[Java Hashmap工作原理及实现](https://yikun.github.io/2015/04/01/Java-HashMap%E5%B7%A5%E4%BD%9C%E5%8E%9F%E7%90%86%E5%8F%8A%E5%AE%9E%E7%8E%B0/)

[ConcurrentHashMap和HashTable的区别](http://www.importnew.com/7166.html)

[HashMap 1.7和1.8的区别](https://juejin.im/post/5aa5d8d26fb9a028d2079264" \l "heading-28)

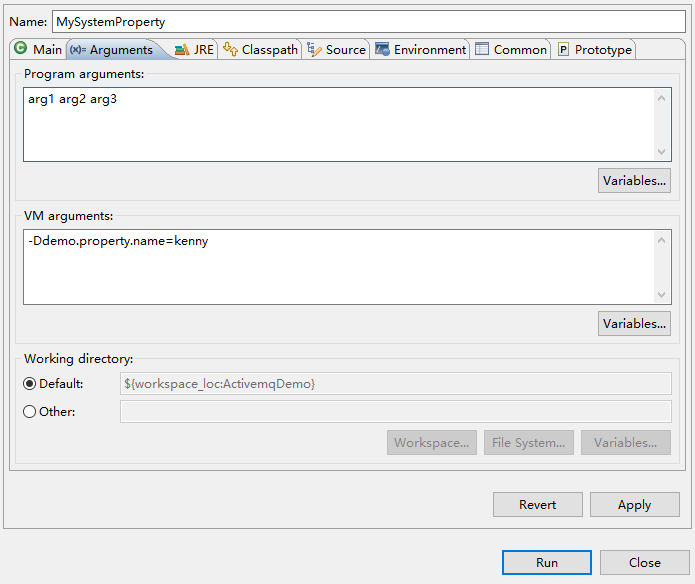
# java命令参数

## -D<propertyName>=value

在虚拟机的系统属性中设置属性键/值对，运行在此虚拟机之上的应用程序可用System.getProperty(“propertyName”)得到value的值。

如果value中有空格，则需要用双引号将该值括起来，如-Dname=”space string”。

该参数通常用于设置系统级全局变量值，如配置文件路径，因为该属性在程序中任何地方都可访问。



# Spring框架

## Spring发展的过程

第一阶段：xml配置，Spring 1.x

第二阶段：注解配置，Spring 2.x

第三阶段：Java配置，Spring 3.x到现在（Spring 4.x和Spring Boot都推荐使用Java配置）

## Spring框架概述

Spring框架是一个轻量级的企业级开发的一站式解决方案，可以基于Spring解决JavaEE开发的所有问题，提供了IoC容器、AOP、数据访问、Web开发、消息、测试等相关技术的支持。

## POJO

Plain Old Java Object，无任何限制的普通Java对象

## Spring模块

1. 核心容器（Core Container）

|  |  |
| --- | --- |
| Spring-Core | 核心工具类，Spring其他模块大量使用Spring-Core |
| Spring-Bean | Spring定义Bean的支持 |
| Spring-Context | 运行时Spring容器 |
| Spring-Context-Support | Spring容器对第三方包的集成支持 |

1. AOP

|  |  |
| --- | --- |
| Spring-AOP | 基于代理的AOP支持 |
| Spring-Aspects | 基于AspectJ的AOP支持 |

1. 消息（Messaging）

|  |  |
| --- | --- |
| Spring-Messaging | 对消息架构和协议的支持 |

1. Web

|  |  |
| --- | --- |
| Spring-Web | 提供基础的Web集成的功能，在Web项目中提供Spring的容器 |
| Spring-Webmvc | 提供基于Servlet的Spring MVC |
| Spring-WebSocket | 提供WebSocket功能 |
| Spring-Webmvc-Portlet | 提供Portlet环境支持 |

1. 数据访问/集成（Data Access/Integration）

|  |  |
| --- | --- |
| Spring-JDBC | 提供以JDBC访问数据库的支持 |
| Spring-TX | 提供编程式和声明式的失误支持 |
| Spring-ORM | 提供对对象/关系映射技术的支持 |
| Spring-OXM | 提供对对象/xml映射技术的支持 |
| Spring-JMS | 提供对JMS的支持 |

## Spring生态

Spring目前提供了大量的基于Spring的项目，可以用来更深入地降低我们的开发难度，提高开发效率。

|  |  |
| --- | --- |
| Spring Boot | 使用默认开发配置来实现快速开发 |
| Spring XD | 用来简化大数据应用开发 |
| Spring Cloud | 为分布式系统开发提供工具集 |
| Spring Data | 对主流的关系型和NoSQL数据库的支持 |
| Spring Integration | 通过消息机制对企业集成模式（EIP）的支持 |
| Spring Batch | 简化及优化大量数据的批处理操作 |
| Spring Security | 通过认证和授权保护应用 |
| Spring HATEOAS | 基于HATEOAS原则简化REST服务开发 |
| Spring Social | 与社交网络API（如Facebook、新浪微博等）的集成 |
| Spring AMQP | 对基于AMQP的消息的支持 |
| Spring Mobile | 提供对手机设备检测的功能，给不同的设备返回不同的页面的支持 |
| Spring for Android | 主要提供在Android上消费RESTful API的功能 |
| Spring Web Flow | 基于Spring MVC提供基于向导流程式的Web应用开发 |
| Spring Web Service | 提供了基于协议有限的SOAP/Web服务 |
| Spring LDAP | 简化使用LDAP开发 |
| Spring Session | 提供一个API及实现来管理用户会话信息 |

## 构建工具

目前主流的项目构建工具有：Ant、Maven、Gradle等。

Apache Maven，软件项目管理工具，基于项目对象模型（Project Object Model，POM）的概念，可用来管理项目的依赖、编译、文档等信息。

## Maven的pom.xml

|  |  |
| --- | --- |
| dependencies | 包含多个项目依赖需要使用的dependency |
| dependency | 通过groupId、artifactId以及version确定唯一的依赖 |
| groupId | 组织的唯一标识 |
| artifactId | 项目的唯一标识 |
| version | 项目的版本 |

变量定义：<properties></properties>可定义变量在dependency中的引用

依赖jar包参考库：http://mvnrepository.com

## 将Maven中心库中没有的jar包，打到本地Maven库后应用：

|  |
| --- |
| mvn install:install-file -DgroupId=com.oracle "-DartifactId=ojdbc14" "-Dversion=10.2.0.2.0" "-Dpackaging=jar" "-Dfile=D:\ojdbc14.jar" |

Spring框架四大原则

|  |
| --- |
| 使用POJO进行轻量级的最小侵入式开发 |
| 通过依赖注入和基于接口编程实现松耦合 |
| 通过AOP和默认习惯进行声明式编程 |
| 使用AOP和模板（template）减少模式化代码 |

## 声明Bean的注解

|  |  |
| --- | --- |
| @Component | 组件，没有明确的角色 |
| @Service | 在业务逻辑层（Service层）使用 |
| @Repository | 在数据访问层（dao层）使用 |
| @Controller | 在表现层（MVC->Spring MVC）使用 |

## 注入Bean的注解

|  |  |
| --- | --- |
| @Autowired | Spring提供的注解 |
| @Inject | JSR-330提供的注解 |
| @Resource | JSR-250提供的注解 |

@Autowired、@Inject、@Resource可注解在set方法上或者属性上，我习惯注解在属性上，优点是代码更少、层次更清晰。

## Autowired和Resource的区别

共同点：两者都可以写在字段和setter方法上。两者如果都写在字段上，那么就不需要写setter方法。

不同点：

@Autowired注解是按照类型（byType）装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许null值，可以设置它的required属性为false。如果想要按照名称（byName）来装配，可以结合@Qualifier注解一起使用。

@Resource默认按照byName自动注入。Spring将@Resource注解的name属性解析为Bean的名字，将type属性解析为bean的类型。

* 如果使用name属性，则使用byName的自动注入策略。
* 使用type属性时，则使用byType自动注入策略。
* 如果不指定name也不指定Type属性时，将通过反射机制使用byName自动注入策略。

@Resource装配顺序

* 1 如果同时指定了name和type，则从Spring上下文中找到唯一匹配的bean进行装配，找不到则抛出异常。
* 2 如果指定了name，则从上下文中查找名称（id）匹配的bean进行装配，找不到则抛出异常。
* 3 如果指定了type，则从上下文中找到匹配的唯一bean进行装配，找不到或是找到多个，都会抛出异常。
* 4 如果既没有指定name，又没有指定type，则自动按照byName方式进行装配；如果没有匹配，则回退为一个原始类型进行匹配，如果匹配则自动装配
* @Resource的作用相当于@Autowired，只不过@Autowired按照byType自动注入。

## Spring EL和资源调用

Spring EL是Spring表达式语言，支持在xml和注解中使用表达式，类似于JSP的EL表达式语言。

Spring开发中经常涉及调用各种资源的情况，包括普通文件、网址、配置文件、系统环境变量等，我们可以使用Spring的表达式语言实现资源的注入。

Spring主要在注解@Value的参数中使用表达式。

|  |
| --- |
| 注入普通字符 |
| 注入操作系统属性 |
| 注入表达式运算结果 |
| 注入其他Bean的属性 |
| 注入文件内容 |
| 注入网址内容 |
| 注入属性文件 |

## Bean的初始化和销毁

Bean在使用之前或者之后做些必要的操作，Spring对Bean的生命周期的操作提供了支持。

1. Java配置方式：使用@Bean的initMethod和destroyMethod（相当于xml配置的init-method和destroy-method）。initMethod和destroyMethod指定Bean类的init和destroy方法在构造函数之后，Bean销毁之前执行。
2. 注解方式：利用JSR-250的@PostConstruct和@PreDestroy。

@PostConstruct，在构造函数执行完之后执行。

@PreDestroy，在Bean销毁之前执行。

## Profile

Profile为在不同环境下使用不同的配置提供了支持（开发环境下的配置和生产环境下的配置肯定是不同的，例如，数据库配置）。

|  |
| --- |
| 通过设定Environment的ActiveProfiles来设定当前Context需要使用的配置环境。在开发中使用@Profile注解类或者方法，达到在不同情况下选择实例化不同的Bean。 |
| 通过设定jvm的spring.profiles.active参数来设置配置环境。 |
| Web项目设置在Servlet的Context parameter中。 |

## 事件（Application Event）

Spring的事件（Application Event）为Bean与Bean之间的消息通信提供了支持。当一个Bean处理完一个任务之后，希望另一个Bean知道并能做相应的处理，这时我们就需要让另外一个Bean监听当前Bean所发送的事件。

Spring的事件需要遵循如下流程：

|  |
| --- |
| （1）自定义事件，继承ApplicationEvent |
| （2）定义事件监听器，实现ApplicationListener<...> |
| （3）使用容器发布事件，注入ApplicationContext用来发布事件，使用ApplicationContext的publishEvent方法来发布。 |

## Spring Aware

Spring的依赖注入的最大亮点就是所有的Bean对Spring容器的存在是没有意识的。即你可以将你的容器替换成别的容器，如Google Guice，这时Bean之间的耦合度很低。

但是在实际项目中，不可以避免的要用到Spring容器本身的功能资源，这时你的Bean必须要意识到Spring容器的存在，才能调用Spring所提供的资源，这就是所谓的Spring Aware。其实Spring Aware本来就是Spring设计用来框架内部使用的，若使用了Spring Aware，你的Bean将会和Spring框架耦合。

Spring提供的Aware接口

|  |  |
| --- | --- |
| BeanNameAware | 获取到容器中Bean的名称 |
| BeanFactoryAware | 获取当前bean factory，这样可以调用容器的服务 |
| ApplicationContextAware | 当前的Application Context，这样可以调用容器的服务 |
| MessageSourcesAware | 获取Message source，这样可以获得文本信息 |
| ApplicationEventPublisherAware | 应用事件发布器，2.5节的DemoPublisher也可实现这个接口来发布事件 |
| ResourceLoadAware | 获得资源加载器，可以获取外部资源文件 |

Spring Aware的目的是为了让Bean获得Spring容器的服务。因为ApplicationContext接口集成了MessageSource接口、ApplicationEventPublisher接口和ResourceLoader接口，所以Bean集成ApplicationContextAware可以获得Spring容器的所有服务，但原则上我们还是用到什么接口，就实现什么接口。

## 多线程

Spring通过任务执行器（TaskExecutor）来实现多线程和并发编程。使用ThreadPoolTaskExecutor可实现一个基于线程池的TaskExecutor。而实际开发中任务一般是非阻塞的，即异步的，所以我们要在配置类中通过@EnableAsync开启对异步任务的支持，并通过在实际执行的Bean的方法中使用@Async注解来声明其是一个异步任务。

## 计划任务

从Spring 3.1开始，计划任务在Spring中的实现变得异常简单。首先通过在配置类注解@EnableScheduling来开启对计划任务的支持，然后在要执行计划任务的方法上注解@Scheduled，声明这是一个计划任务。

Spring通过@Scheduled支持多种类型的计划任务，包含cron、fixDelay、fixRate等。

|  |
| --- |
| @Scheduled(cron = "0 28 11 ? \* \*") |

cron是UNIX和类UNIX（Linux）系统下的定时任务。

## 条件注解@Conditional

## 组合注解与元注解

## @Enable×注解的工作原理

|  |  |
| --- | --- |
| @EnableAspectJAutoProxy | 开启对AspectJ自动代理的支持。 |
| @EnableAsync | 开启异步方法的支持。 |
| @EnableScheduling | 开启计划任务的支持。 |
| @EnableWebMvc | 开启Web MVC的配置支持。 |
| @EnableConfigurationProperties | 开启对@ConfigurationProperties注解配置Bean的支持。 |
| @EnableJpaRepositories | 开启对Spring Data JPA Repository的支持。 |
| @EnableTransactionManagement | 开启注解式事务的支持。 |
| @EnableCaching | 开启注解式的缓存支持。 |

# Spring MVC 4.x

## 基本概念

MVC：Model + View + Controller（数据模型+视图+控制器）

三层架构：Presentation tier + Application tier + Data tier（展现层+应用层+数据访问层）

实际上MVC只存在三层架构的展现层。

## 使用Servlet 3.0+无web.xml的配置方式

Spring MVC里实现WebApplicationInitializer接口便可实现等同于web.xml的配置。

Spring Boot的页面就放置在src/main/resources下。

1 WebApplicationInitializer是Spring提供用来配置Servlet 3.0+配置的接口，从而实现了替代web.xml的位置。实现此接口将会自动被SpringServletContainerInitializer（用来启动Servlet 3.0容器）获取到。

2 新建WebApplicationContext，注册配置类，并将其和当前ServletContext关联。

3 注册Spring MVC的DispatcherServlet

|  |
| --- |
| mvn package  cd /java/tomcat/bin  ./startup.sh  ./shutdown.sh  ps -ef | grep tomcat  kill -9 [pid]  cd logs  tail -f catalina.out |

Spring MVC的常用注解

|  |  |
| --- | --- |
| @Controller |  |
| @RequestMapping |  |
| @ResponseMapping |  |
| @RequestBody |  |
| @PathVariable |  |
| @RestController |  |

添加Jackson及相关依赖，获得对象和json或xml之间的转换。

# Spring Boot基础

Spring Boot使用“习惯优于配置”的理念让项目快速运行起来。使用Spring Boot很容易创建一个独立运行（运行jar，内嵌Servlet容器）、准生产级别的基于Spring框架的项目，使用Spring Boot你可以不用或者只需要很少的Spring配置。

|  |
| --- |
| mvn spring-boot:run |

## 入口类和@SpringBootApplication

Spring Boot通常有一个名为×Application的入口类，入口类里有一个main方法，这个main方法其实就是一个标准的Java应用的入口方法。

|  |  |
| --- | --- |
| @SpringBootApplication | @Configuration |
| @EnableAutoConfiguration |
| @ComponentScan |

## 关闭特定的自动配置

|  |
| --- |
| @SpringBootApplication(exclude = {DataSourceAutoConfiguration.class}) |

## 定制Banner

修改Banner

在src/main/resources下新建一个banner.txt

关闭Banner

|  |
| --- |
| SpringApplication.*run*(Ch522Application.**class**, args); |

|  |
| --- |
| SpringApplication app = **new** SpringApplication(Ch522Application.**class**);  app.setBannerMode(Banner.Mode.***OFF***);  app.run(args); |

使用fluent API修改：

|  |
| --- |
| **new** SpringApplicationBuilder(Ch522Application.**class**)  .bannerMode(Banner.Mode.***OFF***)  .run(args); |

## SpringBoot的配置文件

全局的配置文件application.properties或application.yml

# Docker

Docker镜像检索

|  |
| --- |
| docker search 镜像名  docker search redis |

镜像下载

|  |
| --- |
| docker pull 镜像名  docker pull redis |

镜像列表

|  |
| --- |
| docker images |

镜像删除

|  |
| --- |
| docker rmi image-id |

删除所有镜像

|  |
| --- |
| docker rmi ${docker image -q} |

最简单的运行镜像为容器的命令

|  |
| --- |
| docker run - - name container-name -d image-name  docker run - - name test-redis -d redis |

Docker会为我们的容器生成唯一的标识。

容器列表

|  |
| --- |
| docker ps  docker ps -a |

停止容器

|  |
| --- |
| docker stop container-name/container-id  docker stop test-redis |

启动容器

|  |
| --- |
| docker start container-name/container-id  docker start test-redis |

端口映射

|  |
| --- |
| docker run -d -p 6378:6379 - -name port-redis redis |

删除容器

|  |
| --- |
| docker rm container-id |

删除所有容器

|  |
| --- |
| docker rm ${docker ps -a -q} |

容器日志

|  |
| --- |
| docker logs container-name/container-id  docker logs port-redis |

登录容器(登录访问当前容器，登录后我们可以在容器中进行常规的Linux系统操作命令，还可以使用exit命令退出登录)

|  |
| --- |
| docker exec -it container-id/container-name bash |

## Docker Oracle

容器提供的安装信息

|  |  |
| --- | --- |
| hostname | localhost |
| port | 1521 |
| SID | XE |
| username | system/sys |
| password | oracle |

管理界面访问

|  |  |
| --- | --- |
| url | http://localhost:9090/apex |
| workspace | internal |
| username | admin |
| password | oracle sai199311 |

|  |
| --- |
| mvn install:install-file -DgroupId=com.oracle "-DartifactId=ojdbc6" "-Dversion=11.2.0.2.0" "-Dpackaging=jar" "-Dfile=./ojdbc6.jar" |

# Spring Boot

Spring Data REST

声明式事务（Spring的事务机制）

数据缓存Cache（Spring缓存支持）

切换缓存技术

非关系型数据库NoSQL

MongoDB

Redis

# Spring Web

## Thymeleaf模板引擎

Spring Boot提供了大量模板引擎，包括FreeMarker、Groovy、Thymeleaf、Velocity和Mustache，Spring Boot中推荐使用Thymeleaf作为模板引擎，因为Thymeleaf提供了完美的Spring MVC的支持。

Thymeleaf是一个Java类库，它是一个xml/xhtml/html5的模板引擎，可以作为MVC的Web应用的View层。

## 配置Tomcat

配置Servlet容器

|  |  |
| --- | --- |
| server.port= | #配置程序端口，默认为8080 |
| server.session-timeout= | #用户会话session过期时间，以秒为单位 |
| server.context-path= | #配置访问路径，默认为/ |

配置Tomcat

|  |  |
| --- | --- |
| server.tomcat.uri-encoding= | #配置Tomcat编码，默认为UTF-8 |
| server.tomcat.uri.compression= | #Tomcat是否开启压缩，默认为关闭off |

特定配置

|  |  |
| --- | --- |
| Tomcat | TomcatEmbeddedServletContainerFactory |
| Jetty | JettyEmbeddedServletContainerFactory |
| Undertow | UndertowEmbeddedServletContainerFactory |

替换Tomcat为Jetty

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jetty</artifactId>  </dependency> |

替换Tomcat为Undertow

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-undertow</artifactId>  </dependency> |

## Favicon配置

默认的Favicon：Spring Boot提供了一个默认的Favicon，每次访问应用的时候都能看到。

关闭Favicon

可以在application.properties中设置关闭Favicon，默认为开启

|  |
| --- |
| spring.mvc.facicon.enabled=false |

设置自己的Favicon

需要设置自己的Favicon，则只需要将自己的favicon.ico（文件名不能变动）文件放置在

类路径根目录下

类路径META-INF/resources/下

类路径resources/下

类路径static/下

类路径public/下

这里将favicon.ico放置在src/main/resources/static下

## WebSocket

Spring Boot对内嵌的Tomcat（7或者8）、Jetty9和Undertow使用WebSocket提供支持。配置源码存于org.springframework.boot.autoconfigure.websocket下。

Spring Boot为WebSocket提供的start pom是spring-boot-starter-websocket。

广播式：即服务端有消息时，会将消息发送所有连接了当前endpoint的浏览器。

|  |
| --- |
| @Configuration  @EnableWebSocketMessageBroker  **public** **class** WebSocketConfig **extends** AbstractWebSocketMessageBrokerConfigurer {  @Override  **public** **void** registerStompEndpoints(StompEndpointRegistry registry) {  registry.addEndpoint("/endpointWisely").withSockJS();  }  @Override  **public** **void** configureMessageBroker(MessageBrokerRegistry registry) {  registry.enableSimpleBroker("/topic");  }  } |

1. 通过@EnableWebSocketMessageBroker注解开启使用STOMP协议来传输基于代理（message broker）的消息，这时控制器支持使用@MessageMapping，就像使用@RequestMapping一样。
2. 注册STOMP协议的节点（endpoint），并映射到指定的URL。
3. 注册一个STOMP的endpoint，并指定使用SockJS协议。
4. 配置消息代理（Message Broker）。
5. 广播式应配置一个/topic消息代理。
6. 当浏览器向服务器发送请求时，通过@MessageMapping映射/welcome这个地址，类似于@RequestMapping。
7. 当服务端有消息时，会对订阅了@SendTo中的路径的浏览器发送消息。
8. 添加脚本。将stomp.min.js（STOMP协议的客户端脚本）、sockjs.min.js（SockJS的客户端脚本）以及JQuery放置在src/main/resources/static下。