SpringBoot Web项目案例篇

主讲：安焱

课程大纲

1. 整体介绍演示
2. 框架集成
3. Mybatis配置
4. Druid配置
5. Shiro配置
6. Kaptcha验证码配置
7. Quartz配置
8. AOP监控用户
9. Mvc拦截器配置
10. Swagger2配置
11. XssFilter配置
12. Hibernate-validator
13. Velocity代码生成器
14. 权限模型设计
15. 权限相关功能实现
16. 资料代码

1 介绍-系统特点

该项目案例是一个轻量级系统，其核心设计目标是开发迅速、学习简单、轻量级、易扩展等。

特点如下：

1、系统只涉及Spring、Shiro、Mybatis后端框架，降低学习使用成本

2、友好的代码结构及注释，便于阅读及二次开发

3、灵活的权限控制，可控制到页面或按钮

4、页面交互使用Vue2.x，极大的提高了开发效率（支持HTML、JSP、Velocity、Freemarker等视图）

5、完善的代码生成机制，使用Velocity可在线生成entity、xml、dao、service、page、js代码，几分钟可以完成一个简单的增删改查页面。

6、引入quartz定时任务，可动态完成任务的添加、修改、删除、暂停、恢复及日志查看等功能

7、实现前后端完全分离，前端再也不用关注后端技术，swagger文档支持，方便编写API接口文档

8、引入API模板，根据token作为登录令牌，极大的方便了APP接口开发

9、引入Hibernate Validator校验框架，轻松实现后端校验

10、使用aop对用户行为进行监控记录

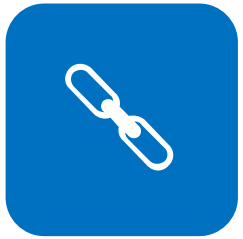
11、对xxs进行拦截过滤等

1 介绍-系统框架

系统架构

核心框架：Spring Framework 4.2
安全框架：Apache Shiro 1.3
视图框架：Spring MVC 4.2
持久层框架：MyBatis 3.3
定时器：Quartz 2.3
数据库连接池：Druid 1.0
日志管理：Logback 1.1
页面交互：Vue2.x
前后端分离：Swagger2
校验工具类：Hibernate-validator5.3
代码生成器：Velocity1.7

2 框架集成-环境准备



JDK1.7+

Maven3.0+

MySQL5.5+

STS (eclipse)

其他

2 框架集成-项目部署

01 建库

创建数据库shiro_boot，数据库编码为UTF-8
执行src/main/resources/db.sql文件，初始化数据
修改application.properties文件，更新MySQL账号和密码

02 编译更新

Eclipse、IDEA执行【Maven-update project】进行jar下载，然后run as--maven clean、run as –maven install打包

03 访问

右键CfApplication类，选择run as – java application启动项目
项目访问路径：http://localhost

默认管理员：admin/admin

建议使用阿里云Maven仓库

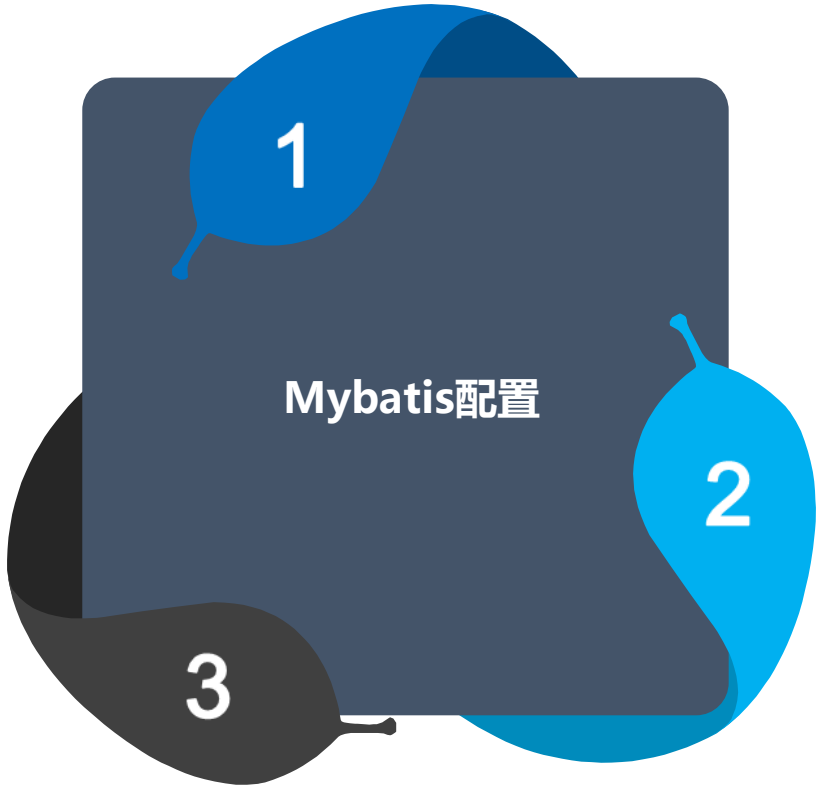
```
<mirror>
<id>alimaven</id>
<name>aliyun maven</name>
<url>http://maven.aliyun.com/nexus/content/groups/public/</url>
<mirrorOf>central</mirrorOf>
</mirror>
```

2 框架集成

系统架构

核心框架：Spring Framework 4.2
安全框架：Apache Shiro 1.3
视图框架：Spring MVC 4.2
持久层框架：MyBatis 3.3
定时器：Quartz 2.3
数据库连接池：Druid 1.0
日志管理：Logback 1.1
页面交互：Vue2.x
前后端分离：Swagger2
校验工具类：Hibernate-validator5.3
代码生成器：Velocity1.7

3 Mybatis



1.导入maven依赖
2.配置数据源
3.创建实体类
4.创建dao（mapper）
5.添加mapper文件
6.配置mybatis自动注入Mapper
7.在启动类中指定@MapperScan

3 Mybatis配置

01

Xml文件配置
Spring bean
Property配置

自动注入Mapper（三种方式配置）

①种方式

1.添加一个mybatisConfig.xml，指定驼峰法转换

2.指定每一个的mappers位置

```
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>  user_id  →userId
        <setting name="mapUnderscoreToCamelCase" value="true" />
    </settings>
    <typeAliases>
        <package name="com.wangnian.mybatis.entity"/>
    </typeAliases>
    <mappers>
        <mapper resource="mybatis/mapper/UserMapper.xml"/>
    </mappers>
</configuration>
```

3.在application.properties里指定mybatisConfig.xml的位置

mybatis.config-location=classPath:mybatisConfig.xml

3 Mybatis配置

02

Xml文件配置
Spring bean
Property配置

②种方式
通过javaBean的方式配置

③种方式
直接通过application.properties中配置

```
#mybatis配置
#指定mapper.xml的位置
mybatis.mapper-locations=classpath:mapper/*.xml
mybatis.configuration.mapUnderscoreToCamelCase=true
mybatis.configuration.useColumnLabel=true
```

4 Druid配置

导入依赖包

配置数据源

注册servlet和过滤器监控

1.导入依赖包

2.配置数据源

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

3.注册servlet和过滤器监控

配置Servlet，在SpringBoot项目中基于注解的配置，如果是web.xml配置，按规则配置即可。

web.xml:StatViewServlet

spring bean

```
    @Bean
    public ServletRegistrationBean druidServlet() {
        ServletRegistrationBean servletRegistrationBean = new ServletRegistrationBean();
        servletRegistrationBean.setServlet(new StatViewServlet());
        servletRegistrationBean.addUrlMappings("/druid/*");

        return servletRegistrationBean;
    }
```

```
# DataSource
spring.datasource.initialSize=5
```

在Spring Boot1.4.0中驱动配置信息没有问题，但是连接池的配置信息不再支持这里的配置项，即无法通过配置项直接支持相应的连接池；这里列出的这些配置项可以通过定制化DataSource来实现。

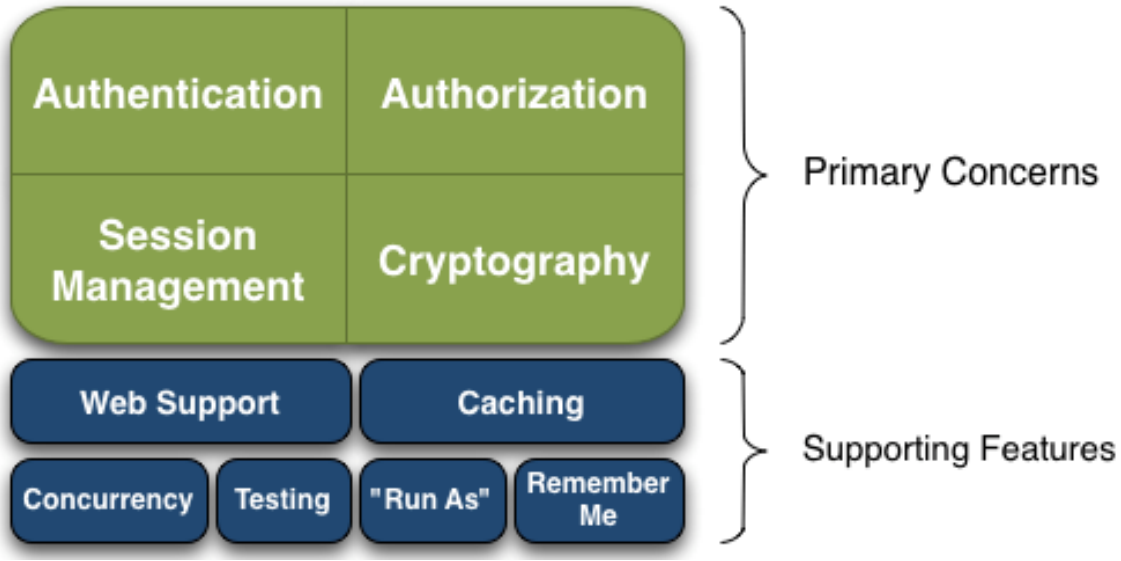
```
public DataSource dataSource(){
        DruidDataSource datasource = new DruidDataSource();

        datasource.setUrl(this.dbUrl);
        datasource.setUsername(username);
}
```

4.启动成功 http://localhost:80/druid/

5 Shiro配置

基本功能



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；即判断用户是否能做事情，常见的如：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环境的，也可以是如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；

Concurrency： shiro支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；

Testing：提供测试支持；

Run As：允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；

Remember Me：记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录了。

注意：Shiro不会去维护用户、维护权限；这些需要我们自己去设计/提供；然后通过相应的接口注入给Shiro即可。

5 Shiro配置

从应用程序角度的来观察如何使用Shiro完成工作

1、应用代码通过Subject来进行认证和授权，而Subject又委托给SecurityManager；
2、我们需要给Shiro的SecurityManager注入Realm，从而让SecurityManager能得到合法的用户及其权限进行判断。
从以上也可以看出，Shiro不提供维护用户/权限，而是通过Realm让开发人员自己注入。

5 Shiro配置

FilterConfig

配置Shiro过滤器,先让Shiro过滤系统接收到的请求

这里filter-name必须对应applicationContext.xml中定义的<bean id="shiroFilter"/>，springboot中对应的@Bean

使用[/*]匹配所有请求,保证所有的可控请求都经过Shiro的过滤

ShiroConfig

配置

securityManager

sessionManager

shiroFilter等

6 验证码Kaptcha配置

添加相关依赖
定义一个返回DefaultKaptcha对象的方法，进行Kaptcha属性封装
因为采用了shiro，需要对将返回的验证码图片进行过滤
在SysLoginController中对captcha.jpg请求进行处理

7 Quartz配置

一般的任务调度使用springs schedule之类的轻量级定时任务可以实现。但对于高级的定时需求，就难以满足了：比如工程运行过程中控制定时任务的开关等。这个时候就需要引入Quartz了

在Spring中使用Quartz有两种方式实现：

第一种是任务类继承QuartzJobBean，

第二种则是在配置文件里定义任务类和要执行的方法，类和方法可以是普通类。

a、加入相关的依赖jar

b、配置一个返回对象SchedulerFactoryBean的方法

c、在SchedulerFactoryBean的方法中进行quartz的参数设置

d、编写相关的共同类对quartz封装

相关的建表语句：

quartz-2.2.3\docs\dbTables

本项目使用Quartz2.2.x实现定时任务功能，支持添加、修改、删除、暂停、恢复、集群及日志查看等功能。

默认支持集群。

7 Quartz配置

1、新增定时任务，只需创建spring bean即可，如下所示：

```
/**
* 新增定时任务
*
* bean的名称为【newTask】
*/
@Component("newTask")
public class NewTask {
		private Logger logger = LoggerFactory.getLogger(getClass());
		public void test1(String params){
			logger.info("我是带参数的test1方法，正在被执行，参数为：" + params);
		}
		public void test2(){
			logger.info("我是不带参数的test2方法，正在被执行");
		}
}
```

2、在管理后台，添加定时任务，如下图所示：

bean名称：newTask
方法名称：test1
参数：shiro
cron表达式：0 0 12 * * ?
备注：测试

完成上面2步，新的定时任务就添加完成了，每天12点都会执行一次

7 Quartz配置

表 5.1. Quartz Cron 表达式支持到七个域

名称	是否必须	允许值	特殊字符
秒	是	0-59	, - * /
分	是	0-59	, - * /
时	是	0-23	, - * /
日	是	1-31	, - * ? / L W C
月	是	1-12 或 JAN-DEC	, - * /
周	是	1-7 或 SUN-SAT	, - * ? / L C #
年	否	空 或 1970-2099	, - * /

Eg:
分钟频度的任务计划 Cron 表达式:
每天的从 5:00 PM 至 5:59 PM 中的每分钟触发 [0 * 17 * * ?]

日的频度的任务计划 Cron 表达式:
每天的 3:00 AM [0 0 3 * * ?]

周和/或月的频度上任务计划的 Cron 表达式:
在每个周一,二, 三和周四的 10:15 AM [0 15 10 ? * MON-FRI]
每月15号的 10:15 AM [0 15 10 15 * ?]

8 AOP监控用户

AOP（Aspect-OrientedProgramming，面向方面编程）
实现AOP的技术，主要分为两大类：
一是采用动态代理技术，利用截取消息的方式，对该消息进行装饰，以取代原有对象行为的执行；
二是采用静态织入的方式，引入特定的语法创建“方面”，从而使得编译器可以在编译期间织入有关“方面”的代码。

AOP用来封装横切关注点，具体可以在下面的场景中使用:

Authentication 权限
Caching 缓存

Context passing 内容传递
Error handling 错误处理
Lazy loading 懒加载
Debugging 调试

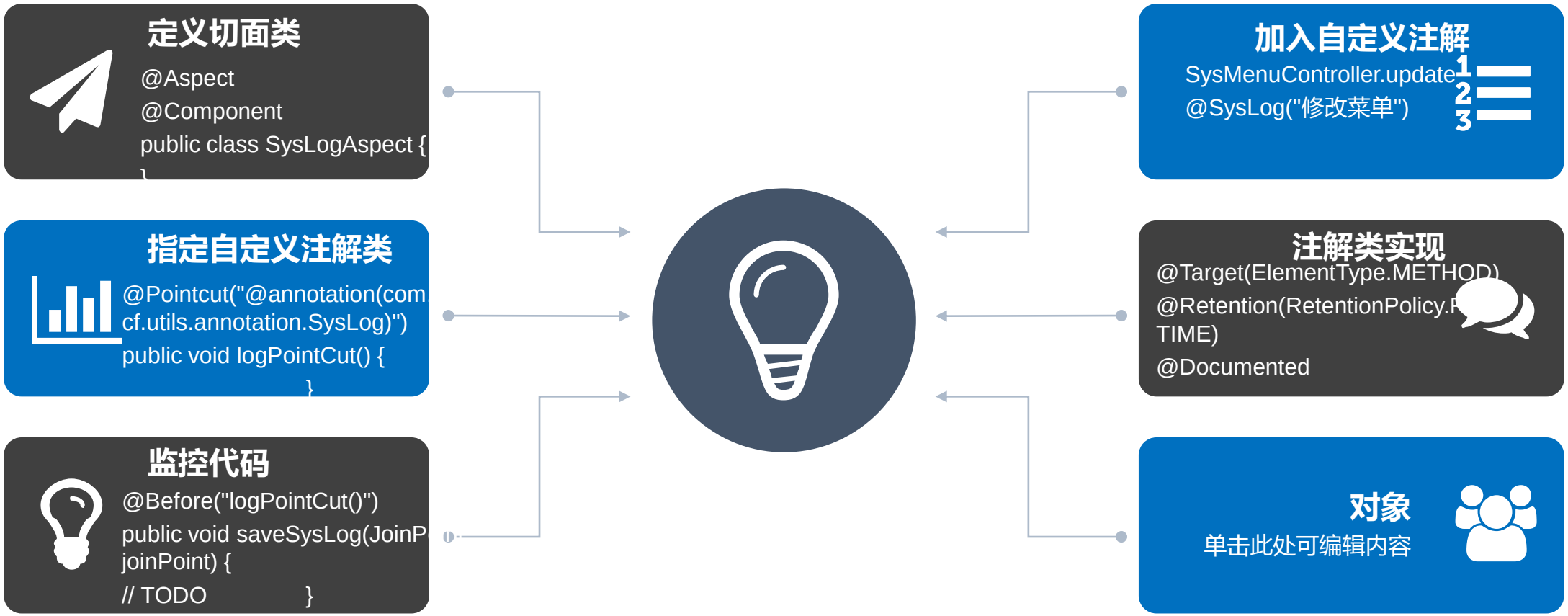
logging, tracing, profiling and monitoring 记录跟踪 优化 校准

Performance optimization 性能优化
Persistence 持久化
Resource pooling 资源池
Synchronization 同步

8 AOP监控用户

相关注解

8 AOP监控用户


定义切面类
@Aspect
@Component
public class SysLogAspect {
}
指定自定义注解类
@Pointcut("@annotation(com.
cf.utils.annotation.SysLog)")
public void logPointCut() {
}
监控代码
@Before("logPointCut()")
public void saveSysLog(JoinP
joinPoint) {
// TODO }
加入自定义注解
SysMenuController.update
@SysLog("修改菜单")
注解类实现
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.R
TIME)
@Documented
对象
单击此处可编辑内容

9 Mvc拦截器配置

Spring MVC提供HandlerInterceptor（拦截器）工具。根据文档，HandlerInterceptor的功能跟过滤器类似，

但拦截器提供更精细的控制能力：在request被响应之前、request被响应之后、视图渲染之前以及request全部结束之后。

spring mvc配置，需要创建一个配置类并实现WebMvcConfigurer接口，WebMvcConfigurerAdapter 抽象类是对WebMvcConfigurer接口的简单抽象。（增加了一些默认实现）

addInterceptors覆盖并重写了addInterceptors(InterceptorRegistory registory)方法，这是典型的回调函数——利用该函数的参数registry来添加自定义的拦截器。

addArgumentResolvers参数解析

根据项目的需要实现接口中特定的方法。

9 Mvc拦截器配置

```
/**
 * 常用的一些方法
 */
public interface WebMvcConfigurer {
    void addFormatters(FormatterRegistry registry);
    void configureMessageConverters(List<HttpMessageConverter<?>> converters);
    void extendMessageConverters(List<HttpMessageConverter<?>> converters);
    Validator getValidator();
    /* 配置内容裁决的一些选项*/
    void configureContentNegotiation(ContentNegotiationConfigurer configurer);
    void configureAsyncSupport(AsyncSupportConfigurer configurer);
    /* @since 4.0.3 */
    void configurePathMatch(PathMatchConfigurer configurer);
    /*参数解析*/
    void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers);
    /*返回值解析*/
    void addReturnValueHandlers(List<HandlerMethodReturnValueHandler> returnValueHandlers);
    /*异常处理*/
    void configureHandlerExceptionResolvers(List<HandlerExceptionResolver> exceptionResolvers);
    void extendHandlerExceptionResolvers(List<HandlerExceptionResolver> exceptionResolvers);
    void addInterceptors(InterceptorRegistry registry);
    MessageCodesResolver getMessageCodesResolver();
    void addViewControllers(ViewControllerRegistry registry);
    /**
     * 这里配置视图解析器
     */
    void configureViewResolvers(ViewResolverRegistry registry);
    /**
     *静态资源处理
     */
    void addResourceHandlers(ResourceHandlerRegistry registry);
    void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer);
    void addCorsMappings(CorsRegistry registry);
}
```

10 Swagger2配置-现状描述

按照现在的趋势，前后端分离几乎已经是业界对开发和部署方式所达成的一种共识。所谓的前后端分离，并不是传统行业中的按部门划分，

一部分人只做前端（HTML/CSS/JavaScript等等），另一部分人只做后端（或者叫服务端），

但是很多团队采取了后端的模板技术（JSP, FreeMarker, ERB等等），前端的开发和调试需要一个后台Web容器的支持，从而无法将前后端开发和部署做到真正的分离。

通常，前后端分别有着自己的开发流程，构建工具，测试等。做前端的谁也不会想要用Maven或者Gradle作为构建工具，

同样的道理，做后端的谁也不会想要用Grunt或者Gulp作为构建工具。

前后端仅仅通过接口来协作，这个接口可能是JSON格式的RESTFul的接口，也可能是XML的，重点是后台只负责数据的提供和计算，而完全不处理展现。

而前端则负责拿到数据，组织数据并展现的工作。这样结构清晰，关注点分离，前后端会变得相对独立并松耦合。

但是这种想法依然还是很理想化，前后端集成往往还是一个很头痛的问题。

比如在最后需要集成的时候，我们才发现最开始商量好的数据结构发生了变化，而且这种变化往往是在所难免的，这样就会增加大量的集成时间。

归根结底，还是前端或者后端感知到变化的时间周期太长，不能“及时协商，尽早解决”，最终导致集中爆发。

怎么解决这个问题呢？我们需要提前协商好一些契约，并将这些契约作为可以被测试的中间产品，

然后前后端都通过自动化测试来检验这些契约，一旦契约发生变化，测试就会失败。

这样，每个失败的测试都会驱动双方再次协商，有效的缩短了反馈周期，并且降低集成风险。

10 Swagger2配置

Swagger包括库、编辑器、代码生成器等很多部分

Swagger是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。

总体目标是使客户端和文件系统作为服务器以同样的速度来更新。文件的方法，参数和模型紧密集成到服务器端的代码，允许API来始终保持同步。

个人觉得，swagger的一个最大的优点是能实时同步api与文档。在项目开发过程中，发生过多次：修改代码但是没有更新文档，

前端还是按照老旧的文档进行开发，在联调过程中才发现问题的情况（当然依据开闭原则，对接口的修改是不允许的，但是在项目不稳定阶段，这种情况很难避免）。

修改pom.xml，添加maven依赖

添加Swagger配置类（可以定义多个组）

http://localhost/swagger-ui.html

Swagger会默认把apis目录下的Controller中的方法都生成API出来，实际上我们一般只需要标准接口的（像返回页面的那种Controller方法我们并不需要）。

10 Swagger2配置

常见swagger注解一览与使用：
@Api：修饰整个类，描述Controller的作用
@ApiOperation：描述一个类的一个方法，或者说一个接口
@ApiParam：单个参数描述
@ApiModel：用对象来接收参数
@ApiProperty：用对象接收参数时，描述对象的一个字段
@ApiResponse：HTTP响应其中1个描述
@ApiResponses：HTTP响应整体描述
@ApiIgnore：使用该注解忽略这个API
@ApiClass
@ApiError
@ApiErrors
@ApiParamImplicit
@ApiParamsImplicit

注：swagger对restful风格的api支持的比较好，非restful风格的api支持的不是很好，
对于非restful风格的api或者其他语言（非java语言）可以采用 http://editor.swagger 编辑器来收工完成相关的API编写
会配合后面的Mvc配置部分进行讲解

11 XssFilter配置

shiroFilter之前已经讲过

xssFilter：跨站点脚本攻击(XSS)

a、在springboot中注册一个FilterRegistrationBean，将自定义的xssFilter注册进去

b、在自定义XssFilter中，并且将封装好的过滤器添加到FilterChain

c、关键是XssHttpServletRequestWrapper的实现方式，继承servlet的HttpServletRequestWrapper，并重写相应的几个有可能带xss攻击的方法

HTMLFilter

采用Java实现的开源类库。基于filter拦截,将特殊字符替换为html转意字符, 需要拦截的点如:请求头 requestHeader、请求体 requestBody、请求参数 requestParameter

SQLFilter

防止sql注入风险。

12 Hibernate-validator

hibernate Validator 是 Bean Validation 的参考实现 。Hibernate Validator 提供了 JSR 303 规范中所有内置 constraint 的实现，除此之外还有一些附加的 constraint。

在日常开发中，Hibernate Validator经常用来验证bean的字段，基于注解，方便快捷高效。

1. Bean Validation 中内置的 constraint

注解	作用
@Valid	被注释的元素是一个对象，需要检查此对象的所有字段值
@Null	被注释的元素必须为 null
@NotNull	被注释的元素必须不为 null
@AssertTrue	被注释的元素必须为 true
@AssertFalse	被注释的元素必须为 false
@Min(value)	被注释的元素必须是一个数字，其值必须大于等于指定的最小值
@Max(value)	被注释的元素必须是一个数字，其值必须小于等于指定的最大值
@DecimalMin(value)	被注释的元素必须是一个数字，其值必须大于等于指定的最小值
@DecimalMax(value)	被注释的元素必须是一个数字，其值必须小于等于指定的最大值
@Size(max, min)	被注释的元素的大小必须在指定的范围内
@Digits (integer, fraction)	被注释的元素必须是一个数字，其值必须在可接受的范围内
@Past	被注释的元素必须是一个过去的日期
@Future	被注释的元素必须是一个将来的日期
@Pattern(value)	被注释的元素必须符合指定的正则表达式

12 Hibernate-validator

2. Hibernate Validator 附加的 constraint

注解	作用
@Email	被注释的元素必须是电子邮箱地址
@Length(min=, max=)	被注释的字符串的大小必须在指定的范围内
@NotEmpty	被注释的字符串的必须非空
@Range(min=, max=)	被注释的元素必须在合适的范围内
@NotBlank	被注释的字符串的必须非空
@URL(protocol=, host=, port=, regexp=, flags=)	被注释的字符串必须是一个有效的url
@CreditCardNumber	被注释的字符串必须通过Luhn校验算法，银行卡，信用卡等号码一般都用Luhn计算合法性
@ScriptAssert (lang=, script=, alias=) ("Scripting for the JavaTM Platform")的实现	要有Java Scripting API 即JSR 223
@SafeHtml (whitelistType=, additionalTags=)	classpath中要有jsoup包

hibernate补充的注解中，最后3个不常用，可忽略。

主要区分下@NotNull @NotEmpty @NotBlank 3个注解的区别：

@NotNull	任何对象的value不能为null
@NotEmpty	集合对象的元素不为0，即集合不为空，也可以用于字符串不为null
@NotBlank	只能用于字符串不为null，并且字符串trim()以后length要大于0

13 Velocity代码生成器

Velocity是一种Java模版引擎技术。它允许任何人仅仅简单的使用模板语言来引用由 Java 代码定义的对象，
从而实现界面和 Java 代码的分离，使得界面设计人员可以和 Java 程序开发人员同步开发一个遵循 MVC 架构的 web 站点。
但是在实际应用过程中，Velocity 又不仅仅被用在了 MVC 的架构中。

1. Web 应用：开发者在不使用 JSP 的情况下，可以用 Velocity 让 HTML 具有动态内容的特性。
2. 源代码生成：Velocity 可以被用来生成 Java 代码、SQL 或者 PostScript。有很多开源和商业开发的软件是使用 Velocity 来开发的。
3. 自动 Email：很多软件的用户注册、密码提醒或者报表都是使用 Velocity 来自动生成的。使用 Velocity 可以在文本文件里面生成邮件内容，而不是在 Java 代码中拼接字符串。
4. 转换 xml：Velocity 提供一个叫 Anakia 的 ant 任务，可以读取 XML 文件并让它能够被 Velocity 模板读取。一个比较普遍的应用是将 xdoc 文档转换成带样式的 HTML 文件。

模板语言语法使用
1. 变量定义
#set($name =“velocity”)
2. 变量的使用
${person.name}
3. 变量赋值
#set($foo = $bar)
4. 循环
#foreach($element in $list)
 This is $element
 $velocityCount
#end

13 Velocity代码生成器

5. 条件语句
#if(condition)
...
#elseif(condition)
…
#else
…
#end
6. 关系操作符
Velocity 引擎提供了 AND、OR 和 NOT 操作符，分别对应&&、||和! 例如：
#if($foo && $bar)
#end
7. 宏
Velocity 中的宏可以理解为函数定义。
#macro(macroName arg1 arg2 …)
...
#end
8. 文件引入
#parse 和 #include 指令的功能都是在外部引用文件，而两者的区别是，
#parse 会将引用的内容当成类似于源码文件，会将内容在引入的地方进行解析，
#include 是将引入文件当成资源文件，会将引入内容原封不动地以文本输出。

13 Velocity代码生成器

代码生成器使用

代码生成器是根据表结构，自动生成相应的代码，需先在MySQL中建好表结构，再使用代码生成器

代码生成器是通过velocity模板实现的，依赖velocity所需jar包，可在线生成entity、xml、dao、service、page、js代码的zip压缩文件

修改包名、作者、作者邮箱，需在generator.properties 中配置

如需去掉表tb_user 前缀tb_ ，可配置tablePrefix 项，如： tablePrefix=tb_ ，则生成的实体类为UserEntity ，否则生成TbUserEntity

可根据自己的需求，自行修改模板，模板代码位置：【resources\template】

模板数据封装在如下所示map里，其中tableEntity 对象为TableEntity.java 实例， config 为generator.properties 配置文件数据

13 Velocity代码生成器

```
//模板数据
Map<String, Object> map = new HashMap<>();
map.put("tableName", tableEntity.getTableName());
map.put("comments", tableEntity.getComments());
map.put("pk", tableEntity.getPk());//数据库主键，没有则为第一个字段
map.put("className", tableEntity.getClassName());
map.put("classname", tableEntity.getClassname());
map.put("pathName", tableEntity.getClassname().toLowerCase());
map.put("columns", tableEntity.getColumns());
map.put("package", config.getString("package"));
map.put("author", config.getString("author"));
map.put("email", config.getString("email"));
map.put("datetime", DateUtils.format(new Date(), DateUtils.DATE_TIME_PATTERN));
```

MySQL数据类型与Java数据类型转换，在generator.properties 中配置，如有些类型的转换关系不存在，则需在generator.properties 中添加，如： bigint=Long

生成的html、js代码，需要修改html代码里的js路径，避免404错误

添加相应的菜单即可【权限标识一定要添加，不然没权限访问接口】

13 Velocity代码生成器

演示快速开发一个模块步骤：

1、手动创建一张表（需要添加comments）

2、登录系统并且使用自动生成代码模块

3、将生成的java代码放入对应的工程目录中

4、将生成的js文件放入对应工程目录中

5、将生成的html文件放入对应工程目录中，并修改加载js文件目录路径

6、使用系统中的[菜单管理]-->新增菜单-->sys/config.html

7、使用系统中的[菜单管理]-->新增按钮-->依次添加按钮【新增】、【修改】、【删除】、【查看】

并且进行授权，多个权限用逗号,隔开（或者使用生成的sql脚本插入数据库）

14 权限设计

1、权限？

权限管理往往是一个极其复杂的问题，但也可简单表述为这样的逻辑表达式：判断“Who对What(Which)进行How的操作”的逻辑表达式是否为真。

针对不同的应用，需要根据项目的实际情况和具体架构，在维护性、灵活性、完整性等N多个方案之间比较权衡，选择符合的方案。

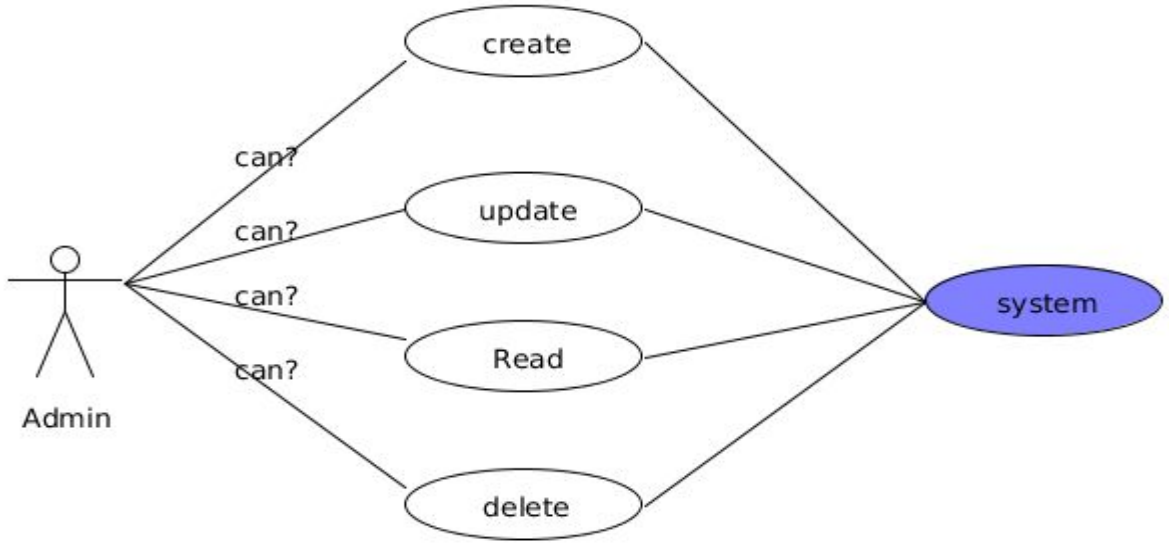
2、目标：

直观，因为系统最终会由最终用户来维护，权限分配的直观和容易理解，显得比较重要简单，

包括概念数量上的简单和意义上的简单还有功能上的简单。

想用一个权限系统解决所有的权限问题是不现实的。

设计中将常常变化的“定制”特点比较强的部分判断为业务逻辑，而将常常相同的“通用”特点比较强的部分判断为权限逻辑就是基于这样的思路。



14 权限设计

3、思想：

权限系统的核心由以下三部分构成：1.创造权限，2.分配权限，3.使用权限，

然后，系统各部分的主要参与者对照如下：

1.创造权限 - Creator创造，一个子系统或称为模块，应该有哪些权限。

2.分配权限 - Administrator 分配，如，创建角色，创建用户组，给用户组分配用户，将用户组与角色关联等等...这些操作都是由 Administrator 来完成的。

3.使用权限 - User：就是什么权限可以访问什么资源

14 权限设计

4、常见的角色模型：

A一般简单的角色模型为：用户 -> 权限

B最常用的角色模型为：用户-〉角色-〉权限

C较复制的关系模型包括了用户、部门、角色、权限、模块

14 权限设计

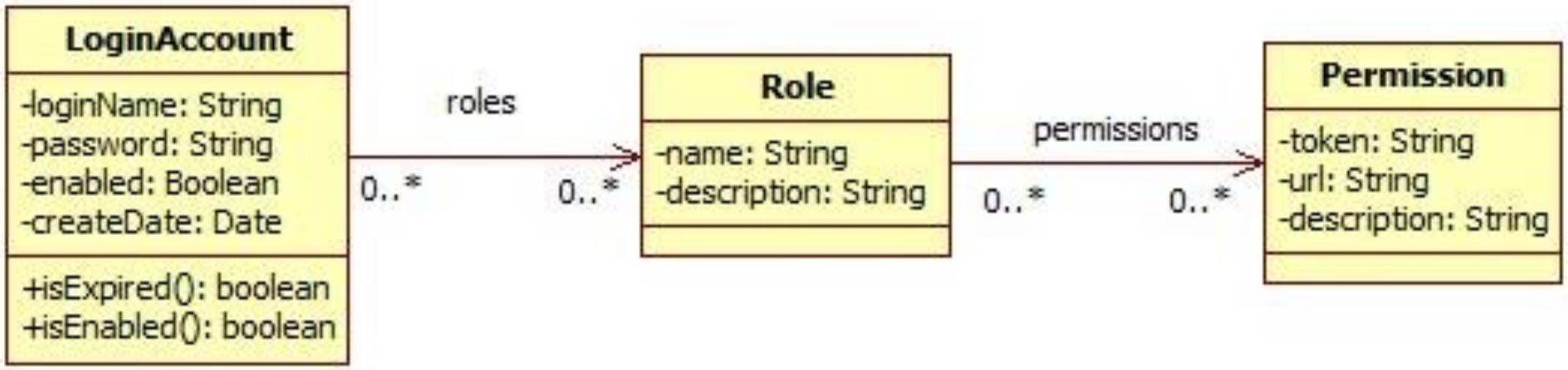
认识用户权限模型：

这里所提到用户权限模型，指的是用来表达用户信息及用户权限信息的数据模型。即能证明“你是谁？”、“你能访问多少受保护资源？”。为实现一个较为灵活的用户权限数据模型，通常把用户信息单独用一个实体表示，用户权限信息用两个实体表示。

1、用户信息用 LoginAccount 表示，最简单的用户信息可能只包含用户名 loginName 及密码 password 两个属性。实际应用中可能会包含用户是否被禁用，用户信息是否过期等信息。

2、用户权限信息用 Role 与 Permission 表示，Role 与 Permission 之间构成多对多关系。Permission 可以理解为对一个资源的操作，Role 可以简单理解为 Permission 的集合。

3、用户信息与 Role 之间构成多对多关系。表示同一个用户可以拥有多个 Role，一个 Role 可以被多个用户所拥有。



14 权限设计

数据库设计：

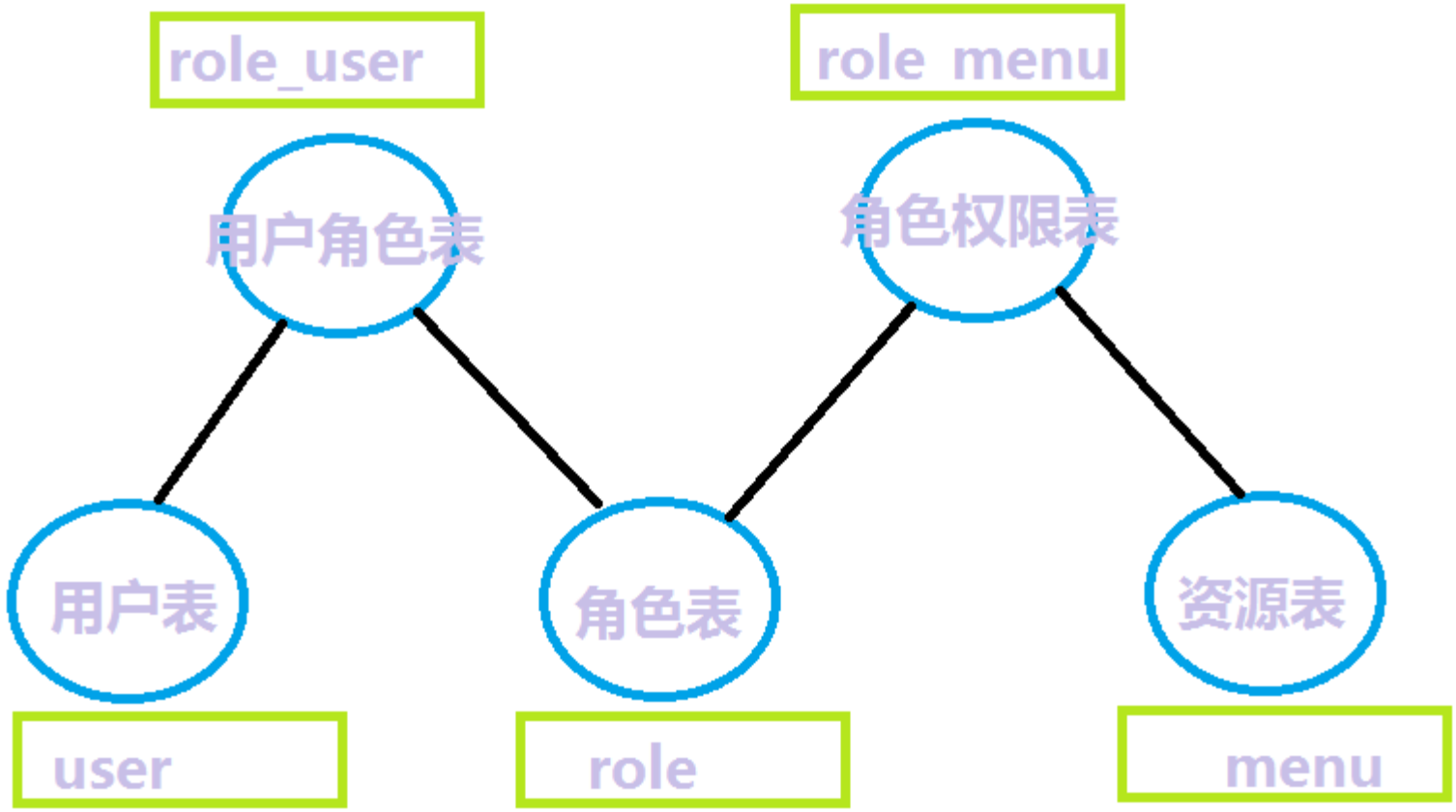
sys_menu定义目录、菜单、按钮（权限许可范围）

sys_role定义角色信息

sys_role_menu存放角色与菜单对应关系

sys_user定义系统用户

sys_user_role用户与角色对应关系



15 权限相关功能实现-菜单管理

管理系统的目录、菜单、按钮资源，提供新增、修改、删除功能。

目录：指的是一个抽拉项目，即【系统管理】
菜单：目录下的菜单项，即导航栏【角色列表】
按钮：页面的具体功能，新增、修改、删除、暂停、恢复等

添加顺序：
目录→菜单→按钮

相关表：
sys_menu

15 权限相关功能实现-菜单管理

注意：
【权限标识一定要在这里添加，不然没权限访问接口；多个权限的情况，用逗号,分隔】
如：【管理员列表】：
查看-->sys:schedule:list,sys:schedule:info
新增-->sys:schedule:save
修改-->sys:schedule:update
删除-->sys:schedule:delete
暂停-->sys:schedule:pause
恢复-->sys:schedule:resume
立即执行-->sys:schedule:run
日志列表-->sys:schedule:log

15 权限相关功能实现-角色管理

定义系统角色，并且对访问资源进行授权。提供新增、修改、删除功能。

相关表：
sys_role
sys_role_menu

15 权限相关功能实现-用户管理

管理系统人员帐号信息，并提供新增、修改、删除功能。帐号启用、禁用。
可以绑定系统角色。

关联表：
sys_user
sys_user_role

16 资料和代码

注意：代码和资料在视频课程最后章节。
请在电脑浏览器上进行下载！

感谢您的参与!

主讲：安燚