Spring 概述

Spring 是个 java 企业级应用的开源开发框架。Spring 主要用来开发 Java 应用，但是有些扩展是针对构建 J2EE 平台的 web 应用。Spring 框架目标是简化 Java 企业级应用开发，并通过 POJO 为基础的编程模型促进良好的编程习惯。

轻量：Spring 是轻量的，基本的版本大约 2MB
控制反转：Spring 通过控制反转实现了松散耦合，对象们给出它们的依赖，而不是创建或查找依赖的对象们
面向切面的编程(AOP)：Spring 支持面向切面的编程，并且把应用业务逻辑和系统服务分开
容器：Spring 包含并管理应用中对象的生命周期和配置
MVC 框架：Spring 的 WEB 框架是个精心设计的框架，是 Web 框架的一个很好的替代品
事务管理：Spring 提供一个持续的事务管理接口，可以扩展到上至本地事务下至全局事务（JTA）
异常处理：Spring 提供方便的 API 把具体技术相关的异常（比如由 JDBC，Hibernate or JDO 抛出的）转化为一致的 unchecked 异常

以下是 Spring 框架的基本模块：
Core module
Bean module
Context module
Expression Language module
JDBC module
ORM module
OXM module
Java Messaging Service(JMS) module
Transaction module
Web module
Web-Servlet module
Web-Struts module
Web-Portlet module

这是基本的 Spring 模块，提供 spring 框架的基础功能，BeanFactory 是 任何以 spring 为基础的应用的核心。Spring 框架建立在此模块之上，它使 Spring 成为一个容器。

Bean 工厂是工厂模式的一个实现，提供了控制反转功能，用来把应用的配置和依赖从正真的应用代码中分离。最常用的 BeanFactory 实现是 XmlBeanFactory 类。

最常用的就是 org.springframework.beans.factory.xml.XmlBeanFactory ，它根据 XML 文件中的定义加载 beans。该容器从 XML 文件读取配置元数据并用它去创建一个完全配置的系统或应用。

AOP 模块用于发给我们的 Spring 应用做面向切面的开发，很多支持由 AOP 联盟提供，这样就确保了 Spring 和其他 AOP 框架的共通性。这个模块将元数据编程引入 Spring。

通过使用 JDBC 抽象和 DAO 模块，保证数据库代码的简洁，并能避免数据库资源错误关闭导致的问题，它在各种不同的数据库的错误信息之上，提供了一个统一的异常访问层。它还利用 Spring 的 AOP 模块给 Spring 应用中的对象提供事务管理服务。

Spring 通过提供 ORM 模块，支持我们在直接 JDBC 之上使用一个对象/关系映射映射(ORM)工具，Spring 支持集成主流的 ORM 框架，如 Hiberate,JDO 和 iBATIS SQL Maps。Spring 的事务管理同样支持以上所有 ORM 框架及 JDBC。

Spring 的 WEB 模块是构建在 application context 模块基础之上，提供一个适合 web 应用的上下文。这个模块也包括支持多种面向 web 的任务，如透明地处理多个文件上传请求和程序级请求参数的绑定到你的业务对象。它也有对 Jakarta Struts 的支持。

Spring 配置文件是个 XML 文件，这个文件包含了类信息，描述了如何配置它们，以及如何相互调用。

Spring IOC 负责创建对象，管理对象（通过依赖注入（DI），装配对象，配置对象，并且管理这些对象的整个生命周期。

IOC 或 依赖注入把应用的代码量降到最低。它使应用容易测试，单元测试不再需要单例和 JNDI 查找机制。最小的代价和最小的侵入性使松散耦合得以实现。IOC 容器支持加载服务时的饿汉式初始化和懒加载。

- FileSystemXmlApplicationContext ：此容器从一个 XML 文件中加载 beans 的定义，XML Bean 配置文件的全路径名必须提供给它的构造函数。
- ClassPathXmlApplicationContext：此容器也从一个 XML 文件中加载 beans 的定义，这里，你需要正确设置 classpath 因为这个容器将在 classpath 里找 bean 配置。
- WebXmlApplicationContext：此容器加载一个 XML 文件，此文件定义了一个 WEB 应用的所有 bean。

Application contexts 提供一种方法处理文本消息，一个通常的做法是加载文件资源（比如镜像），它们可以向注册为监听器的 bean 发布事件。另外，在容器或容器内的对象上执行的那些不得不由 bean 工厂以程序化方式处理的操作，可以在 Application contexts 中以声明的方式处理。Application contexts 实现了 MessageSource 接口，该接口的实现以可插拔的方式提供获取本地化消息的方法。

- 一个定义了一些功能的接口
- 这实现包括属性，它的 Setter ， getter 方法和函数等
- Spring AOP
- Spring 的 XML 配置文件
- 使用以上功能的客户端程序

依赖注入

依赖注入，是 IOC 的一个方面，是个通常的概念，它有多种解释。这概念是说你不用创建对象，而只需要描述它如何被创建。你不在代码里直接组装你的组件和服务，但是要在配置文件里描述哪些组件需要哪些服务，之后一个容器（IOC 容器）负责把他们组装起来。

构造器依赖注入：构造器依赖注入通过容器触发一个类的构造器来实现的，该类有一系列参数，每个参数代表一个对其他类的依赖。
Setter 方法注入：Setter 方法注入是容器通过调用无参构造器或无参 static 工厂 方法实例化 bean 之后，调用该 bean 的 setter 方法，即实现了基于 setter 的依赖注入。

你两种依赖方式都可以使用，构造器注入和 Setter 方法注入。最好的解决方案是用构造器参数实现强制依赖，setter 方法实现可选依赖。

Spring Beans

Spring beans 是那些形成 Spring 应用的主干的 java 对象。它们被 Spring IOC 容器初始化，装配，和管理。这些 beans 通过容器中配置的元数据创建。比如，以 XML 文件中<bean/> 的形式定义。
Spring 框架定义的 beans 都是单件 beans。在 bean tag 中有个属性”singleton”，如果它被 为 TRUE，bean 就是单件，否则就是一个 prototype bean。默认是 TRUE，所以所有在 Spring 框架中的 beans 缺省都是单件。点击这里一图 Spring Bean 的生命周期。

一个 Spring Bean 的定义包含容器必知的所有配置元数据，包括如何创建一个 bean，它的生命周期详情及它的依赖。

这里有三种重要的方法给 Spring 容器提供配置元数据。

- XML 配置文件。
- 基于注解的配置。
- 基于 java 的配置。

当定义一个<bean> 在 Spring 里，我们还能给这个 bean 声明一个作用域。它可以通过 bean 定义中的 scope 属性来定义。如，当 Spring 要在需要的时候每次生产一个新的 bean 实例，bean 的 scope 属性被指定为 prototype。另一方面，一个 bean 每次使用的时候必须返回同一个实例，这个 bean 的 scope 属性 必须设为 singleton。

Spring 框架支持以下五种 bean 的作用域：

- **singleton**：bean 在每个 Spring ioc 容器中只有一个实例。
- **prototype**：一个 bean 的定义可以有多个实例。
- **request**：每次 http 请求都会创建一个 bean，该作用域仅在基于 web 的 Spring ApplicationContext 情形下有效。
- **session**：在一个 HTTP Session 中，一个 bean 定义对应一个实例。该作用域仅在基于 web 的 Spring ApplicationContext 情形下有效。
- **global-session**：在一个全局的 HTTP Session 中，一个 bean 定义对应一个实例。该作用域仅在基于 web 的 Spring ApplicationContext 情形下有效。
- **缺省的 Spring bean 的作用域是 Singleton。**

不，Spring 框架中的单例 bean 不是线程安全的。

Spring 容器 从 XML 文件中读取 bean 的定义，并实例化 bean。
Spring 根据 bean 的定义填充所有的属性。
如果 bean 实现了 BeanNameAware 接口，Spring 传递 bean 的 ID 到 setBeanName 方法。
如果 Bean 实现了 BeanFactoryAware 接口， Spring 传递 beanfactory 给 setBeanFactory 方法。
如果有任何与 bean 相关联的 BeanPostProcessors，Spring 会在 postProcesserBeforeInitialization()方法内调用它们。
如果 bean 实现 IntializingBean 了，调用它的 afterPropertySet 方法，如果 bean 声明了初始化方法，调用此初始化方法。
如果有 BeanPostProcessors 和 bean 关联，这些 bean 的 postProcessAfterInitialization() 方法将被调用。
如果 bean 实现了 DisposableBean，它将调用 destroy()方法。
点击这里一图 Spring Bean 的生命周期。

有两个重要的 bean 生命周期方法，第一个是 setup ， 它是在容器加载 bean 的时候被调用。第二个方法是 teardown 它是在容器卸载类的时候被调用。
The bean 标签有两个重要的属性（init-method 和 destroy-method）。用它们你可以自己定制初始化和注销方法。它们也有相应的注解（@PostConstruct 和 @PreDestroy）。

当一个 bean 仅被用作另一个 bean 的属性时，它能被声明为一个内部 bean，为了定义 inner bean，在 Spring 的 基于 XML 的 配置元数据中，可以在 <property/> 或 <constructor-arg/> 元素内使用<bean/> 元素，内部 bean 通常是匿名的，它们的 Scope 一般是 prototype。

Spring 提供以下几种集合的配置元素：
<list>类型用于注入一列值，允许有相同的值。
<set> 类型用于注入一组值，不允许有相同的值。
<map> 类型用于注入一组键值对，键和值都可以为任意类型。
<props>类型用于注入一组键值对，键和值都只能为 String 类型。

装配，或 bean 装配是指在 Spring 容器中把 bean 组装到一起，前提是容器需要知道 bean 的依赖关系，如何通过依赖注入来把它们装配到一起。

Spring 容器能够自动装配相互合作的 bean，这意味着容器不需要<constructor-arg>和<property>配置，能通过 Bean 工厂自动处理 bean 之间的协作。

有五种自动装配的方式，可以用来指导 Spring 容器用自动装配方式来进行依赖注入
no：默认的方式是不进行自动装配，通过显式设置 ref 属性来进行装配。
byName：通过参数名 自动装配，Spring 容器在配置文件中发现 bean 的 autowire 属性被设置成 byname，之后容器试图匹配、装配和该 bean 的属性具有相同名字的 bean。
byType：通过参数类型自动装配，Spring 容器在配置文件中发现 bean 的 autowire 属性被设置成 byType，之后容器试图匹配、装配和该 bean 的属性具有相同类型的 bean。如果有多个 bean 符合条件，则抛出错误。
constructor：这个方式类似于 byType， 但是要提供给构造器参数，如果没有确定的带参数的构造器参数类型，将会抛出异常。
autodetect：首先尝试使用 constructor 来自动装配，如果无法工作，则使用 byType 方式。

自动装配的局限性是：
重写：你仍需用 <constructor-arg>和 <property> 配置来定义依赖，意味着总要重写自动装配。
基本数据类型：你不能自动装配简单的属性，如基本数据类型，String 字符串，和类。
模糊特性：自动装配不如显式装配精确，如果有可能，建议使用显式装配。

可以。

Spring 注解

基于 Java 的配置，允许你在少量的 Java 注解的帮助下，进行你的大部分 Spring 配置而非通过 XML 文件。
以@Configuration 注解为例，它用来标记类可以当做一个 bean 的定义，被 Spring IOC 容器使用。另一个例子是@Bean 注解，它表示此方法将要返回一个对象，作为一个 bean 注册进 Spring 应用上下文。点击这里学习 JAVA 几大元注解。

相对于 XML 文件，注解型的配置依赖于通过字节码元数据装配组件，而非尖括号的声明。
开发者通过在相应的类，方法或属性上使用注解的方式，直接组件类中进行配置，而不是使用 xml 表述 bean 的装配关系。

注解装配在默认情况下是不开启的，为了使用注解装配，我们必须在 Spring 配置文件中配置 <context:annotation-config/>元素。

这个注解表明 bean 的属性必须在配置的时候设置，通过一个 bean 定义的显式的属性值或通过自动装配，若@Required 注解的 bean 属性未被设置，容器将抛出 BeanInitializationException。

@Autowired 注解提供了更细粒度的控制，包括在何处以及如何完成自动装配。它的用法和@Required 一样，修饰 setter 方法、构造器、属性或者具有任意名称和/或多个参数的 PN 方法。

当有多个相同类型的 bean 却只有一个需要自动装配时，将@Qualifier 注解和@Autowire 注解结合使用以消除这种混淆，指定需要装配的确切的 bean。点击这里学习更多常用注解。

Spring 数据访问

使用 SpringJDBC 框架，资源管理和错误处理的代价都会被减轻。所以开发者只需写 statements 和 queries 从数据存取数据，JDBC 也可以在 Spring 框架提供的模板类的帮助下更有效地被使用，这个模板叫 JdbcTemplate

JdbcTemplate 类提供了很多便利的方法解决诸如把数据库数据转变成基本数据类型或对象，执行写好的或可调用的数据库操作语句，提供自定义的数据错误处理。

Spring 对数据访问对象（DAO）的支持旨在简化它和数据访问技术如 JDBC，Hibernate or JDO 结合使用。这使我们可以方便切换持久层。编码时也不用担心会捕获每种技术特有的异常。

在 Spring 中有两种方式访问 Hibernate：
控制反转 Hibernate Template 和 Callback
继承 HibernateDAOSupport 提供一个 AOP 拦截器

Spring 支持以下 ORM：

Hibernate
iBatis
JPA (Jav

关注点是应用中一个模块的行为，一个关注点可能会被定义成一个我们想实现的一个功能。
横切关注点是一个关注点，此关注点是整个应用都会使用的功能，并影响整个应用，比如日志，安全和数据传输，几乎应用的每个模块都需要的功能。因此这些都属于横切关注点。

连接点代表一个应用程序的某个位置，在这个位置我们可以插入一个 AOP 切面，它实际上是个应用程序执行 Spring AOP 的位置。

通知是个在方法执行前或执行后要做的动作，实际上是程序执行时要通过 SpringAOP 框架触发的代码段。
Spring 切面可以应用五种类型的通知：
before：前置通知，在一个方法执行前被调用
after：在方法执行之后调用的通知，无论方法执行是否成功
after-returning：仅当方法成功完成后执行的通知
after-throwing：在方法抛出异常退出时执行的通知
around：在方法执行之前和之后调用的通知

切入点是一个或一组连接点，通知将在这些位置执行。可以通过表达式或匹配的方式指明切入点。

引入允许我们在已存在的类中增加新的方法和属性。

被一个或者多个切面所通知的对象。它通常是一个代理对象。也指被通知（advised）对象。

代理是通知目标对象后创建的对象。从客户端的角度看，代理对象和目标对象是一样的。

BeanNameAutoProxyCreator
DefaultAdvisorAutoProxyCreator
Metadata autoproxying

织入是将切面和到其他应用类型或对象连接或创建一个被通知对象的过程。

织入可以在编译时，加载时，或运行时完成。

在这种情况下，切面由常规类以及基于 XML 的配置实现。

在这种情况下(基于@AspectJ 的实现)，涉及到的切面声明的风格与带有 java5 标注的普通 java 类一致。

Spring 的 MVC

Spring 配备构建 Web 应用的全功能 MVC 框架。Spring 可以很便捷地和其他 MVC 框架集成，如 Struts，Spring 的 MVC 框架用控制反转把业务对象和控制逻辑清晰地隔离。它也允许以声明的方式把请求参数和业务对象绑定。

Spring 的 MVC 框架是围绕 DispatcherServlet 来设计的，它用来处理所有的 HTTP 请求和响应。

WebApplicationContext 继承了 ApplicationContext 并增加了一些 WEB 应用必备的特有功能，它不同于一般的 ApplicationContext ，因为它能处理主题，并找到被关联的 servlet。

控制器提供一个访问应用程序的行为，此行为通常通过服务接口实现。控制器解析用户输入并将其转换为一个由视图呈现给用户的模型。Spring 用一个非常抽象的方式实现了一个控制层，允许用户创建多种用途的控制器。

该注解表明该类扮演控制器的角色，Spring 不需要你继承任何其他控制器基类或引用 Servlet API。

该注解是用来映射一个 URL 到一个类或一个特定的方处理法上。