅可以用在JavaSE 环境，也可以用在 JavaEE 环境。
  功能强大且易用，可以快速轻松地保护任何应用程序 （ 从最小的移动应用程序到最大的Web和企业应用程序。）
- 方便的与Web 集成和搭建缓存。
- 下载：http://shiro.apache.org/

1.2 功能简介

- 基本功能点如下图所示：

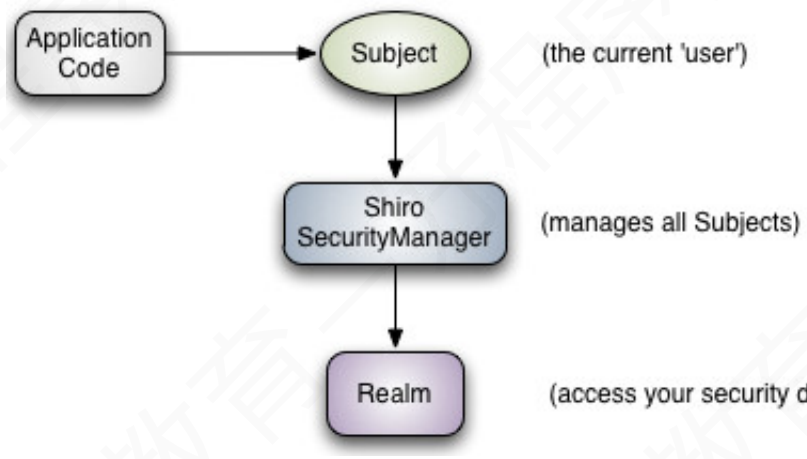
- Authentication
  身份认证/登录，验证用户是不是拥有相应的身份；
- Authorization
  授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能进行什么操作
  如：验证某个用户是否拥有某个角色。或者细粒度的验证某个用户对某个资源是否具有某个权限；
- Session Manager
  会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有
  信息都在会话中；会话可以是普通 JavaSE 环境，也可以是 Web 环境的；
- Cryptography
  加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；
- Web Support
  Web 支持，可以非常容易的集成到Web 环境；
- Caching
  缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；
- Remember Me
  记住我，这个是非常常见的功能，即一次登录后，下次再来的话可以立即知道你是哪个用户

。。。。

二、Shiro 架构

2.1 工作流程

> shiro 运行流程中，3个核心的组件:
>
> `Subject、SecutiryManager、Realm`
>
> shiro使用中，必须有的3个概念！！



- Subject
  安全校验中，最常见的问题是"当前用户是谁?" "当前用户是否有权做x事?",所以考虑安全校验过程最自然的方式就是基于当前用户。Subject 代表了当前“用户”，
  应用代码直接交互的对象是 Subject，只要得到Subject对象马上可以做绝大多数的shiro操作。
  也就是说 Shiro 的对外API 核心就是 Subject。
  Subject 会将所有交互都会委托给 SecurityManager。
  ==Subject是安全管理中直接操作的对象==

- SecurityManager
  安全管理器；即所有与安全有关的操作都会与SecurityManager 交互；
  且其管理着所有 Subject；它是 Shiro的核心，
  它负责与 Shiro 的其他组件进行交互，它相当于 SpringMVC 中DispatcherServlet 的角色

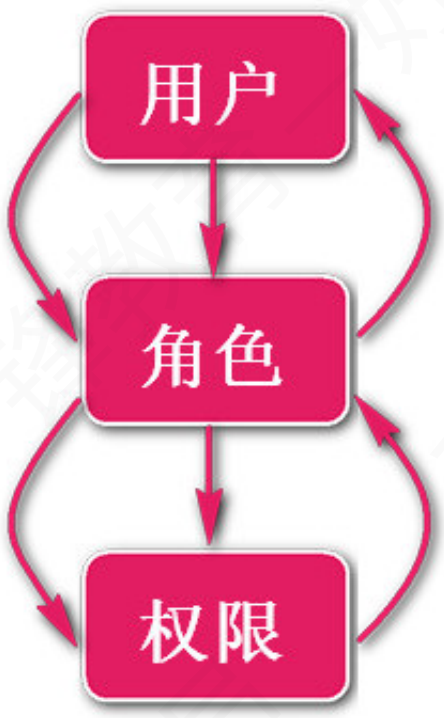
- Realm
  Shiro 从 Realm 获取安全数据（如用户、角色、权限），就是说SecurityManager 要验证用户身份，那么它需要从 Realm 获取相应的用户进行比较以确定用户身份是否合法；
  也需要从 Realm 得到用户相应的角色/权限进行验证用户是否能进行操作；
  可以把 Realm 看成 DAO，(数据访问入口)

2.2 RBAC模型

> Role Base Access Controll 基于角色的访问控制

> shiro采用的安全管理 模型

> 模型中3个主体：用户、角色、权限
>
> 每个角色可以有多个权限，每个权限可以分配给多个角色
>
> 每个用户可以有多个角色，每个角色可以分配给多个用户
>
> *两个多对多*



则权限访问控制，做的事是：

1. 身份校验：判断是否为合法用户
2. 权限校验：用户要做做某件事或使用某些资源，必须要拥有某角色，或必须拥有某权限

在访问控制管理过程中，是要对项目中的资源(功能，数据，页面元素等)的访问、使用进行安全管理。

1> 首先照旧记录用户信息

2> 然后制定角色

如 “千锋学员”，“千锋讲师”，“保洁员”

3> 然后会对"资源"制定权限：即能对资源做的所有操作

如 "教室"-资源，"进入教室"，“在教室学习”，就是该资源的两个权限。

4> 然后将权限分配给不同角色

如 “进入教室” 分配给 “千锋学员”，“千锋讲师”，“保洁员” 三种角色

“在教室学习” 分配给 “千锋学员” 角色

5> 最后将角色分配给具体用户

如 “张三” 报名后分配 “千锋学员” 角色

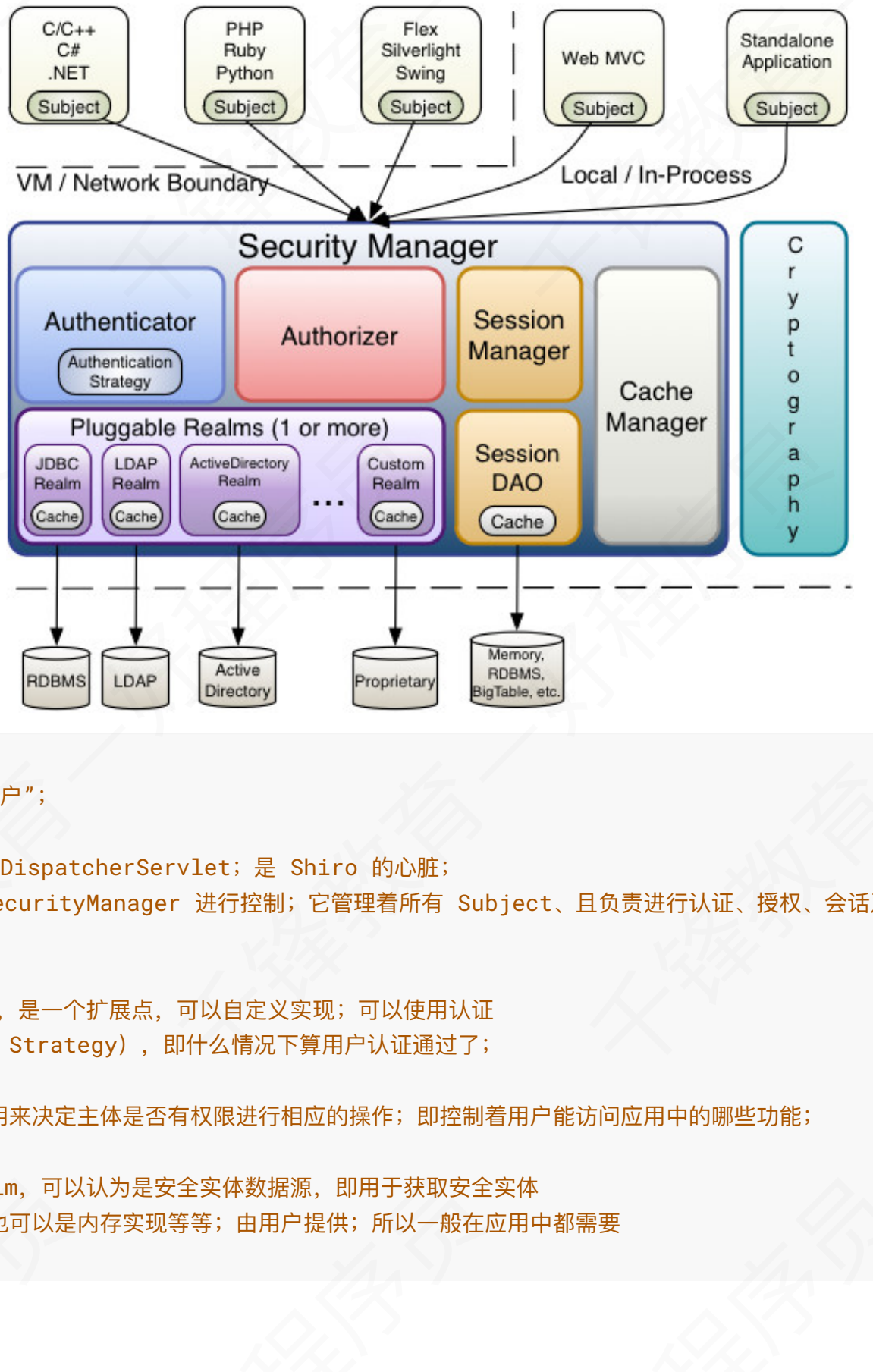
“李四” 入职千锋分配 "千锋保洁员" 角色

6> 如此完成对 用户(李四，张三)使用某资源的访问控制

则：张三能不能进教室？李四能不能在教室学习？

2.3 架构(了解)

简单了解，在对shiro有了完整的应用体验后可以重点了解！



- Subject
  任何可以与应用交互的“用户”；
- SecurityManager
  相当于SpringMVC 中的 DispatcherServlet；是 Shiro 的心脏；
  所有具体的交互都通过 SecurityManager 进行控制；它管理着所有 Subject、且负责进行认证、授权、会话及缓存的管理。
- Authenticator
  负责 Subject 身份认证，是一个扩展点，可以自定义实现；可以使用认证策略（Authentication Strategy），即什么情况下算用户认证通过了；
- Authorizer
  授权器、即访问控制器，用来决定主体是否有权限进行相应的操作；即控制着用户能访问应用中的哪些功能；
- Realm
  可以有 1 个或多个 Realm，可以认为是安全实体数据源，即用于获取安全实体的；可以是JDBC 实现，也可以是内存实现等等；由用户提供；所以一般在应用中都需要实现自己的 Realm；

- SessionManager
  管理 Session 生命周期的组件；而 Shiro 并不仅仅可以用在 Web环境，也可以用在如普通的 JavaSE 环境
- CacheManager
  缓存控制器，来管理如用户、角色、权限等的缓存的；因为这些数据基本上很少改变，
  放到缓存中后可以提高访问的性能
- Cryptography
  密码模块，Shiro 提供了一些常见的加密组件用于如密码加密/解密。

三、HelloWorld

3.1 pom 文件

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-core</artifactId>
    <version>1.4.0</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.12</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.12</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
</dependency>
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.2</version>
</dependency>
```

> ops : require JDK 1.6+ and Maven 3.0.3+

3.2 配置

> 创建一个 `shiro.ini` 文件

```
#定义用户信息
#格式：用户名=密码,角色1,角色2,....
[users]
zhangsan=123,admin
lisi=456,manager,seller
wangwu=789,clerk
# ---------------------------------------------------------------------------
# 角色及其权限信息
# 预定权限: user:query
#          user:detail:query
#          user:update
#          user:delete
#          user:insert
#          order:update
#          ....
[roles]
# admin 拥有所有权限,用*表示
admin=*
# clerk 只有查询权限
clerk=user:query,user:detail:query
# manager 有 user 的所有权限
manager=user:*
```

3.3 代码

> - 通过在ini中设置的 身份信息、角色、权限，做安全管理
> - 1.身份认证: subject.login(token)
> - 2.是否认证身份判断：subject.isAuthenticated()
> - 3.角色校验：subject.hasRole(String:role); subject.hasRoles(List:roles)
> - 4.权限校验：subject.isPermitted(String:perm); subject.isPermittedAll(List:perms)

```
// 定义main函数测试效果
// 创建 "SecurityFactory",加载ini配置,并通过它创建SecurityManager
Factory<SecurityManager> factory = new IniSecurityManagerFactory("classpath:shiro.ini");
SecurityManager securityManager = factory.getInstance();

// 将SecurityManager托管到SecurityUtils工具类中(ops:之后可以不必关心SecurityManager)
SecurityUtils.setSecurityManager(securityManager);

// 获得Subject，通过subject可以执行shiro的相关功能操作(身份认证或权限校验等)
Subject currentUser = SecurityUtils.getSubject();

// 身份认证( 类似登录逻辑 )
if (!currentUser.isAuthenticated()) {//判断是否已经登录
    //如果未登录，则封装一个token，其中包含 用户名和密码
    UsernamePasswordToken token = new UsernamePasswordToken("zhangsan", "123");
    try {
        //将token传入login方法，进行身份认证 (判断用户名和密码是否正确)
        currentUser.login(token);//如果失败则会抛出异常
    } catch (UnknownAccountException uae) {//用户不存在
        System.out.println("There is no user with username of " + token.getPrincipal());
    } catch (IncorrectCredentialsException ice) {//密码错误
        System.out.println("Password for account " + token.getPrincipal() + " was incorrect!");
    } catch (LockedAccountException lae) {//账户冻结
        System.out.println("The account for username " + token.getPrincipal() + " is locked.  "
                           +"Please contact your administrator to unlock it.");
    }catch (AuthenticationException ae) {//其他认证异常
        }
}

// 认证成功则用户信息会存入 currentUser (Subject)
System.out.println("User [" + currentUser.getPrincipal() + "] logged in successfully.");

// 可以进一步进行角色校验 和 权限校验
if (currentUser.hasRole("admin")) { //校验角色
    System.out.println("hello,boss");
} else {
    System.out.println("Hello, you");
}
if (currentUser.isPermitted("user:update")) { //校验权限
    System.out.println("you can update user");
} else {
    System.out.println("Sorry, you can not update.");
}

// 用户退出，会清除用户状态
currentUser.logout();

// System.exit(0);
```

##3.4 权限规则

> 最常用的权限标识: **【资源 : 操作】**

```
1> user:query , user:insert , order:delete , menu:show
    【:】 作为分隔符，分隔资源和操作【资源:操作】
    【,】 作为分隔，分隔多个权限【权限1,权限2,权限3】
2> user:* , *:query
    【*】 作为通配符，代表所有操作、资源
    【user:* 即user的所有操作】
    【*:query 即所有资源的查询操作】
3> *
```

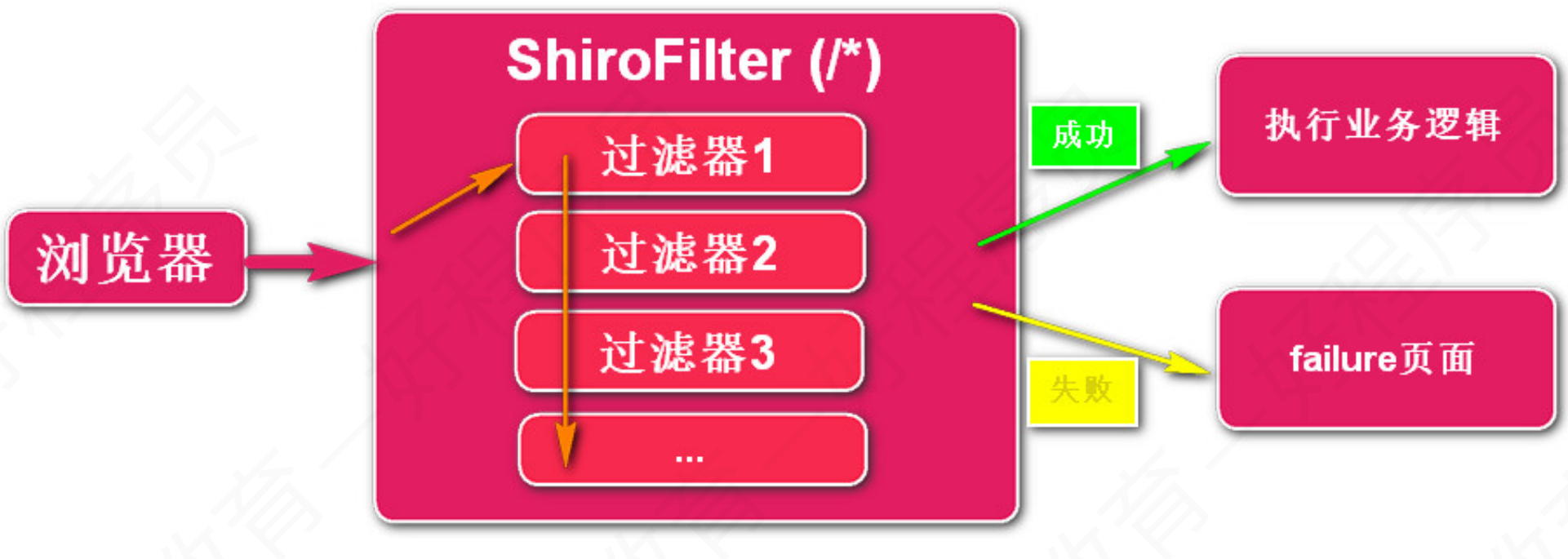
```
    【代表一切资源的一切权限 = 最高权限】
4> 细节:
  1)【user:*  可以匹配 user:xx, user:xx:xxx】
      【*:query 只可以匹配 xx:query,不能匹配 xx:xx:query. 除非 *:*:query】
    2)【user:update,user:insert】 可以简写为 【"user:update,insert"】
      [roles]
      manager1=user:query,user:update,user:insert
      manager2="user:query,update,insert" #注意要加引号
      #如上manager1和manager2权限等价
      #subject.isPermittedAll("user:update","user:insert","user:query")
```

实例级权限标识: 【**资源： 操作： 实例**】，粒度细化到具体某个资源实例

```
1> user:update:1 , user:delete:1
   # 对用户1可以update,对用户1可以delete
2> "user:update,delete:1" #和上面等价
   # subject.isPermittedAll("user:update,delete:1","user:update:1","user:delete:1")
3> user:*:1 , user:update:* , user:*:*
```

四、与Web 集成

与web项目集成后，shiro的工作模式如下：



如上：ShiroFilter拦截所有请求，对于请求做访问控制

如请求对应的功能是否需要有 认证的身份，是否需要某种角色，是否需要某种权限。

1> 如果没有做 身份认证，则将请求强制跳转到登录页面。

如果没有充分的角色或权限，则将请求跳转到权限不足的页面。

2> 如果校验成功，则执行请求的业务逻辑

4.1 pom

```
<!-- ============ Servlet ============ -->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.1</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>
```

```
<!-- ============== SpringMVC ============== -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.3.6.RELEASE</version>
</dependency>

<!-- ============ shiro ============ -->
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-core</artifactId>
    <version>1.4.0</version>
</dependency>
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-web</artifactId>
    <version>1.4.0</version>
</dependency>

<!-- ============ log ============ -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.12</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.12</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
</dependency>
```

4.2 代码： Servlet

> 定义一些Servlet 或 springMVC的controller，发送请求，验证shiro的验证

```
@Controller
@RequestMapping("/user")
public class ShiroController {
    @RequestMapping("/delete")
    public String deleteUser(){//访问此删除功能时要先经过shiro的安全校验
        System.out.println("delete User");
        return "forward:/xx.jsp";
    }
    @RequestMapping("/update")
    public String updateUser(){//访问此更新功能时要先经过shiro的安全校验
        System.out.println("update User");
        return "forward:/xx.jsp";
    }
    @RequestMapping("/insert")
    public String insertUser(){//访问此增加功能时要先经过shiro的安全校验
        System.out.println("insert User");
        return "forward:/xx.jsp";
    }
    @RequestMapping("/login/page")
    public String login(){
      System.out.println("goto login.jsp");
        return "forward:/login.jsp";
    }
    @RequestMapping("/login/logic")
    public String login(String username,String password){//登录功能不能被shiro校验，否则永不能登录
        try{
            Subject subject = SecurityUtils.getSubject();
            subject.login(new UsernamePasswordToken(username,password));
            String uname = (String)subject.getPrincipal();
            System.out.println("uname:"+uname);
```

```
        }catch (AuthenticationException e){
            e.printStackTrace();
            return "redirect:/login.jsp";
        }
        return "forward:/success.jsp";
    }
}
```

> springMVC的相关配置照旧。

4.3 配置

4.3.1 web.xml

> 安装 **ShiroFilter！！！**

```
<!-- 接收所有请求，以通过请求路径 识别是否需要 安全校验，如果需要则触发安全校验
     做访问校验时，会遍历过滤器链。(链中包含shiro.ini中urls内使用的过滤器)

     会通过ThreadContext在当前线程中绑定一个subject和SecurityManager，供请求内使用
     可以通过SecurityUtils.getSubject()获得Subject
-->
<filter>
    <filter-name>shiroFilter</filter-name>
    <filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- 在项目启动时，加载web-info 或 classpath下的 shiro.ini ，并构建WebSecurityManager。
     构建所有配置中使用的过滤器链(anon,authc等)，ShiroFilter会获取此过滤器链
-->
<listener>
    <listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>
</listener>
<!-- 自定义ini文件名称和位置
<context-param>
    <param-name>shiroConfigLocations</param-name>
    <param-value>classpath:shiro9.ini</param-value>
</context-param>-->
<!-- springMVC的配置照旧，此处省略...-->
```

4.3.2 shiro.ini

> 新增两个部分(section), [main] 和 [urls]

```
[users]
zhangsan=123,admin
lisi=456,manager,seller
wangwu=789,clerk

[roles]
admin=*
clerk=user:query,user:detail:query
manager=user:*

[main]
#没有身份认证时，跳转地址
shiro.loginUrl = /user/login/page
#角色或权限校验不通过时，跳转地址
shiro.unauthorizedUrl=/author/error
#登出后的跳转地址,回首页
shiro.redirectUrl=/

[urls]
# 如下格式："访问路径 = 过滤器"
#【1.ant路径：？ * ** 细节如下】
```

```
# /user/login/page , /user/login/logic 是普通路径
# /user/* 代表/user后还有一级任意路径 ： /user/a , /user/b , /user/c , /user/xxxxxxxxxx
# /user/** 代表/user后还有任意多级任意路径: /user/a , /user/a/b/c , /user/xxxx/xxxxxx/xxxxx
# /user/hello? 代表hello后还有一个任意字符: /user/helloa , /user/hellob , /user/hellox

#【2.过滤器，细节如下】
# anon => 不需要身份认证
# authc => 指定路径的访问，会验证是否已经认证身份，如果没有则会强制转发到 最上面配置的loginUrl上
#         ( ops:登录逻辑本身千万不要被认证拦截，否则无法登录 )
# logout => 访问指定的路径，可以登出,不用定义handler。
# roles["manager","seller"] => 指定路径的访问需要subject有这两个角色
# perms["user:update","user:delete"] => 指定路径的访问需要subject有这两个权限
/user/login/page = anon
/user/login/logic = anon
/user/query = authc
/user/update = authc,roles["manager","seller"]
/user/delete = authc, perms["user:update","user:delete"]
/user/logout = logout
#其余路径都需要身份认证【用此路径需谨慎】
/** = authc
#【3.注意】
# url的匹配，是从上到下匹配，一旦找到可以匹配的则停止，所以，通配范围大的url要往后放，
# 如"/user/delete" 和 "/user/**"
```

4.3.3 其他默认过滤器

过滤器名称	过滤器类	描述	例子
anon	org.apache.shiro.web.filter.authc.AnonymousFilter	没有参数，表示可以匿名使用	/admins/**=anon
authc	org.apache.shiro.web.filter.authc.FormAuthenticationFilter	没有参数，表示需要认证(登录)才能使用	/user/**=authc
authcBasic	org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter	没有参数，表示需通过httpBasic 验证，如果不通过，跳转回登录页面	/user/**=authcBasic
logout	org.apache.shiro.web.filter.authc.LogoutFilter	注销登录的时候，完成一定的功能：任何现有的Session 都将会失效，而且任何身份都将会失去关联（在Web 应用程序中，RememberMe cookie 也将被删除）	
noSessionCreation	org.apache.shiro.web.filter.session.NoSessionCreationFilter	阻止在请求期间创建新的会话。以保证无状态的体验	
perms	org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter	参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割。当有多个参数时必须每个参数都通过才通过，想当于isPermitedAll()方法。	/admins/**=perms[user:add:*] /admins/user/**=perms["user:add:*,user:modify:*"]
port	org.apache.shiro.web.filter.authz.PortFilter	指定请求访问的端口。如果不匹配则跳转到登录页面	/admins/**=port[8081]
rest	org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter	根据请求的方法	/admins/user/**=perms[user:method],其中method为post，get，delete等
roles	org.apache.shiro.web.filter.authz.RolesAuthorizationFilter	角色过滤器，判断当前用户是否指定角色。参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，当有多个参数时，每个参数通过才算通过，相当于hasAllRoles()方法	admins/**=roles["admin,guest"]
ssl	org.apache.shiro.web.filter.authz.SslFilter	没有参数，表示安全的url请求，协议为 https	
user	org.apache.shiro.web.filter.authc.UserFilter	没有参数表示必须存在用户	


```
public enum DefaultFilter {
  // 定义了诸多  过滤器
    anon(AnonymousFilter.class),
    authc(FormAuthenticationFilter.class),
    authcBasic(BasicHttpAuthenticationFilter.class),
    logout(LogoutFilter.class),
    noSessionCreation(NoSessionCreationFilter.class),
    perms(PermissionsAuthorizationFilter.class),
    port(PortFilter.class),
    rest(HttpMethodPermissionFilter.class),
    roles(RolesAuthorizationFilter.class),
    ssl(SslFilter.class),
    user(UserFilter.class);
    ....
}
```

4.4 JSP

> 定义 Controller中和 shiro.ini中需要的jsp

4.5 总结

> 通过ShiroFilter和定义在shiro.ini中的配置信息，即可在项目接收用户访问时，进行身份，角色，权限进行访问控制啦！！！！

五、shiro标签

> shiro提供了很多标签，用于在jsp中做安全校验。
>
> 完成对页面元素的访问控制

5.1 导入shiro标签库

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="shiro" uri="http://shiro.apache.org/tags"%>
<html>
    ....
</html>
```

5.2 身份认证

> `<shiro:authenticated>` 已登录
>
> `<shiro:user>` 已登录或记住我
>
> `<shiro:guest>` 游客(未登录 且 未记住我)
>
> `<shiro:notAuthenticated>` 未登录
>
> `<shiro:principal>` 获取用户身份信息

```
<shiro:authenticated>
    欢迎您，<shiro:principal/>
</shiro:authenticated>

<shiro:user> <!-- 常用，包含已登录 且配合记住我，用户体验好 -->
    欢迎您,<shiro:principal/>
</shiro:user>

<shiro:guest>
    欢迎您，未登录，请<a href="<c:url value="/user/login/page"/>">登录</a>
</shiro:guest>

<shiro:notAuthenticated>
    您尚未登录(记住我也算在未登录中)
</shiro:notAuthenticated>
```

5.3 角色校验

> `<shiro:hasAnyRoles name="admin,manager">` 是其中任何一种角色
>
> `<shiro:hasRole name="admin">` 是指定角色
>
> `<shiro:lacksRole name="admin">` 不是指定角色

```
<table>
    <tr>
        <td>id</td>
        <td>name</td>
```

```
        <td>operation</td>
    </tr>
    <tr>
        <td>001</td>
        <td>张三</td>
        <td>
            <shiro:hasAnyRoles name="admin,manager">
                <a href="#" style="text-decoration:none">详情</a>
            </shiro:hasAnyRoles>
            <shiro:hasRole name="admin">
                <a href="#" style="text-decoration: none">删除</a>
            </shiro:hasRole>
            <shiro:lacksRole name="admin">
                <a href="#" style="text-decoration: none">点击升级</a>
            </shiro:lacksRole>
        </td>
    </tr>
</table>
```

5.4 权限校验

> `<shiro:hasPermission name="user:delete">`
> 有指定权限
>
> `<shiro:lacksPermission name="user:delete">`
> 缺失指定权限

```
...
<td>
    <a href="#" style="text-decoration:none">查看详情</a>
    <shiro:hasPermission name="user:delete">
        <a href="#" style="text-decoration: none">删除</a>
    </shiro:hasPermission>
    <shiro:lacksPermission name="user:delete">
        <a href="#" style="text-decoration: none" >无权删除</a>
    </shiro:lacksPermission>
</td>
...
```

5.5 自定义标签(了解)

> jsp中允许自定义标签，所以可以根据需求 自定义一些shiro标签。

5.5.1 定义标签类

```
public class MyAllRoleTag extends RoleTag {
    // jsp中使用: <xxx:xx name="角色参数1,角色参数2,..."/>
    private String name;//存储传入的角色参数
    @Override
    public String getName() {
        return name;
    }
    @Override
    public void setName(String name) {
        this.name = name;
    }

    @Override
    protected boolean showTagBody(String name) {
        System.out.println("验证角色:"+name);
        String[] roles = name.split(",");
        Subject subject = getSubject();
        for(String role:roles) {
            if(!subject.hasRole(role)){
                return false;
            }
        }
```

```
        return true;
    }
}
```

5.5.2 定义tld文件

> 要放在 `WEB-INF` 目录下，名称任意

```
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
        "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
    <tlib-version>1.1.2</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>zhj</short-name>
    <uri>http://zhj.apache.org/tags</uri>
    <description>Apache Shiro JSP Tag Library.</description>
    <tag>
        <!-- 标签名 <zhj:hasAllRoles .../> -->
        <name>hasAllRoles</name>
        <!-- 自定义Tag类路径 -->
        <tag-class>com.zhj.tag.MyAllRoleTag</tag-class>
        <body-content>JSP</body-content>
        <description>Displays body content only if the current Subject (user)
            'has' (implies) all the specified roles
        </description>
        <!-- 自定义Tag中属性名：name
           <zhj:hasAllRoles name="role1,role2"/>
        -->
        <attribute>
            <name>name</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

5.5.3 使用

```
<%@ taglib prefix="zhj" uri="http://zhj.apache.org/tags" %>
<!-- tld中定义的标签名：hasAllRoles，向Tag类中传递参数：“seller,manager” -->
<zhj:hasAllRoles name="seller,manager">
    有所有的角色2：manager，seller
</zhj:hasAllRoles>
```

六、自定义Realm

> **存在的问题：目前所有的 用户、角色、权限数据都在ini文件中，不利于管理。**
>
> **实际项目开发中这些信息，应该在数据库中。所以需要为这3类信息建表**

6.1 建表

> **用户表，角色表，权限表**

```
create table t_user(
  id int primary key auto_increment,
    username varchar(20) not null unique,
    password varchar(100) not null
)engine=innodb default charset=utf8;

create table t_role(
  id int primary key auto_increment,
  role_name varchar(50) not null unique,
  create_time timestamp not null
```

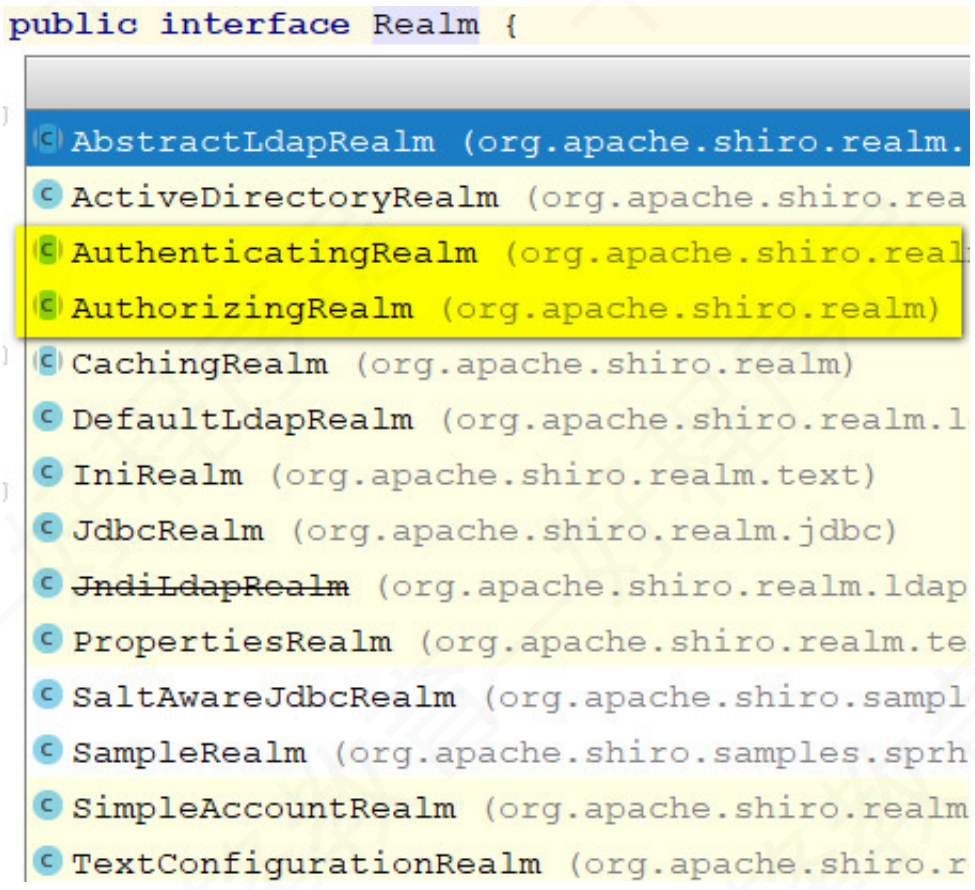
```
)engine=innodb default charset=utf8;

create table t_permission(
  id int primary key auto_increment,
  permission_name varchar(50) not null unique,
  create_time timestamp
)engine=innodb default charset=utf8;

create table t_user_role(
  id int primary key auto_increment,
  user_id int references t_user(id),
  role_id int references t_role(id),
  unique(user_id,role_id)
)engine=innodb default charset=utf8;

create table t_role_permission(
  id int primary key auto_increment,
  permission_id int references t_permission(id),
  role_id int references t_role(id),
  unique(permission_id,role_id)
)engine=innodb default charset=utf8;
```

6.2 自定义Realm

> Realm的职责是，为shiro加载 用户，角色，权限数据，以供shiro内部校验。
>
> 之前定义在ini中的数据，默认有IniRealm去加载。
>
> 现在库中的数据，需要自定义Realm去加载。
>
> **ops : 没必要在Realm中定义大量的查询数据的代码，可以为Realm定义好查询数据的DAO和Service。**

6.2.1 父类

> 如下是Realm接口的所有子类，其中IniRealm是默认的Realm，负责加载shiro.ini中的[users]和[roles]信息,当shiro需要用户，角色，权限信息时，会通过IniRealm获得。
>
> **自定义realm有两个父类可以选择：**
>
> **1> 如果realm只负责做身份认证，则可以继承：AuthenticatingRealm**
>
> **2> 如果realm要负责身份认证和权限校验，则可以继承：AuthorizingRealm**



6.2.2 定义Realm

```
public class MyRealm extends AuthorizingRealm {
    /**
     * 是否支持某种token
     * @param token
     * @return
     */
```

```java
@Override
public boolean supports(AuthenticationToken token) {
    System.out.println("is support in realm1");
    if(token instanceof UsernamePasswordToken){
        return true;
    }
    return false;
}

/**
 * 当subject.login()时，shiro会调用Realm的此方法做用户信息的查询，然后做校验
 * 职责：通过用户传递来的用户名查询用户表，获得用户信息
 * 返回值：将查到的用户信息（用户名+密码）封装在AuthenticationInfo对象中返回
 * 异常：如果没有查到用户可抛出用户不存在异常；如果用户被锁定可抛出用户被锁异常；或其它自定义异常.
 * @param token
 * @return
 * @throws AuthenticationException
 */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws
AuthenticationException {
        //获得用户名
        String username = (String) token.getPrincipal();
        System.out.println("user:"+username+" is authenticating~~");
        UserService userService =
            (UserService)ContextLoader.getCurrentWebApplicationContext().getBean("userService");
        //身份认证
        User user = userService.queryUser(username);
        System.out.println("user:"+user);
        /**
        如下代码可以省略，如果查询结果为空，直接返回null即可，
        shiro的后续流程已有类似判断逻辑，也会抛出UnknownAccountException
        if(user==null){//如果用户信息非法，则抛出异常
            System.out.println("用户不存在");
            throw new UnknownAccountException("username:"+username+"不存在");
        }
        **/
        //省略如上代码后，可以直接写：
        if(user == null){
            return null;
        }
        // 将 当前用户的认证信息存入 SimpleAuthenticationInfo 并返回
        // 注意此方法的本职工作就是查询用户的信息，所以查到后不用比对密码是否正确，那是shiro后续流程的职责。
        // 如果密码错误，shiro的后续流程中会抛出异常IncorrectCredentialsException
        return new SimpleAuthenticationInfo(user.getUsername(),user.getPassword(),getName());
        /**
        补充：可以在user表中增加一列，用于存储用户是否被锁定，则查询的User对象中会有是否锁定的属性
             如果发现锁定则可以在此方法中抛出异常：LockedAccountException,
        **/
}

/**
 * 当触发权限或角色校验时：subject.isPermitted() / subject.checkPermission();
 *                       subject.hasRole() / subject.checkRole() 等。
 * 此时需要数据库中的 权限和角色数据，shiro会调用Realm的此方法来查询
 * 角色和权限信息存入SimpleAuthorizationInfo对象
 * @param principals
 * @return
 */
@Override
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
    //获得username
    String username  = (String)principals.getPrimaryPrincipal();
    //新建SimpleAuthorizationInfo对象
    SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
    //查询当前用户的所有 "角色" 和 "权限"
    UserService userService =
        (UserService)ContextLoader.getCurrentWebApplicationContext().getBean("userService");
    Set<String> roles = userService.queryRolesByUsername(username);
    Set<String> perms = userService.queryPermissionsByUsername(username);
    //“角色” 和 “权限” 存入 SimpleAuthorizationInfo对象
    info.setRoles(roles);
```

```
        info.setStringPermissions(perms);
        //返回SimpleAuthorizationInfo
        return info;
    }
}
```

6.3 配置Realm

> shiro.ini中 配置自定义Realm
>
> 注意：[users] [roles] 两个部分不再需要

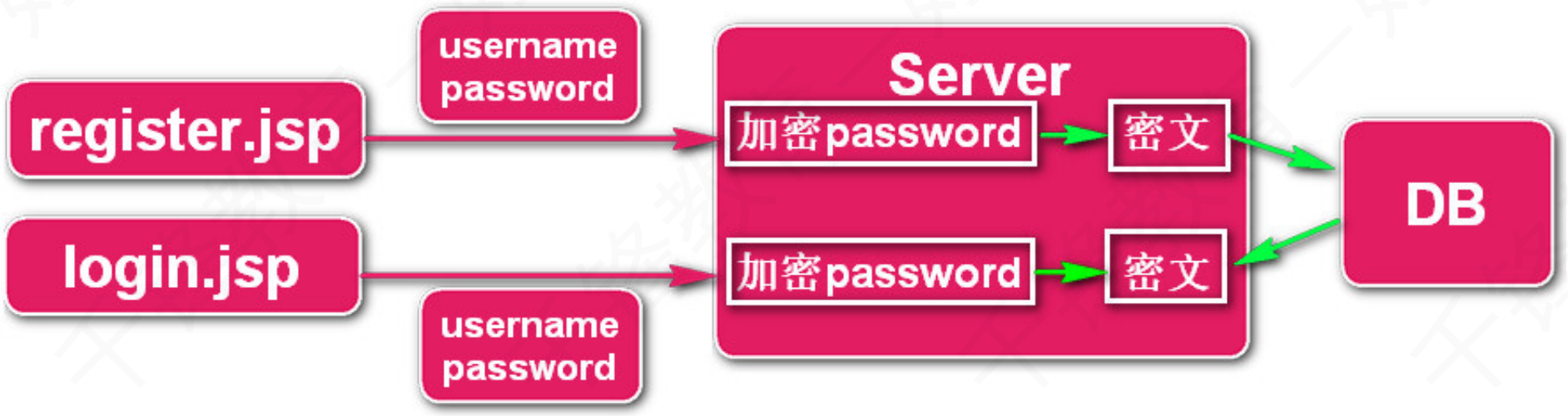
```
[main]
shiro.loginUrl = /login.jsp
shiro.unauthorizedUrl=/login.jsp
shiro.redirectUrl=/logout.jsp
shiro.postOnlyLogout = true
#注意：此处实在安装自定义Realm 指定realm
#声明Realm 名称 = Realm类路径
realm1 = com.zhj.realm.MyRealm
realm2 = com.zhj.realm.MyRealm2
#安装Reaml 关联到SecurityManager
securityManager.realms=$realm1,$realm2

[urls]
#照旧
```

> web.xml配置不变

Shiro_自定义Realm后的 项目架构

七、加密

> 用户的密码是不允许明文存储的，因为一旦数据泄露，用户的隐私信息会完全暴露。
>
> 密码必须结果加密，生成密文，然后数据库中只存储用户的密码的密文。



> 在加密过程中需要使用到一些**"不可逆加密"**，如 **md5,sha**等
>
> 所谓不可逆是指：

- 加密函数A, 明文 “abc”, A("abc") = "密文"，不能通过 "密文" 反推出 "abc"，即使密文泄露密码仍然安全。

7.1 加密介绍

shiro支持hash(散列)加密，常见的如 md5, sha等

- **基本加密过程**

  md5(明文)，sha(明文) 得到明文的密文，但明文可能比较简单导致密文容易被破解。

- **加盐加密过程**

  系统生成一个随机salt="xxxxxx", md5(明文+salt) ,sha(明文+salt)，则提升了密文的复杂度。

- **加盐多次迭代加密过程**

  如果迭代次数为2，则加密2次： md5(明文+salt)=密文a ， md5(密文a+salt)=最终密文

  sha(明文+salt)=密文a ， sha(密文a+salt)=最终密文

  则进一步提升了密文的复杂度，和被破解的难度。

加密过程中建议使用salt，并指定迭代次数，迭代次数的建议值1000+

实例代码：

```
String password="abc";//密码明文
String salt=UUID.randomUUID().toString();//盐
Integer iter = 1000;//迭代次数
String pwd = new Md5Hash(password, salt,iter).toString(); //md5加密
String pwd = new Md5Hash(password, salt, iter).toBase64(); //加密后转base64

String pwd = new Sha256Hash(password, salt, iter).toString();//sha256加密
String pwd = new Sha256Hash(password, salt, iter).toBase64();//加密后转base64

String pwd = new Sha512Hash(password, salt, iter).toString();//sha256加密
String pwd = new Sha512Hash(password, salt, iter).toBase64()//加密后转base64
```

7.2 加密

增加用户，或修改用户密码时，涉及到密码的**加密**

在注册用户的业务中，对用户提交的密码加密即可。

注意：之前的用户表，并未考虑存储加密相关信息，所以此时需要对用户表做出改进，

加一列【 salt varchar(50) 】，用于存储每个用户的盐。

id	username	password	salt
1	zhangsan	ceb34b63869cef8	JqNqz6rhKsAO8oS
2	lisi	0472bd414ab549	QWD9zHw9CKnFR(
3	wangwu	reff48koFTmCnJE	zrNLY4ac74MRog(

```
class UserServiceImpl implements UserService{
    @Autowired
    private UserDAO userDAO;
    public void createUser(User user){
        user.setSalt(UUID.randomUUID().toString());//设置随机盐
        //设置加密属性：sha256算法，随机盐，迭代1000次
        Sha256Hash sha256Hash = new Sha256Hash(user.getPassword(),user.getSalt(),1000);
        //将用户信息（包括密码的密文 和 盐）存入数据库
        user.setPassword(sha256Hash.toBase64());//密文采用base64格式化
        userDAO.createUser(user);
    }
}
```

7.3 密码比对

> 登录认证身份时，涉及到密码 **比对** 过程
>
> **注意，加密过程中使用的加密属性，和此处使用的加密属性 必须一致：**
>
> **【sha256,迭代1000次，使用base64格式化密文】**

7.3.1 指定比对器

```
[main]
...
#声明密码比对器
credentialsMatcher=org.apache.shiro.authc.credential.HashedCredentialsMatcher
credentialsMatcher.hashAlgorithmName=sha-256
credentialsMatcher.hashIterations=1000
#true=hex格式  false=base64格式
credentialsMatcher.storedCredentialsHexEncoded=false
#比对器关联给realm,则realm中对用户做身份认证时，可以使用加密比对器，对密文做比对
realm1 = com.zhj.realm.MyRealm
realm1.credentialsMatcher=$credentialsMatcher
#realm关联给securityManager
securityManager.realms=$realm1
```

7.3.2 修改Realm

> doGetAuthenticationInfo方法的返回值需要做修改

```
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws
AuthenticationException {
    String username = (String) token.getPrincipal();
    UserService userService =
        (UserService)ContextLoader.getCurrentWebApplicationContext().getBean("userService");
    User user = userService.queryUser(username);
    System.out.println("user:"+user);
    if(user==null){
        System.out.println("用户不存在");
        throw new UnknownAccountException("username:"+username+"不存在");
    }
    //以上逻辑不变
    //在最后返回用户认证info时，添加一个属性：ByteSource.Util.bytes(user.getSalt()) = 盐
    //用于密码比对
    return new SimpleAuthenticationInfo(user.getUsername(),
                                        user.getPassword(),
                                        ByteSource.Util.bytes(user.getSalt()),
                                        getName());
}
```

> **至此，可以进行注册，注册中已经会加密密码。**
>
> **然后登陆认证身份，认证时realm会调用比对器比对密文。**

八、Spring集成

> web项目的核心组件都在spring工厂中管理，利用IOC和AOP，组建了关系松散，稳健的系统。
>
> **shiro的诸多组件也需要由spring统一管理，进而可以更好的和其他组件协作。**

> **之前的Realm中一直有如下代码：**
>
> ```
> //由于Realm还未进入spring工厂，所以无法直接注入工厂内部的DAO组件
> UserService userService =(UserService)ContextLoader.getCurrentWebApplicationContext().getBean("xx");
> ```

> **ops：shiro的组件都是pojo组件，非常容易用spring管理，可以方便的从ini迁移到spring**

8.1 pom

```
<!-- 其他依赖和web集成中 一致 , 此处省略-->
<!-- 新增一个依赖 用于在工厂中生产 ShiroFilter-->
<!-- 会传递导入shiro-core 和 shiro-web -->
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.4.0</version>
</dependency>
```

8.2 aplicationContext.xml

> 将SecurityManager和Realm和ShiroFilter 都迁移到applicationContext.xml中
>
> 建议将如下配置，单独一个配置文件：shiro-spring.xml,然后在applicationContext.xml中引入：
>
> `<import resource="classpath:shiro-spring.xml"/>`

```
<!-- 整合mybaits，事务控制等 配置不变 -->
<!-- 添加配置 -->
<!-- shiro -->
<!-- 声明realm -->
<bean id="realm1" class="com.zhj.realm.MyRealm">
    <property name="userService" ref="userService"/>
    <!-- 此属性如果通过注解注入，则需要将注解加载set方法上，不能用在属性上。
     此属性是父类属性，所以只有在set方法上注入，才能覆盖父类属性值。
  -->
    <property name="credentialsMatcher">
        <bean class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">
            <property name="hashAlgorithmName" value="SHA-256"/>
            <!-- true means hex encoded, false means base64 encoded -->
            <property name="storedCredentialsHexEncoded" value="false"/>
            <property name="hashIterations" value="1000"/>
        </bean>
    </property>
</bean>
<!-- 声明SecurityManager -->
<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
    <property name="realm" ref="realm1"/>
</bean>
<!-- 生产SpringShiroFilter
     ( 持有shiro的过滤相关规则，可进行请求的过滤校验，校验请求是否合法 )
-->
<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
    <property name="securityManager" ref="securityManager"/>
    <property name="loginUrl" value="/user/login/page"/>
    <property name="unauthorizedUrl" value="/error.jsp"/>
    <property name="filterChainDefinitions">
        <value>
            /user/query=anon
            /user/insert=authc,roles["banfu"]
            /user/update=authc,perms[""student:update""]
            /order/insert=authc,roles["xuewei"]
            /user/logout=logout
        </value>
    </property>
</bean>
```

8.3 web.xml

```
<!-- 会从spring工厂中获取和它同名的bean，（id="shiroFilter"）
     接到请求后调用bean的doFilter方法，进行访问控制。
-->
<filter>
    <filter-name>shiroFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
```

```
    <init-param>
        <param-name>targetFilterLifecycle</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- EnvironmentLoaderListener不再需要，因为shiro环境已由spring初始化
     springMVC,spring配置不变 -->
```

九、记住我

> 在登录后，可以将用户名存在cookie中，下次访问时，可以先不登录，就可以识别身份。
>
> 在确实需要身份认证时，比如购买，支付或其他一些重要操作时，再要求用户登录即可，用户体验好。
>
> 由于可以保持用户信息，系统后台也可以更好的监控、记录用户行为，积累数据。

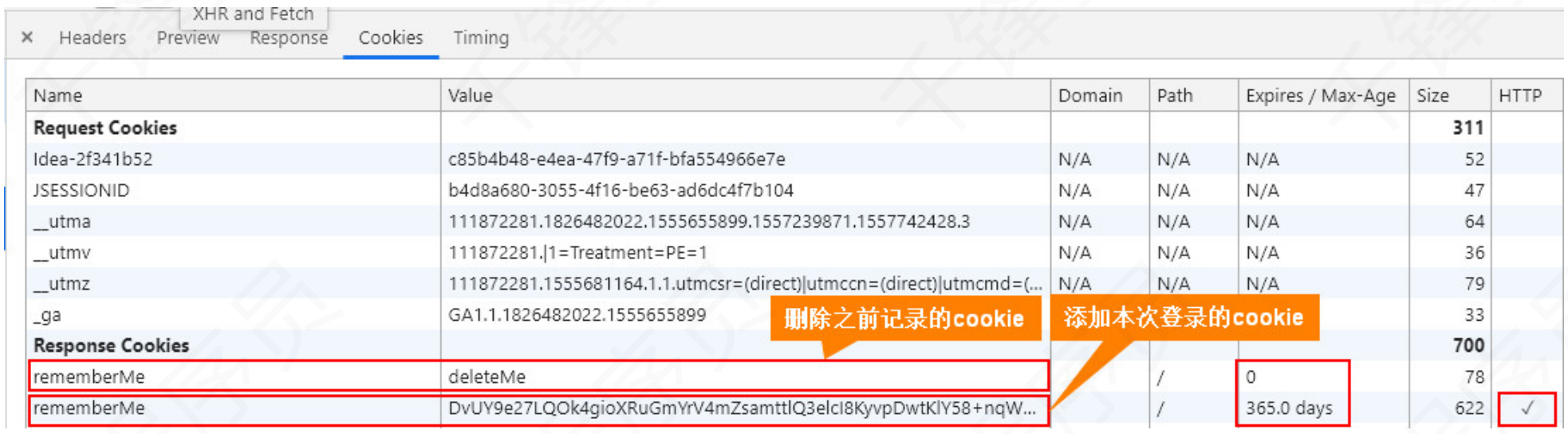
9.1 代码

> **”记住我“ 起点在登录时刻：Subject.login(UsernameAndPasswordToken)**
>
> **而是否确定要“记住我”，由登录时的token控制开关：`token.setRememberMe(true);`**

```
Subject subject = SecurityUtils.getSubject();
UsernamePasswordToken token = new UsernamePasswordToken(username, password);
//如果需要记住我的话，需要在token中设置
token.setRememberMe(true);//shiro默认支持”记住我“，只要有此设置则自动运作。
subject.login(token);
```

9.2 效果

> **登录后效果**



Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP
Request Cookies					**311**	
Idea-2f341b52	c85b4b48-e4ea-47f9-a71f-bfa554966e7e	N/A	N/A	N/A	52	
JSESSIONID	b4d8a680-3055-4f16-be63-ad6dc4f7b104	N/A	N/A	N/A	47	
__utma	111872281.1826482022.1555655899.1557239871.1557742428.3	N/A	N/A	N/A	64	
__utmv	111872281.|1=Treatment=PE=1	N/A	N/A	N/A	36	
__utmz	111872281.1555681164.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(...	N/A	N/A	N/A	79	
_ga	GA1.1.1826482022.1555655899				33	
Response Cookies					**700**	
rememberMe	deleteMe		/	0	78	
rememberMe	DvUY9e27LQOk4gioXRuGmYrV4mZsamttlQ3elcI8KyvpDwtKlY58+nqW...		/	365.0 days	622	✓

9.3 页面中显示

> 在页面中显示，cookie中记录的用户信息
>
> <shiro:user> 当有记住我信息，或已登录，则显示标签体内容
>
> <shiro:principal> 获取用户信息
>
> 注意：首页的访问路径的过滤器 不能是 authc，只能是 user 或 anon

```jsp
<!-- 首页 xx.jsp -->
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@taglib prefix="shiro" uri="http://shiro.apache.org/tags" %>
<html>
    <body>
        <h2>Hello World! </h2>
        <!-- 重点在此：通过如下shiro标签显示 -->
        <shiro:user>
            欢迎您，<shiro:principal/>  <a href="#">退出登录</a>
        </shiro:user>
    </body>
</html>
```

```jsp
<!-- 登录页面 login.jsp 自动填充用户名 -->
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="shiro" uri="http://shiro.apache.org/tags" %>
<html>
    <head>
        <title>Title</title>
    </head>
    <body>
        <form action="<c:url value="/user/login"/>" method="post">
            <!-- 重点在此：<shiro:principal/> -->
            username:<input type="text" name="username" value="<shiro:principal/>"> <br>
            password:<input type="password" name="password"><br>
            <input type="submit" value="登录">
        </form>
    </body>
</html>
```

9.4 自定义

> **如果需要做自定义，可以明确定义如下两个组件：**
>
> `SimpleCookie`：封装cookie的相关属性，定制cookie写出逻辑(`详见：addCookieHeader()`)
>
> `CookieRememberMeManager`：接受SecurityManager调度，获取用户信息，加密数据，并调度SimpleCookie写出cookie。

```jsp
<!-- 之前的配置不变，添加如下配置 -->
<!-- remember me -->
<bean id="rememberMeCookie" class="org.apache.shiro.web.servlet.SimpleCookie">
    <!-- rememberMe是cookie值中的key，value时用户名的密文
     cookie["rememberMe":"deleteMe"] 此cookie每次登陆后都会写出，用于清除之前的cookie
         cookie["rememberMe":username的密文] 此cookie也会在登录后写出，用于记录最新的username
     (ops: 如上设计，既能保证每次登陆后重新记录cookie，也能保证切换账号时，记录最新账号)
    -->
    <property name="name" value="rememberMe"/>
    <!-- cookie只在http请求中可用，那么通过js脚本将无法读取到cookie信息，有效防止cookie被窃取 -->
    <property name="httpOnly" value="true"/>
    <!-- cookie的生命周期，单位：秒 -->
    <property name="maxAge" value="2592000"/><!-- 30天 -->
</bean>
<bean id="rememberMeManager" class="org.apache.shiro.web.mgt.CookieRememberMeManager">
    <!-- 对cookie的value加密的密钥 建议每个项目都不一样
    加密方式AES(对称加密)
    密钥生成：【KeyGenerator keygen = KeyGenerator.getInstance("AES");
              SecretKey deskey = keygen.generateKey();
              System.out.println(Base64.encodeToString(deskey.getEncoded()));】
    SpEL:Spring Expression Language  #{表达式}  #{T(类型)...}

    <property name="cipherKey"
              value="#{T(org.apache.shiro.codec.Base64).decode('c+3hFGPjbgzGdrC+MHgoRQ==')}"/>
  此配置可以省略，CookieRememberMeManager自动完成秘钥生成
  -->
    <!-- 注入SimpleCookie -->
    <property name="cookie" ref="rememberMeCookie"/>
</bean>
<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
  ...
```

```
    <!-- CookieRememberMeManager注入给SecurityManager,
         SecurityManager在处理login时，如果login成功，则会通过rememberMeManager做"记住我",
         将用户名存入cookie
  -->
    <property name="rememberMeManager" ref="rememberMeManager"/>
</bean>
```

十、Session管理

> shiro作为一款安全管理框架，对状态保持有很强的需要。
>
> 比如最常用的用户认证，就必需状态的保持，以及其他的一些功能实现的需要。
>
> 【shiro需要：认证中的 **记住我中的用户名** **正式登陆的用户名** 】【 开发者需要：其他功能中需要存入session的值 】

> **shiro提供了一整套session管理方案.**
>
> ```
> **1. shiro的session方案和任何`容器无关`(如servlet容器)；**
>
> **2. javaSE也可以使用；相关组件都是pojo对ioc极其友好(方便的管理对象和满足依赖关系，定制参数)**
>
> **3. 可以方便的扩展定制存储位置(`内存，缓存，数据库`等)**
>
> **4. 对`web透明`支持：用了shiro的session后，项目中关于session的代码完全不用任何改动**
>
> **5. 提供了全面的session`监听`机制，和session`检测`机制，对session可以细粒度操作**
> ```
>
> **即，使用了shiro后，采用shiro的session方案是最优的方案。**

10.1 javaSE环境 （了解）

> shiro 的session管理方案，可以在javaSE中使用，实用价值不大。

```
Subject subject = SecurityUtils.getSubject();
//获取session
Session session = subject.getSession();
//session超时时间，单位：毫秒；0，马上过期；正数，则空闲对应毫秒后过期；负数，则不会过期
session.setTimeout(10000);
//session存、取值
session.setAttribute("name","zhj");
session.getAttribute("name");
//获取sessionID
getSession().getId();
//销毁session
session.stop();
```

> **原理，核心对象：**
>
> **1. SimpleSession**
>
> ```
> Session的实现类，完成session基本功能。
> ```
>
> **2. SimpleSessionFactory**
>
> ```
> 生产SimpleSession
> ```
>
> **3. SessionDAO**
>
> ```
> 默认的实现类：MemorySessionDAO，由SessionManager创建，
> ```
>
> 负责存储所有session对象，存储位置：内存
>
> **4. DefaultSessionManager**
>
> ```
> 由SecurityManager创建，负责创建、管理SessionFactory和SessionDAO。
> ```

```
//核心类演示:  ( ops:实际开发不用手动创建, shiro会初始化 )
//通过SecurityManager 获得 SessionManager
DefaultSessionManager sessionManager = (DefaultSessionManager)securityManager.getSessionManager();
//通过SessionManager获得SessionFactory
SimpleSessionFactory sessionFactory = (SimpleSessionFactory) sessionManager.getSessionFactory();
//通过SessionManager获得SessionDAO
MemorySessionDAO sessionDAO = (MemorySessionDAO)sessionManager.getSessionDAO();
//通过SessionFactory获得Session
SimpleSession session1 = (SimpleSession) sessionFactory.createSession(null);
//通过Session做具体操作
session1.setAttribute("name","zhangsan");
System.out.println(session1.getAttribute("name"));
```

10.2 javaEE环境

10.2.1 applicationContext.xml

```
<!-- 增加session管理相关配置 -->
<!-- 会话Cookie模板 默认可省-->
<bean id="sessionIdCookie" class="org.apache.shiro.web.servlet.SimpleCookie">
    <!-- cookie的 key="sid" -->
    <property name="name" value="JSESSIONID"/>
    <!-- 只允许http请求访问cookie -->
    <property name="httpOnly" value="true"/>
    <!-- cookie过期时间, -1:存活一个会话 ,单位:秒 , 默认为-1-->
    <property name="maxAge" value="-1"/>
</bean>

<bean id="sessionManager"
      class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">
    <!-- 默认值和配置中给出的一致, 所bean:sessionIdCookie 可以省略 -->
    <property name="sessionIdCookie" ref="sessionIdCookie"/>
    <!-- session全局超时时间, 单位:毫秒 , 30分钟 默认值为1800000-->
    <property name="globalSessionTimeout" value="1800000"/>
</bean>

<!-- 将sessionManager关联到SecurityManager -->
<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
    ...
    <!-- 增加配置sessionManager -->
    <property name="sessionManager" ref="sessionManager"/>
</bean>
```

10.3 Session监听

> session有三个核心过程：创建、过期、停止
>
> **过期：**session的默认过期时间为30分钟。通过比对最近一次使用时间和当前使用时间判断
>
> session不会自动报告过期，需检测器检测时，或再次访问时，才可以识别是否过期并移除。
>
> **停止：**用户主动`logout`；主动调用`session.stop()`；两种情况会将session标志为停止状态。

```
// 定义监听类 exentends SessionListenerAdapter
public class MySessionListener extends SessionListenerAdapter{
    //当有session创建时 触发
    @Override
    public void onStart(Session session) {
        System.out.println("session:"+session.getId()+" start");
    }
    //当有session停止时 触发
    @Override
    public void onStop(Session session) {
        System.out.println("session:"+session.getId()+" stop");
    }
    //当有session过期时 触发
    // 但不会主动触发, 需要再次访问时, 即又要使用session时才会发现session过期, 并触发。
```

```
    @Override
    public void onExpiration(Session session) {
        System.out.println("session:"+session.getId()+" expired");
    }
}
```

> 配置监听类，关联给SessionManager

```
<bean id="sessionManager" class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">
    ...
    <property name="sessionListeners">
        <list>
            <bean class="com.zhj.listener.MySessionListener"></bean>
        </list>
    </property>
    ...
</bean>
```

10.4 Session检测

> 用户如果没有主动退出登录，只是关闭浏览器，则session是否过期无法获知，也就不能停止session。
>
> 为此，shiro提供了session的检测机制，可以定时发起检测，**识别session过期 并停止session**。

```
<!-- sessionManager默认开启session检测机制 -->
<bean id="sessionManager" class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">
    ...
    <!-- 开启检测器，默认开启 -->
    <property name="sessionValidationSchedulerEnabled" value="true"/>
    <!--- 检测器运行间隔，单位：毫秒  默认1小时
        //检测到过期后，会直接将session删除
        protected void afterExpired(Session session) {
                if (isDeleteInvalidSessions()) {
                    delete(session);
                }
            }
    -->
    <property name="sessionValidationInterval" value="3600000"/>
    ...
</bean>
```

> 如上，通过检测器，定时的检测session，并及时移除无效session，释放资源。

十一、注解开发

> **shiro提供了一系列的访问控制的注解，可以简化开发过程。**

```
<!-- 注解加载Controller中，原理是会对Controller做增强，切入访问控制逻辑，所以需要如下依赖 -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aspects</artifactId>
    <version>4.3.6.RELEASE</version>
</dependency>
```

11.1 配置mvc.xml

```
<!-- enable shiro's annotation-->
<bean id="lifecycleBeanPostProcessor"
      class="org.apache.shiro.spring.LifecycleBeanPostProcessor"/>
<!-- 自动代理生成器，等价于aop:config;
     aop:config 或 AutoProxyCreator两者选其一，spring官方提醒千万不要同时使用。

<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"
      depends-on="lifecycleBeanPostProcessor"/>
-->
```

```
<aop:config></aop:config>
<!-- 在此bean的构建过程中，初始化了一些额外功能和piontcut
     interceptors.add(new RoleAnnotationMethodInterceptor(resolver));
     interceptors.add(new PermissionAnnotationMethodInterceptor(resolver));
     interceptors.add(new AuthenticatedAnnotationMethodInterceptor(resolver));
     interceptors.add(new UserAnnotationMethodInterceptor(resolver));
     interceptors.add(new GuestAnnotationMethodInterceptor(resolver));
-->
<bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">
  <property name="securityManager" ref="securityManager"/>
</bean>
```

11.2 注解使用

> 加在类上

```
@Controller
@RequestMapping("/user")
@RequiresAuthentication //类中的所有方法都需要身份认证
@RequiresRoles(value={"manager","admin"},logical= Logical.OR)//类中的所有方法都需要角色,"或"
public class ShiroController {
  ...
}
```

> 加在方法上

```
@Controller
@RequestMapping("/user")
public class ShiroController2{
  ...
    @RequiresPermissions({"user:query",“user:delete"}) //有对应权限，默认是 "且"
    @RequiresUser //记住我 或 已身份认证
    public String hello(){
    ...
    }
    @RequiresGuest //游客身份
    public String hello2(){
    ...
    }
}
```

> 配置修改

```
<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
    <property name="securityManager" ref="securityManager"/>
    <!-- 不再需要，此时如果身份或权限不通过，会抛出异常，需要异常解析器处理
    <property name="loginUrl" value="/user/login/page"/>
    <property name="unauthorizedUrl" value="/error.jsp"/>
    <property name="filterChainDefinitions">
        <value>
            /user/query=anon  如下不再需要，登出可以保留，也可以自己写handler中subject.logout()
            /user/insert=authc,roles["banfu"]
            /user/update=authc,perms[""student:update""]
            /order/insert=authc,roles["xuewei"]
            /user/logout=logout
        </value>
    </property>-->
</bean>
```

11.3 异常处理

```
<!-- mvc.xml中: MVC的自定义异常处理器,用于处理权限或身份认证不通过时的异常处理-->
<bean class="com.zhj.ex.handler.MyExHandler"></bean>
```

```java
public class MyExceptionResolver implements HandlerExceptionResolver{
    @Override
    public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object
handler, Exception ex) {
        System.out.println(ex.getClass());
        ex.printStackTrace();//开发时必需
        ModelAndView mv = new ModelAndView();
        if(ex instanceof IncorrectCredentialsException || ex instanceof UnknownAccountException){
            //跳转登录页面，重新登录
            mv.setViewName("redirect:/user/login");
        }else if(ex instanceof UnauthorizedException){// 角色不足  权限不足
            //跳转权限不足的页面
            mv.setViewName("redirect:/user/perms/error");
        }else if(ex instanceof UnauthenticatedException){//没有登录 没有合法身份
            //跳转登录页面，重新登录
            mv.setViewName("redirect:/user/login");
        }
        return mv;
    }
}
```