**# 一般问题**

**1.1. 不同版本的 Spring Framework 有哪些主要功能？**

Version                    Feature

| Spring 2.5 | 发布于 2007 年。这是第一个支持注解的版本。 |
| --- | --- |
| Spring 3.0 | 发布于 2009 年。它完全利用了 Java5 中的改进，并为 JEE6 提供了支持。 |
| Spring 4.0 | 发布于 2013 年。这是第一个完全支持 JAVA8 的版本。 |

**1.2. 什么是 Spring Framework？**

Spring 是一个开源应用框架，旨在降低应用程序开发的复杂度。  
它是轻量级、松散耦合的。  
它具有分层体系结构，允许用户选择组件，同时还为 J2EE 应用程序开发提供了一个有凝聚力的框架。  
它可以集成其他框架，如 Structs、Hibernate、EJB 等，所以又称为框架的框架。

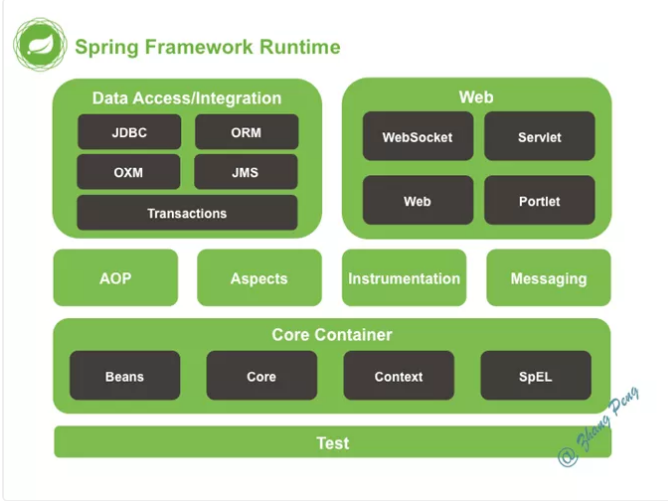
**1.3. 列举 Spring Framework 的优点。**

由于 Spring Frameworks 的分层架构，用户可以自由选择自己需要的组件。  
Spring Framework 支持 POJO(Plain Old Java Object) 编程，从而具备持续集成和可测试性。  
由于依赖注入和控制反转，JDBC 得以简化。  
它是开源免费的。

**1.4. Spring Framework 有哪些不同的功能？**

轻量级 - Spring 在代码量和透明度方面都很轻便。  
  
IOC - 控制反转  
AOP - 面向切面编程可以将应用业务逻辑和系统服务分离，以实现高内聚。  
容器 - Spring 负责创建和管理对象（Bean）的生命周期和配置。  
MVC - 对 web 应用提供了高度可配置性，其他框架的集成也十分方便。  
事务管理 - 提供了用于事务管理的通用抽象层。Spring 的事务支持也可用于容器较少的环境。  
JDBC 异常 - Spring 的 JDBC 抽象层提供了一个异常层次结构，简化了错误处理策略。

**1.5. Spring Framework 中有多少个模块，它们分别是什么？**



Spring 核心容器 – 该层基本上是 Spring Framework 的核心。它包含以下模块：Spring CoreSpring BeanSpEL (Spring Expression Language)Spring Context

数据访问/集成 – 该层提供与数据库交互的支持。它包含以下模块：JDBC (Java DataBase Connectivity)ORM (Object Relational Mapping)OXM (Object XML Mappers)JMS (Java Messaging Service)Transaction

Web – 该层提供了创建 Web 应用程序的支持。它包含以下模块：WebWeb – ServletWeb – SocketWeb – Portlet

**AOP** – 该层支持面向切面编程**Instrumentation**– 该层为类检测和类加载器实现提供支持。**Test** – 该层为使用 JUnit 和 TestNG 进行测试提供支持。

几个杂项模块:  
Messaging – 该模块为 STOMP 提供支持。它还支持注解编程模型，该模型用于从 WebSocket 客户端路由和处理 STOMP 消息  
Aspects – 该模块为与 AspectJ 的集成提供支持。

**1.6. 什么是 Spring 配置文件？**

Spring 配置文件是 XML 文件。该文件主要包含类信息。它描述了这些类是如何配置以及相互引入的。但是，XML 配置文件冗长且更加干净。如果没有正确规划和编写，那么在大项目中管理变得非常困难。

**1.7. Spring 应用程序有哪些不同组件？**

Spring 应用一般有以下组件：  
**接口**- 定义功能。**Bean 类** - 它包含属性，setter 和 getter 方法，函数等。**Spring 面向切面编程（AOP）** - 提供面向切面编程的功能。**Bean 配置文件** - 包含类的信息以及如何配置它们。**用户程序** - 它使用接口。

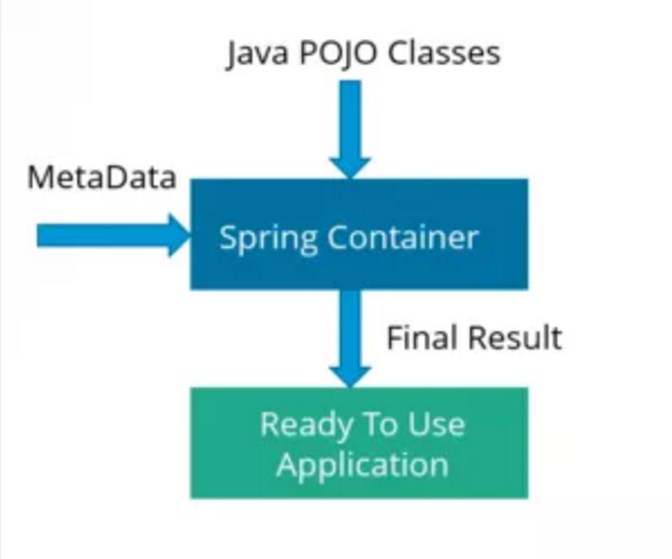
**1.8. 使用 Spring 有哪些方式？**

使用 Spring 有以下方式：  
作为一个成熟的 Spring Web 应用程序。作为第三方 Web 框架，使用 Spring Frameworks 中间层。用于远程使用。作为企业级 Java Bean，它可以包装现有的 POJO（Plain Old Java Objects）。

**# 依赖注入（Ioc）**

**1. 什么是 Spring IOC 容器？**

Spring 框架的核心是 Spring 容器。容器创建对象，将它们装配在一起，配置它们并管理它们的完整生命周期。Spring 容器使用依赖注入来管理组成应用程序的组件。  
容器通过读取提供的配置元数据来接收对象进行实例化，配置和组装的指令。该元数据可以通过 XML，Java 注解或 Java 代码提供。



**2. 什么是依赖注入？**

在依赖注入中，您不必创建对象，但必须描述如何创建它们。您不是直接在代码中将组件和服务连接在一起，而是描述配置文件中哪些组件需要哪些服务。由 IoC 容器将它们装配在一起。

**3. 可以通过多少种方式完成依赖注入？**

通常，依赖注入可以通过三种方式完成，即：  
\*构造函数注入\*setter 注入\*接口注入  
在 Spring Framework 中，仅使用构造函数和 setter 注入。

**4. 区分构造函数注入和 setter 注入。**

| 构造函数注入 | setter 注入 |
| --- | --- |
| 没有部分注入 | 有部分注入 |
| 不会覆盖 setter 属性 | 会覆盖 setter 属性 |
| 任意修改都会创建一个新实例 | 任意修改不会创建一个新实例 |
| 适用于设置很多属性 | 适用于设置少量属性 |

**5. spring 中有多少种 IOC 容器？**

\* BeanFactory - BeanFactory 就像一个包含 bean 集合的工厂类。它会在客户端要求时实例化 bean。\* ApplicationContext - ApplicationContext 接口扩展了 BeanFactory 接口。它在 BeanFactory 基础上提供了一些额外的功能。

**6. 区分 BeanFactory 和 ApplicationContext。**

| BeanFactory | ApplicationContext |
| --- | --- |
| 它使用懒加载 | 它使用即时加载 |
| 它使用语法显式提供资源对象 | 它自己创建和管理资源对象 |
| 不支持国际化 | 支持国际化 |
| 不支持基于依赖的注解 | 支持基于依赖的注解 |

**7. 列举 IoC 的一些好处。**

IoC 的一些好处是：  
\*它将最小化应用程序中的代码量。\*它将使您的应用程序易于测试，因为它不需要单元测试用例中的任何单例或 JNDI 查找机制。\*它以最小的影响和最少的侵入机制促进松耦合。\*它支持即时的实例化和延迟加载服务。

**8. Spring IoC 的实现机制。**

Spring 中的 IoC 的实现原理就是工厂模式加反射机制。示例：

**interface** **Fruit** {  
     **public** **abstract** **void** **eat**();  
}  
**class** **Apple** **implements** **Fruit** {  
    **public** **void** **eat**(){  
        System.out.println("Apple");  
    }  
}  
**class** **Orange** **implements** **Fruit** {  
    **public** **void** **eat**(){  
        System.out.println("Orange");  
    }  
}  
**class** **Factory** {  
    **public** **static** Fruit **getInstance**(String ClassName) {  
        Fruit f=**null**;  
        **try** {  
            f=(Fruit)Class.forName(ClassName).newInstance();  
        } **catch** (Exception e) {  
            e.printStackTrace();  
        }  
        **return** f;  
    }  
}  
**class** **Client** {  
    **public** **static** **void** **main**(String[] a) {  
        Fruit f=Factory.getInstance("io.github.dunwu.spring.Apple");  
        **if**(f!=**null**){  
            f.eat();  
        }  
    }  
}

**#  Beans**

**1. 什么是 spring bean？**

\* 它们是构成用户应用程序主干的对象。\* Bean 由 Spring IoC 容器管理。\* 它们由 Spring IoC 容器实例化，配置，装配和管理。\* Bean 是基于用户提供给容器的配置元数据创建。

**2. spring 提供了哪些配置方式？**

\* 基于 xml 配置  
bean 所需的依赖项和服务在 XML 格式的配置文件中指定。这些配置文件通常包含许多 bean 定义和特定于应用程序的配置选项。它们通常以 bean 标签开头。例如：

<bean id="studentbean" **class**="org.edureka.firstSpring.StudentBean">  
 <property name="name" value="Edureka"></property>  
</bean>

\* 基于注解配置

您可以通过在相关的类，方法或字段声明上使用注解，将 bean 配置为组件类本身，而不是使用 XML 来描述 bean 装配。默认情况下，Spring 容器中未打开注解装配。因此，您需要在使用它之前在 Spring 配置文件中启用它。例如：

<beans>  
<context:annotation-config/>  
*<!-- bean definitions go here -->*  
</beans>

\* 基于 Java API 配置

Spring 的 Java 配置是通过使用 @Bean 和 @Configuration 来实现。

1. @Bean 注解扮演与 元素相同的角色。
2. @Configuration 类允许通过简单地调用同一个类中的其他 @Bean 方法来定义 bean 间依赖关系。

例如：

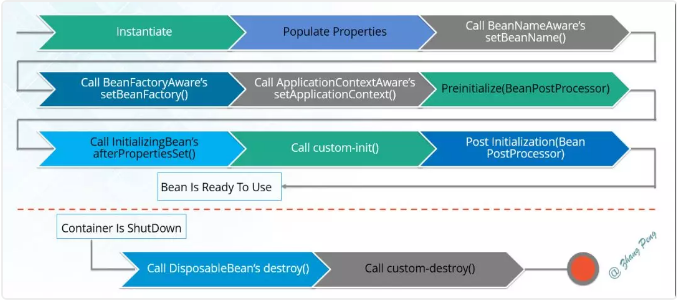
**@Configuration**  
**public** **class** **StudentConfig** {  
    **@Bean**  
    **public** StudentBean **myStudent**() {  
        **return** **new** StudentBean();  
    }  
}

**3. spring 支持集中 bean scope？**

Spring bean 支持 5 种 scope：  
\* Singleton - 每个 Spring IoC 容器仅有一个单实例。\* Prototype - 每次请求都会产生一个新的实例。\* Request - 每一次 HTTP 请求都会产生一个新的实例，并且该 bean 仅在当前 HTTP 请求内有效。\* Session - 每一次 HTTP 请求都会产生一个新的 bean，同时该 bean 仅在当前 HTTP session 内有效。\* Global-session - 类似于标准的 HTTP Session 作用域，不过它仅仅在基于 portlet 的 web 应用中才有意义。Portlet 规范定义了全局 Session 的概念，它被所有构成某个 portlet web 应用的各种不同的 portlet 所共享。在 global session 作用域中定义的 bean 被限定于全局 portlet Session 的生命周期范围内。如果你在 web 中使用 global session 作用域来标识 bean，那么 web 会自动当成 session 类型来使用。  
仅当用户使用支持 Web 的 ApplicationContext 时，最后三个才可用。更多spring内容

**4. spring bean 容器的生命周期是什么样的？**

spring bean 容器的生命周期流程如下：  
1.Spring 容器根据配置中的 bean 定义中实例化 bean  
2.Spring 使用依赖注入填充所有属性，如 bean 中所定义的配置。  
3.如果 bean 实现 BeanNameAware 接口，则工厂通过传递 bean 的 ID 来调用 setBeanName()。  
4.如果 bean 实现 BeanFactoryAware 接口，工厂通过传递自身的实例来调用 setBeanFactory()。  
5.如果存在与 bean 关联的任何 BeanPostProcessors，则调用 preProcessBeforeInitialization() 方法。  
6.如果为 bean 指定了 init 方法（ 的 init-method 属性），那么将调用它。  
7.最后，如果存在与 bean 关联的任何 BeanPostProcessors，则将调用 postProcessAfterInitialization() 方法。  
8.如果 bean 实现 DisposableBean 接口，当 spring 容器关闭时，会调用 destory()。  
9.如果为 bean 指定了 destroy 方法（ 的 destroy-method 属性），那么将调用它。



**5. 什么是 spring 的内部 bean？**

只有将 bean 用作另一个 bean 的属性时，才能将 bean 声明为内部 bean。为了定义 bean，Spring 的基于 XML 的配置元数据在 <property> 或 <constructor-arg> 中提供了 <bean> 元素的使用。内部 bean 总是匿名的，它们总是作为原型。  
例如，假设我们有一个 Student 类，其中引用了 Person 类。这里我们将只创建一个 Person 类实例并在 Student 中使用它。  
Student.java

public class Student {  
    private Person person;  
    //Setters and Getters  
}  
public class Person {  
    private String name;  
    private String address;  
    //Setters and Getters  
}  
bean.xml<bean id=“StudentBean" class="com.edureka.Student">  
    <property name="person">  
        *<!--This is inner bean -->*  
        <bean class="com.edureka.Person">  
            <property name="name" value=“Scott"></property>  
            <property name="address" value=“Bangalore"></property>  
        </bean>  
    </property>  
</bean>

**6. 什么是 spring 装配**

当 bean 在 Spring 容器中组合在一起时，它被称为装配或 bean 装配。Spring 容器需要知道需要什么 bean 以及容器应该如何使用依赖注入来将 bean 绑定在一起，同时装配 bean。

**7. 自动装配有哪些方式？**

Spring 容器能够自动装配 bean。也就是说，可以通过检查 BeanFactory 的内容让 Spring 自动解析 bean 的协作者。  
自动装配的不同模式：\* no - 这是默认设置，表示没有自动装配。应使用显式 bean 引用进行装配。\* byName - 它根据 bean 的名称注入对象依赖项。它匹配并装配其属性与 XML 文件中由相同名称定义的 bean。\* byType - 它根据类型注入对象依赖项。如果属性的类型与 XML 文件中的一个 bean 名称匹配，则匹配并装配属性。\* 构造函数 - 它通过调用类的构造函数来注入依赖项。它有大量的参数。\* autodetect - 首先容器尝试通过构造函数使用 autowire 装配，如果不能，则尝试通过 byType 自动装配。

**8. 自动装配有什么局限？**

\* 覆盖的可能性 - 您始终可以使用 和 设置指定依赖项，这将覆盖自动装配。\* 基本元数据类型 - 简单属性（如原数据类型，字符串和类）无法自动装配。\* 令人困惑的性质 - 总是喜欢使用明确的装配，因为自动装配不太精确。

**# 注解**

**1. 你用过哪些重要的 Spring 注解？**

\* @Controller - 用于 Spring MVC 项目中的控制器类。\* @Service - 用于服务类。\* @RequestMapping - 用于在控制器处理程序方法中配置 URI 映射。\* @ResponseBody - 用于发送 Object 作为响应，通常用于发送 XML 或 JSON 数据作为响应。\* @PathVariable - 用于将动态值从 URI 映射到处理程序方法参数。\* @Autowired - 用于在 spring bean 中自动装配依赖项。\* @Qualifier - 使用 @Autowired 注解，以避免在存在多个 bean 类型实例时出现混淆。\* @Scope - 用于配置 spring bean 的范围。\* @Configuration，@ComponentScan 和 @Bean - 用于基于 java 的配置。\* @Aspect，@Before，@After，@Around，@Pointcut - 用于切面编程（AOP）。

**2. 如何在 spring 中启动注解装配？**

默认情况下，Spring 容器中未打开注解装配。因此，要使用基于注解装配，我们必须通过配置<context：annotation-config /> 元素在 Spring 配置文件中启用它。

**3. @Component, @Controller, @Repository, @Service 有何区别？**

\* @Component：这将 java 类标记为 bean。它是任何 Spring 管理组件的通用构造型。spring 的组件扫描机制现在可以将其拾取并将其拉入应用程序环境中。\* @Controller：这将一个类标记为 Spring Web MVC 控制器。标有它的 Bean 会自动导入到 IoC 容器中。\* @Service：此注解是组件注解的特化。它不会对 @Component 注解提供任何其他行为。您可以在服务层类中使用 @Service 而不是 @Component，因为它以更好的方式指定了意图。  
\* @Repository：这个注解是具有类似用途和功能的 @Component 注解的特化。它为 DAO 提供了额外的好处。它将 DAO 导入 IoC 容器，并使未经检查的异常有资格转换为 Spring DataAccessException。

**4. @Required 注解有什么用？**

@Required 应用于 bean 属性 setter 方法。此注解仅指示必须在配置时使用 bean 定义中的显式属性值或使用自动装配填充受影响的 bean 属性。如果尚未填充受影响的 bean 属性，则容器将抛出 BeanInitializationException。  
示例：

**public** **class** **Employee** {  
    **private** String name;  
    @Required  
    **public** **void** **setName**(String name){  
        **this**.name=name;  
    }  
    **public** string **getName**(){  
        **return** name;  
    }  
}

**5. @Autowired 注解有什么用？**

@Autowired 可以更准确地控制应该在何处以及如何进行自动装配。此注解用于在 setter 方法，构造函数，具有任意名称或多个参数的属性或方法上自动装配 bean。默认情况下，它是类型驱动的注入。

**public** **class** **Employee** {  
    **private** String name;  
    @Autowired  
    **public** **void** **setName**(String name) {  
        **this**.name=name;  
    }  
    **public** string **getName**(){  
        **return** name;  
    }  
}

**6. @Qualifier 注解有什么用？**

当您创建多个相同类型的 bean 并希望仅使用属性装配其中一个 bean 时，您可以使用@Qualifier 注解和 @Autowired 通过指定应该装配哪个确切的 bean 来消除歧义。

例如，这里我们分别有两个类，Employee 和 EmpAccount。在 EmpAccount 中，使用@Qualifier 指定了必须装配 id 为 emp1 的 bean。

Employee.javapublic **class** **Employee** {  
    **private** String name;  
    @Autowired  
    **public** **void** **setName**(String name) {  
        **this**.name=name;  
    }  
    **public** string **getName**() {  
        **return** name;  
    }  
}

EmpAccount.java

**public** **class** **EmpAccount** {  
    **private** Employee emp;  
    **@Autowired**  
    **@Qualifier**(emp1)  
    **public** **void** **showName**() {  
        System.out.println(“Employee name : ”+emp.getName);  
    }  
}

**7. @RequestMapping 注解有什么用？**

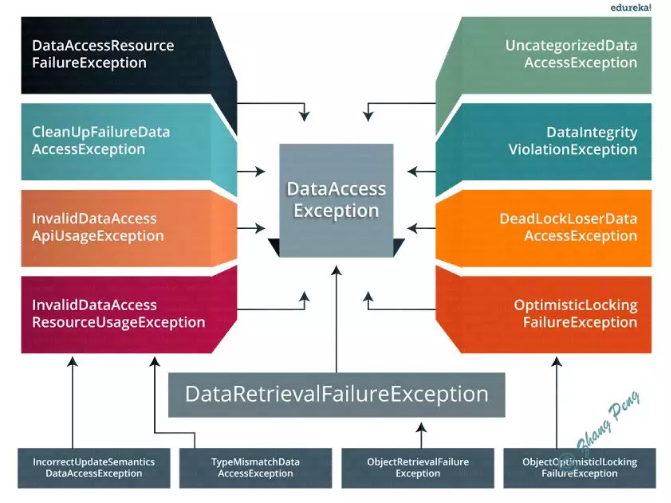
@RequestMapping 注解用于将特定 HTTP 请求方法映射到将处理相应请求的控制器中的特定类/方法。此注解可应用于两个级别：  
\* 类级别：映射请求的 URL\* 方法级别：映射 URL 以及 HTTP 请求方法

**# 数据访问**

**1. spring DAO 有什么用？**

Spring DAO 使得 JDBC，Hibernate 或 JDO 这样的数据访问技术更容易以一种统一的方式工作。这使得用户容易在持久性技术之间切换。它还允许您在编写代码时，无需考虑捕获每种技术不同的异常。

**2. 列举 Spring DAO 抛出的异常。**



**3. spring JDBC API 中存在哪些类？**

JdbcTemplateSimpleJdbcTemplateNamedParameterJdbcTemplateSimpleJdbcInsertSimpleJdbcCall

**4. 使用 Spring 访问 Hibernate 的方法有哪些？**

我们可以通过两种方式使用 Spring 访问 Hibernate：  
1.使用 Hibernate 模板和回调进行控制反转2.扩展 HibernateDAOSupport 并应用 AOP 拦截器节点

**5. 列举 spring 支持的事务管理类型**

Spring 支持两种类型的事务管理：  
1.程序化事务管理：在此过程中，在编程的帮助下管理事务。它为您提供极大的灵活性，但维护起来非常困难。  
2.声明式事务管理：在此，事务管理与业务代码分离。仅使用注解或基于 XML 的配置来管理事务。

**6. spring 支持哪些 ORM 框架**

HibernateiBatisJPAJDOOJB

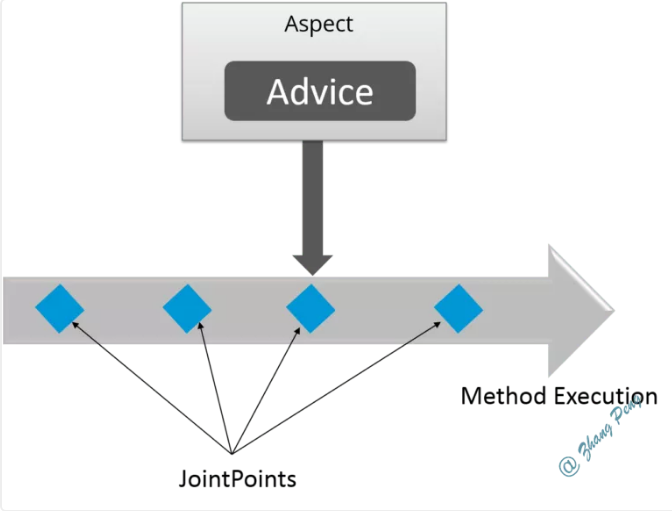
**# AOP**

**1. 什么是 AOP？**

AOP(Aspect-Oriented Programming), 即 面向切面编程, 它与 OOP( Object-Oriented Programming, 面向对象编程) 相辅相成, 提供了与 OOP 不同的抽象软件结构的视角.

在 OOP 中, 我们以类(class)作为我们的基本单元, 而 AOP 中的基本单元是 Aspect(切面)

**2. AOP 中的 Aspect、Advice、Pointcut、JointPoint 和 Advice 参数分别是什么？**



1.Aspect - Aspect 是一个实现交叉问题的类，例如事务管理。方面可以是配置的普通类，然后在 Spring Bean 配置文件中配置，或者我们可以使用 Spring AspectJ 支持使用 @Aspect 注解将类声明为 Aspect。  
2.Advice - Advice 是针对特定 JoinPoint 采取的操作。在编程方面，它们是在应用程序中达到具有匹配切入点的特定 JoinPoint 时执行的方法。您可以将 Advice 视为 Spring 拦截器（Interceptor）或 Servlet 过滤器（filter）。  
3.Advice Arguments - 我们可以在 advice 方法中传递参数。我们可以在切入点中使用 args() 表达式来应用于与参数模式匹配的任何方法。如果我们使用它，那么我们需要在确定参数类型的 advice 方法中使用相同的名称。  
4.Pointcut - Pointcut 是与 JoinPoint 匹配的正则表达式，用于确定是否需要执行 Advice。Pointcut 使用与 JoinPoint 匹配的不同类型的表达式。Spring 框架使用 AspectJ Pointcut 表达式语言来确定将应用通知方法的 JoinPoint。  
5.JoinPoint - JoinPoint 是应用程序中的特定点，例如方法执行，异常处理，更改对象变量值等。在 Spring AOP 中，JoinPoint 始终是方法的执行器。

**3. 什么是通知（Advice）？**

特定 JoinPoint 处的 Aspect 所采取的动作称为 Advice。Spring AOP 使用一个 Advice 作为拦截器，在 JoinPoint “周围”维护一系列的拦截器。

**4. 有哪些类型的通知（Advice）？**

\* Before - 这些类型的 Advice 在 joinpoint 方法之前执行，并使用 @Before 注解标记进行配置。\* After Returning - 这些类型的 Advice 在连接点方法正常执行后执行，并使用@AfterReturning 注解标记进行配置。\* After Throwing - 这些类型的 Advice 仅在 joinpoint 方法通过抛出异常退出并使用 @AfterThrowing 注解标记配置时执行。\* After (finally) - 这些类型的 Advice 在连接点方法之后执行，无论方法退出是正常还是异常返回，并使用 @After 注解标记进行配置。\* Around - 这些类型的 Advice 在连接点之前和之后执行，并使用 @Around 注解标记进行配置。

**5. 指出在 spring aop 中 concern 和 cross-cutting concern 的不同之处。**

concern 是我们想要在应用程序的特定模块中定义的行为。它可以定义为我们想要实现的功能。  
cross-cutting concern 是一个适用于整个应用的行为，这会影响整个应用程序。例如，日志记录，安全性和数据传输是应用程序几乎每个模块都需要关注的问题，因此它们是跨领域的问题。

**6. AOP 有哪些实现方式？**

实现 AOP 的技术，主要分为两大类：\* 静态代理 - 指使用 AOP 框架提供的命令进行编译，从而在编译阶段就可生成 AOP 代理类，因此也称为编译时增强；\* 编译时编织（特殊编译器实现）\* 类加载时编织（特殊的类加载器实现）。\* 动态代理 - 在运行时在内存中“临时”生成 AOP 动态代理类，因此也被称为运行时增强。\* JDK 动态代理\* CGLIB

**7. Spring AOP and AspectJ AOP 有什么区别？**

Spring AOP 基于动态代理方式实现；AspectJ 基于静态代理方式实现。  
Spring AOP 仅支持方法级别的 PointCut；提供了完全的 AOP 支持，它还支持属性级别的 PointCut。

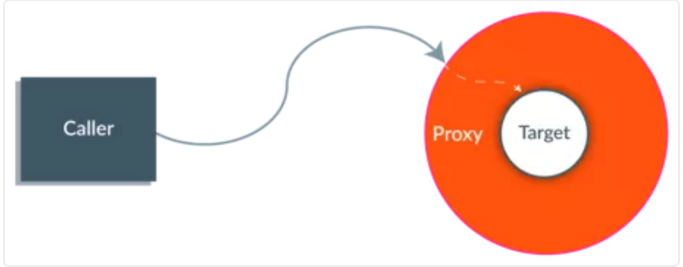
**8. 如何理解 Spring 中的代理？**

将 Advice 应用于目标对象后创建的对象称为代理。在客户端对象的情况下，目标对象和代理对象是相同的。

Advice + Target Object = Proxy

**9. 什么是编织（Weaving）？**

为了创建一个 advice 对象而链接一个 aspect 和其它应用类型或对象，称为编织（Weaving）。在 Spring AOP 中，编织在运行时执行。请参考下图：



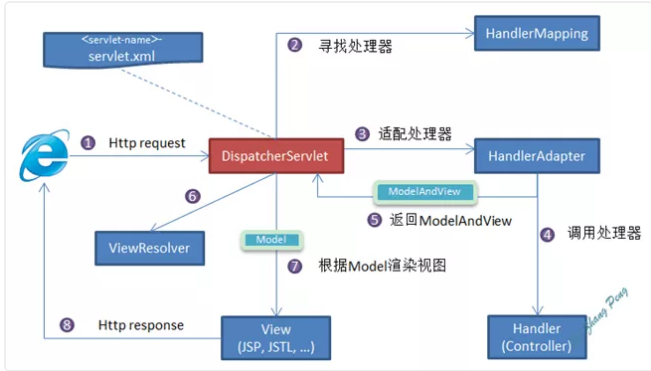
**# MVC**

**1. Spring MVC 框架有什么用？**

Spring Web MVC 框架提供 模型-视图-控制器 架构和随时可用的组件，用于开发灵活且松散耦合的 Web 应用程序。MVC 模式有助于分离应用程序的不同方面，如输入逻辑，业务逻辑和 UI 逻辑，同时在所有这些元素之间提供松散耦合。

**2. 描述一下 DispatcherServlet 的工作流程**

DispatcherServlet 的工作流程可以用一幅图来说明：



1.向服务器发送 HTTP 请求，请求被前端控制器 DispatcherServlet 捕获。  
2.DispatcherServlet 根据 -servlet.xml 中的配置对请求的 URL 进行解析，得到请求资源标识符（URI）。然后根据该 URI，调用 HandlerMapping 获得该 Handler 配置的所有相关的对象（包括 Handler 对象以及 Handler 对象对应的拦截器），最后以HandlerExecutionChain 对象的形式返回。  
3.DispatcherServlet 根据获得的Handler，选择一个合适的 HandlerAdapter。（附注：如果成功获得HandlerAdapter后，此时将开始执行拦截器的 preHandler(…)方法）。  
4.提取Request中的模型数据，填充Handler入参，开始执行Handler（Controller)。在填充Handler的入参过程中，根据你的配置，Spring 将帮你做一些额外的工作：  
\* HttpMessageConveter：将请求消息（如 Json、xml 等数据）转换成一个对象，将对象转换为指定的响应信息。\* 数据转换：对请求消息进行数据转换。如`String`转换成`Integer`、`Double`等。\* 数据根式化：对请求消息进行数据格式化。如将字符串转换成格式化数字或格式化日期等。\* 数据验证：验证数据的有效性（长度、格式等），验证结果存储到`BindingResult`或`Error`中。  
5.Handler(Controller)执行完成后，向 DispatcherServlet 返回一个 ModelAndView 对象；  
6.根据返回的ModelAndView，选择一个适合的 ViewResolver（必须是已经注册到 Spring 容器中的ViewResolver)返回给DispatcherServlet。  
7.ViewResolver 结合Model和View，来渲染视图。  
8.视图负责将渲染结果返回给客户端。

**3. 介绍一下 WebApplicationContext**

WebApplicationContext 是 ApplicationContext 的扩展。它具有 Web 应用程序所需的一些额外功能。它与普通的 ApplicationContext 在解析主题和决定与哪个 servlet 关联的能力方面有所不同。