目录

[1 Spring Boot介绍 3](#_Toc41251067)

[1.1 Spring Boot的优点 3](#_Toc41251068)

[1.2 Starter 3](#_Toc41251069)

[1.2.1 介绍 3](#_Toc41251070)

[1.2.2 常用starter 3](#_Toc41251071)

[2 Spring Boot项目结构介绍 4](#_Toc41251072)

[2.1 目录结构 4](#_Toc41251073)

[2.2 特殊说明 5](#_Toc41251074)

[2.3 静态资源与动态资源 5](#_Toc41251075)

[2.3.1 静态资源 5](#_Toc41251076)

[2.3.2 动态资源 5](#_Toc41251077)

[2.4 webapp目录与资源目录(resources) 5](#_Toc41251078)

[3 application.properties常用配置 6](#_Toc41251079)

[3.1 工作原理 6](#_Toc41251080)

[3.2 视图相关配置 6](#_Toc41251081)

[3.3 数据库相关配置 6](#_Toc41251082)

[4 基于注解的Spring容器 7](#_Toc41251083)

[4.1 容器的创建过程 7](#_Toc41251084)

[4.2 常用注解 7](#_Toc41251085)

[4.3 表达式语言Spring EL 9](#_Toc41251086)

[5 基于注解的数据库操作 11](#_Toc41251087)

[5.1 基于注解的JPA操作 11](#_Toc41251088)

[5.1.1 说明 11](#_Toc41251089)

[5.1.2 使用流程 11](#_Toc41251090)

[5.1.3 实体类常用注解 12](#_Toc41251091)

[5.2 基于注解的Mybatis 12](#_Toc41251092)

[5.2.1 一般流程 12](#_Toc41251093)

[5.2.2 Spring Boot改进 12](#_Toc41251094)

[5.3 基于注解的Spring事务处理 13](#_Toc41251095)

[5.3.1 说明 13](#_Toc41251096)

[5.3.2 声明式事务处理 13](#_Toc41251097)

[5.3.3 自调用的失效问题 14](#_Toc41251098)

[5.4 Spring中Redis的使用 14](#_Toc41251099)

[5.4.1 常用概念 14](#_Toc41251100)

[5.4.2 Spring Boot中使用Redis 15](#_Toc41251101)

[6 Spring Boot中的SpringMVC 16](#_Toc41251102)

[6.1 定制初始化 16](#_Toc41251103)

[7 Spring Security 17](#_Toc41251104)

[7.1 理论介绍 17](#_Toc41251105)

[7.2 工作流程 17](#_Toc41251106)

[7.3 Spring Boot中使用Spring Security 17](#_Toc41251107)

[7.4 Spring Security的配置类WebSecurityConfigurerAdapter 18](#_Toc41251108)

[7.4.1 configure(HttpSecurity http)方法 18](#_Toc41251109)

[7.4.2 configureGlobal(AuthenticationManagerBuilder auth) 方法 18](#_Toc41251110)

[7.4.3 configure(WebSecurity web) 方法 19](#_Toc41251111)

[8 Spring其他功能 20](#_Toc41251112)

[8.1 Spring 异步线程处理 20](#_Toc41251113)

[8.1.1 介绍 20](#_Toc41251114)

[8.1.2 Spring中异常线程池的使用 20](#_Toc41251115)

[8.2 Spring定时任务 20](#_Toc41251116)

# 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。

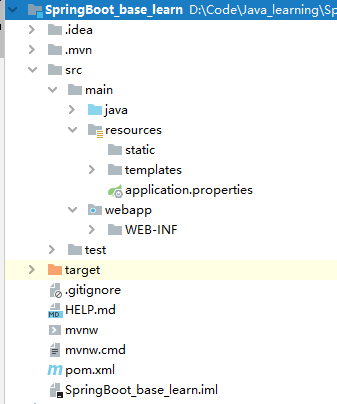
1. 不需要在pom.xml中一个个引入功能所需jar包
2. 设置好了默认的配置参数，用户只需要根据需求进行少量修改覆盖
3. 与Spring Boot项目无缝对接，不需要额外的操作

### 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

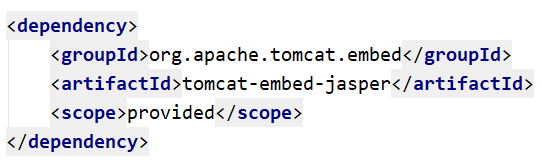
# Spring Boot项目结构介绍

## 目录结构



1. pom.xml
   1. Maven项目管理与依赖管理配置
2. SpringBoot\_base\_learn.iml
3. src/main/java/package\_name/xxxApplication.java
   1. Spring启动类，启动IoC容器
4. src/main/java/package\_name/ ServletInitializer.java
   1. web服务器启动类
5. 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

## 特殊说明

1. webapp目录不是必须的，需要自己创建
2. Spring Boot本身不支持jsp文件格式，需要额外的插件解析jsp文件，或者使用
   1. 

## 静态资源与动态资源

### 静态资源

1. 静态HTML、图片、文件等
2. 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)就可以使用${…}获取到容器中的上下文参数
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

## 视图相关配置

## 数据库相关配置

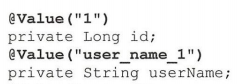
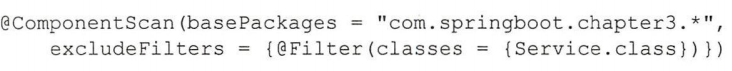
# 基于注解的Spring容器

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

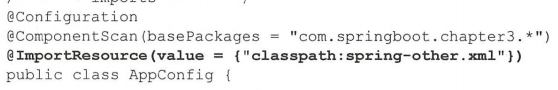
1. 创建一个配置类，比如AppConfig，使用注解@Configuration标明，在该类中通过注解声明各个Bean
2. 创建AnnotationConfigApplicationContex对象实例，并传入配置类AppConfig.class
3. 通过getBean()就可以获取对象实例

## 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

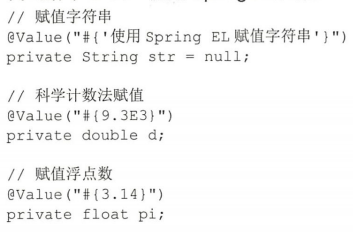


## 表达式语言Spring EL

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

# 基于注解的数据库操作

1. JPA和Mybatis都是用于封装ORM接口，将SQL操作转变为面向对象操作，两者功能上比较冗余
2. JPA通过第三方插件(Hibernate)实现，通过继承相应接口，由Spring能够自动生成基本的CRUD操作
3. Mybatis则通过Mapper定义面向对象操作与SQL语句之间的映射关系

## 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

### 一般流程

1. 创建POJO对象
2. 创建mapper.xml文件，指明ROM映射关系
3. 创建对应的mapper对应的DAO接口以及接口实现类(DAO层)
4. 创建Contoller类，执行SQL操作
   1. 根据数据库配置，创建SqlSessionFactory对象实例
   2. 通过SqlSessionFactory对象实例获取数据库连接SqlSession对象实例
   3. 选择映射mapper，通过SqlSession创建mapper对象实例，执行SQL语句
5. 在配置文件中配置Mybatis参数

### Spring Boot改进

1. 创建POJO对象
2. 创建mapper.xml文件，指明ROM映射关系
3. 创建对应的mapper对应的DAO接口以及接口实现类(DAO层)
4. 创建DAO对应的Service接口以及接口实现类(Service层)
5. 创建Controller，通过自动装配@Autowired获取Service对象实例，进行数据库操作
6. 在启动类通过@MapperScan注解配置Mybatis参数

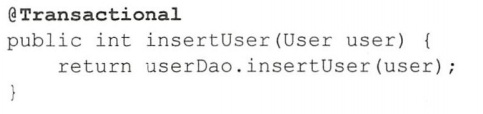
## 基于注解的Spring事务处理

### 说明

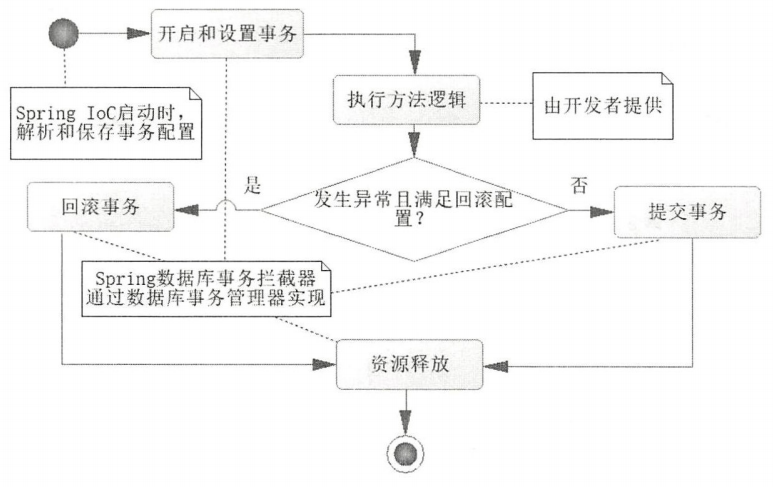
1. 在数据库操作中，一般都会使用try…catch…finally…语句进行事务回滚
2. 这些代码非常冗余，Spring引入了AOP编程减少了这些代码，让开发者专注于业务逻辑开发

### 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



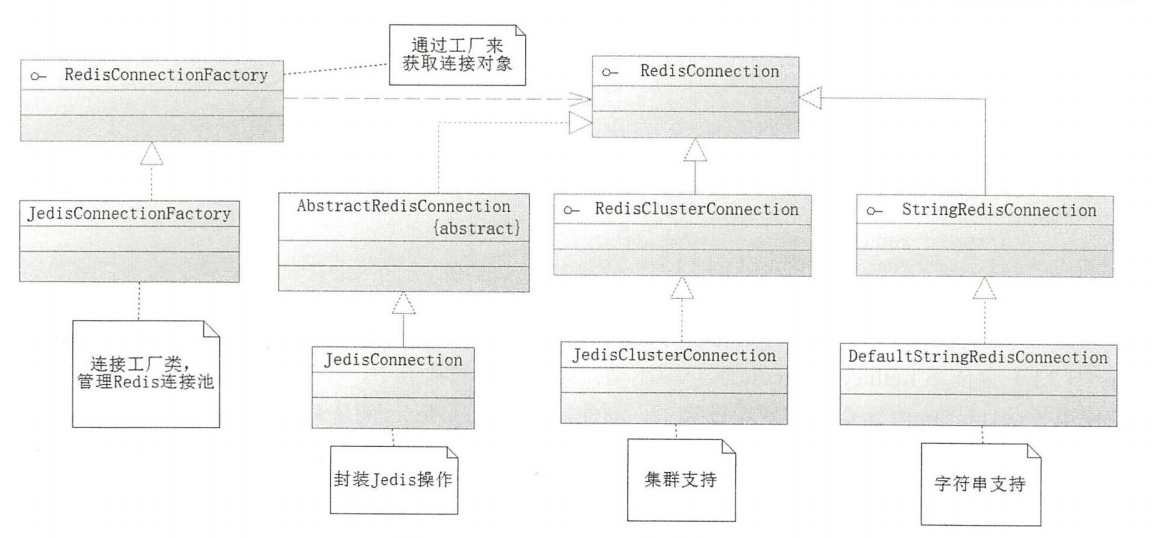
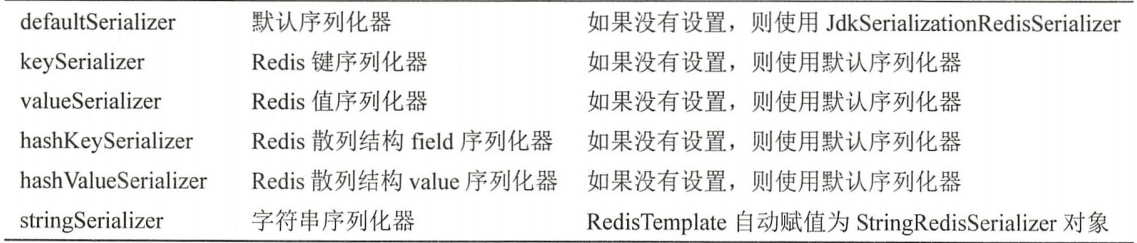
1. ＠Transactional注解还可以配置众多参数，比如事务的隔离级别、传播行为、异常处理等
2. Spring IoC容器会加载这些配置，存储到事务定义器里，记录各个类和方法需要启动事务功能，采取什么策略去执行事务。
3. 当被注解方法被调用时，Spring底层会使用AOP功能完成事务处理，提交事务、回滚事务、资源释放都交由Spring事务管理器完成。



### 自调用的失效问题

## 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

# Spring Boot中的SpringMVC

## 定制初始化

1. Spring Boot中web.xml不再是配置Spring MVC的必要组件，相应的设计了一种初始化配置类接口WebMvcAutoConfiguration
2. Spring Boot容器会自动装配一个静态内部类WebMvcAutoConfigurationAdapter，它会读取配置文件中关于Spring MVC的参数设置，完成自定义初始化工作

# 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

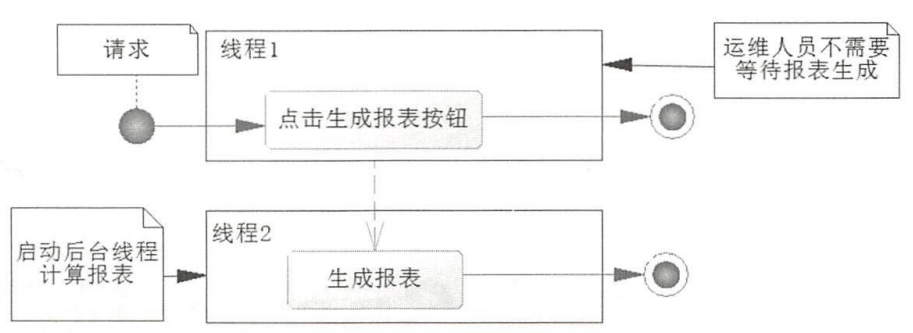
### configure(WebSecurity web) 方法

# 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