课程目标

1、通过对本章内容的学习，可以掌握Spring的基本架构及各子模块之间的依赖关系。
2、了解Spirng的发展历史，启发思维。

3、对Spring形成一个整体的认识，为之后的深入学习做铺垫。
4.通过对本章内容的学习，可以了解Spring版本升级的规律，从而应用到自己的系统升级版本命名。
5、Spring 编程思想总结。

内容定位

Spring 使用经验1-5年，希望深入了解Spring源码的人群。

1. Spring的前世今生

相信经历过不使用框架开发Web项目的70后、80后都会有如此感触，如今的程序员开发项目太轻松了，基本只需要关心业务如何实现，通用技术问题只需要集成框架便可。早在2007年，一个基于Java 语言的开源框架正式发布，取了一个非常有活力且美好的名字，叫做 Spring。它是一个开源的轻量级JavaSE（Java 标准版本）/JavaEE（Java企业版本）开发应用框架，其目的是用于简化企业级应用程序开发。应用程序是由一组相互协作的对象组成。而在传统应用程序开发中，一个完整的应用是由一组相互协作的对象组成。所以开发一个应用除了要开发业务逻辑之外，最多的是关注如何使这些对象协作来完成所需功能，而且要低耦合、高聚合。业务逻辑开发是不可避免的，那如果有个框架出来帮我们来创建对象及管理这些对象之间的依赖关系。可能有人说了，比如“抽象工厂、工厂方法模式"不也可以帮我们创建对象，“生成器模式”帮我们处理对象间的依赖关系，不也能完成这些功能吗？可是这些又需要我们创建另一些工厂类、生成器类，我们又要额外管理这些类，增加了我们的负担，如果能有种通过配置方式来创建对象，管理对象之间依赖关系，我们不需要通过工厂和生成器来创建及管理对象之间的依赖关系，这样我们是不是减少了许多工作，加速了开发，能节省出很多时间来干其他事。Spring框架刚出来时主要就是来完成这个功能。

Spring 框架除了帮我们管理对象及其依赖关系，还提供像通用日志记录、性能统计、安全控制、异常处理等面向切面的能力，还能帮我管理最头疼的数据库事务，本身提供了一套简单的JDBC访问实现，提供与第三方数据访问框架集成（如Hibernate、JPA），与各种JavaEE技术整合（如Java Mail、任务调度等等），提供一套自己的Web 层框架 Spring MVC、而且还能非常简单的与第三方Web框架集成。从这里我们可以认为Spring是一个超级粘合大平台，除了自己提供功能外，还提供粘合其他技术和框架的能力，从而使我们可以更自由的选择到底使用什么技术进行开发。而且不管是JAVASE（C/S架构）应用程序还是JAVAEE（B/S架构）应用程序都可以使用这个平台进行开发。如今的Spring已经不再是一个框架，早已成为了一种生态。SpringBoot 的便捷式开发实现了零配置，SpringCloud全家桶，提供了非常方便的解决方案。接下来，让我们来深入探讨Spring 到底能给我们带来什么？

2. 一切从Bean开始

说到Bean这个概念，还得从Java的起源说起。早在1996年，Java还只是一个新兴的、初出茅庐的编程语言。人们之所以关注她仅仅是因为，可以使用Java的Applet 来开发Web应用，作为浏览器组件。但开发者们很快就发现这个新兴的语言还能做更多的事情。与之前的所有语言不同，Java 让模块化构建复杂的系统成为可能（当时的软件行业虽然在业务上突飞猛进，但当时开发用的是传统的面向过程开发思想，软件的开发效率一直踟蹰不前。伴随着业务复杂性的不断加深，开发也变得越发困难。其实，当时也是OOP思想飞速发展的时期，她在80年代末被提出，成熟于90年代，现今大多数编程语言都已经是面向对象的）。

同年12月，Sun公司发布了当时还名不见经传但后来人尽皆知的JavaBean 1.00-A规范。早期的JavaBean规范针对于Java，她定义了软件组件模型。这个规范规定了一整套编码策略，使简单的Java对象不仅可以被重用，而且还可以轻松地构建更为复杂的应用。尽管JavaBean最初是为重用应用组件而设计的，但当时他们却是主要用作构建窗体控件，毕竟在PC时代那才是主流。但相比于当时正如日中天的Delphi、VB和C++，它看起来还是太简易了，以至于无法胜任任何“实际的“工作需要。

复杂的应用通常需要事务、安全、分布式等服务的支持，但JavaBean并未直接提供。所以到了1998年3月，Sun公司发布了EJB1.0规范，该规范把Java组件的设计理念延伸到了服务器端，并提供了许多必须的企业级服务，但他也不再像早期的JavaBean 那么简单了。实际上，除了名字叫EJB Bean以外，其他的和JavaBean关系不大了。

尽管现实中有很多系统是基于EJB构建的，但EJB从来没有实现它最初的设想：简化开发。EJB的声明式编程模型的确简化了很多基础架构层面的开发，例如事务和安全；但另一方面EJB在部署描述符和配套代码实现等方面变得异常复杂。随着时间的推移，很多开发者对EJB已经不再抱有幻想，开始寻求更简洁的方法。

现在Java组件开发理念重新回归正轨。新的编程技术AOP和DI的不断出现，他们为JavaBean提供了之前EJB才能拥有的强大功能。这些技术为POJO提供了类似EJB的声明式编程模型，而没有引入任何EJB的复杂性。当简单的JavaBean足以胜任时，人们便不愿编写笨重的EJB组件了。客观地讲，EJB的发展甚至促进了基于POJO的编程模型。引入新的理念，最新的EJB规范相比之前的规范有了前所未有的简化，但对很多开发者而言，这一切的一切都来得太迟了。到了EJB3规范发布时，其他基于POJO的开发架构已经成为事实的标准了，而Spring 框架也就是在这样的大环境下出现的。

3. Spring的设计初衷

Spring是为解决企业级应用开发的复杂性而设计，她可以做很多事。但归根到底支撑Spring的仅仅是少许的基本理念，而所有的这些基本理念都能可以追溯到一个最根本的使命：简化开发。这是一个郑重的承诺，其实许多框架都声称在某些方面做了简化。而Spring 则立志于全方面的简化Java开发。

对此，她主要采取了4个关键策略：
1、基于POJO的轻量级和最小侵入性编程；
2、通过依赖注入和面向接口松耦合；
3、基于切面和惯性进行声明式编程；
4、通过切面和模板减少样板式代码；而他主要是通过：面向Bean（BOP）、依赖注入（DI）以及面向切面（AOP）这三种方式来达成的。

4. BOP编程伊始

Spring 是面向Bean的编程（Bean Oriented Programming，BOP），Bean在Spring中才是真正的主角。Bean在Spring中作用就像Object对OOP的意义一样Spring中没有Bean也就没有Spring存在的意义。Spring提供了IOC容器通过配置文件或者注解的方式来管理对象之间的依赖关系。

控制反转（其中最常见的实现方式叫做依赖注入（Dependency Injection，DI），还有一种方式叫“依赖查找”（Dependency Lookup，DL），她在C++、Java、PHP以及.NET中都运用。在最早

的Spring 中是包含有依赖注入方法和依赖查询的，但因为依赖查询使用频率过低，不久就被Spring移除了，所以在Spring中控制反转也被直接称作依赖注入），她的基本概念是：不创建对象，但是描述创建它们的方式。在代码中不直接与对象和服务连接，但在配置文件中描述哪一个组件需要哪一项服务。容器（在Spring框架中是IOC容器）负责将这些联系在一起。

在典型的IOC场景中，容器创建了所有对象，并设置必要的属性将它们连接在一起，决定什么时间调用方法。

5. 依赖注入的基本概念

Spring 设计的核心org.springframework.beans包（架构核心是org.springframework.core包），它的设计目标是与JavaBean组件一起使用。这个包通常不是由用户直接使用，而是由服务器将其用作其他多数功能的底层中介。下一个最高级抽象是BeanFactory接口，它是工厂设计模式的实现，允许通过名称创建和检索对象。BeanFactory也可以管理对象之间的关系。

BeanFactory 最底层支持两个对象模型。
1，单例：提供了具有特定名称的全局共享实例对象，可以在查询时对其进行检索。Singleton是默认的也是最常用的对象模型。
2，原型：确保每次检索都会创建单独的实例对象。在每个用户都需要自己的对象时，采用原型模式。
Bean 工厂的概念是Spring作为IOC 容器的基础。IOC则将处理事情的责任从应用程序代码转移到框架。

6. AOP编程理念

面向切面编程，即AOP，是一种编程思想，它允许程序员对横切关注点或横切典型的职责分界线的行为（例如日志和事务管理）进行模块化。AOP的核心构造是方面（切面），它将那些影响多个类的行为封装到可重用的模块中。

AOP和IOC是补充性的技术，它们都运用模块化方式解决企业应用程序开发中的复杂问题。在典型的面向对象开发方式中，可能要将日志记录语句放在所有方法和Java类中才能实现日志功能。在AOP方式中，可以反过来将日志服务模块化，并以声明的方式将它们应用到需要日志的组件上。当然，优势就是Java类不需要知道日志服务的存在，也不需要考虑相关的代码。所以，用Spring AOP编写的应用程序代码是松散耦合的。

AOP的功能完全集成到了Spring事务管理、日志和其他各种特性的上下文中。

AOP编程的常用场景有：Authentication（权限认证）、Auto Caching（自动缓存处理）、Error Handling （统一错误处理）、Debugging（调试信息输出）、Logging（日志记录）、Transactions（事务处理）

7. Spring中的编程思想总结

7.1. Spring编程思想

Spring 思想	应用场景（特点）	一句话归纳
OOP	Object Oriented Programming（面向对象编程）用程序归纳总结生活中一切事物。	封装、继承、多态。
BOP	Bean Oriented Programming（面向 Bean 编程）面向 Bean（普通的 Java 类）设计程序，解放程序员。	一切从 Bean 开始。
AOP	Aspect Oriented Programming(面向切面编程)找出多个类中有一定规律的代码，开发时拆开，运行时再合并。 面向切面编程，即面向规则编程。	解耦，专人做专事。
IOC	Inversion of Control（控制反转） 将 new 对象的动作交给 Spring 管理，并由 Spring 保存已创建的对象（IOC 容器）。	转交控制权（即控制权反转）
DI/DL	Dependency Injection（依赖注入）或者 Dependency Lookup（依赖查找） 依赖注入、依赖查找，Spring 不仅保存自己创建的对象，而且保存对象与对象之间的关系。 注入即赋值，主要三种方式构造方法、set 方法、直接赋值。	赋值

7.2. Spring注解编程

7.2.1. 1、Spring Framework 1.x注解驱动启蒙时代

从Spring Framework 1.2.0版本开始，开始支持Annotation，虽然框架层面均已支持@managedResource 和@Transactional等，但是其主要的还是以XML配置为准。

7.2.2. 2、Spring Framework 2.X注解驱动过渡时代

Spring Framework 2.0在Annotation 支持方面添加了新的成员，@Required、数据相关的@Repository及AOP相关的@Aspect等，但同时提升了XML配置能力，即“可扩展的XML编写

（Extensible XML authoring）"，当然的，这种扩展能力的出现，无形中为XML的配置增加了筹码。

7.2.3. 3、Spring Framework 2.5开始，引入了新的骨架式Annotation

@Service
@Controller，@RequestMapping 及@ModelAttribute
Spring Framework 2.5还支持了JSR-250（Java规范）。
@Resource注入
@PostConstruct 替代 `<bean init-method="…"/>`
@PreDestroy 替代 `<bean destroy-method="…"/>`

尽管Spring Framework2.X时代提供了为数不少的注解，然而编程的手段却不多，最主要的原因在于框架层面仍未“直接”提供驱动注解的Spring应用上下文，并且仍需要XML驱动，`<context:annotation-config>`和`<context:component-scan>`

7.2.4. 4、Spring Framework 3.x注解驱动黄金时代

Spring Framework3.x是一个里程碑式的时代，3.0除了提升Spring 模式注解“派生”的层次性，首要任务是替换XML配置方式，引入配置类注解@Configuration，该注解是内建的@Component的“派生”注解，遗憾的是，3.0并没有引入 `<context:component-scan>` 的注解，而是选择过渡方案— @ImportResource 和@Import。ImportResource负责导入遗留的XML配置文件，Import允许导入一个或多个类作为Spring Bean。

3.0引入AnnotationConfigApplicationContext 最为前时代ApplicationContext的替代者。3.1新引入注解@ComponentScan，替换XML的 `<context:component-scan>` ，成为全面进入注解驱动时代的一大步。

SpringFramework3.x注解提升还体现在以下方面：
定义声明中，@Bean 允许使用注解@Role 设置其角色使得Spring应用上下文具备条件化Bean定义的能力方面，@RequestHeader，@CookieVale 和@RequestPart出现，使得不必使用Servlet API以及配置属性源抽象PropertySources，奠定了SpringBoot外部化配置的基础。

配套的注解 Caching和Cacheable极大简化数据缓存的开发。

周期异步@Schedule 及异步web 请求DeferredResult。

7.2.5. 5、Spring Framework 4.x 注解驱动完善时代

3.1开始提供@Profile提供了配置化的条件组装，不过这方面还是比较单一的，4.0开始，引入条件化注解@Conditional，通过自定义Condition 实现配合，弥补之前版本条件化装配的短板，4.0开始Profile反过来通过@conditional实现。

Java8开始对提供@Repeatable，Framework4.0巧妙的兼容了JSR-310。根据特性，将@PropertySource、@ComponentScan 提升为可重复使用的注解@PropertySources、@ComponentScans。

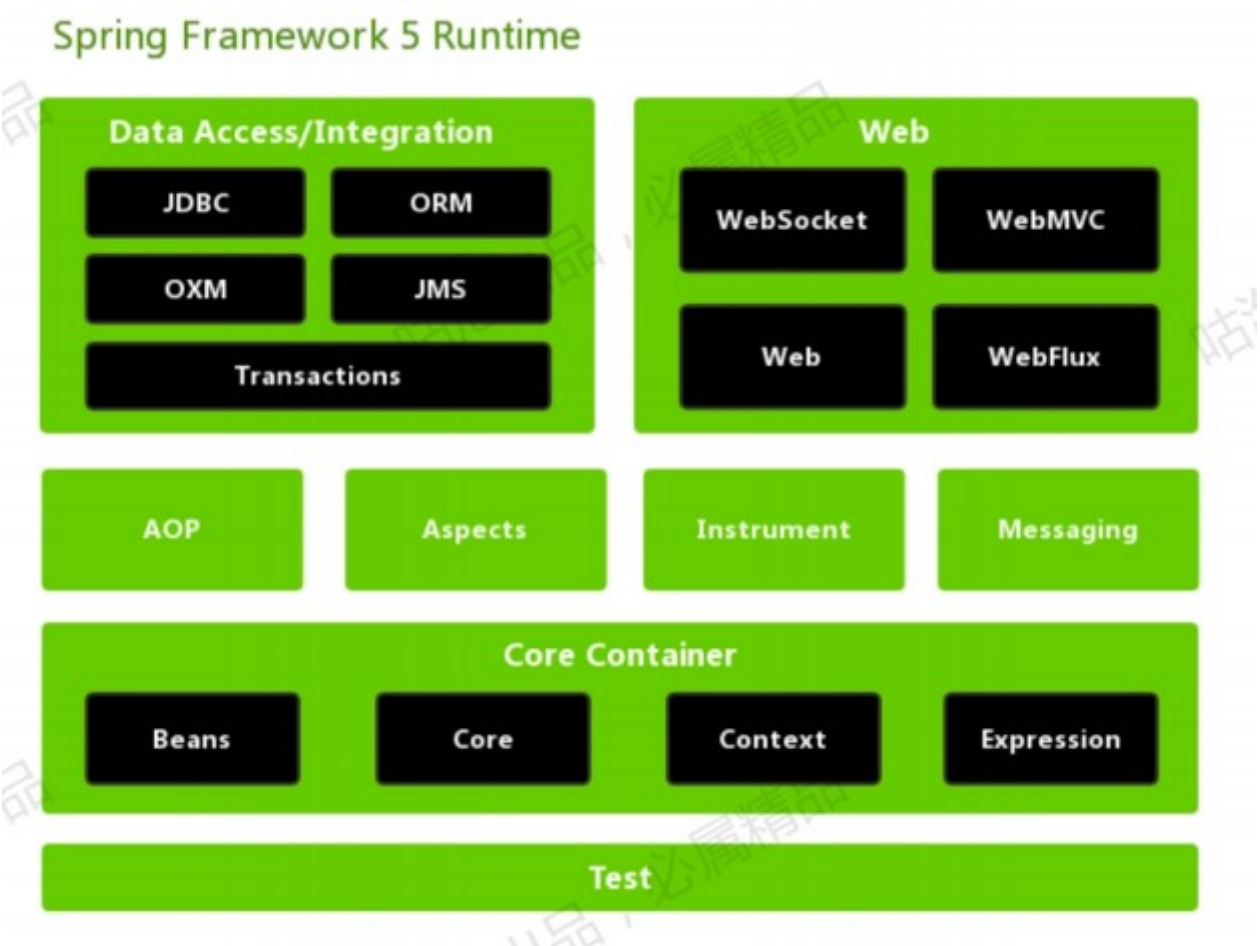
4.2开始新增了事件监听器注解@EventListener，作为ApplicationListener 接口编程的第二选择。
@AliasFor 解除注解派生的时候冲突限制
在浏览器跨域资源访问方面，引入@CrossOrigin，作为CorsRegistry 替换注解方案。

7.2.6. 6、SpringFramework 5.x注解驱动成熟时代

SpringFramework5.0作为Spring Boot2.0的底层，注解驱动的性能提升不是那么明显。在SpringBoot 应用场景中，大量使用@ComponentScan扫描，导致Spring 模式的注解解析时间耗时越长，面对这个问题，5.0引入@Indexed，为Spring 模式注解添加索引。

8. Spring5 系统架构

Spring总共大约有20个模块，由1300多个不同的文件构成。而这些组件被分别整合在核心容器（Core Container）、AOP（Aspect Oriented Programming）和设备支持（Instrmentation）、数据访问及集成（Data Access/Integeration）、Web、报文发送（Messaging）、Test，6个模块集合中。以下是Spring5的模块结构图：



组成Spring框架的每个模块集合或者模块都可以单独存在，也可以一个或多个模块联合实现。每个模块的组成和功能如下：

8.1. 核心容器

由spring-beans、spring-core、spring-context 和 spring-expression（Spring Expression Language，SpEL）4个模块组成。

spring-core 和spring-beans 模块是Spring 框架的核心模块，包含了控制反转（Inversion of Control，IOC）和依赖注入（Dependency Injection，DI）。BeanFactory 接口是Spring框架中的核心接口，它是工厂模式的具体实现。BeanFactory使用控制反转对应用程序的配置和依赖性规范与实际的应用程序代码进行了分离。但BeanFactory 容器实例化后并不会自动实例化Bean，只有当Bean被使用时BeanFactory 容器才会对该Bean进行实例化与依赖关系的装配。

spring-context模块构架于核心模块之上，他扩展了BeanFactory，为她添加了Bean生命周期控制、框架事件体系以及资源加载透明化等功能。此外该模块还提供了许多企业级支持，如邮件访问、远程访问、任务调度等，ApplicationContext是该模块的核心接口，她的超类是BeanFactory。与BeanFactory 不同，ApplicationContext 容器实例化后会自动对所有的单实例Bean进行实例化与依赖关系的装配，使之处于待用状态。

spring-context-support模块是对Spring IOC容器的扩展支持，以及IOC子容器。

spring-context-indexer 模块是Spring的类管理组件和Classpath扫描。

spring-expression 模块是统一表达式语言（EL）的扩展模块，可以查询、管理运行中的对象，同时也方便的可以调用对象方法、操作数组、集合等。它的语法类似于传统EL，但提供了额外的功能，最出色的要数函数调用和简单字符串的模板函数。这种语言的特性是基于Spring产品的需求而设计，他可以非常方便地同Spring IOC进行交互。

8.2. AOP和设备支持

由spring-aop、spring-aspects和spring-instrument 3个模块组成。

spring-aop是Spring的另一个核心模块，是AOP主要的实现模块。作为继OOP后，对程序员影响最大的编程思想之一，AOP极大地开拓了人们对于编程的思路。在Spring中，他是以JVM的动态代理技术为基础，然后设计出了一系列的AOP横切实现，比如前置通知、返回通知、异常通知等，同时，Pointcut接口来匹配切入点，可以使用现有的切入点来设计横切面，也可以扩展相关方法根据需求进行切入。

spring-aspects 模块集成自AspectJ框架，主要是为Spring AOP提供多种AOP实现方法。

spring-instrument模块是基于JAVASE中的“java.lang.instrument“进行设计的，应该算是AOP的一个支援模块，主要作用是在JVM启用时，生成一个代理类，程序员通过代理类在运行时修改类的字节，从而改变一个类的功能，实现AOP的功能。在分类里，我把他分在了AOP模块下，在Spring官方文档里对这个地方也有点含糊不清，这里是纯个人观点。

8.3. 数据访问与集成

由spring-jdbc、spring-tx、spring-orm、spring-jms和spring-oxm 5个模块组成。

spring-jdbc模块是Spring 提供的JDBC抽象框架的主要实现模块用于简化Spring JDBC操作。
主要是提供JDBC模板方式、关系数据库对象化方式、SimpleJdbc方式、事务管理来简化JDBC编程，主要实现类是JdbcTemplate、SimpleJdbcTemplate 以及NamedParameterJdbcTemplate。

spring-tx模块是Spring JDBC事务控制实现模块。使用Spring框架，它对事务做了很好的封装，通过它的AOP配置，可以灵活的配置在任何一层；但是在很多的需求和应用，直接使用JDBC事务控制

还是有其优势的。其实，事务是以业务逻辑为基础的；一个完整的业务应该对应业务层里的一个方法；如果业务操作失败，则整个事务回滚；所以，事务控制是绝对应该放在业务层的；但是，持久层的设计则应该遵循一个很重要的原则：保证操作的原子性，即持久层里的每个方法都应该是不可以分割的。所以，在使用Spring JDBC事务控制时，应该注意其特殊性。

spring-orm模块是ORM框架支持模块，主要集成Hibernate，Java Persistence API(JPA)和Java Data Objects(JDO)用于资源管理、数据访问对象（DAO）的实现和事务策略。

spring-oxm模块主要提供一个抽象层以支撑OXM（OXM是Object-to-XML-Mapping的缩写，它是一个O/M-mapper，将java对象映射成XML数据，或者将XML数据映射成java对象），例如：JAXB，Castor，XMLBeans，JiBX和XStream等。

spring-jms 模块（Java Messaging Service）能够发送和接收信息，自Spring Framework4.1以后，他还提供了对spring-messaging模块的支撑。

8.4. Web组件

由spring-web、spring-webmvc、spring-websocket 和spring-webflux4个模块组成。

spring-web模块为Spring 提供了最基础Web支持，主要建立于核心容器之上，通过Servlet或者Listeners 来初始化IOC容器，也包含一些与Web相关的支持。

spring-webmvc模块众所周知是一个的Web-Servlet模块，实现了Spring MVC（model-view-Controller）的Web应用。

spring-websocket 模块主要是与Web前端的全双工通讯的协议。

spring-webflux是一个新的非堵塞函数式Reactive Web框架，可以用来建立异步的，非阻塞，事件驱动的服务，并且扩展性非常好。

8.5. 通信报文

即spring-messaging模块，是从Spring4开始新加入的一个模块，主要职责是为Spring 框架集成一些基础的报文传送应用。

8.6. 集成测试

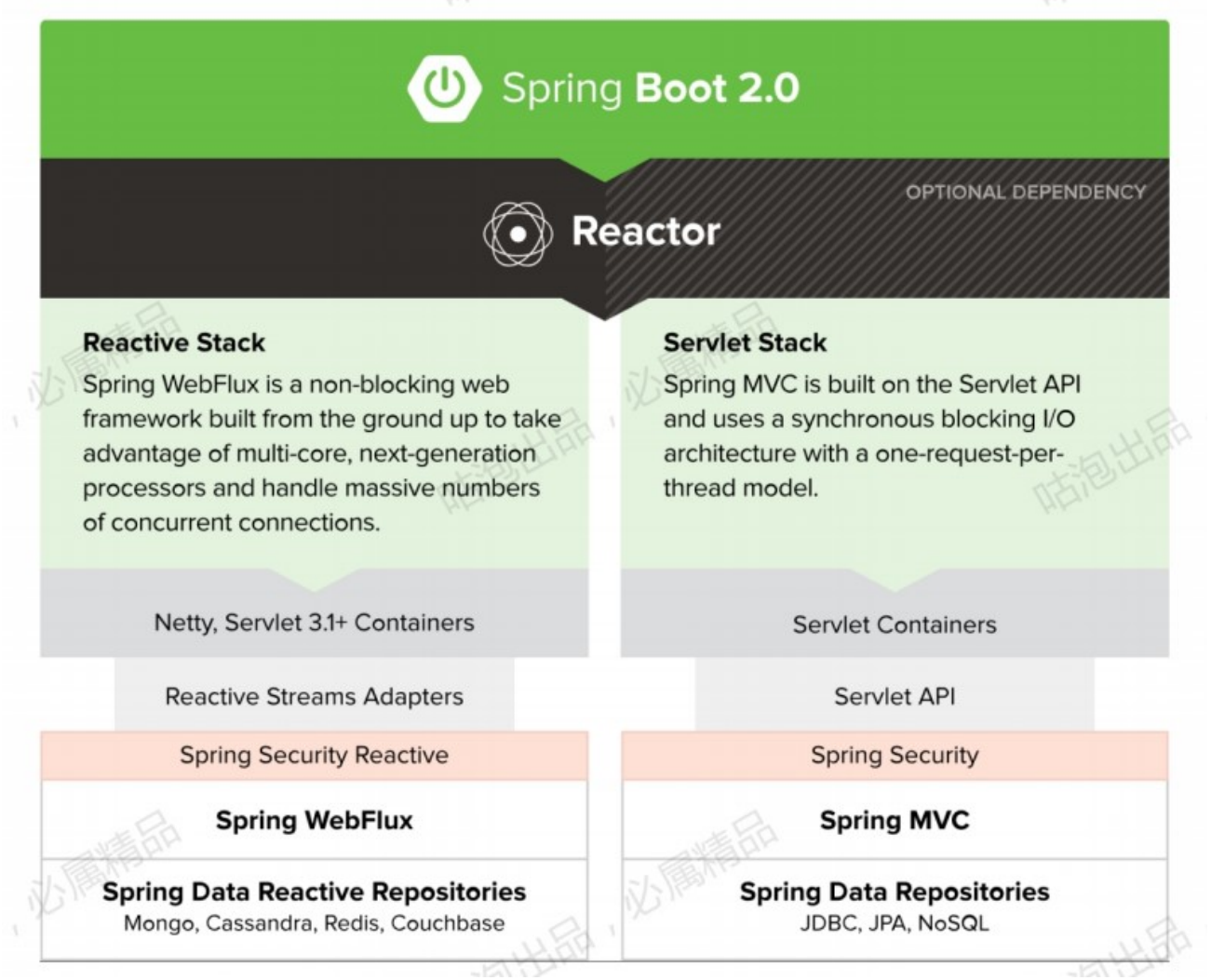
即spring-test模块，主要为测试提供支持的，毕竟在不需要发布（程序）到你的应用服务器或者连接到其他企业设施的情况下能够执行一些集成测试或者其他测试对于任何企业都是非常重要的。

8.7. 集成兼容

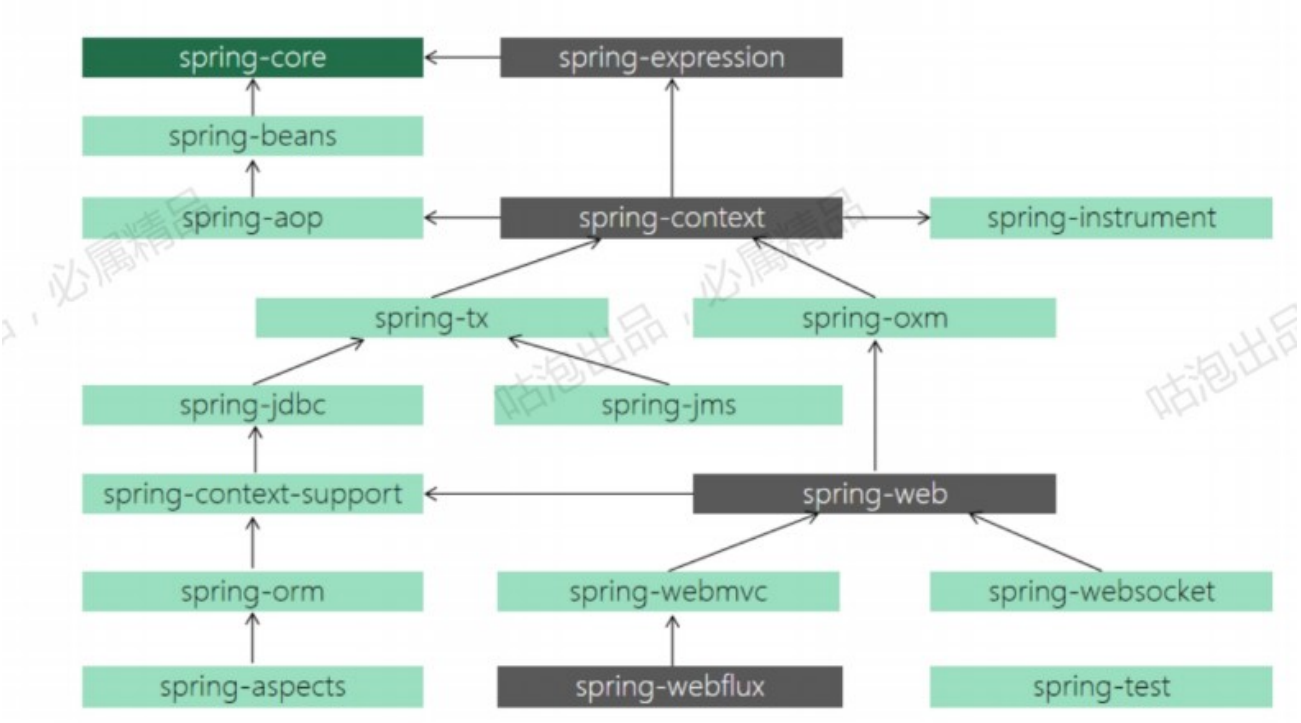
即spring-framework-bom 模块，Bill of Materials.解决Spring的不同模块依赖版本不同问题。

8.8. 各模块之间的依赖关系

Spring 官网对Spring5各模块之间的关系也做了详细说明：



我本人也对Spring5各模块做了一次系统的总结，描述模块之间的依赖关系，希望能对小伙伴们有所帮助。



接下来的课程中，我们将深入了解Spring的核心模块功能。基本学习顺序为：从spring-core入手，其次是spring-beans和spring-aop，随后是spring-context，再其次是spring-tx和spring-orm，最后是spring-web和其他部分。

9. Spring 版本命名规则

常见软件的版本号命名

软件	升级过程	说明
Linux Kernel	0.0.1 1.0.0 2.6.32 3.0.18 ...	若用 X.Y.Z 表示，则偶数 Y 表示稳定版本，奇数 Y 表示开发版本。
Windows	Windows 98 Windows 2000 Windows XP Windows 7 ...	最大的特点是杂乱无章，毫无规律。
SSH Client	0.9.8	
OpenStack	2014.1.3 2015.1.1.dev8	

从上可以看出，不同的软件版本号风格各异，随着系统的规模越大，依赖的软件越多，如果这些软件没有遵循一套规范的命名风格，容易造成Dependency Hell。所以当我们发布版本时，版本号的命名需要遵循某种规则，其中Semantic Versioning 2.0.0定义了一套简单的规则及条件来约束版本号的配置和增长。本文根据 Semantic Versionning 2.0.0和Semantic Versioning 3.0.0选择性的整理出版本号命名规则指南。

9.1. 语义化版本命名通用规则

该规则对版本的迭代顺序命名做了很好的规范，其版本号的格式为X.Y.Z（又称Major.Minor.Patch），递增的规则为：

序号	格式要求	说明
X	非负整数	表示主版本号(Major)，当 API 的兼容性变化时，X 需递增。
Y	非负整数	表示次版本号(Minor)，当增加功能时(不影响 API 的兼容性)，Y 需递增。
Z	非负整数	表示修订号（Patch），当做 Bug 修复时(不影响 API 的兼容性)，Z 需递增。

详细的使用规则如下：
X，Y，Z必须为非负整数，且不得包含前导零，必须按数值递增，如1.9.0->1.10.0->1.11.0

0.Y.Z的版本号表明软件处于初始开发阶段，意味着API可能不稳定；1.0.0表明版本已有稳定的API。

当API的兼容性变化时，X必须递增，Y和Z同时设置为0；当新增功能（不影响API的兼容性）或者API被标记为Deprecated时，Y必须递增，同时Z设置为0；当进行bug fix时，Z必须递增。

先行版本号（Pre-release）意味该版本不稳定，可能存在兼容性问题，其格式为：X.Y.Z.[a-c][正整数]，如1.0.0.a1，1.0.0.b99，1.0.0.c1000。

开发版本号常用于CI-CD，格式为X.Y.Z.dev[正整数]，如1.0.1.dev4。

版本号的排序规则为依次比较主版本号、次版本号和修订号的数值，如1.0.0<1.0.1<1.1.1<2.0.0；对于先行版本号和开发版本号，有：1.0.0.a100<1.0.0，2.1.0.dev3<2.1.0；当存在字母时，以ASCII的排序来比较，如1.0.0.a1<1.0.0.b1。注意：版本一经发布，不得修改其内容，任何修改必须在新版本发布！

9.2. 商业软件中常见的修饰词

描述方式	说明	含义
Snapshot	快照版	尚不不稳定、尚处于开发中的版本
Alpha	内部版	严重缺陷基本完成修正并通过复测，但需要完整的功能测试
Beta	测试版	相对 Alpha 有很大的改进，消除了严重的错误，但还是存在一些缺陷
RC	终测版	Release Candidate（最终测试），即将作为正式版发布。
Demo	演示版	只集成了正式版部分功能升级，无法升级
SP	SP1	是 service pack 的意思表示升级包，相信大家在 windows 中都见过。
Release	稳定版	功能相对稳定，可以对外发行，但有时间限制
Trial	试用版	试用版，仅对部分用户发行
Full Version	完整版	即正式版，已发布。
Unregistered	未注册	有功能或时间限制的版本
Standard	标准版	能满足正常使用的功能的版本
Lite	精简版	只含有正式版的核心功能
Enhance	增强版	正式版，功能优化的版本
Ultimate	旗舰版	在标配版本升级体验感更好的版本
Professiona	专业版	针对更高要求功能，专业性更强的使用群体发行的版本
Free	自由版	自由免费使用的版本
Upgrade	升级版	有功能增强或修复已知 bug
Retail	零售版	单独发售
Cardware	共享版	公用许可证（IOS 签证）
LTS	维护版	该版本需要长期维护

9.3. 软件版本号使用限定

为了方便理解，版本限定的语法简述为为 `[范围描述]<版本号描述>`

范围描述可选必须配合版本描述确定范围，无法独立存在

`<` 小于某一版本号

`<=` 小于等于某一版本号

`>` 大于某一版本号

`>=` 大于等于某一版本号

`=` 等于某一版本号，没有意义和直接写该版本号一样

`~` 基于版本号描述的最新补丁版本

`^` 基于版本号描述的最新兼容版本

`-` 某个范围，他应该出现在两个版本描述中间，实际上语法应为 `<版本描述>-<版本描述>` ，写在此处为了统一

严格来讲对 `~` ， `^` 的表述需要结合具体的包管理工具和版本号规则来确定.但是对于一般使用记住如下原则：

`^` 是确保版本兼容性时，默认对次版本号的限定约束

`~` 是确保版本兼容性时，默认对补丁号的约束

9.4. Spring版本命名规则

描述方式	说明	含义
Snapshot	快照版	尚不不稳定、尚处于开发中的版本
Release	稳定版	功能相对稳定，可以对外发行，但有时间限制
GA	正式版	代表广泛可用的稳定版(General Availability)
M	里程碑版	(M 是 Milestone 的意思）具有一些全新的功能或是具有里程碑意义的版本。
RC	终测版	Release Candidate（最终测试），即将作为正式版发布。