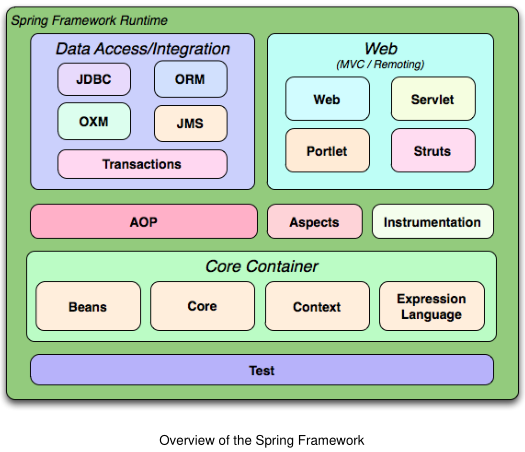
#### Spring 5的框架结构



**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：**

**1.1 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-beans 和 spring-core 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，她是 BeanFactory 的超类，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-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。

### 1.2 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 JAVA SE 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。

### 1.3 数据访问及集成：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。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-jms 模块（Java Messaging Service）能够发送和接受信息，自 Spring Framework 4.1

以后，他还提供了对 spring-messaging 模块的支撑。

  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。

### 1.4 Web：由 spring-web、spring-webmvc、spring-websocket 和 spring-webflux 4 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。（资料缺乏，这是个人理解）

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。

### 1.5 报文发送：即 spring-messaging 模块。

spring-messaging 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。

### 1.6 Test：即 spring-test 模块。

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。

## Spring IoC 和 DI 简介

#### IoC：Inverse of Control（控制反转）

* 读作**“反转控制”**，更好理解，不是什么技术，而是一种**设计思想**，就是**将原本在程序中手动创建对象的控制权，交由Spring框架来管理。**
* **正控：**若要使用某个对象，需要**自己去负责对象的创建**
* **反控：**若要使用某个对象，只需要**从 Spring 容器中获取需要使用的对象，不关心对象的创建过程**，也就是把**创建对象的控制权反转给了Spring框架**

**Spring IoC 阐述**

这就是一种控制反转的理念，上述的例子已经很好的说明了问题，我们再来描述一下控制反转的概念：**控制反转是一种通过描述（在 Java 中可以是 XML 或者注解）并通过第三方（Spring）去产生或获取特定对象的方式。**

* **好处：**  
  降低对象之间的耦合  
  我们不需要理解一个类的具体实现，只需要知道它有什么用就好了（直接向 IoC 容器拿）

主动创建的模式中，责任归于开发者，而在被动的模式下，责任归于 IoC 容器，**基于这样的被动形式，我们就说对象被控制反转了。（也可以说是反转了控制）**

#### DI：Dependency Injection（依赖注入）

* 指 Spring 创建对象的过程中，**将对象依赖属性（简单值，集合，对象）通过配置设值给该对象**

**Spring IoC 容器的设计**

Spring IoC 容器的设计主要是基于以下两个接口：

* **BeanFactory**
* **ApplicationContext**

其中 ApplicationContext 是 BeanFactory 的子接口之一，换句话说：**BeanFactory 是 Spring IoC 容器所定义的最底层接口，**而ApplicationContext 是其最高级接口之一，并对 BeanFactory 功能做了许多的扩展，所以在**绝大部分的工作场景下**，都会使用 ApplicationContext 作为 Spring IoC 容器。

由于这个接口的重要性，所以有必要在这里作一下简短的说明：

* 【getBean】 对应了多个方法来获取配置给 Spring IoC 容器的 Bean。  
  ① 按照类型拿 bean：  
  bean = (Bean) factory.getBean(Bean.class);  
  **注意：**要求在 Spring 中只配置了一个这种类型的实例，否则报错。（如果有多个那 Spring 就懵了，不知道该获取哪一个）  
  ② 按照 bean 的名字拿 bean:  
  bean = (Bean) factory.getBean("beanName");  
  **注意：**这种方法不太安全，IDE 不会检查其安全性（关联性）  
  ③ 按照名字和类型拿 bean：**（推荐）**  
  bean = (Bean) factory.getBean("beanName", Bean.class);
* 【isSingleton】 用于判断是否单例，如果判断为真，其意思是该 Bean 在容器中是作为一个唯一单例存在的。而【isPrototype】则相反，如果判断为真，意思是当你从容器中获取 Bean，容器就为你生成一个新的实例。  
  **注意：**在默认情况下，【isSingleton】为 ture，而【isPrototype】为 false

## 装配 Bean 的概述

在 Spring 中提供了 3 种方法进行配置：

* 在 XML 文件中显式配置
* 在 Java 的接口和类中实现配置
* 隐式 Bean 的发现机制和自动装配原则

在现实的工作中，这 3 种方式都会被用到，并且在学习和工作之中常常混合使用，所以这里给出一些关于这 3 种优先级的建议：

1.**最优先：通过隐式 Bean 的发现机制和自动装配的原则。**  
基于约定由于配置的原则，这种方式应该是最优先的

* **好处：**减少程序开发者的决定权，简单又不失灵活。

2.**其次：Java 接口和类中配置实现配置**  
在没有办法使用自动装配原则的情况下应该优先考虑此类方法

* **好处：**避免 XML 配置的泛滥，也更为容易。
* **典型场景：**一个父类有多个子类，比如学生类有两个子类，一个男学生类和女学生类，通过 IoC 容器初始化一个学生类，容器将无法知道使用哪个子类去初始化，这个时候可以使用 Java 的注解配置去指定。

3.**最后：XML 方式配置**  
在上述方法都无法使用的情况下，那么也只能选择 XML 配置的方式。

* **好处：**简单易懂（当然，特别是对于初学者）
* **典型场景：**当使用第三方类的时候，有些类并不是我们开发的，我们无法修改里面的代码，这个时候就通过 XML 的方式配置使用了。

## 通过 XML 配置装配 Bean

id 属性是 Spring 能找到当前 Bean 的一个依赖的编号，遵守 XML 语法的 ID 唯一性约束。必须以字母开头，可以使用字母、数字、连字符、下划线、句号、冒号，不能以 / 开头。

不过 id 属性不是一个必需的属性，name 属性也可以定义 bean 元素的名称，能以逗号或空格隔开起多个别名，并且可以使用很多的特殊字符。

class 属性显然就是一个类的全限定名。

property 元素是定义类的属性，其中的 name 属性定义的是属性的名称，而 value 是它的值。

## 通过注解装配 Bean

* **优势：**  
  1.可以减少 XML 的配置，当配置项多的时候，臃肿难以维护  
  2.功能更加强大，既能实现 XML 的功能，也提供了自动装配的功能，采用了自动装配后，程序猿所需要做的决断就少了，更加有利于对程序的开发，这就是“约定由于配置”的开发原则

在 Spring 中，它提供了两种方式来让 Spring IoC 容器发现 bean：

* **组件扫描：**通过定义资源的方式，让 Spring IoC 容器扫描对应的包，从而把 bean 装配进来。
* **自动装配：**通过注解定义，使得一些依赖关系可以通过注解完成。

**@Component注解：**  
表示 Spring IoC 会把这个类扫描成一个 bean 实例

**@Value注解：**

表示值的注入，跟在 XML 中写 value 属性是一样的。

1.@Repository，这个注解和@Component、@Controller和我们最常见的@Service注解是一个作用，都可以将一个类声明为一个Spring的Bean。它们的区别到不在于具体的语义上，更多的是在于注解的定位上。之前说过，企业级应用注重分层开发的概念，因此，对这四个相似的注解应当有以下的理解：

* @Repository注解，对应的是持久层即Dao层，其作用是直接和数据库交互，通常来说一个方法对应一条具体的Sql语句
* @Service注解，对应的是服务层即Service层，其作用是对单条/多条Sql语句进行组合处理，当然如果简单的话就直接调用Dao层的某个方法了
* @Controller注解，对应的是控制层即MVC设计模式中的控制层，其作用是接收用户请求，根据请求调用不同的Service取数据，并根据需求对数据进行组合、包装返回给前端
* @Component注解，这个更多对应的是一个组件的概念，如果一个Bean不知道属于拿个层，可以使用@Component注解标注

### 自动装配——@Autowired

所谓自动装配技术是一种**由 Spring 自己发现对应的 Bean，自动完成装配工作的方式，**这个时候 **Spring 会根据类型去寻找定义的 Bean 然后将其注入**

* **问题：** IoC 容器可能会寻找失败，此时会抛出异常（默认情况下，Spring IoC 容器会认为一定要找到对应的 Bean 来注入到这个字段，但有些时候并不是一定需要，比如日志）
* **解决：** 通过配置项 required 来改变，比如 @Autowired(required = false)

**@Qualifier 注解：**

如果我们现在有两个 学生 类型的对象，Spring IoC 就会不知所措，不知道究竟该引入哪一个 Bean

Spring 在寻找依赖注入的时候是按照类型注入引起的。除了按类型查找 Bean，Spring IoC 容器最底层的接口 BeanFactory 还提供了按名字查找的方法。

#### Bean 的作用域

**在默认的情况下，Spring IoC 容器只会对一个 Bean 创建一个实例**，但有时候，我们希望能够通过 Spring IoC 容器获取多个实例，我们可以通过 @Scope 注解或者 <bean> 元素中的 scope 属性来设置

Spring 提供了 5 种作用域，它会根据情况来决定是否生成新的对象：

| **作用域类别** | **描述** |
| --- | --- |
| singleton(单例) | 默认值，单例对象。被标识为单例的对象在Spring容器中只会存在一份实例 |
| prototype(多例) | 多例的。被标识为多例的对象每次在获得时才会创建，并且每次创建都是新的对象。单例对象是在容器启动时就创建了。 |
| request(请求) | WEB 项目中，Spring 创建一个 Bean 的对象，将对象存入到 request 域中。对象与request生命周期一致。 --基本不用 |
| session(会话) | WEB 项目中，Spring 创建一个 Bean 的对象，将对象存入到 session 域中。对象与session生命周期一致。 --基本不用 |
| globalSession(全局会话) | WEB 项目中，应用在 Porlet 环境。如果没有 Porlet 环境那么 globalSession 相当于 session。 --基本不用 |

在未来的开发中，绝大多数scope的取值都是使用默认值