目录

[1 Spring Boot介绍 3](#_Toc42694620)

[1.1 Spring Boot的优点 3](#_Toc42694621)

[1.2 Starter 3](#_Toc42694622)

[1.2.1 介绍 3](#_Toc42694623)

[1.2.2 常用starter 3](#_Toc42694624)

[2 Spring Boot项目结构介绍 5](#_Toc42694625)

[2.1 目录结构 5](#_Toc42694626)

[2.2 静态资源与动态资源 6](#_Toc42694627)

[2.2.1 静态资源 6](#_Toc42694628)

[2.2.2 动态资源 6](#_Toc42694629)

[2.3 webapp目录与资源目录(resources) 6](#_Toc42694630)

[3 application.properties常用配置 7](#_Toc42694631)

[3.1 工作原理 7](#_Toc42694632)

[4 基于注解的Spring容器 8](#_Toc42694633)

[4.1 容器的创建过程 8](#_Toc42694634)

[4.2 常用注解 8](#_Toc42694635)

[4.3 表达式语言Spring EL 10](#_Toc42694636)

[5 SpringBoot数据访问 12](#_Toc42694637)

[5.1 JDBC操作 12](#_Toc42694638)

[5.1.1 jdbcTemplate的使用流程 12](#_Toc42694639)

[5.1.2 jdbcTemplate的主要API 12](#_Toc42694640)

[5.1.3 NamedParameterJdbcTemplate 12](#_Toc42694641)

[5.2 基于注解的JPA操作 13](#_Toc42694642)

[5.2.1 说明 13](#_Toc42694643)

[5.2.2 使用流程 13](#_Toc42694644)

[5.2.3 实体类常用注解 13](#_Toc42694645)

[5.3 基于注解的Mybatis 14](#_Toc42694646)

[5.3.1 基本流程 14](#_Toc42694647)

[5.3.2 基本注解 14](#_Toc42694648)

[5.3.3 增删改查注解 15](#_Toc42694649)

[5.3.4 动态SQL注解 15](#_Toc42694650)

[5.3.5 人工锁 16](#_Toc42694651)

[5.4 基于注解的Spring事务处理 16](#_Toc42694652)

[5.4.1 说明 16](#_Toc42694653)

[5.4.2 声明式事务处理 16](#_Toc42694654)

[5.5 Spring中Redis的使用 17](#_Toc42694655)

[5.5.1 常用概念 17](#_Toc42694656)

[5.5.2 Spring Boot中使用Redis 18](#_Toc42694657)

[5.6 SpringBoot缓存机制 19](#_Toc42694658)

[5.6.1 基本流程 19](#_Toc42694659)

[5.6.2 常用缓存注解 19](#_Toc42694660)

[6 Spring Security 22](#_Toc42694661)

[6.1 理论介绍 22](#_Toc42694662)

[6.2 工作流程 22](#_Toc42694663)

[6.3 Spring Boot中使用Spring Security 22](#_Toc42694664)

[6.4 Spring Security的配置类WebSecurityConfigurerAdapter 23](#_Toc42694665)

[6.4.1 configure(HttpSecurity http)方法 23](#_Toc42694666)

[6.4.2 configureGlobal(AuthenticationManagerBuilder auth) 方法 23](#_Toc42694667)

[6.4.3 configure(WebSecurity web) 方法 24](#_Toc42694668)

[7 Spring其他功能 25](#_Toc42694669)

[7.1 Spring 异步线程处理 25](#_Toc42694670)

[7.1.1 介绍 25](#_Toc42694671)

[7.1.2 Spring中异常线程池的使用 25](#_Toc42694672)

[7.2 Spring定时任务 25](#_Toc42694673)

[8 HTTPS设置 26](#_Toc42694674)

[9 打包与部署 27](#_Toc42694675)

# Spring Boot介绍

## Spring Boot的优点

1. 基于注解的开发Spring应用
2. 嵌入了Tomcat等WEB服务器，不需要再单独配置
3. 通过Maven获取和管理starter
4. 最大化的自动配置Spring，开发者只需要配置需要修改的地方覆盖默认配置
   1. 配置参数加载优先级



1. 提供了监测、自动检测的功能和外部配置

## Starter

### 介绍

Spring Boot将某些功能所需的jar包都封装到一起，并加入了默认配置和启动类，叫做一个starter。

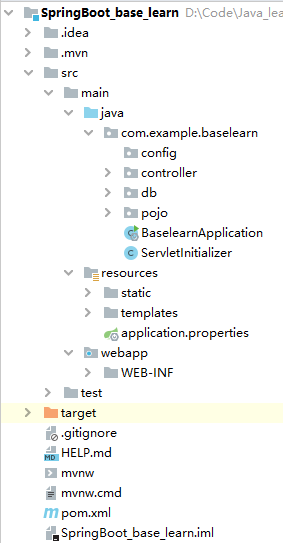
1. 不需要在pom.xml中一个个引入功能所需jar包
2. 设置好了默认的配置参数，用户只需要根据需求进行少量修改覆盖
3. 与Spring Boot项目无缝对接，不需要额外的操作

### 常用starter

1. spring-boot-starter-web
   1. Spring MVC功能所依赖的包
2. spring-boot-starter-tomcat
   1. 内嵌Tomcat服务器所依赖的包
3. spring-boot-starter-parent
   1. 提供相关的Maven默认依赖，使用它之后，常用的包依赖可以省去version标签

# Spring Boot项目结构介绍

## 目录结构



1. pom.xml
   1. Maven项目管理与依赖管理配置
2. src/main/java/package\_name/xxxApplication.java
   1. Spring启动类，启动IoC容器
3. src/main/java/package\_name/ ServletInitializer.java
   1. web服务器启动类
4. src/main/resources/application.properties
   1. 应用配置文件，用于覆盖默认配置
   2. 各种第三方插件的参数也都集中到该文件进行设置
5. webapp
   1. 网络资源存放目录
   2. war项目才会自动生成

## 静态资源与动态资源

### 静态资源

1. 静态HTML、图片、文件等
2. 默认存放路径：src/main/resources/static
3. 默认与http://ip:port/映射绑定，用户可以直接访问
   1. http://ip:port/static\_resource\_name
4. 映射路径可以修改

### 动态资源

1. 动态HTML、JSP等，Spring Boot不推荐使用JSP
2. 动态资源无法直接访问，用户发送请求后，Web服务器先进行处理，然后调用模板渲染引擎对动态页面进行填充渲染后，返回给用户

## webapp目录与资源目录(resources)

1. webroots：src/main/webapp
   1. 
2. sourceRoots：src/main/java、src/main/ resources(默认classpath)
   1. 

# application.properties常用配置

## 工作原理

1. 在配置文件application.properties中设置各种插件参数
2. IoC容器启动时会加载该配置文件中的参数
3. 其他Bean(第三方插件)或者我们自己的代码中都可以使用**${…}**获取到容器中的上下文参数
   1. 
4. 因此，Spring Boot中需要用到什么插件就配置相应参数，该插件注册Bean时会自动加载参数
5. 除了默认的配置文件application.properties，还有其他默认配置文件
   1. 加载优先级

bootstrap.yml > 远程服务器配置文件 > application.yml/application.properties

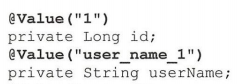
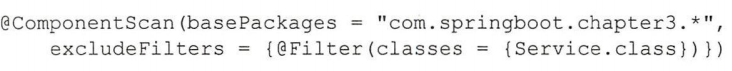
# 基于注解的Spring容器

目前主流的Spring应用大都基于注解开发，包括Spring Boot，这里单独介绍一下基于注解的Spring容器AnnotationConfigApplicationContex

## 容器的创建过程

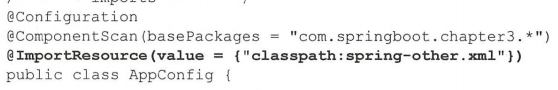
1. 创建一个配置类，比如AppConfig，使用注解@Configuration标明，在该类中通过注解声明各个Bean
2. 创建AnnotationConfigApplicationContex对象实例，并传入配置类AppConfig.class
3. 通过getBean()就可以获取对象实例

## 常用注解

1. @Configuration
   1. 声明该类是一个容器配置类
2. 组件注解
   1. 声明该类是一个Bean
   2. value参数指明Bean的name，默认使用类名，并将第一个字母小写
   3. 常用的组件注解有：@Controller、@Service、@Repository、@Component
      1. @Controller：用于注解控制层组件
      2. @Service：用于注解业务层组件
      3. @Repository：用于注解持久层(数据访问DAO)组件
      4. @Component：用于注解POJO组件或者不易区分功能的组件
   4. 对于被注解类内部的一些变量，可以使用@Value注解进行赋值，使用@Autowired进行自动装配
      1. 
      2. 
3. @ComponentScan
   1. 用于注解**容器配置类**，去扫描带有组件注解的类，将其注入到容器中
   2. 参数
      1. value/ basePackages：指明扫描路径，默认是配置类所在包及其子包
      2. includeFliters：满足过滤器条件时注入至容器
      3. excludeFliters：不满足过滤器条件时注入至容器
      4. lazyinit：是否延迟加载
   3. 
4. @Bean
   1. 注解一个方法，声明该方法的**返回值**是一个Bean，这是与组件注解的最大区别，常用于配置第三方插件Bean
   2. 参数
      1. value/name：Bean name
      2. autowire：自动装配方式
      3. initMethod：类似于构造函数，实例化时自动调用的方法
      4. destroyMethod：实例销毁时自动调用的方法
5. 与@Bean和组件注解搭配使用的一些注解
   1. @Scope：声明Bean的声明周期，singleton、prototype、…
   2. @Lazy：延迟加载
   3. @DependsOn：声明该Bean的依赖，需要先于当前Bean加载
   4. ＠Conditional：指定一个检测类实例，如果不满足检测类实例的matches方法，则该Bean不被加载
      1. 
6. 自动装配注解
   1. 在代码上下文中找到与其匹配的Bean，并自动注入到相应的地方去
   2. @Autowired
      1. 注解类成员变量、方法及构造函数，替代set，get方法
      2. 注解方法时表示方法的参数需要自动装配
      3. Spring注解，默认按照Bean类型自动装配
      4. 如果被注解变量类型是接口时，会有多个实现类，此时会默认按照变量名匹配相应的Bean，匹配不上就报错
   3. @Resource
      1. J2EE注解，与@Autowired功能相同，默认按照Bean name自动装配
      2. 可以使用name或type参数明确声明装配Bean的名称或者类型
   4. @Qualifier
      1. 与@Autowired配合使用，消除歧义性
      2. 当需要自动装配的是一个接口时，因为其有多个实现类，无法自动确定具体装配哪个类，所以需要使用@Qualifier(“**类名**”)明确指明装配的类，类名首字母小写
   5. @Primary
      1. 设置Bean的高优先级，有多个可选Bean时，会被优先选择
      2. 与@Qualifier作用相同，不过一个用于被依赖Bean，一个用于当前Bean
7. @PropertySource
   1. Spring Boot中默认配置文件是application.properties，也可以添加更多的配置文件，在启动类上使用@PropertySource注解，value参数可以设置多个新增配置文件路径



1. @ImportResource
   1. 用于**容器配置类**，添加XML文件路径，将XML文件中配置的Bean加载至IoC容器中

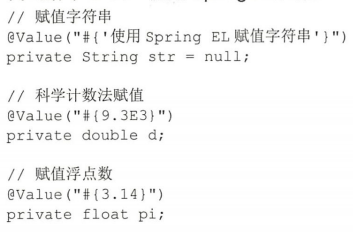


## 表达式语言Spring EL

在使用XML文件或者注解配置Bean时，可以使用表达式对Bean中的属性进行赋值。

1. ${…}：占位符，调用上下文中定义的参数



1. #{…}：启动表达式
   1.  T()调用类
   2. 
   3.  调用其他Bean的属性

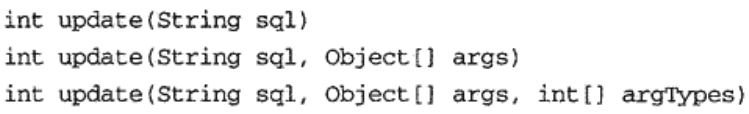
# SpringBoot数据访问

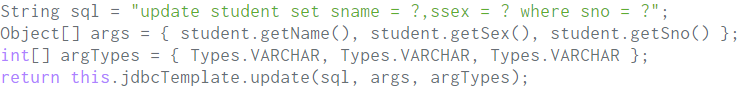
## JDBC操作

### jdbcTemplate的使用流程

1. 引入依赖
   1. spring-boot-starter-jdbc
2. 配置数据源参数
3. 在DAO类中自动装载JdbcTemplate对象实例，使用其执行SQL语句

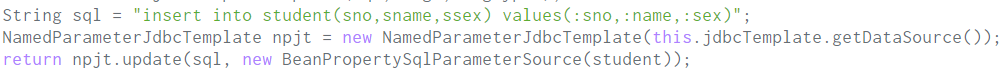
### jdbcTemplate的主要API

1. jdbcTemplate主要提供三类API用于执行数据库操作
   1. execute：执行所有类型的SQL语句
   2. update：执行INSERT、UPDATE、DELETE语句
   3. query：执行查询语句
2. 每类都有多种不同的具体方法，每个方法也有多种不同的重载
   1. 
   2. 一个参数时必须输入完整的SQL语句，使用Statement进行实际的操作
   3. 多个参数时sql语句中使用占位符确定参数位置，然后通过args指定具体的参数，使用PreparedStatement进行实际操作，更加安全



### NamedParameterJdbcTemplate

1. 在jdbcTemplate的基础上进行改进，使用带有语义的符号作为占位符，替代问号



1. 第二个参数是一个SqlParameterSource接口的实例化对象，用于提供参数，有多种实现类

## 基于注解的JPA操作

### 说明

1. Spring Boot的JPA依赖于Hibernate实现
2. 核心是实体(Entity Bean)，通过一个持久化上下文(Persistence Context)来使用，该上下文包括3部分
   1. 对象映射关系(ORM)
   2. 实体操作API，实现对实体对象的CRUD操作
   3. 面向对象查询语言

### 使用流程

1. 创建POJO实体类，使用注解将其与数据库中的表进行对应
2. 创建一个JPA操作定义具体的操作，直接继承与已有接口，比如JpaRepository<T, ID>
   1. <T, ID>其中T是实体类类型，ID是主键字段的类型



* 1. 定义了JPA操作接口之后，**不需要实现类，Spring会自动帮我们创建实现类代码**

1. 在启动类中使用注解扫描JPA操作接口和POJO实体类
   1. @EnableJpaRepositories：指定JPA接口扫描路径
   2. @EntityScan：指定实体类扫描路径
2. 在控制器中自动装配JPA操作接口的对象实例，进行数据库操作

### 实体类常用注解

1. @Entity
   1. 注解该类是一个Entity Bean实体类
2. @Table
   1. 当实体类名与数据库表名不同时，可以使用@Table 注解的name字段指定相应的数据库表
3. @Id
   1. 注解属性，声明该属性是表的主键
   2. 通常会配合注解@GeneratedValue，标注主键的生成策略，通过strategy 属性指定
4. @Column
   1. 注解属性，标明其对应表中的字段名，当属性名与数据库表中字段名不同时使用
5. @ Convert
   1. 注解属性，用于数据库字段存储类型与实体属性的类型之间做类型转换
   2. 通过converter参数指明转换类的class对象

## 基于注解的Mybatis

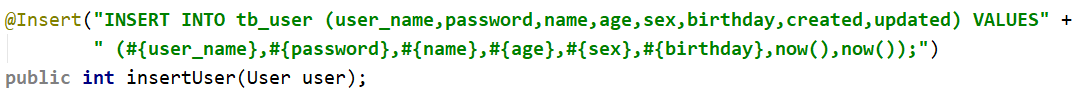
### 基本流程

1. 创建POJO对象
2. 创建Mapper接口
   1. 将DAO和mapper.xml融合在一起
   2. 通过注解的方式声明ROM映射关系
3. Service接口以及接口实现类(Service层)
4. 创建Controller，通过自动装配@Autowired获取Service对象实例，进行数据库操作
5. 在启动类通过@MapperScan注解配置Mybatis参数

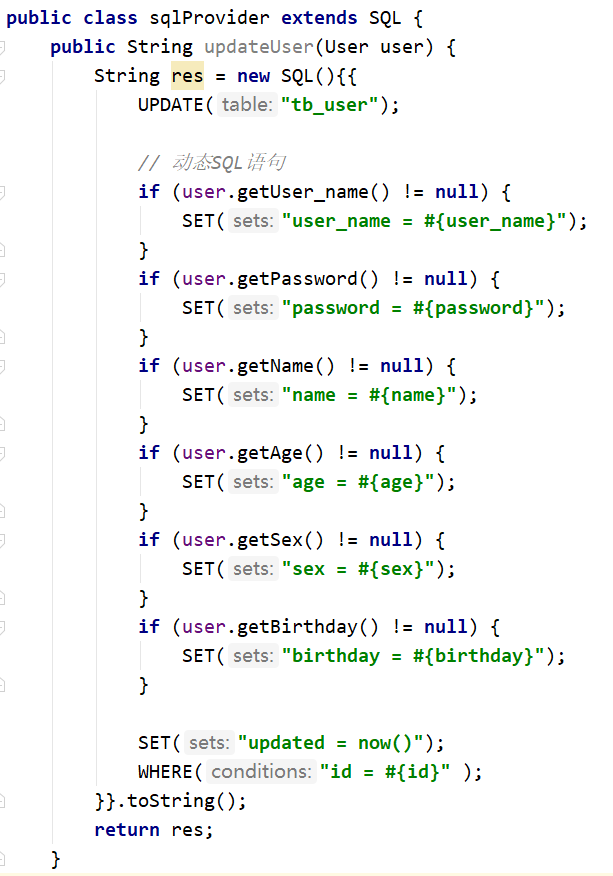
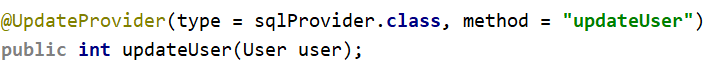
### 基本注解

1. @Mapper
   1. 注解于Mapper接口，会自动生成相应的实现类，并注册到IoC容器中
   2. 接口中不能有重名方法，即不支持重载
   3. 只使用该注解，运行不会有问题，但编辑器(IDEA)可能识别不到该Bean，可以再加个@Component显示的告诉编辑器
2. @MapperScan
   1. 注解于启动类，指定mapper接口的包，启动时会自动扫描这些包下的接口，生成相应的Mapper对象，注册到IoC容器
   2. 与@Mapper功能重复，不过不需要再为每个mapper接口单独注解了

### 增删改查注解

1. @Select
   1. 注解于Mapper接口中定义的查询方法，用于指定ROM映射
   2. 
2. @Param
   1. 指定方法中的形参与Mapper映射语句之间的对应关系
   2. 
3. @Insert
   1. 注解于Mapper接口中定义的插入方法，用于指定ROM映射
   2. 
   3. 传入形参是POJO对象时，会自动提取出其中的各个属性
4. @Update和@Delete同上

### 动态SQL注解

1. 创建一个动态SQL生成类sqlProvider
   1. 该类继承与SQL类
   2. 实现了多种不同的动态SQL生成方法
   3. 
2. 在Mapper中使用@InsertProvider、@UpdateProvider、@DeleteProvider、@SelectProvider注解调用sqlProvider类的相应方法，生成动态SQL语句
   1. 

### 人工锁

MySQL数据库本身就支持锁，因此可以使用特定的SQL映射语句人工加锁

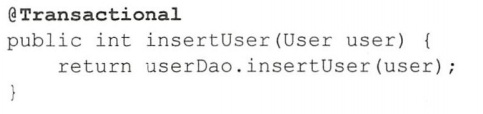
## 基于注解的Spring事务处理

### 说明

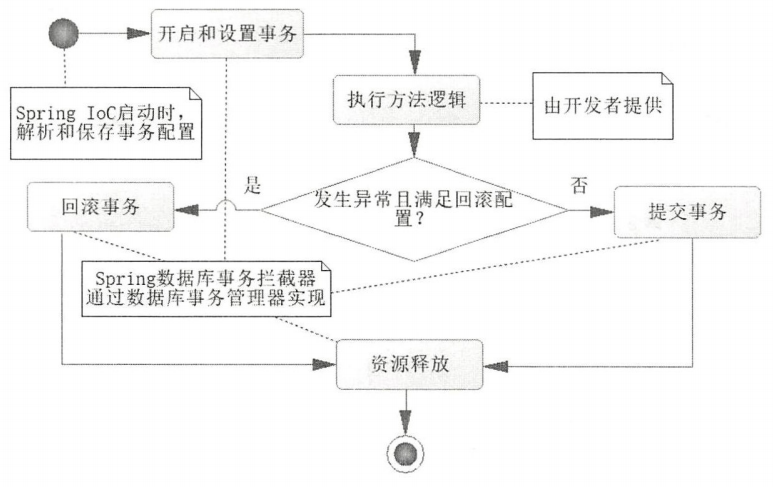
1. 在数据库操作中，一般都会使用try…catch…finally…语句进行事务回滚
2. 这些代码非常冗余，Spring引入了AOP编程减少了这些代码，让开发者专注于业务逻辑开发

### 声明式事务处理

1. 使用＠Transactional注解类或者方法，表明该类内的所有公共(public)非静态方法都要启用事务处理

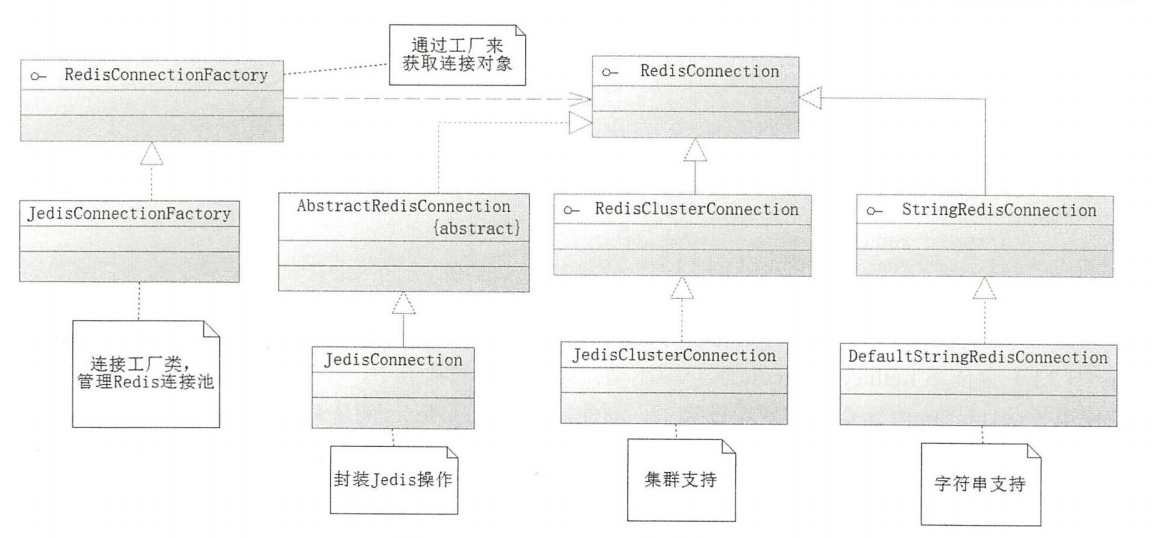
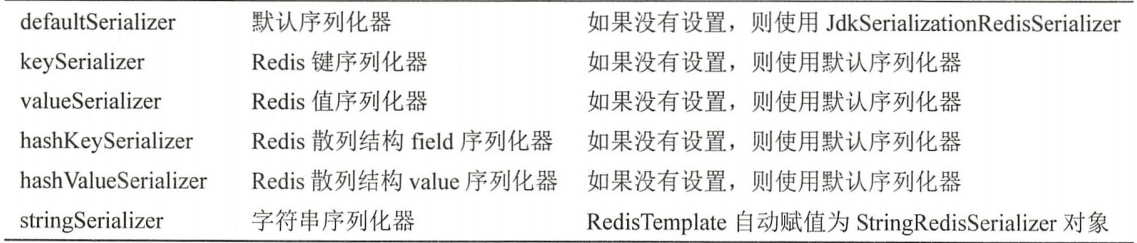


1. ＠Transactional注解还可以配置众多参数，比如事务的隔离级别、传播行为、异常处理等
2. Spring IoC容器会加载这些配置，存储到事务定义器里，记录各个类和方法需要启动事务功能，采取什么策略去执行事务。
3. 当被注解方法被调用时，Spring底层会使用AOP功能完成事务处理，提交事务、回滚事务、资源释放都交由Spring事务管理器完成。



## Spring中Redis的使用

### 常用概念

1. Java与Redis连接的驱动有很多，常用的是Jedis
2. Spring中Redis接口的设计
   1. 
   2. ConnectionFactory类负责建立和管理连接
   3. Connection负责封装具体的操作
3. 通过ConnectionFactory对象获取到的数据库连接使用完需要手动关闭，为了简化这一操作，Spring提供了RedisTemplate。
   1. RedisTemplate对象执行每个数据库操作，都会建立一条新的连接，使用完释放连接
   2. RedisTemplate可以设置不同的序列化器，将Java对象以指定的字符串形式存到Redis数据库中
   3. 
4. 一般一条连接只执行一条数据库操作，为了使得一条连接能够执行多次操作，Spring封装了SessionCallback和RedisCallback接口，可以创建相应实例类完成多次操作(比如事务)。

### Spring Boot中使用Redis

1. 在配置文件中设置Redis参数
2. Spring Boot IoC容器会自动创建RedisConnectionFactory、RedisTemplate、 StringRedisTemplate等常用的对象
3. 使用时直接通过@Autowired自动装配即可

## SpringBoot缓存机制

1. 缓存数据库(redis等)是缓存功能的具体实现，本质上是一种特殊数据库，所以可以像关系数据库那样进行DAO操作
2. SpringBoot的缓存功能是基于缓存数据库，利用注解自动的实现对缓存数据库的DAO操作
3. SpringBoot根据默认顺序去侦测缓存实现，也可以通过配置属性spring.cache.type来强制指定缓存实现类型

### 基本流程

1. 配置缓存数据库，比如redis
   1. 加载依赖
   2. 配置参数
2. 加载缓存依赖
   1. spring-boot-starter-cache
3. 在启动类使用注解@EnableCaching开启缓存功能
4. 在Service接口中加入缓存注解

### 常用缓存注解

1. @CacheConfig
   1. 标注于类上，设置公共配置参数
   2. cacheNames：缓存存储的集合名称，或者叫做CacheManager，可以理解为一个表，表里可以存储多个缓存，每个缓存都通过key来标识
2. @Cacheable
   1. 注解于方法，将方法的返回值放入缓存中
   2. **在查询时，会先从缓存中获取，若不存在才再发起对数据库的访问**
   3. value：缓存集合名称，即cacheName
   4. key：缓存对象存储在Map集合中的key值，非必需，缺省按照函数的所有参数组合作为key值，配置需使用SpEL表达式
      1. key = "#p0"：将函数第一个形参作为key
      2. key=”#userName” ：将函数的形参userName作为key
   5. condition：缓存条件，可以缺省，使用SpEL表达式，返回true或false，返回ture时才存入缓存
3. @CachePut
   1. 与@Cacheable不同的是每次都会真实调用函数，访问数据库，所以主要用于数据新增和修改操作上
   2. 其他都和@Cacheable相同
4. @CacheEvict
   1. 注解于方法，根据方法参数对缓存中的数据进行清除，一般用于删除操作上
   2. 基本参数：value、key、condition
   3. allEntries：布尔值，是否清除所有缓存，缺省为false
   4. beforeInvocation：布尔值，是否在方法执行前就清除缓存，缺省为false

# SpringBoot Web

# Spring Security

## 理论介绍

1. Java Web中通常使用Filter进行安全验证，在请求传入Servlet之前进行处理
2. Spring也基础Filter实现了自己的安全框架Spring Security，本质就是一个特殊的Filter Chain，用户请求需要通过一系列的Filter处理后才能交由后续的控制器进行处理

## 工作流程

1. 用户发送请求
2. Spring Security根据配置参数判断是否需要拦截
3. 需要拦截的请求，则跳转至登录页面(默认页面或者开发者指定页面)，让用户进行登录验证
   1. Spring Security支持两种验证方式，一是通过登录页面提交form表单
   2. 二是将用户名密码放入到HTTP报文头部字段Authorization中
4. 对用户登录信息进行验证，验证失败跳转至错误页面，成功则跳转至之前访问的页面
   1. Spring Security采用两种方式避免重复进行验证
   2. 一是由客户端(浏览器)将用户名和密码保存在头部的Authorization字段中，放在每个用户请求中
   3. 二是在服务端Spring Security配置中设置rememberMe()参数，将用户的验证成功信息保存至cookies字段中，这样用户请求中只需要带有相应cookies就可以了

## Spring Boot中使用Spring Security

1. 引入依赖
   1. <groupId>org.springframework.boot</groupId><artifactId>spring-boot-starter-security</artifactId>
   2. Spring Boot中引入依赖之后会自动启动Spring Security，项目的所有接口都会被自动保护起来，不需要再进行注解
2. 配置Spring Security，创建配置类，继承于WebSecurityConfigurerAdapter
   1. 使用@Configuration进行注解，通知IoC容器进行加载
   2. WebSecurityConfigurerAdapter类实现了基本的Spring Security配置，开发者只需要根据需求重写相应方法即可

## Spring Security的配置类WebSecurityConfigurerAdapter

### configure(HttpSecurity http)方法

作用：

* 1. 定义哪些请求进行拦截，哪些不拦截
  2. 定义不同页面的访问权限
  3. 定义请求访问通道HTTPS或者HTTP
  4. 定义cookies记录功能
  5. 定义拦截时跳转到的登录页面、登出页面等

自定义登录页面注意事项：

1. 将自定义登录页面权限设为permitAll()，否则会死循环，出现跳转次数过多错误
2. 最好设置登录处理URL，loginProcessingUrl()，设置自定义登录页面中form表单的提交地址，表单中的用户名和密码字段名必须是username和password
3. Spring Security会默认开启CSRF服务，访问时会将CSRF参数返回给用户，用户进行登录验证的表单中必须得带有这些CSRF参数
   1. 可以使用http.csrf().disable();关闭该服务，但是不安全。

### configureGlobal(AuthenticationManagerBuilder auth) 方法

作用：

1. 用于配置用户名、密码、用户权限/身份

三种方法：

1. 内存签名服务：
   1. 通过代码指定用户、密码、身份，在程序启动时加载并保存在内存中
2. SQL数据库用户认证服务
   1. 将用户。密码。身份信息存储至SQL数据库表中，数据库中存储的密码也是通过加密器加密后的密码，而不是密码原文
   2. 验证登录时通过SQL语句进行查询验证
3. 自定义验证方式
   1. 查询SQL数据库较慢，经常先查询NoSQL数据库(Redis)，查不到再查SQL数据库
   2. 创建一个用户信息查询服务类，自行实现UserDetailsService接口中的loadUserByUsername方法，用于用户信息查询。然后使用auth.userDetailsService(userDetailsService)启用自定义的验证方式

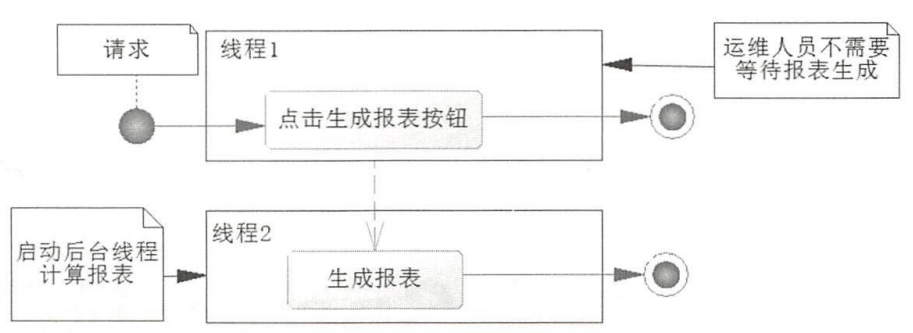
### configure(WebSecurity web) 方法

# Spring其他功能

## Spring 异步线程处理

### 介绍

有些用户请求耗时较长，不希望用户一直等待，而是启动后台线程去处理，之后用户再去获取处理结果，这就是请求的异步线程处理。



### Spring中异常线程池的使用

1. 创建一个配置类继承AsyncConfigurer，使用注解@Configuration和@EnableAsync
   1. @Configuration声明是配置类，交由IoC容器解析加载
   2. @EnableAsync启动Spring异步线程机制
2. 实现getAsyncExecutor()方法，返回线程池，供IoC容器调用，
3. 实现getAsyncUncaughtExceptionHandler()，定义线程内异常的处理方法(可选)
4. 具体的方法使用@Async进行注解，表示该方法需要请求线程池中的线程进行后台处理，然后继续执行当前请求，无需等待

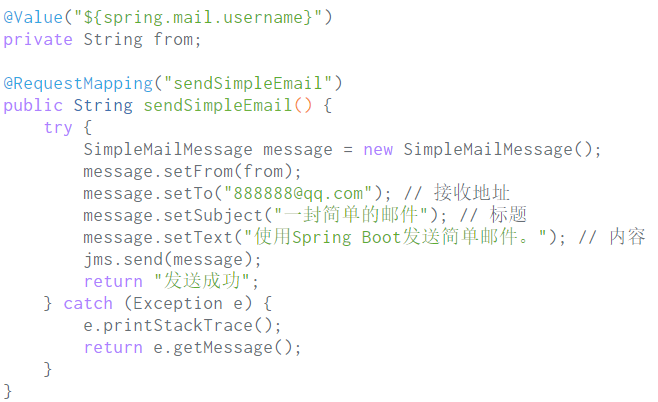
## Spring定时任务

1. 在配置类使用@EnableScheduling注解启用定时任务驱动
2. 在具体方法上使用@Scheduled注解设置具体的定时情况

## Actuator监控应用

1. 引入依赖
   1. spring-boot-starter-actuator
2. 配置参数
   1. 
3. 访问URL接口即可查看监控情况
   1. 

## 邮件发送

1. 引入依赖
   1. spring-boot-starter-mail
2. 配置参数
   1. 
3. 自动装载JavaMailSender对象
   1. 
4. 发送简单邮件
   1. 创建SimpleMailMessage对象实例，使用jms发送
   2. 
5. 发送HTML格式的邮件
   1. 创建MimeMessage对象实例，使用jms发送
   2. 
6. 发送带附件的邮件
   1. 依旧使用MimeMessage，通过addAttachment()添加附件
   2. 
7. 发送带静态资源的邮件
   1. 和附件类似，不过邮件内容是HTML，并且需要使用cid标识静态资源
   2. 通过addInline()方法添加静态资源
   3. 
8. 使用模板发送邮件
   1. 引入依赖spring-boot-starter-thymeleaf
   2. 创建模板emailTemplate.html



* 1. 通过TemplateEngine对象将模板和模型(数据)进行整合，转换成HTML格式字符串
  2. 

# HTTPS设置

# 打包与部署