

一、基本概念面试题集 (Spring 相关概念梳理)	6
1. 谈谈对 Spring IoC 的理解 ?	6
2. 谈谈对 Spring DI 的理解 ?	6
3. BeanFactory 接口和 ApplicationContext 接口不同点是什么 ?	6
4. 请介绍你熟悉的 Spring 核心类 , 并说明有什么作用 ?	6
5. 介绍一下 Spring 的事务的了解 ?	7
6. 介绍一下 Spring 的事务实现方式 ?	8
7. 解释 AOP 模块.....	9
8. Spring 的通知类型有哪些 , 请简单介绍一下 ?	9
9. Spring 通知类型使用场景分别有哪些 ?	10
10. 请介绍一下你对 Spring Beans 的理解?.....	11
11. Spring 有哪些优点?.....	11
12. 在 Spring 中使用 hibernate 的方法步骤.....	11
13. Spring 和 Struts 的区别 ?	11
14. Spring 框架由那几部分组成 ?	12
15. 谈谈你对 BeanFactory 的理解 , BeanFactory 实现举例.....	12
16. 谈谈对 Spring 中的 Web 模块的理解.....	13
17. BeanFactory 和 Application contexts 有什么区别 ?	13
18. 谈谈你对 Spring 依赖注入的理解 ?	13
19. 什么是 Bean 装配?.....	14
20. 什么是 Bean 的自动装配 ?	14
21. 介绍一下自动装配有几种方式 ?	14

22. 什么是基于注解的容器配置?	15
23. 简述 JdbcTemplate 类的作用	15
24. 解释 AOP	15
25. 解释 Aspect 切面	15
26. 简述 Spring AOP 中的通知	16
27. Spring AOP 中的织入你怎样理解?	16
28. 请详细介绍一下 Spring MVC 的流程?	17
29. Spring 配置文件?	17
30. @RequestMapping 注解用在类上面有什么作用	17
31. 怎么样把某个请求映射到特定的方法上面	18
32. 谈谈 Spring 对 DAO 的支持	18
二、应用场景面试题集(各知识点不同使用场景选型)	19
33. Spring 配置 Bean 实例化有哪些方式?	19
34. Bean 注入属性有哪几种方式	19
35. 在 Spring 中如何实现时间处理?	19
36. Spring 中如何更高效的使用 JDBC?	20
37. 请介绍一下设计模式在 Spring 框架中的使用?	20
38. 讲讲 Spring 框架的优点有哪些?	21
39. 哪种依赖注入方式你建议使用,构造器注入,还是 Setter 方法注入?	22
40. 你怎样定义类的作用域?	22
41. 解释 Spring 支持的几种 Bean 的作用域	22
42. 在 Spring 中如何注入一个 Java 集合?	23

43. 你可以在 Spring 中注入一个 null 和一个空字符串吗？	23
44. 什么是基于 Java 的 Spring 注解配置？给一些注解的例子	23
45. 你更倾向用那种事务管理类型？	24
46. Bean 的调用方式有哪些？	24
47. Spring MVC 里面拦截器是怎么写的	25
48. 当一个方法向 AJAX 返回特殊对象，譬如 Object、List 等，需要做什么处理？	26
49. 如何使用 Spring MVC 完成 JSON 操作	26
50. Spring 如何整合 Hibernate	26
51. Spring 如何整合 Struts2？	26
52. 开发中主要使用 Spring 的什么技术？	27
53. 介绍一下 Spring MVC 常用的一些注解	27
54. Spring 框架的事务管理有哪些优点	27
三、深度知识面试题集（底层实现原理详解）	28
55. IoC 控制反转设计原理？	28
56. Spring 的生命周期？	29
57. Spring 如何处理线程并发问题？	30
58. 核心容器（应用上下文）模块的理解？	31
59. 为什么说 Spring 是一个容器？	32
60. Spring 的优点？	32
61. Spring 框架中的单例 Beans 是线程安全的么？	32
62. Spring 框架中有哪些不同类型的事件？	33

63. IoC 的优点是什么？	34
64. 解释 Spring 框架中 Bean 的生命周期	34
65. 什么是 Spring 的内部 Bean？	35
66. 自动装配有哪些局限性？	35
67. Spring 框架的事务管理有哪些优点？	35
68. 在 Spring AOP 中，关注点和横切关注的区别是什么？	35
69. 说说 Spring AOP 的底层实现原理？	36
70. 如何给 Spring 容器提供配置元数据？	36
71. 哪些是重要的 Bean 生命周期方法？你能重载它们吗？	37
72. 讲下 Spring MVC 的执行流程	37
73. Spring MVC 的控制器是不是单例模式,如果是,有什么问题,怎么解决？	37
74. Spring 中循环注入的方式？	37
75. Spring MVC 比较 Struts2	40
四、拓展内容面试题集（Spring Boot 相关题集）	40
76. 什么是 Spring Boot？	40
77. Spring Boot 自动配置的原理？	40
78. Spring Boot 读取配置文件的方式？	41
79. 什么是微服务架构？	41
80. Ribbon 和 Feign 的区别？	41
81. Spring Cloud 断路器的作用？	42
82. 为什么要用 Spring Boot？	42
83. Spring Boot 的核心配置文件有哪几个？它们的区别是什么？	42

84. Spring Boot 的配置文件有哪几种格式？它们有什么区别？	43
85. Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的？	43
86. 开启 Spring Boot 特性有哪几种方式？	44
87. Spring Boot 需要独立的容器运行吗？	44
88. 运行 Spring Boot 有哪几种方式？	44
89. 你如何理解 Spring Boot 中的 Starters？	44
90. 如何在 Spring Boot 启动的时候运行一些特定的代码？	45
91. Spring Boot 有哪几种读取配置的方式？	45
92. Spring Boot 实现热部署有哪几种方式？	45
93. Spring Boot 多套不同环境如何配置？	47
94. Spring Boot 可以兼容老 Spring 项目吗，如何做？	48
95. 什么是 Spring Cloud？	48
96. 介绍一下 Spring Cloud 常用的组件？	48
97. Spring Cloud 如何实现服务注册的？	48
98. 什么是负载均衡？有什么作用？	48
99. 什么是服务熔断？	49
100. 请介绍一下 Ribbon 的主要作用？	49

一、基本概念面试题集（ Spring 相关概念梳理 ）

1. 谈谈对 Spring IoC 的理解？

IoC Inverse of Control 反转控制的概念。将之前程序中需要手动创建对象的操作，交由 Spring 框架来实现，创建对象的操作被反转到了 Spring 框架。对象的生命周期由 Spring 来管理，直接从 Spring 那里去获取一个对象。

2. 谈谈对 Spring DI 的理解？

DI Dependency Injection 依赖注入。Spring 框架创建 Bean 对象时，动态的将依赖对象注入到 Bean 组件中，实现依赖对象的注入。

3. BeanFactory 接口和 ApplicationContext 接口不同点是什么？

1. ApplicationContext 接口继承 BeanFactory 接口，Spring 核心工厂是 BeanFactory，BeanFactory 采取延迟加载，第一次 getBean 时才会初始化 Bean，ApplicationContext 是会在加载配置文件时初始化 Bean。
2. ApplicationContext 是对 BeanFactory 扩展，它可以进行国际化处理、事件传递和 Bean 自动装配以及各种不同应用层的 Context 实现。

开发中基本都在使用 ApplicationContext，Web 项目使用 WebApplicationContext，很少用到 BeanFactory。

4. 请介绍你熟悉的 Spring 核心类，并说明有什么作用？

1. BeanFactory : 产生一个新的实例 , 可以实现单例模式
2. BeanWrapper : 提供统一的 get 及 set 方法
3. ApplicationContext 提供框架的实现 , 包括 BeanFactory 的所有功能。

5. 介绍一下 Spring 的事务的了解 ?

事务是数据库操作的最小工作单元 , 是作为单个逻辑工作单元执行的一系列操作 ; 这些操作作为一个整体一起向系统提交 , 要么都执行、要么都不执行 ; 事务是一组不可再分割的操作集合 (工作逻辑单元) 。

事务特性 : ACID

1. 原子性 : 事务不可分割
2. 一致性 : 事务执行的前后 , 数据完整性保持一致
3. 隔离性 : 一个事务执行的时候 , 不应该受到其他事务的打扰
4. 持久性 : 一旦结束 , 数据就永久的保存到数据库

Spring 中有自己的事务管理机制 , 使用 TransactionManager 进行管理 , 可以通过 Spring 的注入来完成此功能。提供了以下几个事务处理的类 :

1. TransactionDefinition : 事务属性定义 , 定义了事务传播行为 类型 (7 种) , 事务隔离类型 (5 种) , 超时设置等。
2. TransactionStatus : 代表了当前的事务 , 可以提交、回滚。

3. PlatformTransactionManager :这个是 Spring 提供的用于管理事务的基础接口。通过这个接口 , Spring 可以为如 JDBC、Hibernate 等提供了对应的事务管理器 , 具体的实现就是各个平台来实现。

事务定义样例 :

```
TransactionDefinition td = new TransactionDefinition();
```

```
TransactionStatus ts = transactionManager.getTransaction(td); try{
```

```
//do sth
```

```
transactionManager.commit(ts);
```

```
}catch(Exception e){
```

```
transactionManager.rollback(ts);
```

```
}
```

6. 介绍一下 Spring 的事务实现方式 ?

Spring 事务管理实现方式有两种 : 编程式事务管理和声明式事务。

1. 编程式事务

使用 `TransactionTemplate` 或使用底层的 `PlatformTransactionManager` , 显式的方式调用 `beginTransaction()` 开启事务、`commit()` 提交事务、`rollback()` 回滚事务, 编写代码形式来声明事务。

2. 声明式事务

底层是建立在 Spring AOP 的基础上, 在方式执行前后进行拦截, 并在目标方法开始执行前创建新事务或加入一个已存在事务, 最后在目标方法执行完后根据情况提交或者回滚事务。

声明式事务的优点: 不需要编程, 减少了代码的耦合, 在配置文件中配置并在目标方法上添加 `@Transactional` 注解来实现。

7. 解释 AOP 模块

AOP (Aspect-Oriented Programming) 是一种程序设计类型, 它通过分离横切关注点来增加程序的模块化。AOP 在不修改现有代码的情况下对现有代码添加功能, 这个是 AOP 最重要的能力。

8. Spring 的通知类型有哪些, 请简单介绍一下?

Spring 的通知类型总共有 5 种: 前置通知、环绕通知、后置通知、异常通知、最终通知。

1. **前置通知 (Before advice)** : 在目标方法执行之前执行的通知。在某连接点 (join point) 之前执行的通知, 但这个通知不能阻止连接点前的执行 (除非它抛出一个异常)。
2. **环绕通知 (Around Advice)** : 在目标方法执行之前和之后都可以执行额外代码的通知。也可以选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。
3. **后置通知 (After (finally) advice)** : 目标方法执行之后 (某连接点退出的时候) 执行的通知 (不论是正常返回还是异常退出)。
4. **异常后通知 (After throwing advice)** : 在方法抛出异常退出时执行的通知。
5. **最终通知 (After returning advice)** : 在某连接点 (join point) 正常完成后执行的通知: 例如, 一个方法没有抛出任何异常, 正常返回。

9. Spring 通知类型使用场景分别有哪些?

通知类型	使用场景
环绕通知	控制事物 权限控制
后置通知	记录日志 (方法已经成功调用)
异常通知	异常处理 控制事物
最终通知	记录日志 (方法已经调用, 但不一定成功)

10. 请介绍一下你对 Spring Beans 的理解?

1. Spring Beans 是被实例的, 组装的及被 Spring 容器管理的 Java 对象
2. Spring Beans 会被 Spring 容器自动完成 @bean 对象的实例化
3. Spring 框架定义的 Beans 都是默认为单例, 也可以配置为多例

11. Spring 有哪些优点?

1. 提供控制反转能力, 将对象的创建交给了 Spring, 降低了代码耦合性、侵入性
2. Spring 是 POJO 编程, 使得可持续构建和可测试能力提高
3. Spring 是开源免费的
4. 方便集成各种优秀的框架

12. 在 Spring 中使用 hibernate 的方法步骤

1. 在 context 中定义 dataSource, 创建 SessionFactory, 设置参数
2. DAO 继承 hibernateDaoSupport, 实现具体的接口, 从中获得 HibernateTemplate 进行具体操作
3. 在使用中如果遇到 OpenSessionInView 的问题, 可以添加 OpenSessionInViewFilter 或 OpenSessionInViewInterceptor

13. Spring 和 Struts 的区别?

Spring :

1. 具备 IOC/DI、AOP 等通用能力，提高研发效率
2. 除了支持 Web 层建设以外，还提供了 J2EE 整体服务
3. 方便与其他不同技术结合使用，例如：Hibernate，Mybatis 等
4. Spring 拦截机制是方法级别

Struts：

1. 是一个基于 MVC 模式的一个 Web 层的处理
2. Struts 拦截机制是类级别

14. Spring 框架由那几部分组成？

主要七大模块介绍

1. Spring AOP 面相切面编程
2. Spring ORM Hibernate|mybatis|JDO
3. Spring Core 提供 Bean 工厂 IOC
4. Spring Dao JDBC 支持
5. Spring Context 提供了关于 UI 支持、邮件支持等
6. Spring Web 提供了 Web 的一些工具类的支持
7. Spring MVC 提供了 Web MVC、Webviews、JSP、PDF、Export

15. 谈谈你对 BeanFactory 的理解，BeanFactory 实现举例

- BeanFactory 用于管理 Bean 的，通过 BeanFactory 可以从 Spring 中获取注册到其中的 Bean 来使用了。
- BeanFactory 实现基于工厂模式，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、XmlWebApplicationContext、ClassPathXmlApplicationContext 等。

16. 谈谈对 Spring 中的 Web 模块的理解

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。

17. BeanFactory 和 Application contexts 有什么区别？

1. BeanFactory 提供了最简单的容器功能，只提供了实例化对象和拿对象的功能。
2. Application contexts 应用上下文，继承 BeanFactory 接口，它是 Spring 的一个更高级的容器，提供了更多有用的功能。

18. 谈谈你对 Spring 依赖注入的理解？

通常情况下，当需要调用一个其他对象的时候，采用 new 的方式进行对象的创建，导致对象之间耦合性增强，后续代码维护比较困难。

Spring 框提供了依赖注入的能力，对象统一由 Spring 容器创建，Spring 容器会负责程序之间的关系。这样控制权由应用代码转移到 Spring 容器，控制权发生了反转，就是 Spring 的控制反转。创建依赖对象的工作交由 Spring 容器来完成，然后注入调用者，这就是依赖注入。

19. 什么是 Bean 装配？

Spring 容器根据 Bean 之间的依赖关系，将 Bean 通过依赖注入进行组装在一起。这就是 Bean 装配。

20. 什么是 Bean 的自动装配？

在 Spring 框架里面是使用 `<constructor-arg>` 和 `<property>` 配置方式进行依赖注入，如果 Bean 对象较多的情况下注入工作就比较麻烦，XML 文件会变得很难维护，所以为了简化 XML 配置文件。

提高开发效率我们可以使用 `autowire`（自动装配），能通过 Bean 工厂自动处理 Bean 之间的协作。

21. 介绍一下自动装配有几种方式？

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。

1. `no`：默认设置，表示没有自动装配，通过显式设置 Bean 引用来进行装配。

2. `byName` : 根据 Bean 的名称注入对象依赖项。
3. `byType` : 根据类型注入对象依赖项。
4. `constructor` : 通过调用类的构造函数来注入依赖项
5. `autodetect` : 先尝试 `constructor` 来自动装配, 若不成功, 则使用 `byType` 方式。

22. 什么是基于注解的容器配置?

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, 直接在组件类中进行配置, 而不是使用 XML 表述 Bean 的装配关系。

23. 简述 JdbcTemplate 类的作用

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。

24. 解释 AOP

面向切面的编程, 或 AOP 是一种编程技术, 允许程序模块化横向切割关注点, 或横切典型的责任划分, 如日志和事务管理。

25. 解释 Aspect 切面

AOP 核心就是切面，它将多个类的通用行为封装成可重用的模块，该模块含有一组 API 提供横切功能。比如，一个日志模块可以被称作日志的 AOP 切面。根据需求的不同，一个应用程序可以有若干切面。在 Spring AOP 中，切面通过带有 @Aspect 注解的类实现。

26. 简述 Spring AOP 中的通知

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 Spring AOP 框架触发的代码段。

Spring 切面可以应用五种类型的通知：

1. before：前置通知，在一个方法执行前被调用。
2. after：在方法执行之后调用的通知，无论方法执行是否成功。
3. after-returning：仅当方法成功完成后执行的通知。
4. after-throwing：在方法抛出异常退出时执行的通知。
5. around：在方法执行之前和之后调用的通知。

27. Spring AOP 中的织入你怎样理解？

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。

28. 请详细介绍一下 Spring MVC 的流程？

1. 用户发送请求至前端控制器 DispatcherServlet
2. DispatcherServlet 收到请求调用 HandlerMapping 处理器映射器。
3. 处理器映射器根据请求 url 找到具体的处理器，生成处理器对象及处理器拦截器（如果有则生成）可以一并返回给 DispatcherServlet。
4. DispatcherServlet 通过 HandlerAdapter 处理器适配器调用处理器
5. 执行处理器（Controller，也叫后端控制器）。
6. Controller 执行完成返回 ModelAndView。
7. HandlerAdapter 将 controller 执行结果 ModelAndView 返回给 DispatcherServlet。
8. DispatcherServlet 将 ModelAndView 传给 View Resolver 视图解析器。
9. View Resolver 解析后返回具体 View。
10. DispatcherServlet 对 View 进行渲染视图（即将模型数据填充至视图中）。
11. DispatcherServlet 响应用户。

29. Spring 配置文件？

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。

30. @RequestMapping 注解用在类上面有什么作用

用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

31. 怎么样把某个请求映射到特定的方法上面

在方法上面加上注解 `@RequestMapping`，并且在这个注解里面写上要拦截的路径。

32. 谈谈 Spring 对 DAO 的支持

Spring 提供的 DAO（数据访问对象）支持主要的目的是便于以标准的方式使用不同的数据访问技术。

- 简化 DAO 组件的开发。Spring 提供了一套抽象 DAO 类供你扩展。这些抽象类提供了一些方法，用来简化代码开发。
- IoC 容器的使用，提供了 DAO 组件与业务逻辑组件之间的解耦。所有的 DAO 组件，都由容器负责注入到业务逻辑组件中，其业务组件无须关心 DAO 组件的实现。
- 面向接口编程及 DAO 模式的使用，提高了系统组件之间的解耦，降低了系统重构的成本。
- 方便的事务管理：Spring 的声明式事务管理力度是方法级。
- 异常包装：Spring 能够包装 Hibernate 异常，把它们从 `CheckedException` 变为 `RuntimeException`；开发者可选择在恰当的层处理数据中不可恢复的异常，从而避免烦琐的 `catch/throw` 及异常声明。

二、应用场景面试题集（各知识点不同使用场景选型）

33. Spring 配置 Bean 实例化有哪些方式？

1. 使用类构造器实例化（默认无参数）

```
<bean id="bean1" class="cn.itcast.spring.b_instance.Bean1"></bean>
```

2. 使用静态工厂方法实例化（简单工厂模式）

//下面这段配置的含义：调用 Bean2Factory 的 getBean2 方法得到 bean2

```
<bean id="bean2" class="cn.itcast.spring.b_instance.Bean2Factory"
```

```
factorymethod="getBean2"></bean>
```

3. 使用实例工厂方法实例化（工厂方法模式）

//先创建工厂实例 bean3Factory，再通过工厂实例创建目标 bean 实例

```
<bean id="bean3Factory" class="cn.itcast.spring.b_instance.Bean3Factory"/><bean
```

```
id="bean3" factorybean="bean3Factory" factorymethod="getBean3"></bean>
```

34. Bean 注入属性有哪几种方式

1. 属性注入方式，通过 setXXX() 方法注入 Bean 的属性值或者依赖对象
2. 构造函数注入方式，使用的前提：Bean 必须提供带参的构造函数
3. 工厂方法注入方式

35. 在 Spring 中如何实现时间处理？

在 applicationContext.xml 中配置事件源、监听器，先得到事件源，调用事件源的方法，通知监听器。

36. Spring 中如何更高效的使用 JDBC ？

传统的 JDBC 实现有两个不足之处：

- 连接需要自己管理操
- JDBC 操作代码封装与编写重复实现

Spring 实现了 JdbcTemplate，在 JDBC API 的基础做了科学的封装。

JdbcTemplate 的优点有：

1. 配置基于模板化处理
2. JdbcTemplate 是线程安全类
3. 实例化操作比较简单，仅需要传递 DataSource
4. 自动完成资源的创建和释放工作
5. 对 JDBC 的核心流程进行了封装，简化了对 JDBC 的操作
6. 创建一次 JdbcTemplate，到处可用，避免重复开发

37. 请介绍一下设计模式在 Spring 框架中的使用？

1. 工厂模式：BeanFactory 就是简单工厂模式的体现，用来创建对象的实例。
2. 单例模式：Bean 默认为单例模式。

3. 代理模式：Spring 的 AOP 功能用到了 JDK 的动态代理和 CGLIB 字节码生成技术。
4. 模板方法：用来解决代码重复的问题。比如：RestTemplate, JmsTemplate, JpaTemplate。
5. 观察者模式：定义对象间一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都会得到通知并被更新，如 Spring 中 listener 的实现：ApplicationListener。

38. 讲讲 Spring 框架的优点有哪些？

1. 非侵入式设计：代码与框架的依赖性比较低。
2. 代码解耦：提供控制反转能力，对象的创建和依赖关系的维护工作都交给 Spring 容器的管理，大大的降低了对象之间的耦合性。
3. 可复用性提高：支持 AOP，允许将一些通用能力（打印日志、事务处理、安全操作等）进行集中式处理。
4. 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。
5. 事务支持方便：Spring 提供一个持续的事务管理接口，配置化完成对事物的管理，减少手动编程。
6. 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。
7. 方便程序测试：提供了对 Junit4 的支持，可以通过注解方便的测试 Spring 程序。

39. 哪种依赖注入方式你建议使用，构造器注入，还是 Setter 方法注入？

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。

40. 你怎样定义类的作用域？

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。

41. 解释 Spring 支持的几种 Bean 的作用域

Spring 框架支持以下五种 Bean 的作用域：

1. singleton : Bean 在每个 Spring IoC 容器中只有一个实例。
2. prototype : 一个 Bean 的定义可以有多个实例。
3. 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。
4. 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。

5. `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。

42. 在 Spring 中如何注入一个 Java 集合？

Spring 提供以下几种集合的配置元素：

1. `<list>` 类型用于注入一系列值，允许有相同的值。
2. `<set>` 类型用于注入一组值，不允许有相同的值。
3. `<map>` 类型用于注入一组键值对，键和值都可以为任意类型。
4. `<props>` 类型用于注入一组键值对，键和值都只能为 String 类型。

还总结出了各大互联网公司 java 程序员面试涉及到的绝大部分面试题及答案做成了文档和学习笔记文件以及架构视频资料免费分享给大家（包括 Dubbo、Redis、Netty、zookeeper、Spring cloud、分布式、高并发等架构技术资料），加 QQ 群：857710406 或加管理小姐姐 VX：xy196xy 免费获取！

43. 你可以在 Spring 中注入一个 null 和一个空字符串吗？

可以

44. 什么是基于 Java 的 Spring 注解配置？给一些注解的例子

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。

45. 你更倾向用那种事务管理类型？

声明式事务管理，因为它对应用代码侵入性很少，更符合一个无侵入的轻量级容器的思想。

声明式事务管理要优于编程式事务管理，虽然比编程式事务管理（这种方式允许你通过代码控制事务）少了一点灵活性。

46. Bean 的调用方式有哪些？

有三种方式可以得到 Bean 并进行调用。

1. 使用 BeanWrapper

```
HelloWorld hw=new HelloWorld();
```

```
BeanWrapper bw=new BeanWrapperImpl(hw);
```

```
bw.setPropertyvalue("msg","HelloWorld");
```

```
system.out.println(bw.getPropertyCalue("msg"));
```

2. 使用 BeanFactory

```
InputStream is=new FileInputStream("config.xml");
```



```
XmlBeanFactory factory=new XmlBeanFactory(is);
```

```
HelloWorld hw=(HelloWorld) factory.getBean("HelloWorld");
```

```
system.out.println(hw.getMsg());
```

3. 使用 ApplicationContext

```
ApplicationContext actx=new FileSystemXmlApplicationContext("config.xml");
```

```
HelloWorld hw=(HelloWorld) actx.getBean("HelloWorld");
```

```
System.out.println(hw.getMsg());
```

使用 @ResponseBody 注解。

47. Spring MVC 里面拦截器是怎么写的

有两种写法，一种是实现接口，另外一种是继承适配器类，然后在 Spring MVC 的配置文件中配置拦截器即可：

```
<!-- 配置 SpringMvc 的拦截器 --> <mvc:interceptors>
```

```
<!-- 配置一个拦截器的 Bean 就可以了 默认是对所有请求都拦截 -->
```

```
<bean id="myInterceptor" class="com.et.action.MyHandlerInterceptor"/>
```

```
<!-- 只针对部分请求拦截 -->
```

```
<mvc:interceptor>
```

```
<mvc:mapping path="/modelMap.do" />
```

```
<bean class="com.et.action.MyHandlerInterceptorAdapter" />
```

```
</mvc:interceptor> </mvc:interceptors>
```

48. 当一个方法向 AJAX 返回特殊对象，譬如 Object、List 等，需要做什么处理？

要加上 @ResponseBody 注解。

49. 如何使用 Spring MVC 完成 JSON 操作

1. 配置 MappingJacksonHttpMessageConverter
2. 使用 @RequestBody 注解或 ResponseEntity 作为返回值

50. Spring 如何整合 Hibernate

整合 Hibernate，即由 IoC 容器生成 SessionFactory 对象，并使用 Spring 的声明式事务。

1. 利用 LocalSessionFactoryBean 工厂 Bean，声明一个使用 XML 映射文件的 SessionFactory 实例
2. 利用 HibernateTransactionManager 配置 Hibernate 的事务管理器

51. Spring 如何整合 Struts2？

整合 Struts2，即由 IoC 容器管理 Struts2 的 Action：

- 安装 Spring 插件：把 struts2-spring-plugin-2.2.1.jar 复制到当前 Web 应用的 WEB-INF/lib 目录下
- 在 Spring 的配置文件中配置 Struts2 的 Action 实例
- 在 Struts 配置文件中配置 action ,但其 class 属性不再指向该 Action 的实现类，而是指向 Spring 容器中 Action 实例的 ID

52. 开发中主要使用 Spring 的什么技术？

1. IoC 容器管理各层的组件
2. 使用 AOP 配置声明式事务
3. 整合其他框架

53. 介绍一下 Spring MVC 常用的一些注解

1. @RequestMapping：处理请求地址映射的注解，可用于类或方法上。
2. @PathVariable：绑定 URI 模板变量值是用来获得请求 url 中的动态参数
3. @RequestParam：用于将指定的请求参数赋值给方法中的形参
4. @RequestBody：读取 Request 请求的 body 部分数据
5. @ResponseBody：用于将 Controller 的方法返回的对象，通过适当的 HttpMessageConverter 转换为指定格式后，写入到 Response 对象的 body 数据区

54. Spring 框架的事务管理有哪些优点

1. 为不同的事务 API (JDBC、Hibernate、JPA) 提供统一的编程模型
2. 封装了简单统一的 API 对事物进行管理操作
3. 同时支持声明式事物和编程时事物两种方式

三、深度知识面试题集（底层实现原理详解）

55. IoC 控制反转设计原理？

具体设计原理如下图：



我们需要在 Web 层操作业务层，业务层操作数据层类



在 Web 层创建业务层类对象

```
UserService userService = new UserService();
```

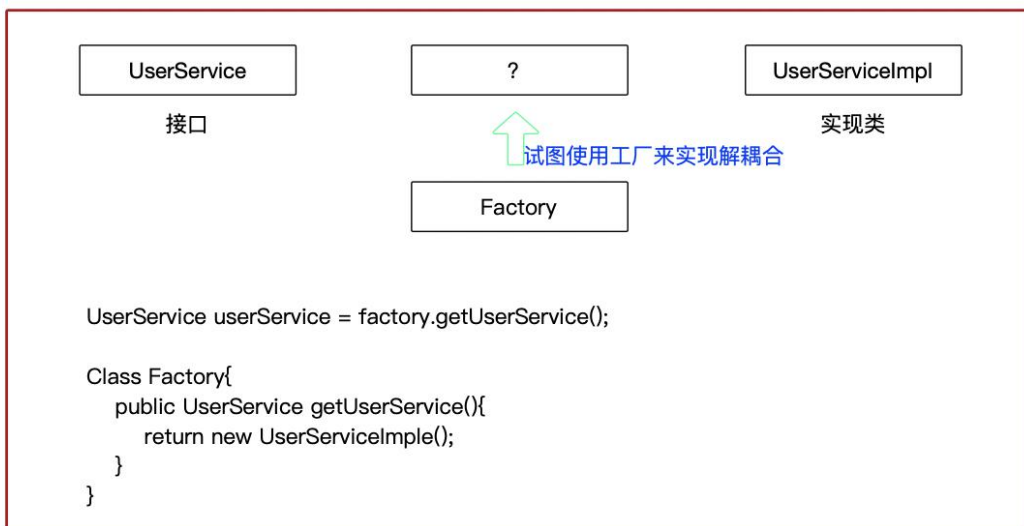
面向接口编程 —> 为了程序的扩展和维护

```
UserService userService = new UserServiceImpl();
```

方便维护，为接口提供了不同的实现类，这种直接在程序中让接口 = 实现类的做法造成了 程序的紧密耦合



紧密耦合在程序更换实现类时，需要修改源程序，而 Java 中有 OCP 原则（open-close），建议不需修改原先已经写好的代码，可以通过添加新的代码实现新的功能（对修改关闭，对扩展开放）



工厂和实现类通过反射和配置文件来实现解耦和

```
public UserService getUserService(){
    //读取配置文件，或得类的完全名称

    return Class.forName(类名).newInstance();
}
```

XML 或者 properties

```
userService = cn.itcast.service.UserServiceImpl
```

56. Spring 的生命周期？

1. instantiate Bean 对象实例化
2. populate properties 封装属性
3. 如果 Bean 实现 BeanNameAware 执行 setBeanName
4. 如果 Bean 实现 BeanFactoryAware 或者 ApplicationContextAware 设置工厂 setBeanFactory 或者上下文对象 setApplicationContext
5. 如果存在类实现 BeanPostProcessor (后处理 Bean) , 执行 postProcessBeforeInitialization , BeanPostProcessor 接口提供钩子函数, 用来动态扩展修改 Bean。(程序自动调用后处理 Bean)
6. 如果 Bean 实现 InitializingBean 执行 afterPropertiesSet
7. 调用 `<bean init-method="init">` 指定初始化方法 init
8. 如果存在类实现 BeanPostProcessor (处理 Bean) , 执行 postProcessAfterInitialization
9. 执行业务处理
10. 如果 Bean 实现 DisposableBean 执行 destroy
11. 调用 `<bean destroy-method="customerDestroy">` 指定销毁方法 customerDestroy

57. Spring 如何处理线程并发问题？

Spring 使用 ThreadLocal 解决线程安全问题。

我们知道在一般情况下，只有无状态的 Bean 才可以在多线程环境下共享，在 Spring 中，绝大部分 Bean 都可以声明为 singleton 作用域。就是因为

Spring 对一些 Bean (如 RequestContextHolder 、 TransactionSynchronizationManager、LocaleContextHolder 等) 中非线程安全状态采用 ThreadLocal 进行处理,让它们也成为线程安全的状态,因为有状态的 Bean 就可以在多线程中共享了。

ThreadLocal 和线程同步机制都是为了解决多线程中相同变量的访问冲突问题。在同步机制中,通过对象的锁机制保证同一时间只有一个线程访问变量。这时该变量是多个线程共享的,使用同步机制要求程序缜密地分析什么时候对变量进行读写,什么时候需要锁定某个对象,什么时候释放对象锁等繁杂的问题,程序设计和编写难度相对较大。而 ThreadLocal 则从另一个角度来解决多线程的并发访问。ThreadLocal 会为每一个线程提供一个独立的变量副本,从而隔离了多个线程对数据的访问冲突。因为每一个线程都拥有自己的变量副本,从而也就没有必要对该变量进行同步了。ThreadLocal 提供了线程安全的共享对象,在编写多线程代码时,可以把不安全的变量封装进 ThreadLocal。

由于 ThreadLocal 中可以持有任何类型的对象,低版本 JDK 所提供的 get() 返回的是 Object 对象,需要强制类型转换。但 JDK 5.0 通过泛型很好的解决了这个问题,在一定程度地简化 ThreadLocal 的使用。概括起来说,对于多线程资源共享的问题,同步机制采用了“以时间换空间”的方式,而 ThreadLocal 采用了“以空间换时间”的方式。前者仅提供一份变量,让不同的线程排队访问,而后者为每一个线程都提供了一份变量,因此可以同时访问而互不影响。

58. 核心容器 (应用上下文) 模块的理解 ?

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。

还总结出了各大互联网公司 java 程序员面试涉及到的绝大部分面试题及答案做成了文档和学习笔记文件以及架构视频资料免费分享给大家（包括 Dubbo、Redis、Netty、zookeeper、Spring cloud、分布式、高并发等架构技术资料），加 QQ 群：857710406 或加管理小姐姐 VX：xy196xy 免费获取！

59. 为什么说 Spring 是一个容器？

Spring 容器是整个 Spring 框架的核心，通常我们说的 Spring 容器就是 Bean 工厂，Bean 工厂负责创建和初始化 Bean、装配 Bean 并且管理应用程序中的 bean.spring 中提供了两个核心接口——BeanFactory 和 ApplicationContext，ApplicationContext 是 BeanFactory 子接口，它提供了比 BeanFactory 更完善的功能。

60. Spring 的优点？

1. Spring 属于低侵入式设计，代码的污染极低；
2. Spring 的 DI 机制将对象之间的依赖关系交由框架处理，减低组件的耦合性；
3. Spring 提供了 AOP 技术，支持将一些通用任务，如安全、事务、日志、权限等进行集中式管理，从而提供更好的复用。
4. Spring 对于主流的应用框架提供了集成支持。

61. Spring 框架中的单例 Beans 是线程安全的么？

Spring 框架并没有对单例 Bean 进行任何多线程的封装处理。关于单例 Bean 的线程安全和并发问题需要开发者自行去搞定。但实际上，大部分的 Spring Bean 并没有可变的状态(比如 Service 类和 DAO 类)，所以在某种程度上说 Spring 的单例 Bean 是线程安全的。如果你的 Bean 有多种状态的话（比如 View Model 对象），就需要自行保证线程安全。最浅显的解决办法就是将多态 Bean 的作用域由 “singleton” 变更为 “prototype”。

62. Spring 框架中有哪些不同类型的事件？

Spring 提供了以下 5 种标准的事件：

1. 上下文更新事件（ ContextRefreshedEvent ）：在调用 ConfigurableApplicationContext 接口中的 refresh() 方法时被触发。
2. 上下文开始事件（ ContextStartedEvent ）：当容器调用 ConfigurableApplicationContext 的 Start() 方法开始/重新开始容器时触发该事件。
3. 上下文停止事件（ ContextStoppedEvent ）：当容器调用 ConfigurableApplicationContext 的 Stop() 方法停止容器时触发该事件。
4. 上下文关闭事件(ContextClosedEvent) :当 ApplicationContext 被关闭时触发该事件。容器被关闭时，其管理的所有单例 Bean 都被销毁。
5. 请求处理事件（ RequestHandledEvent ）：在 Web 应用中，当一个 HTTP 请求（ request ）结束触发该事件。 如果一个 Bean 实现了

ApplicationListener 接口，当一个 ApplicationEvent 被发布以后，Bean 会自动被通知。

63. IoC 的优点是什么？

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。

64. 解释 Spring 框架中 Bean 的生命周期

1. Spring 容器从 XML 文件中读取 Bean 的定义，并实例化 Bean。
2. Spring 根据 Bean 的定义填充所有的属性。
3. 如果 Bean 实现了 BeanNameAware 接口，Spring 传递 Bean 的 ID 到 setBeanName 方法。
4. 如果 Bean 实现了 BeanFactoryAware 接口，Spring 传递 beanfactory 给 setBeanFactory 方法。
5. 如果有任何与 Bean 相关联的 BeanPostProcessors，Spring 会在 postProcessorBeforeInitialization() 方法内调用它们。
6. 如果 Bean 实现 InitializingBean 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。
7. 如果有 BeanPostProcessors 和 Bean 关联，这些 Bean 的 postProcessAfterInitialization() 方法将被调用。
8. 如果 Bean 实现了 DisposableBean，它将调用 destroy() 方法。

65. 什么是 Spring 的内部 Bean ?

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。

66. 自动装配有哪些局限性 ?

自动装配的局限性是 :

1. 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。
2. 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。
3. 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。

67. Spring 框架的事务管理有哪些优点 ?

1. 它为不同的事务 API 如 JTA、JDBC、Hibernate、JPA 和 JDO , 提供一个不变的编程模式。
2. 它为编程式事务管理提供了一套简单的 API 而不是一些复杂的事务 API。
3. 它支持声明式事务管理。
4. 它和 Spring 各种数据访问抽象层很好得集成。

68. 在 Spring AOP 中 , 关注点和横切关注的区别是什么 ?

1. 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。
2. 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。

69. 说说 Spring AOP 的底层实现原理？

Spring AOP 中的动态代理主要有两种方式，JDK 动态代理和 CGLIB 动态代理。

- JDK 动态代理通过反射来接收被代理的类，并且要求被代理的类必须实现一个接口。JDK 动态代理的核心是 `InvocationHandler` 接口和 `Proxy` 类。
- 如果目标类没有实现接口，那么 Spring AOP 会选择使用 CGLIB 来动态代理目标类。CGLIB (Code Generation Library)，是一个代码生成的类库，可以在运行时动态的生成某个类的子类，注意，CGLIB 是通过继承的方式做的动态代理，因此如果某个类被标记为 `final`，那么它是无法使用 CGLIB 做动态代理的。

70. 如何给 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

71. 哪些是重要的 Bean 生命周期方法？你能重载它们吗？

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`）。

72. 讲下 Spring MVC 的执行流程

系统启动的时候根据配置文件创建 Spring 的容器，首先是发送 http 请求到核心控制器 `disPatherServlet`，Spring 容器通过映射器去寻找业务控制器，使用适配器找到相应的业务类，在进入业务类时进行数据封装，在封装前可能会涉及到类型转换，执行完业务类后使用 `ModelAndView` 进行视图转发，数据放在 `model` 中，用 `map` 传递数据进行页面显示。

73. Spring MVC 的控制器是不是单例模式,如果是,有什么问题,怎么解决？

是单例模式，所以在多线程访问的时候有线程安全问题，不要用同步，会影响性能的，解决方案是在控制器里面不能写字段。

74. Spring 中循环注入的方式？

什么是循环注入？举个例子我有一个类 A ,A 有一个构造器里面的参数是类 B ,然后类 B 里面有个构造器参数是类 C ,类 C 里面有个构造器参数是类 A ,就是我们会发现其实引用循环了 A 里面有 B 的引用 ,B 里面有 C 的引用 ,C 里面又有 A 的引用。

当我们用 Spring 来加载 A 的时候 Spring 的流程是这样的：

1. Spring 创建 A 首先去当前创建池中去查找当前 A 是否在创建 ,如果发现没有创建则准备其构造器需要的参数 B ,然后把创建 A 的标识放入当前创建池中。
2. Spring 创建 B 首先去当前创建池中去查找当前 B 是否在创建 ,如果发现没有创建则准备其构造器需要的参数 C ,然后把创建 B 的标识放入当前创建池中。
3. Spring 创建 C 首先去当前创建池中去查找当前 C 是否在创建 ,如果发现没有创建则准备其构造器需要的参数 A ,然后把创建 C 的标识放入当前创建池中。
4. Spring 创建 C 需要的 A ,这个时候会发现在当前创建池中已经有 A 的标识 ,A 正在创建中则抛出 `BeanCurrentlyInCreationException`。

器的循环注入是没有办法解决的，所以只能我们避免。

接下来看下 set 方式的循环注入。

set 方式的循环注入分 2 种情况，第一种情况是可以解决的循环注入就是单列情况下。第二种情况就是无法解决的循环注入就是多列情况下，下面分析一下原因。

先看第一种情况，还是拿上面的 A、B、C3 个类来说明问题，只不过这次不是构造器里面的参数，而是换成他们的成员变量，然后通过 set 方式类注入，这里代码就不写了直接讲下。

单列下 set 方式的注入流程是这样的：

1. Spring 创建 A，首先根据其无参构造器创建一个对象 A，然后提前暴露出创建出来的这个 A 对象，然后再当前的创建池中放入创建 A 的标识，然后进行 set 方法注入 B。
2. Spring 创建 B，首先根据其无参构造器创建一个对象 B，然后提前暴露出创建出来的这个 B 对象，然后在当前的创建池中放入创建 B 的标识，然后进行 set 方法的注入 C。
3. Spring 创建 C，首先根据其无参构造器创建一个对象 C，然后提前暴露出创建处理的这个 C 对象，然后在当前的创建池中放入创建 C 的标识，然后进行 set 方法的注入 A。
4. 在第三步注入 A 的时候由于提前暴露出来了创建出来的 A 对象所以不会报 BeanCurrentlyInCreationException 的错误。

多列下 set 方式的循环注入不能解决的原因是在多列的情况下，当创建对象的时候 Spring 不会提前暴露创建处理的对象 A，这样的话则会和构造器循环注入出现一样的情况最终导致报错。

75. Spring MVC 比较 Struts2

1. Spring MVC 的入口是 Servlet，而 Struts2 是 Filter。
2. Spring MVC 会稍微比 Struts2 快些。Spring MVC 是基于方法设计，而 Struts2 是基于类，每次发一次请求都会实例一个 Action。
3. Spring MVC 使用更加简洁，开发效率 Spring MVC 确实比 struts2 高，支持 JSR303，处理 Ajax 的请求更方便。
4. Struts2 的 OGNL 表达式使页面的开发效率相比 Spring MVC 更高些。
- 5.

四、拓展内容面试题集（Spring Boot 相关题集）

76. 什么是 Spring Boot？

能够简化 Spring 应用的初始搭建以及开发过程，使用特定的方式来进行配置（properties 或 yml 文件），如：创建独立的 Spring 应用程序 main 方法运行、嵌入的 Tomcat 无需部署 war 文件、简化 Maven 配置、自动配置 Spring 添加对应功能 starter 自动化配置等。

77. Spring Boot 自动配置的原理？

在 Spring 程序 main 方法中，添加 @Spring BootApplication 或者 @EnableAutoConfiguration 注解，程序启动时会通过扫描注解自动去 maven 中读取每个 starter 中的 spring.factories 文件，此文件里配置了所有需要被创建 Spring 容器中的 Bean。

78. Spring Boot 读取配置文件的方式？

Spring Boot 默认读取配置文件为 application.properties 或者是 application.yml。

79. 什么是微服务架构？

微服务架构（Microservice Architecture）是一种架构概念，通过将功能或者业务将统一服务拆分到各个不同的服务中以实现系统服务的解耦，更加灵活的服务支持。拆分的不同服务可以独立的进行开发、管理、迭代。

80. Ribbon 和 Feign 的区别？

Ribbon 和 Feign 都是用于调用其他服务的，不过方式不同。

1. 启动类使用的注解不同：Ribbon 用的是 @RibbonClient，Feign 用的是 @EnableFeignClients。
2. 服务的指定位置不同：Ribbon 是在 @RibbonClient 注解上声明，Feign 则是在定义抽象方法的接口中使用 @FeignClient 声明。
3. 调用方式不同：Ribbon 需要自己构建 http 请求，模拟 http 请求然后使用 RestTemplate 发送给其他服务，步骤相当繁琐。Feign 则是在

Ribbon 的基础上进行了一次改进，采用接口的方式，将需要调用的其他服务的方法定义成抽象方法即可，不需要自己构建 HTTP 请求。

81. Spring Cloud 断路器的作用？

当一个服务调用另一个服务由于网络原因或者自身原因出现问题时，调用者就会等待被调用者的响应。当更多的服务请求到这些资源时，导致更多的请求等待，这样就会发生连锁效应（雪崩效应），断路器就是解决这一问题。

82. 为什么要用 Spring Boot ？

独立运行、简化配置、自动配置、代码生成和 XML 配置、应用监控、上手容易等。

83. Spring Boot 的核心配置文件有哪几个？它们的区别是什么？

Spring Boot 的核心配置文件是 application 和 bootstrap 配置文件。

application 配置文件这个容易理解，主要用于 Spring Boot 项目的自动化配置。

bootstrap 配置文件有以下几个应用场景。

使用 Spring Cloud Config 配置中心时，这时需要在 bootstrap 配置文件中添加连接到配置中心的配置属性来加载外部配置中心的配置信息；一些固定的不能被覆盖的属性；一些加密/解密的场景；

84. Spring Boot 的配置文件有哪几种格式？它们有什么区别？

.properties 和 .yaml，它们的区别主要是书写格式不同。

1. .properties

```
app.user.name = javastack
```

2. yaml

```
app:
```

```
  user:
```

```
    name: javastack
```

85. Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的？

启动类上面的注解是 @SpringBootApplication，它也是 Spring Boot 的核心注解，主要组合包含了以下 3 个注解：

- @SpringBootConfiguration：组合了 @Configuration 注解，实现配置文件的功能。

- `@EnableAutoConfiguration` : 打开自动配置的功能，也可以关闭某个自动配置的选项，如关闭数据源自动配置功能：`@SpringBootApplication(exclude={ DataSourceAutoConfiguration.class })`。
- `@ComponentScan` : Spring 组件扫描。

86. 开启 Spring Boot 特性有哪几种方式？

1. 继承 `spring-boot-starter-parent` 项目
2. 导入 `spring-boot-dependencies` 项目依赖

87. Spring Boot 需要独立的容器运行吗？

可以不需要，内置了 Tomcat/Jetty 等容器。

88. 运行 Spring Boot 有哪几种方式？

1. 打包用命令或者放到容器中运行
2. 用 Maven/Gradle 插件运行
3. 直接执行 `main` 方法运行

89. 你如何理解 Spring Boot 中的 Starters？

Starters 理解为启动器，它包含了一系列可以集成到应用里面的依赖包，方便开发者快速集成 Spring 及其他技术，避免投入很多精力寻找依赖包或者代码的工作。

Spring 的官方启动器都是以 `spring-boot-starter-` 命名的，代表了一个特定的应用类型。

90. 如何在 Spring Boot 启动的时候运行一些特定的代码？

通过实现接口 `ApplicationRunner` 或者 `CommandLineRunner`，来实现 `SpringApplication` 执行之后开始执行一部分功能。

这两个接口实现方式一样，它们都只提供了一个 `run` 方法。

`ApplicationRunner` 通过 `ApplicationArguments` 用来接收参数的。

`CommandLineRunner` 接收字符串数组的命令行参数。

91. Spring Boot 有哪几种读取配置的方式？

Spring Boot 可以通过 `@PropertySource`、`@Value`、`@Environment`、`@ConfigurationProperties` 来绑定变量。

92. Spring Boot 实现热部署有哪几种方式？

主要有两种方式：`Spring Loaded`、`Spring-boot-devtools`。

方式 1：`Spring Loaded`

`pom.xml` 中直接在 `spring-boot` 插件中添加依赖。

```
<plugin>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-maven-plugin</artifactId>
```

```
<dependencies>
```

```
<dependency>
```

```
<groupId>org.springframework</groupId>
```

```
<artifactId>springloaded</artifactId>
```

```
<version>1.2.6.RELEASE</version>
```

```
</dependency>
```

```
</dependencies>
```

```
<configuration>
```

```
<mainClass>此处为入口类</mainClass>
```

```
</configuration>
```

```
</plugin>
```

方式 2 : Spring-boot-devtools

pom.xml 中直接添加依赖即可。

```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-devtools</artifactId>
```

```
<scope>provided</scope>
```

```
<optional>true</optional></dependency>
```

93. Spring Boot 多套不同环境如何配置？

真实的开发环境，一个工程一般有几个不同的环境，比如：dev、test、prod 等。

不同环境的配置信息各不相同，例如：数据库地址信息、三方接口域名信息、缓存地址配置信息等、端口信息。

如果不同环境部署程序的时候，需要不断的修改这些配置文件将会非常繁琐。因此 Spring Boot 就提供了不同环境的配置文件的操作。

在 Spring Boot 中多环境配置文件名需要满足 application-{profile}.properties 的格式，其中 {profile} 对应不同环境的标识。如：

1. application.properties
2. application-dev.properties
3. application-test.properties
4. application-prod.properties

具体哪个配置文件会被加载，由 application.properties 文件中的 spring.profiles.active 属性来设置，其值对应 {profile} 值。

94. Spring Boot 可以兼容老 Spring 项目吗，如何做？

可以兼容，使用 `@ImportResource` 注解导入老 Spring 项目配置文件。

95. 什么是 Spring Cloud？

Spring Cloud 是微服务架构提供者，将一系列优秀的组件进行了整合。基于 Spring Boot 构建，通过一些简单的注解，开发者可以快速的在应用中配置一下常用模块并构建合适的分布式系统。

96. 介绍一下 Spring Cloud 常用的组件？

- 服务发现——Netflix Eureka
- 客户端负载均衡——Netflix Ribbon
- 断路器——Netflix Hystrix
- 服务网关——Netflix Zuul
- 分布式配置——Spring Cloud Config

97. Spring Cloud 如何实现服务注册的？

所有服务都在 Eureka 服务器上注册并通过调用 Eureka 服务器完成查找。

98. 什么是负载均衡？有什么作用？

负载均衡主要思想是优化资源利用，最大化吞吐量，最小化响应时间并避免任何服务单一资源的过载。

负载均衡可以优化单机、服务集群、网络链接等多种计算资源的使用负载分布。
使用多个组件实现负载均衡而不是单个组件，通过冗余来提高可靠性和可用性。

99. 什么是服务熔断？

在应用系统服务中，当依赖服务因访问压力过大而响应变慢或失败，上游服务为了保护系统整体的可用性，临时切断对下游服务的调用。这种牺牲局部，保全整体的措施就叫做熔断。

100. 请介绍一下 Ribbon 的主要作用？

Ribbon，提供 Client 端的访问负载均衡算法。基于 HTTP 和 TCP 的 Client 端负载均衡工具，基于 Netflix Ribbon 实现。通过 Spring Cloud 封装，可以让开发者非常方便地面向服务的 REST 模版请求自动转换成客户端负载均衡的服务调用。

读者分享

诚邀您加 QQ 群：[578486082](#) 免费领取

VX：[rxh8515](#)，全部都可以免费领取










验证消息回答：“[Spring](#)” 否在不予通过好友，后续资料也会不断更新

Java 架构进阶资料展示



有面试复习资料还有整理了面试高频问题的视频解析和大咖架构进阶笔记

【Mark老师】面试必备—服务器推送技术.mp4	2019-05-11 12:00	972.72MB
【James老师】面试必备—轻松搞定AOP面试从Spring热插件实战开始.m...	2019-05-11 12:00	793.46MB
【James老师】面试必备—快速搞定RabbitMq中间件.mp4	2019-05-11 12:00	786.27MB
【James老师】面试必备—手写SpringMVC框架, 从害怕到喜欢只需1小...	2019-05-11 12:00	771.43MB
【Peter老师】面试必备—API接口安全.mp4	2019-05-11 13:55	756.11MB
【King老师】面试必备—JVM性能优化.mp4	2019-05-11 11:59	633.64MB
【King老师】面试必备—深入理解JVM.mp4	2019-05-11 11:59	601.06MB
【Lison老师】面试必备—匠心独运手写MyBatis框架.mp4	2019-05-11 11:59	577.78MB
【Deer老师】面试必备—zk分布式锁.mp4	2019-05-11 11:58	554.66MB
【Lison老师】面试必备—Spring AOP源码讲解.mp4	2019-05-11 11:58	513.67MB
【Deer老师】面试必备—Redis进阶问题讲解.mp4	2019-05-11 11:59	487.04MB

-  Dubbo服务框架知识点PDF文档整理
-  Java架构进阶150道面试题PDF文档整理
-  Java架构师面试技术点整理及学习笔记
-  Java面试高频试题汇总（整理答案适合1-5年开发）
-  Java面试知识点体系PDF文档整理
-  JVM及性能优化知识点笔记PDF文档整理
-  Redis知识点笔记PDF文档整理
-  Spring全家桶知识点PDF文档整理
-  分布式架构知识点笔记PDF文档整理