spring.http.log-request-details 启用调试日志，默认位false

spring-boot-devtools默认禁用缓存

spring.devtools.add-properties：使用spring-boot-devtools时关闭属性默认值

关于Spring-boot-devtools的使用：

[https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/using-spring-boot.html#using-boot-devtools](https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/using-spring-boot.html" \l "using-boot-devtools)

# 打包应用用于生产环境：

[https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/using-spring-boot.html#using-boot-packaging-for-production](https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/using-spring-boot.html" \l "using-boot-packaging-for-production)

# 生产环境中的监控配置：

Spring Boot Actuator 介绍：

<http://ypk1226.com/2019/01/17/spring-boot/spring-boot-actuator-1/>

官方文档介绍：

[https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/production-ready-features.html#production-ready](https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/production-ready-features.html" \l "production-ready)

# SpringBoot的各种配置介绍（偏向于从头讲出，适合阅读理解）：

[https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/spring-boot-features.html#boot-features](https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/spring-boot-features.html" \l "boot-features)

# SpringBoot应用的配置（偏向于具体的参数配置，适合查阅）

[https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/appendix-application-properties.html#web-properties](https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/appendix-application-properties.html" \l "web-properties)

# SpringBoot指定激活哪个配置：

<https://blog.csdn.net/weixin_40317062/article/details/82194191?utm_source=blogxgwz2>

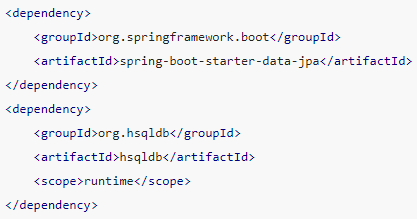
官方文档介绍：

[https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/spring-boot-features.html#boot-features-profile-specific-configuration](https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/spring-boot-features.html" \l "boot-features-profile-specific-configuration)

# SpringBoot中嵌入数据库

SpringBoot中能自动嵌入h2、HSQL和Derby数据库，只要在pom文件中添加相关的依赖就是了。如果使用这个特性在你的测试中，那么在一个测试用例套件中，默认是用同一个上下文中的数据库，如果希望每个上下文都是单读的嵌入一个数据库，可以设置：spring.datasource.generate-unique-name=true。

在一个SpringBoot应用中，引入数据库支持的写法类似如下：



不管出于什么原因，如果使用了url连接的方式在SpringBoot应用中来嵌入了一个数据库，应该保证数据的自动关闭是被关闭的。如果使用h2，可以配置DB\_CLOSE\_ON\_EXIT=FALSE来避免这种情况；使用HSQLDB需要确定shotdown=true。因为数据库的自动关闭会让SpringBoot对数据库的控制自动关闭。

## 数据源的配置

SpringBoot默认自动嵌入的数据库连接池优先级：

1. [HikariCP](https://github.com/brettwooldridge/HikariCP) （如果添加了它的依赖）
2. Tomcat的数据库连接池（前提是此数据源可用）
3. 如果[HikariCP](https://github.com/brettwooldridge/HikariCP)和Tomcat数据源都不可用，如果 [Commons DBCP2](https://commons.apache.org/proper/commons-dbcp/) 可用，那么选择它

4、如果使用

spring-boot-starter-jdbc 或者 spring-boot-starter-data-jpa 作为“starters”，那么会自动注入[HikariCP](https://github.com/brettwooldridge/HikariCP) 的数据库连接池。

5、自定义数据源类型：spring.datasource.type，如使用jdbc的数据源：

spring.datasource.type=org.apache.tomcat.jdbc.pool.DataSource

1. 特别注意一点，如果SpringBoot应用跑在tomcat中，那么会默认使用tomcat的jdbc连接池

对于大多数数据库，SpringBoot可以从url中识别driver-class-name。

### 自定义连接池配置

更多的datasourceproperties的配置：

<https://github.com/spring-projects/spring-boot/blob/v2.2.2.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceProperties.java>

可以通过前缀的方式来对数据源的具体特性进行微调，如：

spring.datasource.hikari.\*,

 spring.datasource.tomcat.\*,

spring.datasource.dbcp2.\*’

更多tomcat连接池的配置详情：

[https://tomcat.apache.org/tomcat-8.0-doc/jdbc-pool.html#Common\_Attributes](https://tomcat.apache.org/tomcat-8.0-doc/jdbc-pool.html" \l "Common_Attributes)

## 自定义配置数据源（一个或多个数据源）：

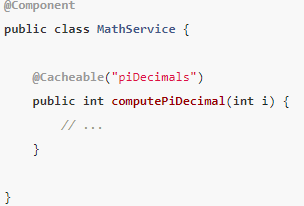
[https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/howto.html#howto-data-access](https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/howto.html" \l "howto-data-access)

# SpringBoot中关于jpa的一些配置

默认情况下，SpringBoot是默认开启“[OpenEntityManagerInViewInterceptor](https://docs.spring.io/spring/docs/5.2.2.RELEASE/javadoc-api/org/springframework/orm/jpa/support/OpenEntityManagerInViewInterceptor.html)”的，如果想要关闭，可以这么设置：spring.jpa.open-in-view=false

# SpringBoot中关于缓存的配置

在SpringBoot中使用了@Cacheable注解的地方就会开启它自动带的缓存了（SpringBoot默认会提供一种支持并发的简单缓存，不推荐在生产环境中使用。这种缓存默认使用ConcurrentHashMap实现），下面看一个例子：



这个例子演示了在一个开销可能会很大的操作上使用缓存。在调用computePiDecimal方法之前，会在缓存中去寻找寻符合参数i的实体，如果找到就立马返回，没找到就执行查询方法，然后更新缓存，再返回结果（piDecimals是缓存的名字，以便于快速定位）

**缓存需要的starter：spring-boot-starter-cache**

## 缓存相关的注解：

关于缓存的注解Spring建议用于注释类或类方法，当然也可以注释接口和接口方法，但前提是缓存要基于proxy的代理方式，即基于接口的代理方式（jdk的代理方式）。

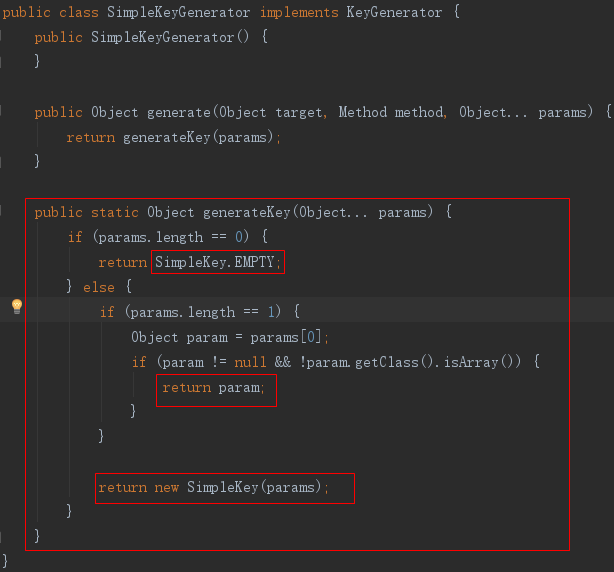
### @Cacheable

#### 默认的key生成规则：

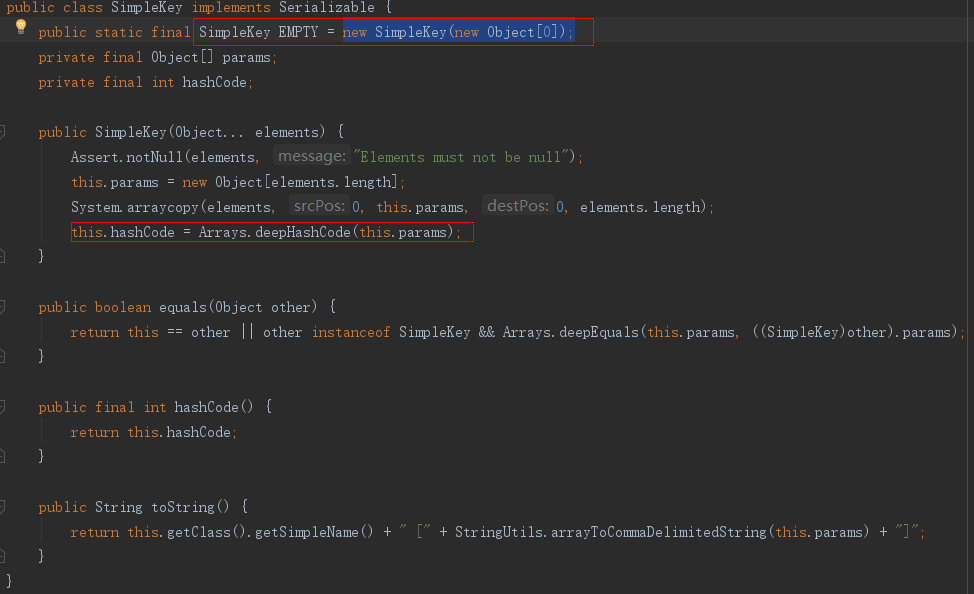
**@Cacheable缓存的key默认（就是通过目标方法是否有参数来确定的）是通过KeyGenerator的实现类SimpleKeyGenerator生成的（关于SimpleKey的生成策略源码：**

<https://www.cnblogs.com/wzdnwyyu/p/11180465.html>**）**，源码如下：

1. **SimpleKeyGenerator：**



1. **SimpleKey：**

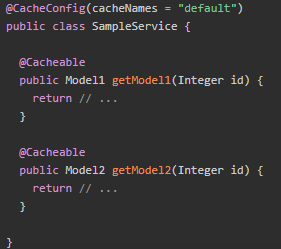


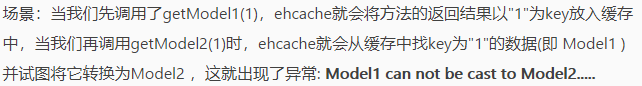
其默认生成策略为（下面的规则有一个前提，就是参数有自然键且有效的实现了hashCode()和equals()方法）：

1. 如果方法没有参数，则使用0作为key
2. 如果只有一个参数，则使用该参数作为key
3. 如果参数多于一个，则使用所有参数的hashCode作为key

可以看出默认的key生成策略中并没有涉及方法名称和类，这就意味着如果我们有两个参数列表相同的方法，我们用相同的参数分别调用两个方法，当调用第二个方法的时候，spring cache将会返回缓存中的第一个方法的缓存值，因为他们的key是一样的。下面我们看两个例子：

**例1、**





所以我们需要自定义key策略来解决这个问题，将类名和方法名和参数列表一起来生成key：<https://www.jianshu.com/p/2a584aaafad3>

生成自定义的秘钥生成器需要实现（从Spring4.0开始引入了采用了新的策略接口），

org.springframework.cache.interceptor.KeyGenerator接口，如果想要采用以前的策略，实现，

org.springframework.cache.interceptor.DefaultKeyGenerator

[https://docs.spring.io/spring/docs/5.2.2.RELEASE/spring-framework-reference/integration.html#cache-annotations-cacheable](https://docs.spring.io/spring/docs/5.2.2.RELEASE/spring-framework-reference/integration.html" \l "cache-annotations-cacheable)

**例2、**

在大多数情况下，只声明一个缓存，但注释允许指定多个名称，以便使用多个缓存。在这种情况下，在执行方法之前检查每个缓存--------如果至少命中一个缓存，则返回关联的值,



@Cacheable(“books”)

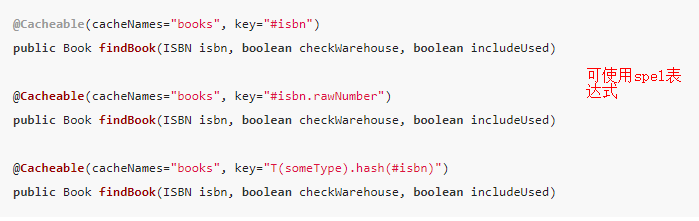
public Book findBookById(int id)

findBook方法就有多个缓存名，如果存在其他地方也有相同缓存名，那么缓存也会同步更新，即使那个方法没有执行。换句话说就是，上面两段代码，不管哪个方法执行后更新了名为books的缓存，另一个方法关联的缓存也会跟着同步，因为都是名为books的同一个缓存，而且缓存本质上是键值存储，所以每次调用缓存的方法都需要转换成合适的键来进行缓存访问。

#### 自定义的秘钥生成规则：

生成秘钥的方式：

1、采用key（秘钥）



上**面调用方法参数生成key的方式，还可以写成别名的方式来调用方法参数，如，调用方法中的第一个参数可写成#iban or #a0 (也可以写成：#p0 or #p<#arg> )（采用下标的方式时，从0开始）。**使用spel表达式不需要编写更多代码或实现某个接口。**推荐使用。除了使用上面代码中的使用方法参数作为key之外，Spring还提供了一个root对象可用来生成key，通过该root对象可获取到如下信息：**



2、采用自定义的秘钥生成器



#### 注解属性介绍：

<https://www.jianshu.com/p/c148b4b7e8a4>

#### 关于在多线程环境下，锁定缓存：



#### 控制在特定环境中，进行缓存



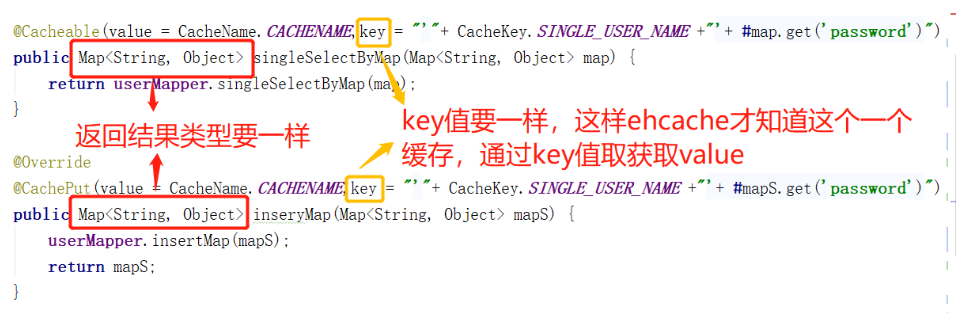
其中condition表达式是在方法调用前执行，unless表达式是在方法调用后执行（其中#result表示执行方法执行结果）

### @CachePut

当需要更新缓存而不影响方法的正常执行，这意味着方法总是执行，并且把结果放入缓存中（依据注解参数而定）。这个注解有着和@Cacheable一样的参数，且它们的用法都差不多。这个注解用于缓存填充，而不是用于方法流程优化。

此注解和@Cacheable的区别在于，它是先调用方法然后再把结果存到缓存中去，而@Cacheable则是先去缓存中找是否有合适的结果，有就直接返回，没有则执行方法，把新的结果存到缓存中，再返回结果。说白了就是@Cacheable用于跳过方法执行，而@CachePut则正好相反。

如果要在@Cacheable注解中取出由@CachePut注解更新的缓存，则必须保证缓存的key和对应的方法结果类型是一样的。如下代码所示：



### @CacheEvict

用于清除缓存，它所标注的方法没有返回值，用void修饰。具有allEntries参数，表示是否清除整个缓存区域，默认false否。beforeInvocation参数，清除操作默认是在对应方法成功执行之后触发的，即方法如果因为抛出异常而未能成功返回时也不会触发清除操作。使用beforeInvocation可以改变触发清除操作的时间，当我们指定该属性值为true时，Spring会在调用该方法之前清除缓存中的指定元素。

### @Caching

用于多个注解的嵌套使用，如：



### @CacheConfig

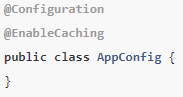
@CacheConfig是一个类级注释，允许共享缓存名称、自定义密钥生成器、自定义CacheManager和自定义CacheResolver。将此注释放在类上不会打开任何缓存操作。**它提供了三层含义**：

1. 全局配置，配置相同的CacheManager, KeyGenerator
2. 在类界别上使用
3. 属于操作层面

## 开启Spring的缓存注解

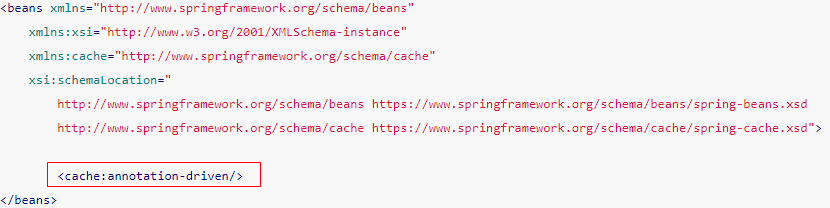
如果不需要开启缓存注解，只需要不配置就行

1. SpringBoot方式，在@Configuration注解上加上@EnableCaching注解，如下所示：



**@EnableCaching注解属性（使用aspectj需要引入spring-aspects）：**

1. mode：代理模式，默认使用proxy，最好是使用aspectj
2. proxyTargetClass：默认为false（使用jdk代理方式）（使用aspectj代理需要设置为true，注意：**如果设置为true，整个spring托管的bean都会改为cglib方式，不仅仅是采用了@Cacheable的包括@Transactional注解的等**）
3. order() default 2147483647
4. xml方式：



上面两种方式都允许通过aop来将缓存添加到应用程序中（换句话说，就是都支持aop）

### Cache配置

#### 1、

通过继承CachingConfigurerSupport类实现自定义cacheManager、keyGenerator、cacheResolver、errorHandler：

方式1、采用重写方法的方式：

<https://docs.spring.io/spring-framework/docs/5.2.2.RELEASE/javadoc-api/org/springframework/cache/annotation/EnableCaching.html>

方式2、采用@Bean的方式：

[https://docs.spring.io/spring-framework/docs/5.2.2.RELEASE/javadoc-api/org/springframework/cache/annotation/CachingConfigurer.html#cacheManager--](https://docs.spring.io/spring-framework/docs/5.2.2.RELEASE/javadoc-api/org/springframework/cache/annotation/CachingConfigurer.html" \l "cacheManager--)

#### 2、

**SpringBoot配置多个CacheManager：**

<https://blog.csdn.net/qq_26440803/article/details/90145543><https://www.cnblogs.com/pengthrree/p/10956078.html>

## 缓存的代理方式

Spring中，缓存默认的代理方式是jdk的方式也就是基于接口的代理方式，这意味着只会拦截通过代理传入的外部方法调用。换句话说，自调用（目标对象上的一个方法调用目标对象上的另一个方法）不会在运行时，产生实际的缓存，即使加了@Cacheable注解。如果要解决这个问题，可以改为使用AspectJ的代理模式。

## 定义和使用自定义的缓存注解

1. 定义：

[https://docs.spring.io/spring/docs/5.2.2.RELEASE/spring-framework-reference/integration.html#cache-annotation-stereotype](https://docs.spring.io/spring/docs/5.2.2.RELEASE/spring-framework-reference/integration.html" \l "cache-annotation-stereotype)

# SpringBoot中的任务调度器

在SpringBoot中只要引入了spring-boot-starter-quartz，那么一个Scheduler是会被自动配置的（通过SchedulerFactoryBean）。JobDetail、Calendar、 Trigger是将会自动获取（前提是配置了它们）并与Scheduler关联。

**官方文档介绍：**

[https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/spring-boot-features.html#boot-features-quartz](https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/spring-boot-features.html" \l "boot-features-quartz)

**简单使用Quartz：**

1. <https://blog.csdn.net/qq_34675369/article/details/90691551>
2. <https://blog.csdn.net/weixin_39723544/article/details/83382000>
3. <https://blog.csdn.net/qq_39669058/article/details/90411497>

**关于jobStores：**

各种jobStores的介绍：

<https://www.jianshu.com/p/ce5eb0eddd39>

配置分布式quartz：

<http://www.itkeyword.com/doc/9814361177729683103/quartz-spring-mysql>

配置jobStoreCMT程序被锁：

<http://www.blogjava.net/ldd600/archive/2009/02/06/253604.html>

Spring配置分布式quartz（xml方式）

<https://blog.csdn.net/qq_21434959/article/details/83119981>

SpringBoot配置分布式quartz：

<https://my.oschina.net/u/3164861/blog/1812663>