# SpringBoot

## Spring Boot介绍

1. Spring Boot的目的在于创建和启动新的基于Spring框架的项目。Spring Boot会选择最合适的Spring子项目和第三方开源库进行整合。大部分Spring Boot应用只需要非常少的配置就可以快速运行起来。
2. Spring Boot是伴随着Spring4.0诞生的，Spring Boot提供了一种快速使用Spring的方式。

## Spring Boot特点

1. 为基于Spring的开发提供更快的入门体验
2. 创建可以独立运行的Spring应用
3. 直接嵌入Tomcat或Jetty服务器，不需要打包成WAR文件
4. 提供推荐的基础POM（starter）来简化Apache Maven配置
5. 提供可以直接在生产环境中使用的功能，如性能指标、应用信息和应用健康检查
6. 尽可能的根据项目依赖来自动配置Spring框架
7. 开箱即用，没有代码生成，无需XML配置。同时也可以修改默认值来满足特定的需求
8. 其它大量的项目都是基于Spring Boot之上，如Spring Cloud。

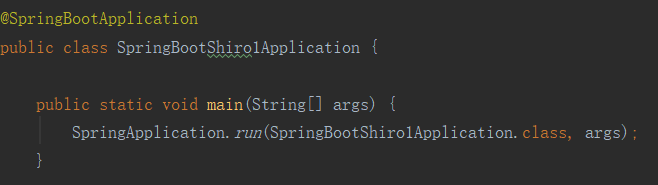
## Spring Boot缺点

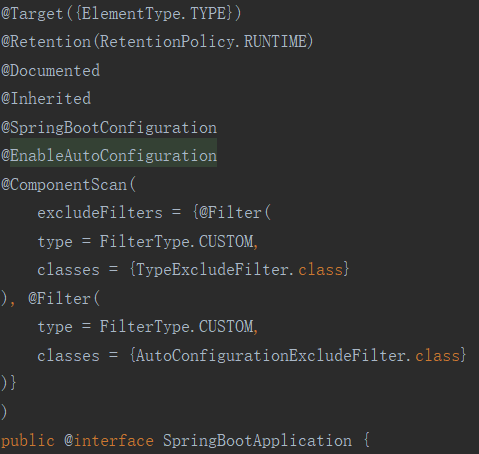
1. 依赖太多，随便的一个Spring Boot应用都有几十M
2. 确少服务的注册和发现等解决方案
3. 缺少监控集成方案、安全管理方案

## Spring Boot应用场景

1. Spring能够应用的场景
2. Java Web应用
3. 微服务

# SpringBoot入口

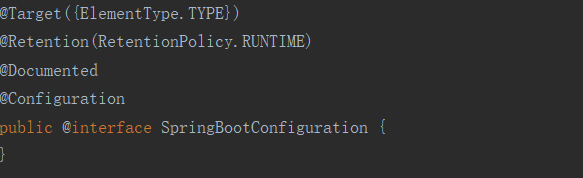




SpringBoot入口类都使用@SpringBootApplication标注，其实这也是一个配置类。查看@SpringBootApplication我们可以发现，它继承了多个注解，其中有三个核心注解：

1. @SpringBootConfiguration
2. @ComponentScan
3. @EnableAutoConfiguration

## @SpringBootConfiguration



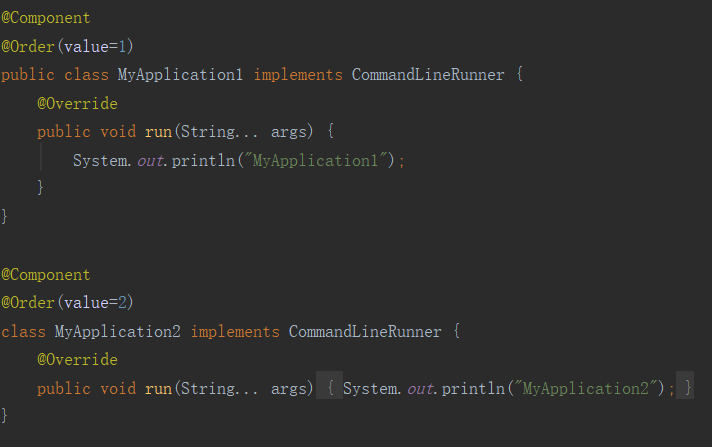
1. 进入SpringBootConfiguration可以发现它继承了@Configuration，也是一个配置类。所以在入口类下配置@Bean。

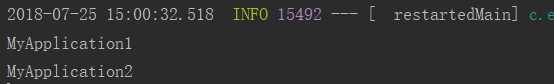
## @ComponentScan

1. 从自定义路径中找出标志需要装配的类(@Component标志)，自动装配到Bean容器中。
2. @Controller，@Service，@Repository注解都继承了@Component,所以会和@Component标志的类一起装配到Bean容器。

## @EnableAutoConfiguration

# SpringBootApplication启动加载



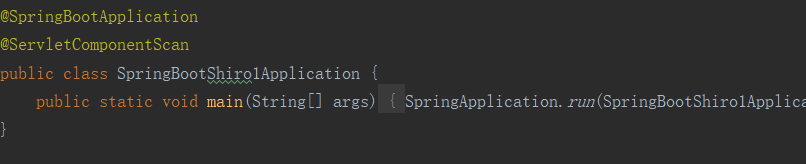


1. 我们希望把某些数据在项目启动的时候就添加进去，可以将相关类实现CommandLineRunner接口，重写它的run()方法,并使用@Order(value=x)配置加载顺序。

# Servlet

虽然Controller可以满足我们绝大部分的需求，但是如果我们要拓展Controller的功能或Controller不适合需求时，可能需要使用到Servlet满足我们的需求。下面通过注解式实现页面访问Servlet。





SpringBoot的核心控制器DispatcherServlet会处理所有的请求，所以我们需要通过注册来让核心控制器知道这个Servlet存在以及它所处理的URL：

1. 创建一个Servlet,继承HttpServlet，写上注解@WebServlet(“url”)
2. 在项目启动类注解@ServletCompentScan

# Filter

管理服务器Web资源，实现URL级别上的权限访问控制，同时可以对请求做预处理。



1. 创建过滤器，实现Filter接口
2. 重写doFilter方法，最后做do.Filter(req,resp)放行操作。

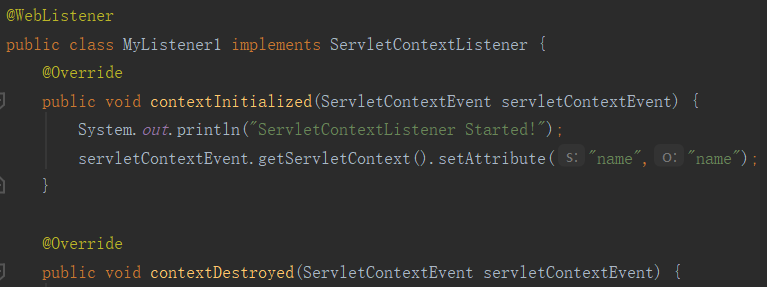
# Listener

## Application域

SpringBoot提供了对Application域的监听机制，监听内容包括Application的创建和销毁、Application域属性的添加、修改、删除。

### Application的创建和销毁

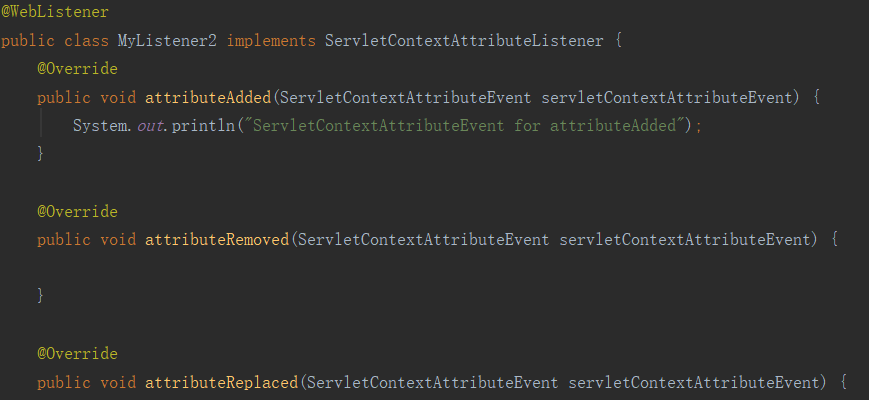
在一些业务场景中，当容器初始化完成之后，需要处理一些操作，比如一些数据的加载、初始化缓存、特定任务的注册等等。



1. 创建MyListener1，实现ServletContextListener接口，覆盖里面的两个方法：contextInitialized(…)、contextDestroyed(…)。
2. 通过@WebListener注册监听器。

### Application域属性的添加、修改、删除

在一些业务场景中，我们可能需要对应用的域属性进行监管，方便日后维护追踪。



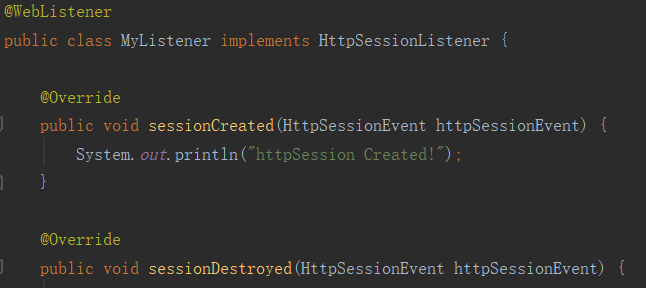
1. 创建MyListener2，实现ServletContextAttributeListener接口，覆写里面的三个方法。
2. 通过@WebListener注册监听

## Session域

SpringBoot提供了对Session域的监听机制，监听内容包括Session的创建和销毁、Session域属性的添加、修改、删除。

### Session的创建和销毁

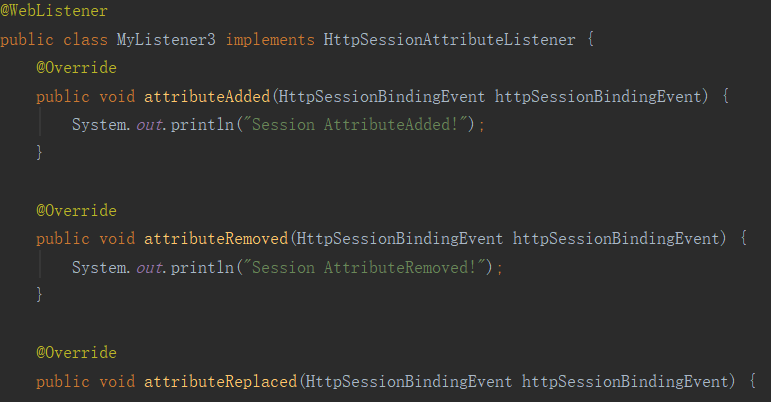
应用场景中，我们可能需要在Session创建时添加某些统一的操作，这时使用监听机制可以很方便的实现。



1. 创建MyListener，实现HttpSessionListener接口，覆写接口的两个方法。
2. 通过@WebListener注册监听器

### Session域属性的添加、修改和删除

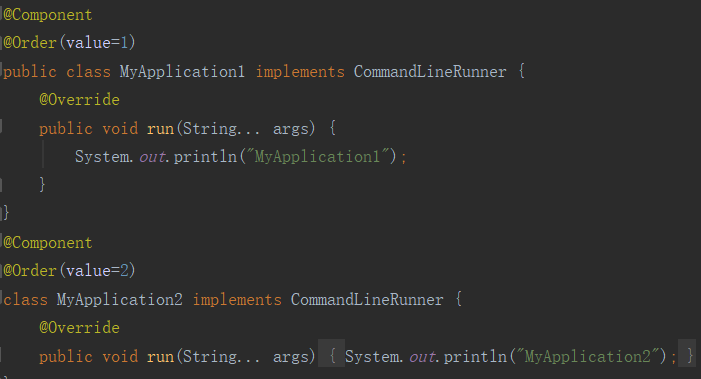
如果Session的域属性发生了改变，我们可以在log下找到具体改变内容，方便日后的维护追踪。



1. 创建MyListener3，实现HeepSessionAttributeListener接口，覆写三个方法。
2. 通过@WebListener注册监听。

# SpringBoot启动加载

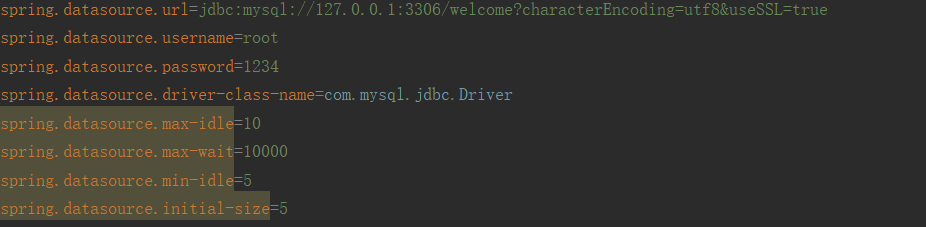
如果我们需要在项目启动完成之后自动进行某些操作，这时候我们可以通过SpringBoot的启动加载功能实现。



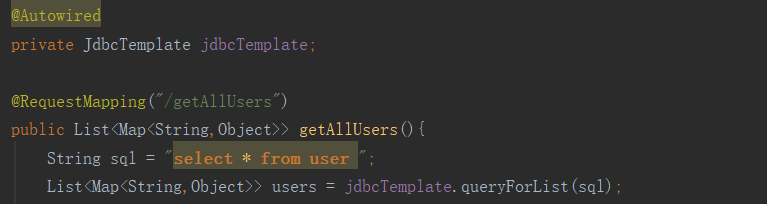
1. 创建MyApplication1和MyApplication2，实现CommandLineRunner接口，覆写run()方法。
2. 通过@Order(value=x)来设置CommandLineRunner实现类的加载顺序，通过@Component注册。

# JDBC访问数据库

1. 配置文件

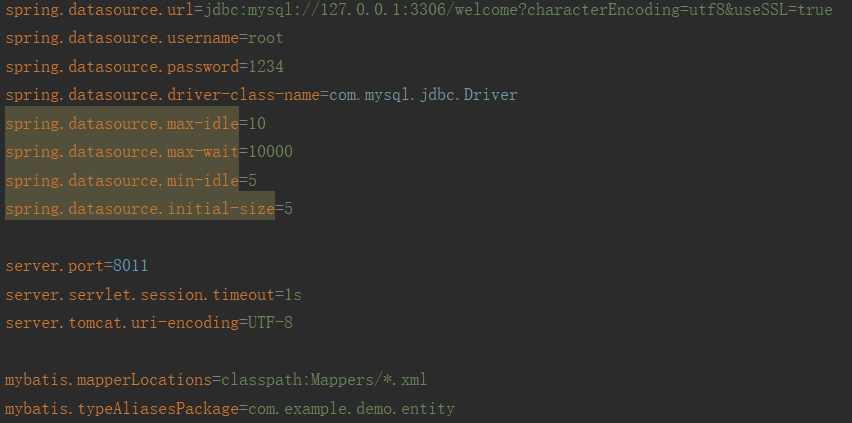


1. 通过JDBCTemplate访问数据库

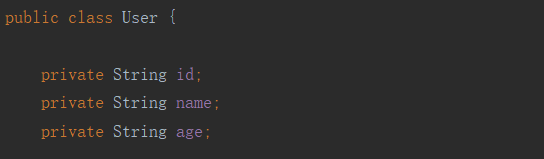


# Mybatis数据库连接

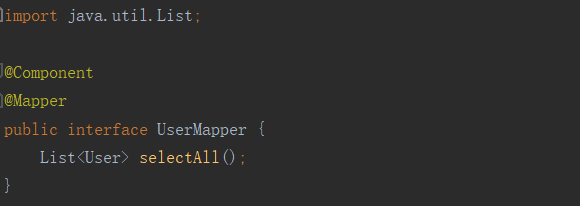
1. 首先配置数据库连接、Mybatis连接



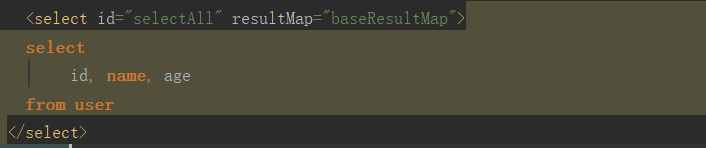
1. 创建实体类



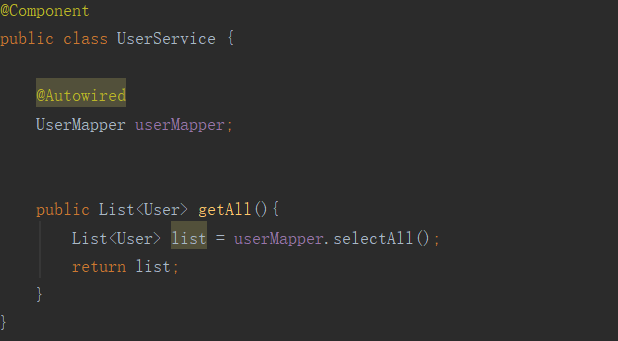
1. 创建Mapper接口



1. 写配置文件UserMapper.xml



1. 访问数据库



Mybatis本身是对JDBC的封装。相对JDBC来说，优点有：

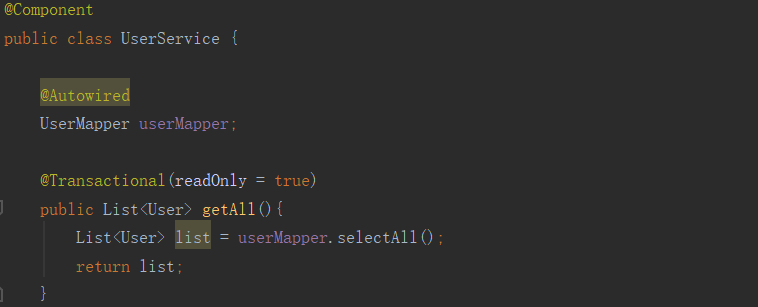
1. 将数据库连接进行统一管理，不必频繁创建和销毁连接，从而节省资源。
2. 将数据库的SQL语句都放在配置文件中统一管理，不必像JDBC那样分布在各个JAVA类中。改善可读性，易于维护。
3. JDBC访问数据库需要实现每个DAO层的接口，由于不同程序员的编码风格不一致从而导致代码可读性以及健壮性不如Mybatis。

# 事务控制

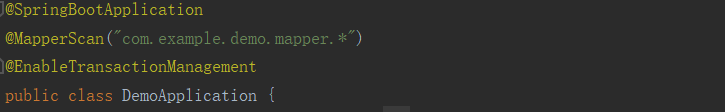
在业务场景一中，往往一个业务需要多条SQL语句协同完成。如果部分SQL执行成功，其它SQL执行失败了，我们会希望执行成功的那部分SQL能回滚，维持数据的正确性。SpringBoot的事务控制能将这多条SQL组成一个事务，仅当所有SQL都成功执行时才提交，否则回滚。

在业务场景二中，我们像对某些事物添加不一样的权限控制，比如仅可读、超时时间等，可以使用事务控制进行设置。

1. 在业务逻辑层的类中添加@Transaction注解，并添加权限控制。



1. 在应用入口类中添加@EnableTransactionManager注解，管理事务。

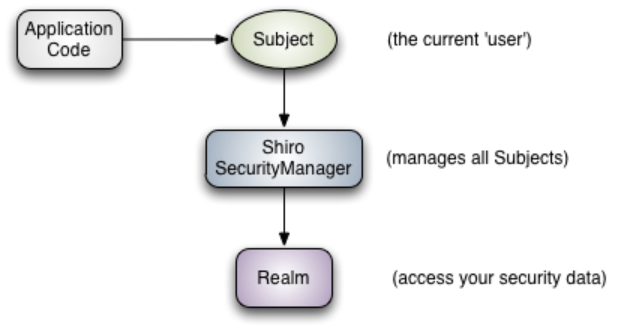


# Shiro权限管理

Apache Shiro是Java的一个安全框架。它相当简单，对比Spring Security，可能没有Spring Security做的功能强大，但是在实际工作时可能并不需要那么复杂的东西。

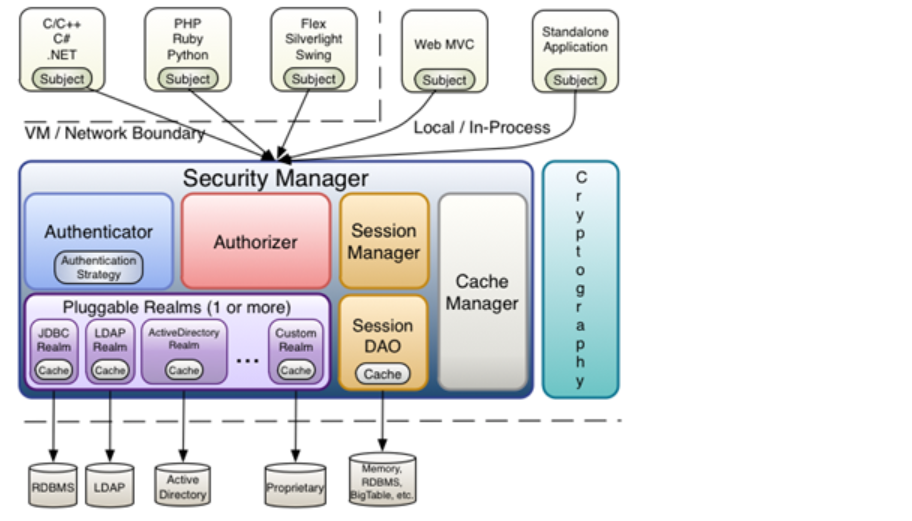
Shiro的一个重要思想是：不同subject具有它自己的角色，而不同的角色具有它自己的权限。所有Shiro只需要对权限进行访问控制即可。

## Shiro外部框架



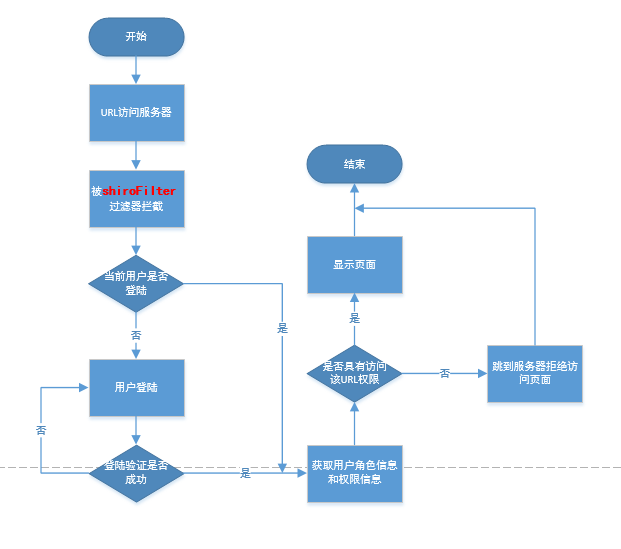
1. Subject：主体，代表当前“用户”，可以是人、网络爬虫、机器人等。
2. Shiro SecurityManage：安全管理器，即所有与安全有关的操作都会与SecurityManager交互，且它管理着所有Subject。它是Shiro的核心，它负责与后边介绍的其他组件进行交互。
3. Reaml：域，Shiro从从Realm获取安全数据（如用户、角色、权限），就是说SecurityManager要验证用户身份，那么它需要从Realm获取相应的用户进行比较以确定用户身份是否合法；也需要从Realm得到用户相应的角色/权限进行验证用户是否能进行操作。

## Shiro的内部框架



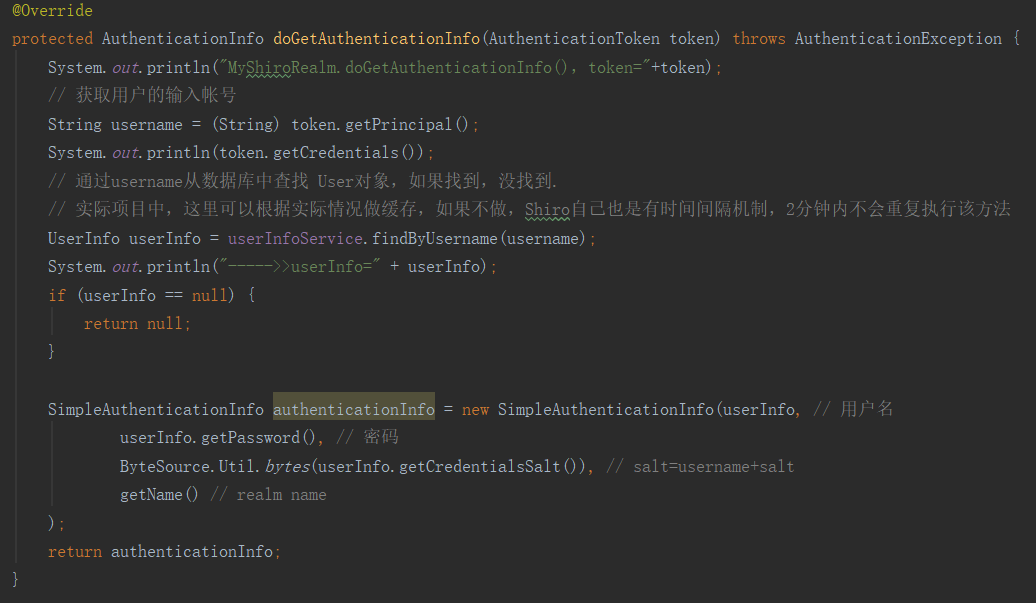
上图可以看出Shiro的组成。其中有身份验证器、权限访问控制器、会话管理、缓存管理、密码模块等。

## Shiro逻辑图



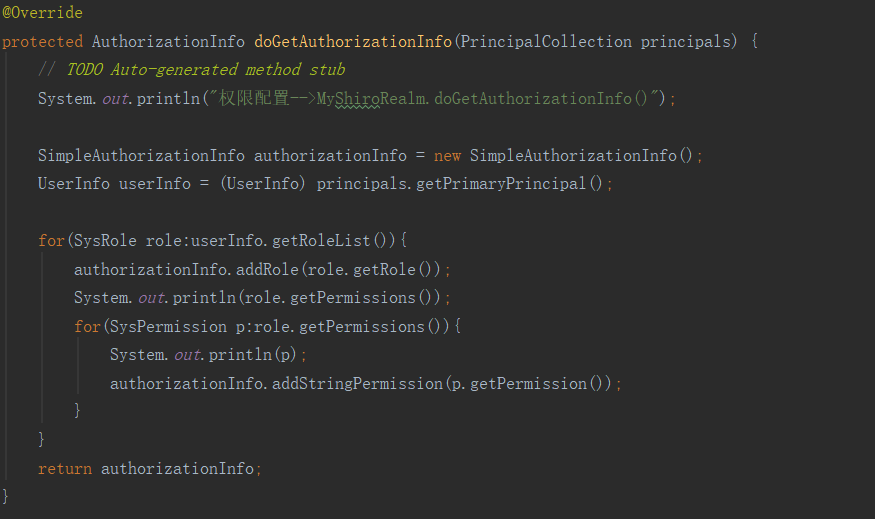
1. Shiro对部分/全部URL进行过滤拦截，检查是否登陆。
2. 当登陆后的角色访问需要权限的界面时，会检查当前角色的权限。若无权限则访问失败。

## ShiroReaml身份认证



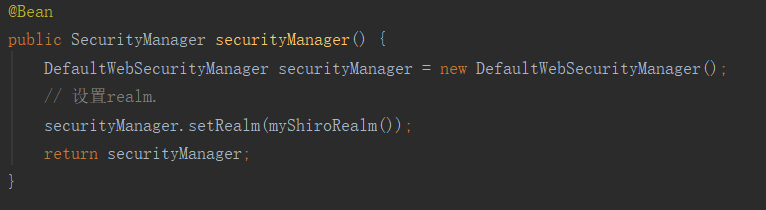
1. 请求访问服务器时，满足过滤器过滤规则的页面，会被一个过滤器拦截下来，检查是否登陆，如果未登录，则跳转到登陆界面
2. 根据用户在页面输入的账号密码，去数据库查找比对。若登陆成功，跳转到登陆成功界面。

## ShiroReaml权限控制



1. 用户访问需要权限的页面时，调用权限控制器。
2. 权限控制器获取当前用户信息，将具有的权限赋给AuthorizationInfo对象，交给Shiro统一管理。
3. 如果该用户具有相应权限，则成功访问该URL页面。

## Shiro安全管理器



1. 安全控制器聚集了ShiroReaml对象，它还可以聚集其它安全机制。
2. 对访问URL的用户进行统一的安全管理。

## 密码安全

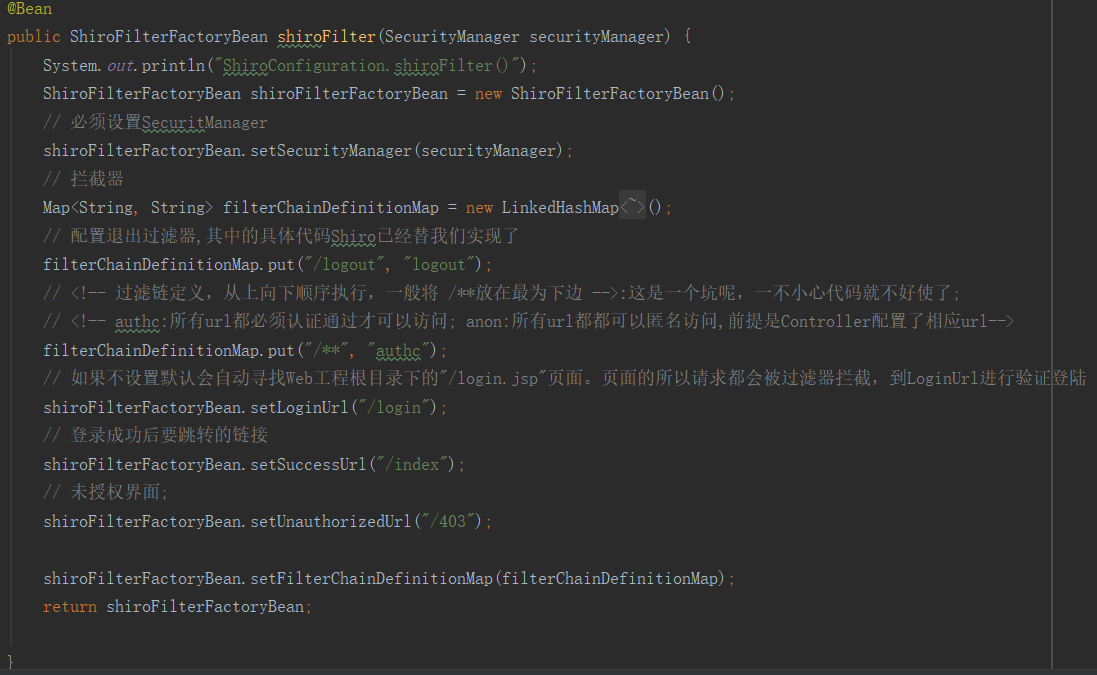
特点：对数据源进行加密，并且易于使用。常用的加密算法为MD5。

## 会话管理



1. Session是一个Shiro特定的实例，它提供了大多数惯常使用的HttpSession实例操作。
2. 我们可以通过Session来进行会话管理。注意，Session可以依赖HTTP环境，也可以在非Web应用程序内运行。它不是必须要HTTP环境。

## Shiro过滤器



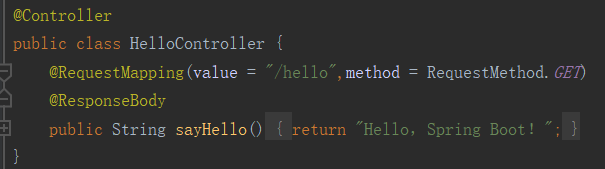
1. SpringBoot的过滤器主要有authc、authcbasic、anon、perms、port、rest、roles、ssl、user。
2. 用户访问服务器时，首先会调用authc过滤器，判断是否具有权限。若没有，则到登陆界面进行登陆。

# 往页面返回一个字符串

1. 方式一



1. 方式二



# Swagger

1. 是一款让你更好的书写API文档的规范且完整框架。
2. 提供描述、生产、消费和可视化RESTful Web Service。
3. 是由庞大工具集合支撑的形式化规范。这个集合涵盖了从终端用户接口、底层代码库到商业API管理的方方面面。

# @ConfigurationProperties

作用：将实体类通过配置文件的形式赋值。

* 创建Bean

@Component

@ConfigurationProperties(prefix = "demo.book")

@Validated

@RestController

public class BookComponent {

    @NotEmpty

private String name;

    @NotNull

    private String writer;

    @RequestMapping(value="/BookName",method= RequestMethod.GET)

    public String getName() {

        return name + " with " +this.getWriter();

    }

    public void setName(String name) {

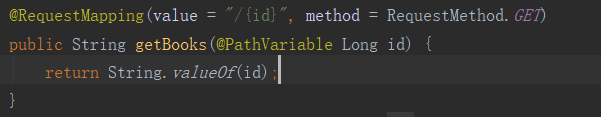
        this.name = name;

    }

* 配置配置文件(application.properties)



# @PathVariable注解作用



1. 通过 @PathVariable 可以将 URL 中的相应参数绑定到控制器对应的方法参数中。

# JPA(Java Persistence API)

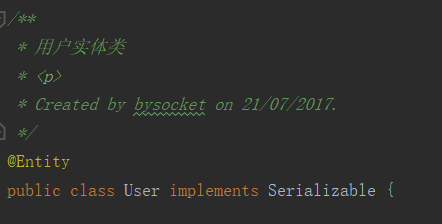
1. JPA(Java Persistence API)是Sun官方提出的Java持久化规范。它为Java开发人员提供了一种对象/关联映射工具来管理Java应用中的关系数据。他的出现主要是为了简化现有的持久化开发工作和整合ORM技术，结束现在Hibernate，TopLink，JDO等ORM框架各自为营的局面。值得注意的是，JPA是在充分吸收了现有Hibernate，TopLink，JDO等ORM框架的基础上发展而来的，具有易于使用，伸缩性强等优点。从目前的开发社区的反应上看，JPA受到了极大的支持和赞扬，其中就包括了Spring与EJB3.0的开发团队。
2. spring data jpa让我们解脱了DAO层的操作，基本上所有CRUD都可以依赖于它来实现

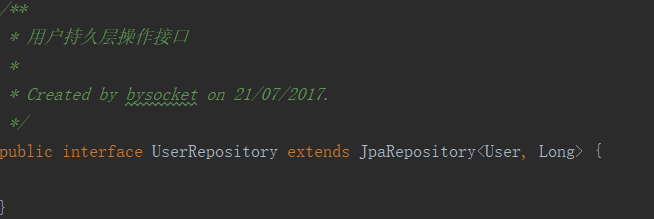
## 查询

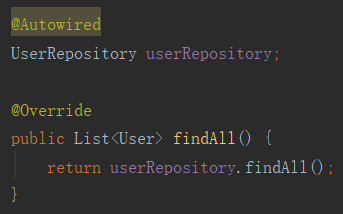
### 简单查询

#### 基本查询

1. spring data jpa 默认预先生成了一些基本的CURD的方法，调用API即可直接使用。例如：增、删、查等等。注意：Entity类要序列化。







#### 自定义简单查询

1. 自定义的简单查询就是根据方法名来自动生成SQL，主要的语法是findXXBy,readAXXBy,queryXXBy,countXXBy, getXXBy，By后面跟属性名称：



### 复杂查询

在实际的开发中我们需要用到分页、删选、连表等查询的时候就需要特殊的方法或者自定义SQL。<https://www.cnblogs.com/ityouknow/p/5891443.html>

#### 分页查询

#### 限制查询

#### 自定义SQL查询

#### 多表查询

## 多数据源的支持

### 同源数据库的多源支持

日常项目中因为使用的分布式开发模式，不同的服务有不同的数据源，常常需要在一个项目中使用多个数据源，因此需要配置sping data jpa对多数据源的使用，一般分一下为三步：

1. 配置多数据源
2. 不同源的实体类放入不同包路径
3. 声明不同的包路径下使用不同的数据源、事务支持

这里有一篇文章写的很清楚：https://www.jianshu.com/p/34730e595a8c

### 异构数据库多源支持

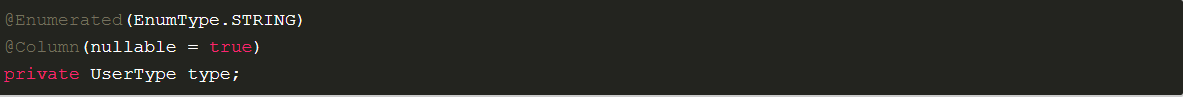
比如我们的项目中，即需要对mysql的支持，也需要对mongodb的查询等。

实体类声明@Entity 关系型数据库支持类型、声明@Document 为mongodb支持类型，不同的数据源使用不同的实体就可以了。

### 其它

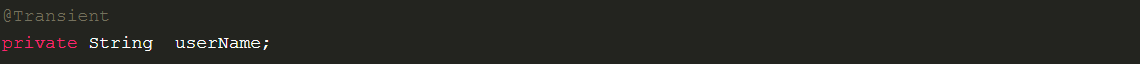
#### 枚举

1. 使用枚举的时候，我们希望数据库中存储的是枚举对应的String类型，而不是枚举的索引值，需要在属性上面添加@Enumerated(EnumType.STRING) 注解。



1. 不需要和数据库映射的属性：

正常情况下我们在实体类上加入注解@Entity，就会让实体类和表相关连如果其中某个属性我们不需要和数据库来关联只是在展示的时候做计算，只需要加上@Transient属性既可。



#### 源码案例

这里有一个开源项目几乎使用了这里介绍的所有标签和布局，大家可以参考：

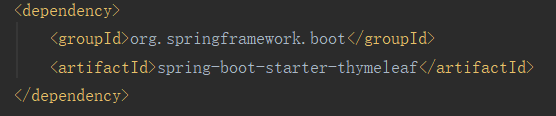
<https://github.com/cloudfavorites/favorites-web>

# Thymeleaf

1. Thymeleaf是Web和独立环境的现代服务器端Java模板引擎，能够处理HTML，XML，JavaScript，CSS甚至纯文本。
2. Thymeleaf将静态页面和动态数据分离，使得在没有网络的条件下也能浏览静态页面。
3. 如果没有在pom下引入thymeleaf，resource/template下的html页面是无法被显示到浏览器的，原因是应用程序无法创建Thymeleaf模板引擎。

## 相关说明

1. 引入依赖包



1. 在html引入命名空间，不加命名空间对Thymeleaf的功能没有任何影响，但编辑器出现html验证错误。



1. 页面使用内置标签



## 静态页面的访问

1. 访问 /resource/static 页面下的html文件，在URL直接输入html文件名即可。

## 动态页面的访问

1. 访问 /resource/templates 下的html文件需要在Controller方法中return文件路径。

## EL

1. 支持EL表达式取域中的值。



## OGNL

1. 支持OGNL取得域中的值，但是需要引入依赖包。（待验证）



# Spring注解

## @SpringBootApplication

1. 申明让spring boot自动给程序进行必要的配置，这个配置等同于：@Configuration ，@EnableAutoConfiguration和 @ComponentScan 三个配置。

## @Configuration

1. 相当于传统的xml配置文件，如果有些第三方库需要用到xml文件，建议仍然通过@Configuration类作为项目的配置主类——可以使用@ImportResource注解加载xml配置文件。

## @ImportResource

1. 用来加载xml配置文件。

## @EnableAutoConfiguration

1. SpringBoot自动配置（auto-configuration）：尝试根据你添加的jar依赖自动配置你的Spring应用。例如，如果你的classpath下存在HSQLDB，并且你没有手动配置任何数据库连接beans，那么我们将自动配置一个内存型（in-memory）数据库”。你可以将@EnableAutoConfiguration或者@SpringBootApplication注解添加到一个@Configuration类上来选择自动配置。如果发现应用了你不想要的特定自动配置类，你可以使用@EnableAutoConfiguration注解的排除属性来禁用它们。

## @Componen

1. 泛指组件，当组件不好归类（Service、Dao等）的时候，我们可以使用这个注解进行标注。

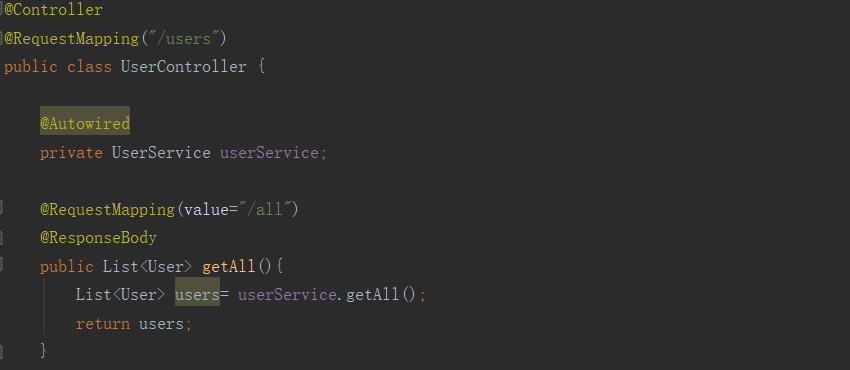
## @ComponentScan

1. 表示将该类自动发现扫描组件。如果扫描到有@Component、@Controller、@Service等这些注解的类，并注册为Bean，结合@Autowired注解注入。

## @ GetMapping

1. 是@RequestMapping(method = RequestMethod.GET)的缩写。

## @ResponseBody、@Controller、@RequestMapping



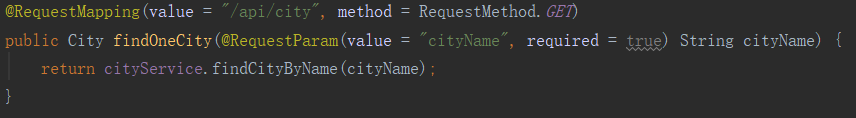
1. @ResponseBody表示将返回值直接写回HTTP response body中，一般在进行异步获取数据的时候使用。
2. @RequestMapping提供路由信息，负责URL到Controller中的具体函数的映射。在使用@RequestMapping后，返回值通常解析为跳转路径，加上@esponsebody后返回结果不会被解析为跳转路径，而是直接写入HTTP response body中。比如异步获取json数据，加上@Responsebody后，会直接返回json数据。
3. @Controlle用于定义控制器类，在spring项目中由控制器负责将用户发来的URL请求转发到对应的服务接口（service层），一般这个注解在类中，通常方法需要配合注解@RequestMapping。

## @WebServlet

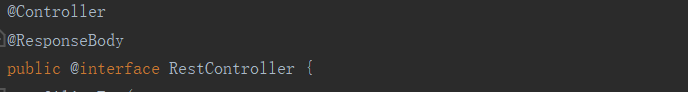
1. 将该类注册为Servlet,处理URL请求。作用和@Controller差不多，只是@Conroller是方法级别上的更细粒度的控制，而@WebServlet属于类级别粒度控制。

## @RequestParam

1. 获取请求参数的值
2. RequestParam可以接受简单类型的属性，也可以接受对象类型。
3. 在GET请求中，不能使用@RequestBody。
4. 在POST请求，可以使用@RequestBody和@RequestParam，但是如果使用@RequestBody，对于参数转化的配置必须统一。
5. 在方法体中使用，能限定方法参数。比如说是否必须传递这个参数等。



## @RestController



1. 用于标注控制层组件(如struts中的action)，@ResponseBody和@Controller的合集。

## @Autowired

1. 自动导入依赖的bean。byType方式。把配置好的Bean拿来用，完成属性、方法的组装，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。当加上（required=false）时，就算找不到bean也不报错。

## @Resource

1. 自动注入bean。默认按byName自动注入。当byName方式找不到时，则按照类型查找。可以通过设置注解的type属性来指定查找方式。

## @Qualifier



1. 当有多个同一类型的Bean时（一个接口有多个实现类，按照接口类型注入），可以用@Qualifier(“name”)来指定。与@Autowired配合使用。@Qualifier限定描述符除了能根据名字进行注入，但能进行更细粒度的控制如何选择候选者

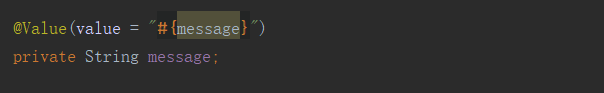
## @Bean

1. 用@Bean标注方法等价于XML中配置的bean。
2. 放在方法的上面，而不是类，意思是产生一个bean,并交给spring管理。
3. 默认是单例的。

## @Import

1. 用来导入其他配置类。

## @Value



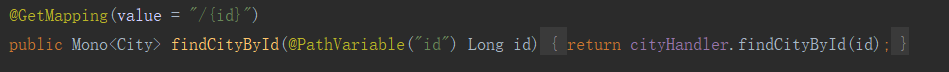
1. 注入Spring boot application.properties配置的属性的值。

## @Inject

1. 等价于默认的@Autowired，只是没有required属性；

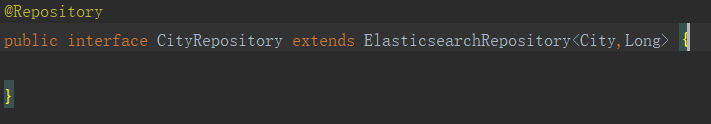
## @PathVariable

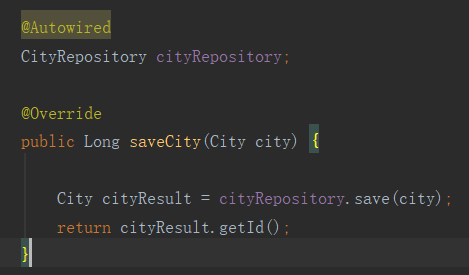
1. 获取url中参数的值



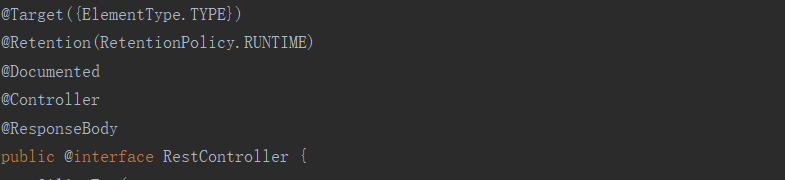
## @CityRepository

1. @Repository用于标注数据访问组件，即DAO组件；创建City类后，能够使用框架自动生成的工具类实现对数据库的增删改查操作。
2. 在一个稍大的项目中，如果组件采用xml的bean定义来配置，显然会增加配置文件的体积，查找以及维护起来也不太方便。





## 思考





1. 既然注解本质上是类，满足继承机制。那么为什么要重复写？

比如说@RestController=@ResponseBody+@Controller，那RestController接口上为什么还要写@Target等注解呢？有什么其他含义？！

# JPA注解

## @Entity

1. @Table(name=” ”)：表明这是一个实体类。一般用于jpa这两个注解一般一块使用，但是如果表名和实体类名相同的话，@Table可以省略

## @MappedSuperClass

1. 用在确定是父类的entity上。父类的属性子类可以继承。

## @NoRepositoryBean

1. 一般用作父类的repository，有这个注解，spring不会去实例化该repository。

## @Column

1. 如果字段名与列名相同，则可以省略。

## @Id

表示该属性为主键。

## @GeneratedValue

1. GeneratedValue(strategy = GenerationType.SEQUENCE,generator = “repair\_seq”)：表示主键生成策略是sequence（可以为Auto、IDENTITY、native等，Auto表示可在多个数据库间切换），指定sequence的名字是repair\_seq。

## @SequenceGeneretor

1. @SequenceGeneretor(name = “repair\_seq”, sequenceName = “seq\_repair”, allocationSize = 1)：name为sequence的名称，以便使用，sequenceName为数据库的sequence名称，两个名称可以一致。

## @Transient

1. 表示该属性并非一个到数据库表的字段的映射,ORM框架将忽略该属性。如果一个属性并非数据库表的字段映射,就务必将其标示为@Transient,否则,ORM框架默认其注解为@Basic。@Basic(fetch=FetchType.LAZY)：标记可以指定实体属性的加载方式

## @JsonIgnore

1. 作用是json序列化时将[Java](http://lib.csdn.net/base/java)bean中的一些属性忽略掉,序列化和反序列化都受影响。

## @JoinColumn（name=”loginId”）

1. 一对一：本表中指向另一个表的外键。一对多：另一个表指向本表的外键。

## @OneToOne、@OneToMany、@ManyToOne

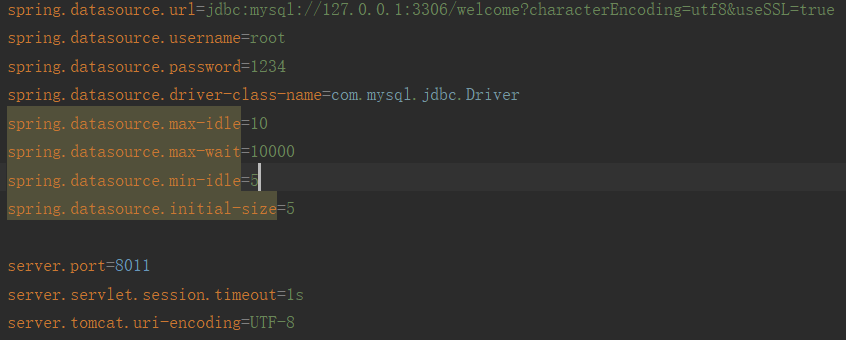
1. 对应[hibernate](http://lib.csdn.net/base/javaee)配置文件中的一对一，一对多，多对一。

# 全局异常处理注解

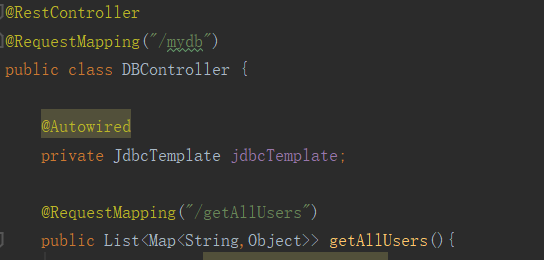
1. @ControllerAdvice：包含@Component。可以被扫描到。统一处理异常。
2. @ExceptionHandler（Exception.class）：用在方法上面表示遇到这个异常就执行以下方法。

# MySQL数据库连接

1. 添加Mysql依赖
2. 配置文件



1. 编写Controller



# 事务控制

1. 数据库的每一条sql语句处于一个单独的事务中，如果执行成功则自动提交，失败则自动回滚。但是我们往往想将多条sql组成一个事务，在发生异常的时候希望事务回滚，以免破坏数据库数据的正确性。（spring关闭了数据库默认自动提交的功能）
2. 在service类上添加@Transactional(...)注解，表示该类下的所以方法都受事务控制。一般会在增删改的方法上添加事务控制，因为查询是不需要事务控制的。
   1. Transactional(timeout=30)，超时时间，默认是30秒
   2. Transactional(Isolation=Isolation.SERIALIZABLE)。串行化
      1. @Transactional(isolation = Isolation.READ\_UNCOMMITTED)：读取未提交数据(会出现脏读, 不可重复读) 基本不使用
      2. @Transactional(isolation = Isolation.READ\_COMMITTED)：读取已提交数据(会出现不可重复读和幻读)
      3. @Transactional(isolation = Isolation.REPEATABLE\_READ)：可重复读(会出现幻读)
      4. @Transactional(isolation = Isolation.SERIALIZABLE)：串行化
   3. 其它：readOnly、[no]RollBackFor、[no]RollBackForClassname 、propagation。
3. 在启动类下添加@EnableTransactionalManager注解，表示开启事务管理。

# Freemarker

1. 文件的后缀为 .ftl；
2. Freemarker其实是一种比较简单的网页展示技术，说白了就是网页模板和数据模型的结合体。这种结合模式的好处就是，分离了网页[界面设计](http://baike.baidu.com/view/119481.htm" \t "_blank)人员和编程人员的工作，让他们各司其职。

# WebFlux特性

1. 一部非阻塞
   1. SpringMVC是同步阻塞的IO模型，资源浪费相对来说比较严重，当我们在处理一个比较耗时的任务时，例如上传一个比较大的文件，服务器的线程只能一直等待，直到文件上传完成。服务器把拿到的文件写入磁盘时，也需要等待直到文件写入完成，期间无法做其他事。
   2. Spring WebFlux可以做到异步非阻塞。还是上面那上传文件的例子，Spring WebFlux是这样做的：线程发现文件还没准备好，就先去做其它事情，当文件准备好之后，通知这根线程来处理，当接收完毕写入磁盘的时候（根据具体情况选择是否做异步非阻塞），写入完毕后通知这根线程再来处理（异步非阻塞情况下）。
2. 响应式函数编程
   1. 如果你觉得java8的lambda写起来很爽，那么，你会再次喜欢上Spring WebFlux，因为它支持函数式编程，得益于对于reactive-stream的支持（通过reactor框架来实现的）
3. 不再拘泥于Servlet容器
   1. 以前，我们的应用都运行于Servlet容器之中，例如我们大家最为熟悉的Tomcat, Jetty...等等。而现在Spring WebFlux不仅能运行于传统的Servlet容器中（前提是容器要支持Servlet3.1，因为非阻塞IO是使用了Servlet3.1的特性），还能运行在支持NIO的Netty和Undertow中。

# Dubbo

<https://www.cnblogs.com/Alandre/p/6490142.html>

# RPC架构

RPC（Remote Promote Call） 一种进程间通信方式。允许像调用本地服务一样调用远程服务。

RPC框架的主要目标就是让远程服务调用更简单、透明。RPC框架负责屏蔽底层的传输方式（TCP或者UDP）、序列化方式（XML/JSON/二进制）和通信细节。开发人员在使用的时候只需要了解谁在什么位置提供了什么样的远程服务接口即可，并不需要关心底层通信细节和调用过程。

<https://www.cnblogs.com/ChrisMurphy/p/6550184.html>

# 注解开发

Annontation是Java5开始引入的新特征。中文名称一般叫注解。它提供了一种安全的类似注释的机制，用来将任何的信息或元数据（metadata）与程序元素（类、方法、成员变量等）进行关联。

Annontation像一种修饰符一样，应用于包、类型、构造方法、方法、成员变量、参数及本地变量的声明语句中。

<https://blog.csdn.net/hotdust/article/details/53032152>

# Bean的三种装配方式

1. 通过容器getBean
2. RunableFactoryBean
3. Jeep

# Bean的三种初始化和销毁方式

# Bean的依赖注入