

如果一个分片停止了，除非查询设置了“Partial”选项，否则查询会返回一个错误。如果一个分片响应很慢，MongoDB会等待它的响应。

42、分析器在MongoDB中的作用是什么？

分析器就是explain 显示每次操作性能特点的数据库分析器。通过分析器可能查找比预期慢的操作

43、如果用户移除对象的属性，该属性是否从存储层中删除？

是的，用户移除属性然后对象会重新保存（re-save()）。

44、能否使用日志特征进行安全备份？

是的

45、更新操作立刻fsync到磁盘？

一般磁盘的写操作都是延迟执行的

46、如何执行事务/加锁？

因为mongodb设计就是轻量高性能，所以没有传统的锁和复杂的事务的回滚

47、什么是master或primary？

当前备份集群负责所有的写入操作的主要节点，在集群中，当主节点（master）失效，另一个成员会变为master

48、getLastError的作用

调用getLastError 可以确认当前的写操作是否成功的提交

49、分片（sharding）和复制（replication）是怎样工作的？

分片可能是单一的服务器或者集群组成，推荐使用集群

50、数据在什么时候才会扩展到多个分片（shard）里？

mongodb分片是基于区域的，所以一个集合的所有对象都放置在同一个块中，只有当存在多余一个块的时候，才会有多个分片获取数据的选项

51、当我试图更新一个正在被迁移的块（chunk）上的文档时会发生什么？

会立即更新旧的分片，然后更改才会在所有权转移前复制到新的分片上

52、我怎么查看 Mongo 正在使用的链接？

```
db._adminCommand("connPoolStats");
```

53、mongodb的结构介绍

数据库中存储的对象设计bson，一种类似json的二进制文件，由键值对组成

54、数据库的整体结构

键值对-》文档-》集合-》数据库

55、MongoDB是由哪种语言写的

MongoDB用c++编写的，流行的开源数据库MySQL也是用C++开发的。C++1983年发行是一种使用广泛的计算机程序设计语言。它是一种通用[程序设计语言](#)，支持[多重编程模式](#)。

56、MongoDB的优势有哪些

- 面向文档的存储：以 JSON 格式的文档保存数据。
- 任何属性都可以建立索引。
- 复制以及高可扩展性。
- 自动分片。
- 丰富的查询功能。
- 快速的即时更新。
- 来自 MongoDB 的专业支持。、

57、什么是集合

集合就是一组 MongoDB 文档。它相当于关系型数据库（RDBMS）中的表这种概念。集合位于单独的一个数据库中。一个集合内的多个文档可以有多个不同的字段。一般来说，集合中的文档都有着相同或相关的目的。

58、什么是文档

文档由一组 key value 组成。文档是动态模式，这意味着同一集合里的文档不需要有相同的字段和结构。在关系型数据库中 table 中的每一条记录相当于 MongoDB 中的一个文档。

59、什么是“mongod”

mongod 是处理 MongoDB 系统的主要进程。它处理数据请求，管理数据存储，和执行后台管理操作。当我们运行 mongod 命令意味着正在启动 MongoDB 进程，并且在后台运行。

60、“mongod”参数有什么

- 传递数据库存储路径，默认是“/data/db”
- 端口号 默认是 “27017”

61、什么是“mongo”

它是一个命令行工具用于连接一个特定的 mongod 实例。当我们没有带参数运行 mongo 命令，它将使用默认的端口号和 localhost 连接。

62、MongoDB哪个命令可以切换数据库

MongoDB 用 use + 数据库名称 的方式来创建数据库。use 会创建一个新的数据库，如果该数据库存在，则返回这个数据库。

```
>use database_name
```

63、什么是非关系型数据库

非关系型数据库是对不同于传统关系型数据库的统称。非关系型数据库的显著特点是不使用 SQL 作为查询语言，数据存储不需要特定的表格模式。由于简单的设计和非常好的性能所以被用于大数据和 Web Apps 等。

64、非关系型数据库有哪些类型

- -Key-Value 存储 Eg:Amazon S3
- 图表 Eg:Neo4j
- 文档存储 Eg:MongoDB
- 基于列存储 Eg:Cassandra

65、为什么用 MongoDB？

- 架构简单
- 没有复杂的连接
- 深度查询能力，MongoDB 支持动态查询。
- 容易调试
- 容易扩展
- 不需要转化/映射应用对象到数据库对象
- 使用内部内存作为存储工作区，以便更快的存取数据。

66、在哪些场景使用 MongoDB

- 大数据
- 内容管理系统
- 移动端App
- 数据管理

67、MongoDB中的命名空间是什么意思？

MongoDB内部有预分配空间的机制，每个预分配的文件都用0进行填充。

数据文件每新分配一次，它的大小都是上一个数据文件大小的2倍，每个数据文件最大2G。

MongoDB每个集合和每个索引都对应一个命名空间，这些命名空间的元数据集中在16M的*.ns文件中，平均每个命名占用约628字节，也即整个数据库的命名空间的上限约为24000。

如果每个集合有一个索引（比如默认的_id索引），那么最多可以创建12000个集合。如果索引数更多，则可创建的集合数就更少了。同时，如果集合数太多，一些操作也会变慢。

要建立更多的集合的话，MongoDB也是支持的，只需要在启动时加上“--nssize”参数，这样对应数据库的命名空间文件就可以变得更大以便保存更多的命名。这个命名空间文件（.ns文件）最大可以为2G。

每个命名空间对应的盘区不一定是连续的。与数据文件增长相同，每个命名空间对应的盘区大小都是随分配次数不断增长的。目的是为了平衡命名空间浪费的空间与保持一个命名空间数据的连续性。

需要注意的一个命名空间\$freelist，这个命名空间用于记录不再使用的盘区（被删除的Collection或索引）。每当命名空间需要分配新盘区时，会先查看\$freelist是否有大小合适的盘区可以使用，如果有就回收空闲的磁盘空间。

68、哪些语言支持MongoDB？

- C
- C++
- C#
- Java
- Node.js
- Perl
- PHP 等

69、在MongoDB中如何创建一个新的数据库

MongoDB用use + 数据库名称的方式来创建数据库。use会创建一个新的数据库，如果该数据库存在，则返回这个数据库。

```
>use mydb  
switched to db mydb
```

70、在MongoDB中如何查看数据库列表

使用命令“show dbs”

```
>show dbs
```

71、MongoDB中的分片是什么意思

分片是将数据水平切分到不同的物理节点。当应用数据越来越大的时候，数据量也会越来越大。当数据量增长时，单台机器有可能无法存储数据或可接受的读取写入吞吐量。利用分片技术可以添加更多的机器来应对数据量增加以及读写操作的要求。

72、如何查看使用MongoDB的连接Sharding - MongoDB Manual21.如何查看使用MongoDB的连接

使用命令“db.adminCommand(“connPoolStats”)”

```
>db.adminCommand("connPoolStats")
```

73、什么是复制

复制是将数据同步到多个服务器的过程，通过多个数据副本存储到多个服务器上增加数据可用性。复制可以保障数据的安全性，灾难恢复，无需停机维护（如备份，重建索引，压缩），分布式读取数据。

74、在MongoDB中如何在集合中插入一个文档

要想将数据插入MongoDB集合中，需要使用insert()或save()方法。

```
>db.collectionName.insert({"key": "value"})
>db.collectionName.save({"key": "value"})
```

75、在MongoDB中如何除去一个数据库Collection Methods

24. 在MongoDB中如何除去一个数据库

MongoDB 的 `dropDatabase()` 命令用于删除已有数据库。

```
>db.dropDatabase()
```

76、在MongoDB中如何创建一个集合。

在 MongoDB 中，创建集合采用 `db.createCollection(name, options)` 方法。 `options` 是一个用来指定集合配置的文档。

```
>db.createcollection("collectionName")db.createcollection() - MongoDB Manual>db.createcollection("
```

77、在MongoDB中如何查看一个已经创建的集合

可以使用 `show collections` 查看当前数据库中的所有集合清单

```
>show collections
```

78、在MongoDB中如何删除一个集合

MongoDB 利用 `db.collection.drop()` 来删除数据库中的集合。

```
>db.collectionName.drop()
```

79、为什么要在MongoDB中使用分析器

数据库分析工具(Database Profiler)会针对正在运行的mongod实例收集数据库命令执行的相关信息。包括增删改查的命令以及配置和管理命令。分析器(profiler)会写入所有收集的数据到 `system.profile` 集合，一个capped集合在管理员数据库。分析器默认是关闭的你能通过per数据库或per实例开启。

80、MongoDB支持主键外键关系吗

默认MongoDB不支持主键和外键关系。用Mongodb本身的API需要硬编码才能实现外键关联，不够直观且难度较大。可以参考以下链接

81、MongoDB支持哪些数据类型

- String
- Integer
- Double
- Boolean
- Object
- Object ID
- Arrays
- Min/Max Keys
- Datetime
- Code
- Regular Expression等

82、为什么要在MongoDB中用"Code"数据类型

"Code"类型用于在文档中存储 JavaScript 代码。

83、为什么要在MongoDB中用"Regular Expression"数据类型

"Regular Expression"类型用于在文档中存储正则表达式

84、为什么在MongoDB中使用"Object ID"数据类型

"`ObjectID`"数据类型用于存储文档id

85、如何在集合中插入一个文档

要想将数据插入 MongoDB 集合中，需要使用`insert()`或`save()`方法。

```
>db.collectionName.insert({ "key": "value" })
>db.collectionName.save({ "key": "value" })
```

86、"`ObjectID`"有哪些部分组成

一共有四部分组成:时间戳、客户端ID、客户进程ID、三个字节的增量计数器

`_id`是一个 12 字节长的十六进制数，它保证了每一个文档的唯一性。在插入文档时，需要提供`_id`。如果你不提供，那么 MongoDB 就会为每一文档提供一个唯一的 id。`_id`的头 4 个字节代表的是当前的时间戳，接着的后 3 个字节表示的是机器 id 号，接着的 2 个字节表示 MongoDB 服务器进程 id，最后的 3 个字节代表递增值。

87、在MongoDb中什么是索引

索引用于高效的执行查询.没有索引MongoDB将扫描查询整个集合中的所有文档这种扫描效率很低，需要处理大量数据。索引是一种特殊的数据结构，将一小块数据集保存为容易遍历的形式。索引能够存储某种特殊字段或字段集的值，并按照索引指定的方式将字段值进行排序。

88、如何添加索引

使用`db.collection.createIndex()`在集合中创建一个索引

```
>db.collectionName.createIndex({columnName:1})
```

89、用什么方法可以格式化输出结果

使用`pretty()`方法可以格式化显示结果

```
>db.collectionName.find().pretty()
```

90、如何使用"AND"或"OR"条件循环查询集合中的文档

在`find()`方法中，如果传入多个键，并用逗号(,)分隔它们，那么 MongoDB 会把它看成是**AND**条件。

```
>db.mycol.find({key1:value1, key2:value2}).pretty()
```

若基于**OR**条件来查询文档，可以使用关键字`$or`。

```
>db.mycol.find(
  {
    $or: [
      {key1: value1}, {key2:value2}
    ]
  }
).pretty()
```

91、在MongoDB中如何更新数据

`update()`与`save()`方法都能用于更新集合中的文档。`update()`方法更新已有文档中的值，而`save()`方法则是用传入该方法的文档来替换已有文档。

```
>db.collectionName.update({key:value}, {$set:{newkey:newValue}})
```

92、如何删除文档

MongoDB 利用`remove()`方法清除集合中的文档。它有 2 个可选参数：

- `deletion criteria`：(可选)删除文档的标准。
- `justOne`：(可选)如果设为 true 或 1，则只删除一个文档。

```
>db.collectionName.remove({key:value})
```

93、在MongoDB中如何排序

MongoDB 中的文档排序是通过 `sort()` 方法来实现的。`sort()` 方法可以通过一些参数来指定要进行排序的字段，并使用 1 和 -1 来指定排序方式，其中 1 表示升序，而 -1 表示降序。

```
>db.connectionName.find({key:value}).sort({columnName:1})
```

94、什么是聚合

聚合操作能够处理数据记录并返回计算结果。聚合操作能将多个文档中的值组合起来，对成组数据执行各种操作，返回单一的结果。它相当于 SQL 中的 `count(*)` 组合 `group by`。对于 MongoDB 中的聚合操作，应该使用 `aggregate()` 方法。

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

95、在MongoDB中什么是副本集

在MongoDB中副本集由一组MongoDB实例组成，包括一个主节点多个次节点，MongoDB客户端的所有数据都写入主节点(Primary)，副节点从主节点同步写入数据，以保持所有复制集内存储相同的数据，提高数据可用性。

String面试题

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。
Spring 5.0	Spring Framework 5.0的最大特点之一是响应式编程 (Reactive Programming)。响应式编程核心功能和对响应式 endpoints 的支持可通过 Spring Framework 5.0 中获得。

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，同时还能为 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。

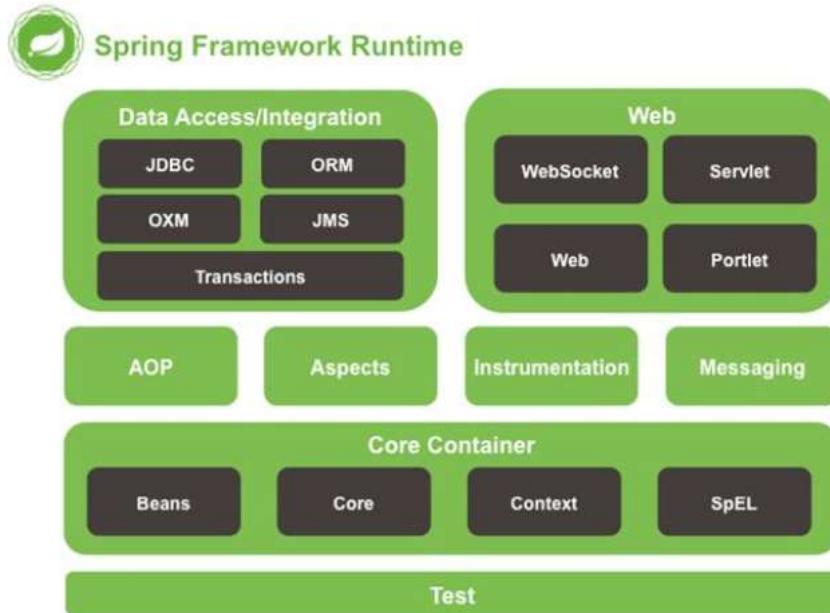
3、列举 Spring Framework 的优点。

由于 Spring Framework 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。

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。

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 Core
- || Spring Bean
- SpEL (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：

- Web
- Web – Servlet
- Web – Socket
- Web – 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。

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。

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。

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)。

9、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。

10、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。

11、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。

12、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

13、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。

14、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

15、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。

16、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。

实例：

```
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[] args) {
        Fruit f=Factory.getInstance("io.github.dunwu.spring.Apple");
        if(f!=null){
            f.eat();
        }
    }
}
```

17、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。

18、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();
    }
}

```

19、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 globalsession 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。

20、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 destroy()。
- 9、如果为 bean 指定了 destroy 方法（的 destroy-method 属性），那么将调用它。

21、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 或 中提供了 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>

```

22、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。

23、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。

24、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。

25、什么是基于注解的容器配置

不使用 XML 来描述 bean 装配，开发人员通过在相关的类，方法或字段声明上使用注解将配置移动到组件类本身。它可以作为 XML 设置的替代方案。例如：Spring 的 Java 配置是通过使用 @Bean 和 @Configuration 来实现。 @Bean 注解扮演与 元素相同的角色。

@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();  
    }  
}
```

26、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。

27、@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。

28、@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;  
    }  
}
```

29、@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;  
    }  
}
```

30、@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.java

```
public 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());  
    }  
}
```

31、@RequestMapping 注解有什么用？

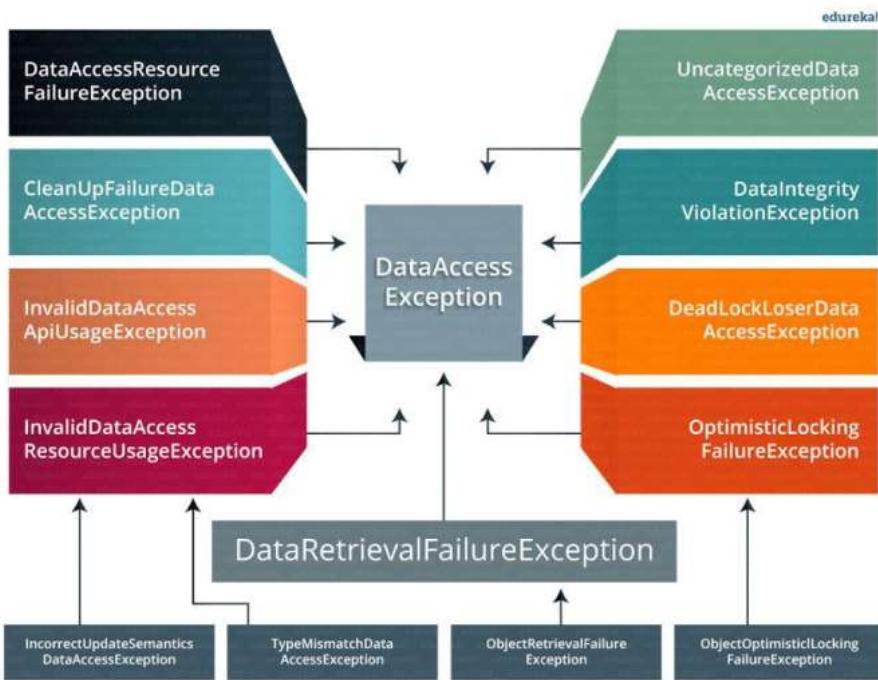
@RequestMapping 注解用于将特定 HTTP 请求方法映射到将处理相应请求的控制器中的特定类/方法。此注释可应用于两个级别：

类级别：映射请求的 URL 方法级别：映射 URL 以及 HTTP 请求方法

32、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。

33、列举 Spring DAO 抛出的异常。



34、spring JDBC API 中存在哪些类？

- JdbcTemplate
- SimpleJdbcTemplate
- NamedParameterJdbcTemplate
- SimpleJdbcInsert
- SimpleJdbcCall

35、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

36、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。

37、spring 支持哪些 ORM 框架

- Hibernate
- iBatis
- JPA
- JDO
- OJB

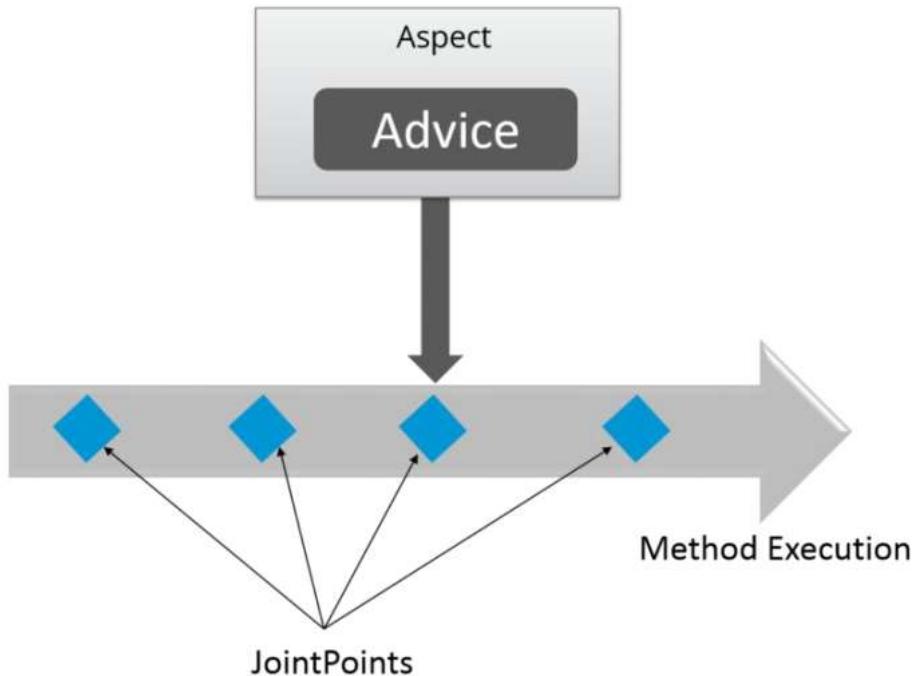
38、什么是 AOP？

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)

39、什么是 Aspect？

Aspect 由 pointcut 和 advice 组成, 它既包含了横切逻辑的定义, 也包括了连接点的定义. Spring AOP 就是负责实施切面的框架, 它将切面所定义的横切逻辑编织到切面所指定的连接点中. AOP 的工作重心在于如何将增强编译目标对象的连接点上, 这里包含两个工作:

- 1、如何通过 pointcut 和 advice 定位到特定的 joinpoint 上
- 2、如何在 advice 中编写切面代码.



以简单地认为, 使用 @Aspect 注解的类就是切面.

40、什么是切点 (JoinPoint)

程序运行中的一些时间点, 例如一个方法的执行, 或者是一个异常的处理. 在 Spring AOP 中, join point 总是方法的执行点。

41、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。

42、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。

43、指出在 spring aop 中 concern 和 cross-cuttingconcern 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

44、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

45、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。SpringAOP 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。

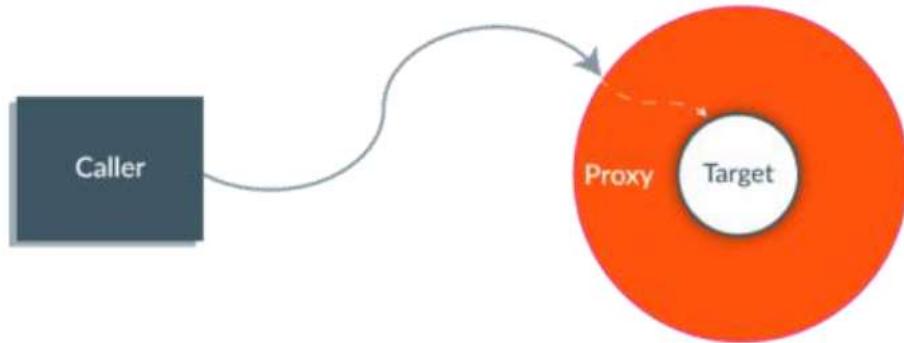
46、如何理解 Spring 中的代理？

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

47、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：

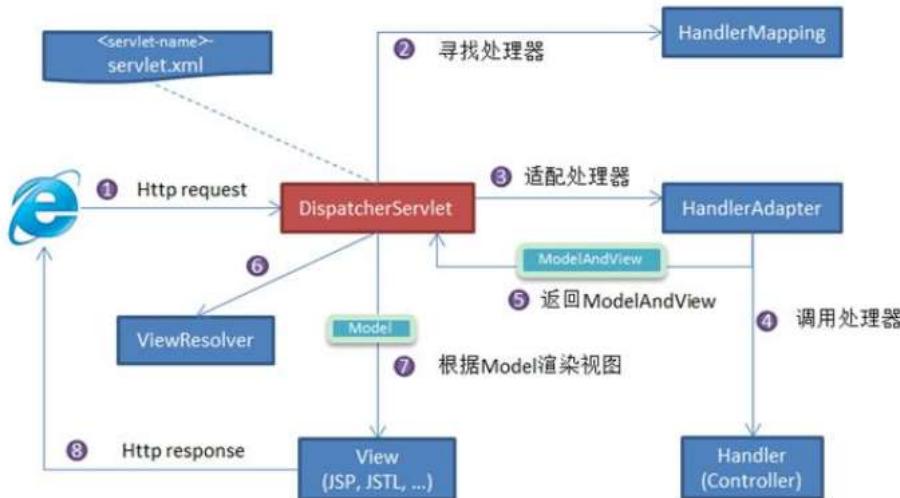


48、Spring MVC 框架有什么用？

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。

49、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：

 □ HttpMessageConverte : 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。

 □ 数据格式化 : 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。

50、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。

英文原文链接：

<https://www.edureka.co/blog/interview-questions/spring-interview-questions/>

51、什么是 spring?

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。

52、使用 Spring 框架的好处是什么？

- 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 或 JDO 抛出的）转化为一致的 unchecked 异常。

53、Spring 由哪些模块组成？

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

54、Spring的IOC和AOP机制

我们是在使用Spring框架的过程中，其实就是为了使用IOC，依赖注入，和AOP，面向切面编程，这两个是Spring的灵魂。

主要用到的设计模式有工厂模式和代理模式。

IOC就是典型的工厂模式，通过sessionfactory去注入实例。

AOP就是典型的代理模式的体现。

代理模式是常用的java设计模式，他的特征是代理类与委托类有同样的接口，代理类主要负责为委托类预处理消息、过滤消息、把消息转发给委托类，以及事后处理消息等。代理类与委托类之间通常会存在关联关系，一个代理类的对象与一个委托类的对象关联，代理类的对象本身并不真正实现服务，而是通过调用委托类的对象的相关方法，来提供特定的服务

spring的IoC容器是spring的核心，spring AOP是spring框架的重要组成部分。

在传统的程序设计中，当调用者需要被调用者的协助时，通常由调用者来创建被调用者的实例。但在spring里创建被调用者的工作不再由调用者来完成，因此控制反转（IoC）；创建被调用者实例的工作通常由spring容器来完成，然后注入调用者，因此也被称为依赖注入（DI），依赖注入和控制反转是同一个概念。

面向方面编程（AOP）是以另一个角度来考虑程序结构，通过分析程序结构的关注点来完善面向对象编程（OOP）。OOP将应用程序分解成各个层次的对象，而AOP将程序分解成多个切面。spring AOP 只实现了方法级别的连接点，在J2EE应用中，AOP拦截到方法级别的操作就已经足够。在spring中，未来使IoC方便地使用健壮、灵活的企业服务，需要利用spring AOP实现为IoC和企业服务之间建立联系。

IOC:控制反转也叫依赖注入。利用了工厂模式

将对象交给容器管理，你只需要在spring配置文件总配置相应的bean，以及设置相关的属性，让spring容器来生成类的实例对象以及管理对象。在spring容器启动的时候，spring会把你配置在配置文件中的bean都初始化好，然后在你需要调用的时候，就把它已经初始化好的那些bean分配给你需要调用这些bean的类（假设这个类名是A），分配的方法就是调用A的setter方法来注入，而不需要你在A里面new这些bean了。

注意：面试的时候，如果有条件，画图，这样更加显得你懂了

AOP:面向切面编程。（Aspect-Oriented Programming）

AOP可以说是对OOP的补充和完善。OOP引入封装、继承和多态性等概念来建立一种对象层次结构，用以模拟公共行为的一个集合。当我们需要为分散的对象引入公共行为的时候，OOP则显得无能为力。也就是说，OOP允许你定义从上到下的关系，但并不适合定义从左到右的关系。例如日志功能。日志代码往往水平地散布在所有对象层次中，而与它所散布到的对象的核心功能毫无关系。在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。将程序中的交叉业务逻辑（比如安全，日志，事务等），封装成一个切面，然后注入到目标对象（具体业务逻辑）中去。

实现AOP的技术，主要分为两大类：一是采用动态代理技术，利用截取消息的方式，对该消息进行装饰，以取代原有对象行为的执行；二是采用静态织入的方式，引入特定的语法创建“方面”，从而使得编译器可以在编译期间织入有关“方面”的代码。

简单点解释，比方说你想在你的biz层所有类中都加上一个打印‘你好的功能，这时就可以用aop思想来做。你先写个类写个类方法，方法经实现打印‘你好’，然后ioc这个类 ref = “biz.*”让每个类都注入即可实现。

55、Spring中Autowired和Resource关键字的区别

@Resource和@Autowired都是做bean的注入时使用，其实@Resource并不是Spring的注解，它的包是javax.annotation.Resource，需要导入，但是Spring支持该注解的注入。

1、共同点

两者都可以写在字段和setter方法上。两者如果都写在字段上，那么就不需要再写setter方法。

2、不同点

(1) @Autowired

@Autowired为Spring提供的注解，需要导入包
org.springframework.beans.factory.annotation.Autowired;只按照byType注入

```
public class TestServiceImpl {  
    // 下面两种@Autowired只要使用一种即可  
    @Autowired  
    private UserDao userDao; // 用于字段上  
    @Autowired  
    public void setUserDao(UserDao userDao) { // 用于属性的方法上  
        this.userDao = userDao;  
    }  
}
```

@Autowired注解是按照类型（byType）装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许null值，可以设置它的required属性为false。如果我们想使用按照名称（byName）来装配，可以结合@Qualifier注解一起使用。如下：

```
public class TestServiceImpl {  
    @Autowired  
    @Qualifier("userDao")  
    private UserDao userDao;  
}
```

(2) @Resource

@Resource默认按照ByName自动注入，由J2EE提供，需要导入包javax.annotation.Resource。@Resource有两个重要的属性：name和type，而Spring将@Resource注解的name属性解析为bean的名字，而type属性则解析为bean的类型。所以，如果使用name属性，则使用ByName的自动注入策略，而使用type属性时则使用byType自动注入策略。如果既不制定name也不制定type属性，这时将通过反射机制使用ByName自动注入策略。

```
public class TestServiceImpl {  
    // 下面两种@Resource只要使用一种即可  
    @Resource(name="userDao")  
    private UserDao userDao; // 用于字段上  
    @Resource(name="userDao")  
    public void setUserDao(UserDao userDao) { // 用于属性的setter方法上  
        this.userDao = userDao;  
    }  
}
```

注：最好是将@Resource放在setter方法上，因为这样更符合面向对象的思想，通过set、get去操作属性，而不是直接去操作属性。

@Resource装配顺序：

①如果同时指定了name和type，则从Spring上下文中找到唯一匹配的bean进行装配，找不到则抛出异常。

②如果指定了name，则从上下文中查找名称（id）匹配的bean进行装配，找不到则抛出异常。

③如果指定了type，则从上下文中找到类似匹配的唯一bean进行装配，找不到或是找到多个，都会抛出异常。

④如果既没有指定name，又没有指定type，则自动按照byName方式进行装配；如果没有匹配，则回退为一个原始类型进行匹配，如果匹配则自动装配。@Resource的作用相当于@Autowired，只不过@Autowired按照byType自动注入。

56、依赖注入的方式有几种，各是什么？

一、构造器注入

将被依赖对象通过构造函数的参数注入给依赖对象，并且在初始化对象的时候注入。

优点：

对象初始化完成后便可获得可使用的对象。

缺点：

当需要注入的对象很多时，构造器参数列表将会很长；不够灵活。若有多种注入方式，每种方式只需注入指定几个依赖，那么就需要提供多个重载的构造函数，麻烦。

二、setter方法注入

IoC Service Provider通过调用成员变量提供的setter函数将被依赖对象注入给依赖类。

优点：

灵活。可以选择性地注入需要的对象。

缺点：

依赖对象初始化完成后由于尚未注入被依赖对象，因此还不能使用。

三、接口注入

依赖类必须要实现指定的接口，然后实现该接口中的一个函数，该函数就是用于依赖注入。该函数的参数就是要注入的对象

优点

接口注入中，接口的名字、函数的名字都不重要，只要保证函数的参数是要注入的对象类型即可。

缺点：

侵入行太强，不建议使用。

PS：什么是侵入行？

如果类A要使用别人提供的一个功能，若为了使用这功能，需要在自己的类中增加额外的代码，这就是侵入性

57、讲一下什么是Spring

Spring是一个轻量级的IoC和AOP容器框架。是为Java应用程序提供基础性服务的一套框架，目的是用于简化企业应用程序的开发，它使得开发者只需要关心业务需求。常见的配置方式有三种：基于XML的配置、基于注解的配置、基于Java的配置。

主要由以下几个模块组成：

Spring Core：核心类库，提供IOC服务；

Spring Context：提供框架式的Bean访问方式，以及企业级功能（JNDI、定时任务等）；

Spring AOP：AOP服务；

Spring DAO：对JDBC的抽象，简化了数据访问异常的处理

Spring ORM：对现有的ORM框架的支持；

Spring Web：提供了基本的面向Web的综合特性，例如多方文件上传；

Spring MVC：提供面向Web应用的Model-View-Controller实现。

58、Spring MVC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、处理器映射器找到具体的处理器(可以根据xml配置、注解进行查找)，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。

4、DispatcherServlet调用HandlerAdapter处理器适配器。

5、HandlerAdapter经过适配调用具体的处理器(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ViewResolver视图解析器。

9、ViewResolver解析后返回具体View。

10、DispatcherServlet根据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

组件说明：

以下组件通常使用框架提供实现：

DispatcherServlet：作为前端控制器，整个流程控制的中心，控制其它组件执行，统一调度，降低组件之间的耦合性，提高每个组件的扩展性

HandlerMapping：通过扩展处理器映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

HandlerAdapter：通过扩展处理器适配器，支持更多类型的处理器。

ViewResolver：通过扩展视图解析器，支持更多类型的视图解析，例如：jsp、freemarker、pdf、excel等。

组件：

1、前端控制器DispatcherServlet（不需要工程师开发），由框架提供

作用：接收请求，响应结果，相当于转发器，中央处理器。有了dispatcherServlet减少了其它组件之间的耦合度。

用户请求到达前端控制器，它就相当于mvc模式中的c，dispatcherServlet是整个流程控制的中心，由它调用其它组件处理用户的请求，dispatcherServlet的存在降低了组件之间的耦合性。

2、处理器映射器HandlerMapping（不需要工程师开发），由框架提供

作用：根据请求的url查找Handler

HandlerMapping负责根据用户请求找到Handler即处理器，springmvc提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

3、处理器适配器HandlerAdapter

作用：按照特定规则（HandlerAdapter要求的规则）去执行Handler

通过HandlerAdapter对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

4、处理器Handler（需要工程师开发）

注意：编写Handler时按照HandlerAdapter的要求去做，这样适配器才可以去正确执行Handler

Handler是继DispatcherServlet前端控制器的后端控制器，在DispatcherServlet的控制下Handler对具体的用户请求进行处理。由于Handler涉及到具体的用户业务请求，所以一般情况需要工程师根据业务需求开发Handler。

5、视图解析器View resolver（不需要工程师开发），由框架提供

作用：进行视图解析，根据逻辑视图名解析成真正的视图（view）

View Resolver负责将处理结果生成View视图，View Resolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。springmvc框架提供了很多的View视图类型，包括：jstlView、freemarkerView、pdfView等。一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由工程师根据业务需求开发具体的页面。

6、视图View（需要工程师开发jsp...）

View是一个接口，实现类支持不同的View类型（jsp、freemarker、pdf...）核心架构的具体流程步骤如下：

1、首先用户发送请求——>DispatcherServlet，前端控制器收到请求后自己不进行处理，而是委托给其他的解析器进行处理，作为统一访问点，进行全局的流程控制；

- 2、DispatcherServlet——>HandlerMapping，HandlerMapping 将会把请求映射为HandlerExecutionChain 对象（包含一个Handler 处理器（页面控制器）对象、多个HandlerInterceptor 拦截器）对象，通过这种策略模式，很容易添加新的映射策略；
- 3、DispatcherServlet——>HandlerAdapter，HandlerAdapter 将会把处理器包装为适配器，从而支持多种类型的处理器，即适配器设计模式的应用，从而很容易支持很多类型的处理器；
- 4、HandlerAdapter——>处理器功能处理方法的调用，HandlerAdapter 将会根据适配的结果调用真正的处理器的功能处理方法，完成功能处理；并返回一个ModelAndView 对象（包含模型数据、逻辑视图名）；
- 5、 ModelAndView的逻辑视图名——> ViewResolver，ViewResolver 将把逻辑视图名解析为具体的View，通过这种策略模式，很容易更换其他视图技术；
- 6、View——>渲染，View会根据传进来的Model模型数据进行渲染，此处的Model实际是一个Map数据结构，因此很容易支持其他视图技术；
- 7、返回控制权给DispatcherServlet，由DispatcherServlet返回响应给用户，到此一个流程结束。下边两个组件通常情况下需要开发：
Handler：处理器，即后端控制器用controller表示。
View：视图，即展示给用户的界面，视图中通常需要标签语言展示模型数据。

59、springMVC是什么

springMVC是一个MVC的开源框架，springMVC=struts2+spring，springMVC就相当于是Struts2加上spring的整合，但是这里有一个疑惑就是，springMVC和spring是什么样的关系呢？这个在百度百科上有一个很好的解释：意思是说，springMVC是spring的一个后续产品，其实就是spring在原有基础上，又提供了web应用的MVC模块，可以简单的把springMVC理解为是spring的一个模块（类似AOP，IOC这样的模块），网络上经常会说springMVC和spring无缝集成，其实springMVC就是spring的一个子模块，所以根本不需要同spring进行整合

60、SpringMVC怎么样设定重定向和转发的？

- (1) 转发：在返回值前面加“forward：“，譬如“forward:user.do?name=method4”
- (2) 重定向：在返回值前面加“redirect：“，譬如“redirect:<http://www.baidu.com>”

61、SpringMVC常用的注解有哪些

@RequestMapping：用于处理请求 url 映射的注解，可用于类或方法上。用于类上，则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。
@RequestBody：注解实现接收http请求的json数据，将json转换为java对象。
@ResponseBody：注解实现将controller方法返回对象转化为json对象响应给客户

62、Spring的AOP理解

OOP面向对象，允许开发者定义纵向的关系，但并不适用于定义横向的关系，导致了大量代码的重复，而不利于各个模块的重用。
AOP，一般称为面向切面，作为面向对象的一种补充，用于将那些与业务无关，但却对多个对象产生影响的公共行为和逻辑，抽取并封装为一个可重用的模块，这个模块被命名为“切面”（Aspect），减少系统中的重复代码，降低了模块间的耦合度，同时提高了系统的可维护性。
可用于权限认证、日志、事务处理。
AOP实现的关键在于 代理模式，AOP代理主要分为静态代理和动态代理。静态代理的代表为AspectJ；动态代理则以Spring AOP为代表。
(1) AspectJ是静态代理的增强，所谓静态代理，就是AOP框架会在编译阶段生成AOP代理类，因此也称为编译时增强，它会在编译阶段将AspectJ(切面)织入到java字节码中，运行的时候就是增强之后的AOP对象。
(2) Spring AOP使用的动态代理，所谓的动态代理就是说AOP框架不会去修改字节码，而是每次运行时在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，并回调原对象的方法。

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：

①JDK动态代理只提供接口的代理，不支持类的代理。核心InvocationHandler接口和Proxy类，InvocationHandler 通过invoke()方法反射来调用目标类中的代码，动态地将横切逻辑和业务编织在一起；接着，Proxy利用 InvocationHandler动态创建一个符合某一接口的实例，生成目标类的代理对象。
②如果代理类没有实现 InvocationHandler 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指定类的一个子类对象，并覆盖其中特定方法并添加增强代码，从而实现AOP。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
③静态代理与动态代理区别在于生成AOP代理对象的时机不同，相对来说AspectJ的静态代理方式具有更好的性能，但是AspectJ需要特定的编译器进行处理，而Spring AOP则无需特定的编译器处理。

63、Spring的IOC理解

1) IOC就是控制反转，是指创建对象的控制权的转移，以前创建对象的主动权和时机是由自己把控的，而现在这种权力转移到Spring容器中，并由容器根据配置文件去创建实例和管理各个实例之间的依赖关系，对象与对象之间松散耦合，也利于功能的复用。DI依赖注入，和控制反转是同一个概念的不同角度的描述，即 应用程序在运行时依赖IoC容器来动态注入对象需要的外部资源。
(2) 最直观的表达就是，IOC让对象的创建不用去new了，可以由spring自动生成，使用java的反射机制，根据配置文件在运行时动态的去创建对象以及管理对象，并调用对象的方法的。
(3) Spring的IOC有三种注入方式：构造器注入、setter方法注入、根据注解注入。

IoC让相互协作的组件保持松散的耦合，而AOP编程允许你把遍布于应用各层的功能分离出来形成可重用的功能组件

64、解释一下spring bean的生命周期

首先说一下Servlet的生命周期：实例化，初始init，接收请求service，销毁destroy；Spring上下文中的Bean生命周期也类似，如下：

(1) 实例化Bean：

对于BeanFactory容器，当客户向容器请求一个尚未初始化的bean时，或初始化bean的时候需要注入另一个尚未初始化的依赖时，容器就会调用createBean进行实例化。对于ApplicationContext容器，当容器启动结束后，通过获取BeanDefinition对象中的信息，实例化所有的bean。

(2) 设置对象属性（依赖注入）：

实例化后的对象被封装在BeanWrapper对象中，紧接着，Spring根据BeanDefinition中的信息以及通过BeanWrapper提供的设置属性的接口完成依赖注入。

(3) 处理Aware接口：

接着，Spring会检测该对象是否实现了xxxAware接口，并将相关的xxxAware实例注入给Bean：

①如果这个Bean已经实现了BeanNameAware接口，会调用它实现的setBeanName(String beanId)方法，此处传递的就是Spring配置文件中Bean的id值；

②如果这个Bean已经实现了BeanFactoryAware接口，会调用它实现的setBeanFactory()方法，传递的是Spring工厂自身。

③如果这个Bean已经实现了ApplicationContextAware接口，会调用setApplicationContext(ApplicationContext)方法，传入Spring上下文；

(4) BeanPostProcessor：

如果想对Bean进行一些自定义的处理，那么可以让Bean实现了BeanPostProcessor接口，那将会调用postProcessBeforeInitialization(Object obj, String s)方法。

(5) InitializingBean 与 init-method：

如果Bean在Spring配置文件中配置了init-method属性，则会自动调用其配置的初始化方法。

(6) 如果这个Bean实现了BeanPostProcessor接口，将会调用postProcessAfterInitialization(Object obj, String s)方法；由于这个方法是在Bean初始化结束时调用的，所以可以被应用于内存或缓存技术

以上几个步骤完成后，Bean就已经被正确创建了，之后就可以使用这个Bean了

7) DisposableBean：

当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean这个接口，会调用其实现的destroy()方法；

(8) destroy-method：

最后，如果这个Bean的Spring配置中配置了destroy-method属性，会自动调用其配置的销毁方法

65、解释Spring支持的几种bean的作用域。

Spring容器中的bean可以分为5个范围：

(1) singleton：默认，每个容器中只有一个bean的实例，单例的模式由BeanFactory自身来维护。

(2) prototype：为每一个bean请求提供一个实例。

(3) request：为每一个网络请求创建一个实例，在请求完成以后，bean会失效并被垃圾回收器回收。

(4) session：与request范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效。

(5) global-session：全局作用域，global-session和Portlet应用相关。当你的应用部署在Portlet容器中工作时，它包含很多portlet。如果你想要声明让所有的portlet共用全局的存储变量的话，那么这全局变量需要存储在global-session中。全局作用域与Servlet中的session作用域效果相同

66、Spring基于xml注入bean的几种方式

1) Set方法注入；

(2) 构造器注入：①通过index设置参数的位置；②通过type设置参数类型；

(3) 静态工厂注入；

(4) 实例工厂；

详细内容可以阅读：<https://blog.csdn.net/a745233700/article/details/89307518>

67、Spring框架中都用到了哪些设计模式

(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：定义对象键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

68、核心容器（应用上下文）模块

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。

69、BeanFactory – BeanFactory实现举例。

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。

70、XMLBeanFactory

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。

71、解释 AOP 模块

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。

72、解释 JDBC 抽象和 DAO 模块。

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。

72、解释对象/关系映射集成模块。

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(ORM)工具，Spring 支持集成主流的 ORM 框架，如 Hibernate,JDO 和 iBATISSQL Maps。Spring 的事务管理同样支持以上所有 ORM 框架及 JDBC。

73、解释 WEB 模块。

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。

74、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

75、什么是 Spring IOC 容器？

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。

76、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。

77、ApplicationContext 通常的实现是什么？

- 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。

78、Bean 工厂和 Application contexts 有什么区别？

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。

79、一个 Spring 的应用看起来象什么？

- 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。

80、什么是 Spring 的依赖注入？

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。

81、有哪些不同类型的 IOC (依赖注入) 方式 ?

- 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。

82、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

83、什么是 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 ” , 如果它被赋为 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。

84、一个 Spring Bean 定义包含什么 ?

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。

85、如何给 Spring 容器提供配置元数据 ?

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。

86、你怎样定义类的作用域 ?

当定义一个在 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 。

87、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 :

- 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 。

88、Spring 框架中的单例 bean 是线程安全的吗 ?

不 , Spring 框架中的单例 bean 不是线程安全的。

89、解释 Spring 框架中 bean 的生命周期。

- 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 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。
- 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

90、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) 。

91、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 或 元素内使用 元素，内部 bean 通常是匿名的，它们的 Scope 一般是 prototype。

92、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：

- 类型用于注入一列值，允许有相同的值。
- 类型用于注入一组值，不允许有相同的值。
- 类型用于注入一组键值对，键和值都可以为任意类型。
- 类型用于注入一组键值对，键和值都只能为 String 类型。

93、什么是 bean 装配?

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。

94、什么是 bean 的自动装配 ?

Spring 容器能够自动装配相互合作的 bean，这意味着容器不需要手动配置，能通过 Bean 工厂自动处理 bean 之间的协作。

95、解释不同方式的自动装配 。

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。

96、自动装配有哪些局限性

自动装配的局限性是：

- 重写：你仍需用 和 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。

97、你可以在 Spring 中注入一个 null 和一个空字符串吗 ?

可以

98、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。

99、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，直接组件类中进行配置，而不是使用 xml 表达 bean 的装配关系。

100、怎样开启注解装配 ?

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

101、@Required 注解

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。

102、@Autowired 注解

@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。

103、@Qualifier 注解

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@Autowired 注解结合使用以消除这种混淆，指定需要装配的确切的 bean。

104、在 Spring 框架中如何更有效地使用 JDBC？

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（例子见[这里](#)here）

105、JdbcTemplate

JdbcTemplate 类提供了许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。

106、Spring 对 DAO 的支持

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。

107、使用 Spring 通过什么方式访问 Hibernate？

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。

108、Spring 支持的 ORM

Spring 支持以下 ORM：

- Hibernate
- || iBatis
- JPA (Java Persistence API)
- TopLink
- JDO (Java Data Objects)
- OJB

109、如何通过 HibernateDaoSupport 将 Spring 和 Hibernate 结合起来？

用 Spring 的 SessionFactory 调用 LocalSessionFactory。集成过程分三步：

- 配置 the Hibernate SessionFactory。
- 继承 HibernateDaoSupport 实现一个 DAO。
- 在 AOP 支持的事务中装配。

110、Spring 支持的事务管理类型

Spring 支持两种类型的事务管理：

- 编程式事务管理：这意味着你通过编程的方式管理事务，给你带来极大的灵活性，但是难维护。
- 声明式事务管理：这意味着你可以将业务代码和事务管理分离，你只需用注解和 XML 配置来管理事务。

111、Spring 框架的事务管理有哪些优点？

□ 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。

 || 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

 如

 □ 它支持声明式事务管理。

 || 它和 Spring 各种数据访问抽象层很好得集成。

112、你更倾向用那种事务管理类型？

大多数 Spring 框架的用户选择声明式事务管理，因为它对应用代码的影响最小，因此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。

113、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。

114、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。

115、在 Spring AOP 中，关注点和横切关注的区别是什么？

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。

116、连接点

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。

117、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。

118、切点

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。

119、什么是引入？

引入允许我们在已存在的类中增加新的方法和属性。

120、什么是目标对象？

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。

121、什么是代理？

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。

122、有几种不同类型的自动代理？

BeanNameAutoProxyCreator
DefaultAdvisorAutoProxyCreator
Metadata autoproxying

123、什么是织入。什么是织入应用的不同点？

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。

124、解释基于 XML Schema 方式的切面实现。

在这种情况下，切面由常规类以及基于 XML 的配置实现。

125、解释基于注解的切面实现

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

126、什么是 Spring 的 MVC 框架？

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。

127、DispatcherServlet

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。

128、WebApplicationContext

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。

129、什么是 Spring MVC 框架的控制器？

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。

130、@Controller 注解

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。

131、@RequestMapping 注解

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

Spring Boot面试题

1、什么是 Spring Boot ?

多年来，随着新功能的增加，spring 变得越来越复杂。只需访问<https://spring.io/projects> 页面，我们就会看到可以在我们的应用程序中使用的所有 Spring 项目的不同功能。如果必须启动一个新的 Spring 项目，我们必须添加构建路径或添加 Maven 依赖关系，配置应用程序服务器，添加 spring 配置。因此，开始一个新的 spring 项目需要很多努力，因为我们现在必须从头开始做所有事情。

Spring Boot 是解决这个问题的方法。Spring Boot 已经建立在现有 spring 框架之上。使用 spring 启动，我们避免了之前我们必须做的所有样板代码和配置。因此，Spring Boot 可以帮助我们以最少的工作量，更加健壮地使用现有的 Spring 功能。

2、为什么要用SpringBoot

Spring Boot 优点非常多，如：

一、独立运行

Spring Boot 而且内嵌了各种servlet容器，Tomcat、Jetty等，现在不再需要打成war包部署到容器中，Spring Boot只要打成一个可执行的jar包就能独立运行，所有的依赖包都在一个jar包内。

二、简化配置

spring-boot-starter-web启动器自动依赖其他组件，简少了maven的配置。

三、自动配置

Spring Boot能根据当前类路径下的类、jar包来自动配置bean，如添加一个spring-boot-starter-web启动器就能拥有web的功能，无需其他配置。

四、无代码生成和XML配置

Spring Boot配置过程中无代码生成，也无需XML配置文件就能完成所有配置工作，这一切都是借助于条件注解完成的，这也是Spring4.x的核心功能之一。

五、应用监控

Spring Boot提供一系列端点可以监控服务及应用，做健康检测

3、Spring Boot 有哪些优点？

Spring Boot 的优点有：

- 1、减少开发，测试时间和努力。
- 2、使用 JavaConfig 有助于避免使用 XML。
- 3、避免大量的 Maven 导入和各种版本冲突。
- 4、提供意见发展方法。
- 5、通过提供默认值快速开始开发。
- 6、没有单独的 Web 服务器需要。这意味着你不再需要启动 Tomcat，Glassfish或其他任何东西。
- 7、需要更少的配置 因为没有 web.xml 文件。只需添加用@ Configuration 注释的类，然后添加用@Bean 注释的方法，Spring 将自动加载对象并像以前一样对其进行管理。您甚至可以将@Autowired 添加到 bean 方法中，以使 Spring 自动装入需要的依赖关系中。
- 8、基于环境的配置 使用这些属性，您可以将您正在使用的环境传递到应用程序：-Dspring.profiles.active = {environment}。在加载主应用程序属性文件后，Spring 将在 (application{environment}.properties) 中加载后续的应用程序属性文件。

4、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

5、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

6、如何理解 Spring Boot 中的 Starters ?

Starters是什么：

Starters可以理解为启动器，它包含了一系列可以集成到应用里面的依赖包，你可以一站式集成Spring及其他技术，而不需要到处找示例代码和依赖包。如你想使用Spring JPA访问数据库，只要加入springboot-starter-data-jpa启动器依赖就能使用了。Starters包含了许多项目中需要用到的依赖，它们能快速持续的运行，都是一系列得到支持的管理传递性依赖。

Starters命名：

Spring Boot官方的启动器都是以spring-boot-starter-命名的，代表了一个特定的应用类型。第三方的启动器不能以spring-boot开头命名，它们都被Spring Boot官方保留。一般一个第三方的应该这样命名，像mybatis的mybatis-spring-boot-starter。

Starters分类：

1. Spring Boot应用类启动器

启动器名称	功能描述
spring-boot-starter	包含自动配置、日志、YAML的支持。
spring-boot-starter-web	使用Spring MVC构建web 工程，包含restful，默认使用Tomcat容器。
...	...

https://blog.csdn.net/Kevin_Gu6

2.Spring Boot生产启动器

启动器名称	功能描述
spring-boot-starter-actuator	提供生产环境特性，能监控管理应用。

3. Spring Boot技术类启动器

启动器名称	功能描述
spring-boot-starter-json	提供对JSON的读写支持。
spring-boot-starter-logging	默认的日志启动器，默认使用Logback。
...	...

4. 其他第三方启动器

7、如何在Spring Boot启动的时候运行一些特定的代码？

如果你想在Spring Boot启动的时候运行一些特定的代码，你可以实现接口ApplicationRunner或者CommandLineRunner，这两个接口实现方式一样，它们都只提供了一个run方法。CommandLineRunner：启动获取命令行参数

8、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

9、Spring Boot中的监视器是什么？

Spring boot actuator是spring启动框架中的重要功能之一。Spring boot监视器可以帮助您访问生产环境中正在运行的应用程序的当前状态。有几个指标必须在生产环境中进行检查和监控。即使一些外部应用程序可能正在使用这些服务来向相关人员触发警报消息。监视器模块公开了一组可直接作为HTTP URL访问的REST端点来检查状态

10、如何使用Spring Boot实现异常处理？

Spring提供了一种使用ControllerAdvice处理异常的非常有用的方法。我们通过实现一个ControllerAdvice类，来处理控制器类抛出的所有异常

11、你如何理解 Spring Boot 中的 Starters

Starters可以理解为启动器，它包含了一系列可以集成到应用里面的依赖包，你可以一站式集成Spring及其他技术，而不需要到处找示例代码和依赖包。如你想使用Spring JPA访问数据库，只要加入spring-boot-starter-data-jpa启动器依赖就能使用了

12、springboot常用的starter有哪些

spring-boot-starter-web 嵌入tomcat和web开发需要servlet与jsp支持
 spring-boot-starter-data-jpa 数据库支持
 spring-boot-starter-data-redis redis数据库支持
 spring-boot-starter-data-solr solr支持
 mybatis-spring-boot-starter 第三方的mybatis集成starter

13、SpringBoot 实现热部署有哪几种方式

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

14、如何理解 Spring Boot 配置加载顺序

在Spring Boot里面，可以使用以下几种方式来加载配置。

- 1) properties文件；
 - 2) YAML文件；
 - 3) 系统环境变量；
 - 4) 命令行参数；
- 等等.....

15、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。

1. 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；
2. 一些固定的不能被覆盖的属性；
3. 一些加密/解密的场景

16、如何集成 Spring Boot 和 ActiveMQ

对于集成 Spring Boot 和 ActiveMQ，我们使用spring-boot-starter-activemq依赖关系。它只需要很少的配置，并且不需要样板代码

17、什么是 JavaConfig？

Spring JavaConfig 是 Spring 社区的产品，它提供了配置 Spring IoC 容器的纯Java 方法。因此它有助于避免使用 XML 配置。使用 JavaConfig 的优点在于：

- 1、面向对象的配置。由于配置被定义为 JavaConfig 中的类，因此用户可以充分利用 Java 中的面向对象功能。一个配置类可以继承另一个，重写它的@Bean 方法等。
- 2、减少或消除 XML 配置。基于依赖注入原则的外化配置的好处已被证明。但是，许多开发人员不希望在 XML 和 Java 之间来回切换。JavaConfig 为开发人员提供了一种纯 Java 方法来配置与 XML 配置概念相似的 Spring 容器。从技术角度来讲，只使用 JavaConfig 配置类来配置容器是可行的，但实际上很多人认为将JavaConfig 与 XML 混合匹配是理想的。
- 3、类型安全和重构友好。JavaConfig 提供了一种类型安全的方法来配置 Spring 容器。由于 Java 5.0 对泛型的支持，现在可以按类型而不是按名称检索 bean，不需要任何强制转换或基于字符串的查找。

18、如何重新加载 Spring Boot 上的更改，而无需重新启动服务器？

这可以使用 DEV 工具来实现。通过这种依赖关系，您可以节省任何更改，嵌入式tomcat 将重新启动。Spring Boot 有一个开发工具（DevTools）模块，它有助于提高开发人员的生产力。Java 开发人员面临的一个主要挑战是将文件更改自动部署到服务器并自动重启服务器。开发人员可以重新加载 Spring Boot 上的更改，而无需重新启动服务器。这将消除每次手动部署更改的需要。Spring Boot 在发布它的第一个版本时没有这个功能。这是开发人员最需要的功能。DevTools 模块完全满足开发人员的需求。该模块将在生产环境中被禁用。它还提供 H2 数据库控制台以更好地测试应用程序。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
</dependency>
```

19、Spring Boot 中的监视器是什么？

Spring boot actuator 是 spring 启动框架中的重要功能之一。Spring boot 监视器可帮助您访问生产环境中正在运行的应用程序的当前状态。有几个指标必须在生产环境中进行检查和监控。即使一些外部应用程序可能正在使用这些服务来向相关人员触发警报消息。监视器模块公开了一组可直接作为 HTTP URL 访问的REST 端点来检查状态。

20、如何在 Spring Boot 中禁用 Actuator 端点安全性？

默认情况下，所有敏感的 HTTP 端点都是安全的，只有具有 ACTUATOR 角色的用户才能访问它们。安全性是使用标准的 HttpServletRequest.isUserInRole 方法实施的。我们可以使用来禁用安全性。只有在执行机构端点在防火墙后访问时，才建议禁用安全性。

21、如何在自定义端口上运行 Spring Boot 应用程序？

为了在自定义端口上运行 Spring Boot 应用程序，您可以在application.properties 中指定端口。

```
server.port = 8090
```

22、什么是 YAML？

YAML 是一种人类可读的数据序列化语言。它通常用于配置文件。

与属性文件相比，如果我们想要在配置文件中添加复杂的属性，YAML 文件就更加结构化，而且更少混淆。可以看出 YAML 具有分层配置数据。

23、如何实现 Spring Boot 应用程序的安全性？

为了实现 Spring Boot 的安全性，我们使用 spring-boot-starter-security 依赖项，并且必须添加安全配置。它只需要很少的代码。配置类将必须扩展 WebSecurityConfigurerAdapter 并覆盖其方法。

24、如何集成 Spring Boot 和 ActiveMQ？

对于集成 Spring Boot 和 ActiveMQ，我们使用
依赖关系。它只需要很少的配置，并且不需要样板代码。

25、如何使用 Spring Boot 实现分页和排序？

使用 Spring Boot 实现分页非常简单。使用 Spring Data-JPA 可以实现将可分页的传递给存储库方法。

26、什么是 Swagger？你用 Spring Boot 实现了它吗？

Swagger 广泛用于可视化 API，使用 Swagger UI 为前端开发人员提供在线沙箱。Swagger 是用于生成 RESTful Web 服务的可视化表示的工具，规范和完整框架实现。它使文档能够以与服务器相同的速度更新。当通过 Swagger 正确定义时，消费者可以使用最少量的实现逻辑来理解远程服务并与其进行交互。因此，Swagger 消除了调用服务时的猜测。

27、什么是 Spring Profiles？

Spring Profiles 允许用户根据配置文件（dev, test, prod 等）来注册 bean。因此，当应用程序在开发中运行时，只有某些 bean 可以加载，而在 PRODUCTION 中，某些其他 bean 可以加载。假设我们的要求是 Swagger 文档仅适用于 QA 环境，并且禁用所有其他文档。这可以使用配置文件来完成。Spring Boot 使得使用配置文件非常简单。

28、什么是 Spring Batch？

Spring Boot Batch 提供可重用的函数，这些函数在处理大量记录时非常重要，包括日志/跟踪，事务管理，作业处理统计信息，作业重新启动，跳过和资源管理。它还提供了更先进的技术服务和功能，通过优化和分区技术，可以实现极高批量和高性能批处理作业。简单以及复杂的大批量批处理作业可以高度可扩展的方式利用框架处理重要大量的信息。

29、什么是 FreeMarker 模板？

FreeMarker 是一个基于 Java 的模板引擎，最初专注于使用 MVC 软件架构进行动态网页生成。使用 Freemaker 的主要优点是表示层和业务层的完全分离。程序员可以处理应用程序代码，而设计人员可以处理 html 页面设计。最后使用freemarker 可以将这些结合起来，给出最终的输出页面。

30、如何使用 Spring Boot 实现异常处理？

Spring 提供了一种使用 ControllerAdvice 处理异常的非常有用的方法。我们通过实现一个 ControllerAdvice 类，来处理控制器类抛出的所有异常。

31、您使用了哪些 starter maven 依赖项？

使用了下面的一些依赖项

```
spring-boot-starter-activemq
spring-boot-starter-security
```

这有助于增加更少的依赖关系，并减少版本的冲突。

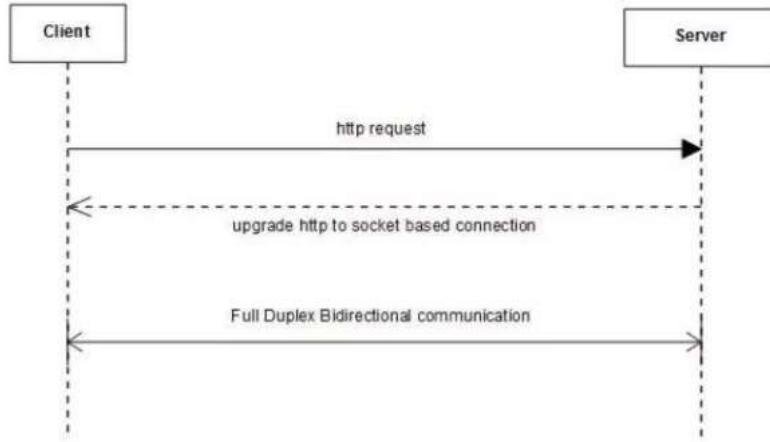
32、什么是 CSRF 攻击？

CSRF 代表跨站请求伪造。这是一种攻击，迫使最终用户在当前通过身份验证的 Web 应用程序上执行不需要的操作。CSRF 攻击专门针对状态改变请求，而不是数据窃取，因为攻击者无法查看对伪造请求的响应。

这有助于增加更少的依赖关系，并减少版本的冲突。

33、什么是 WebSockets？

WebSocket 是一种计算机通信协议，通过单个 TCP 连接提供全双工通信信道。



- 1、WebSocket 是双向的 - 使用 WebSocket 客户端或服务器可以发起消息发送。
- 2、WebSocket 是全双工的 - 客户端和服务器通信是相互独立的。
- 3、单个 TCP 连接 - 初始连接使用 HTTP，然后将此连接升级到基于套接字的连接。然后这个单一连接用于所有未来的通信
- 4、Light - 与 http 相比，WebSocket 消息数据交换要轻得多。

34、什么是 AOP ?

在软件开发过程中，跨越应用程序多个点的功能称为交叉问题。这些交叉问题与应用程序的主要业务逻辑不同。因此，将这些横切关注与业务逻辑分开是面向方面编程（AOP）的地方。

35、什么是 Apache Kafka ?

Apache Kafka 是一个分布式发布 - 订阅消息系统。它是一个可扩展的，容错的发布 - 订阅消息系统，它使我们能够构建分布式应用程序。这是一个 Apache 顶级项目。Kafka 适合离线和在线消息消费。

36、我们如何监视所有 Spring Boot 微服务？

Spring Boot 提供监视器端点以监控各个微服务的度量。这些端点对于获取有关应用程序的信息（如它们是否已启动）以及它们的组件（如数据库等）是否正常运行很有帮助。但是，使用监视器的一个主要缺点或困难是，我们必须单独打开应用程序的知识点以了解其状态或健康状况。想象一下涉及 50 个应用程序的微服务，管理员将不得不击中所有 50 个应用程序的执行终端。为了帮助我们处理这种情况，我们将使用位于的开源项目。它建立在 Spring Boot Actuator 之上，它提供了一个 Web UI，使我们能够可视化多个应用程序的度量。

37、Spring Boot 的配置文件有哪几种格式？它们有什么区别？

.properties 和 .yml，它们的区别主要是书写格式不同。

1).properties

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

2).yml

```
app:
  user:
    name: javastack
```

另外，.yml 格式不支持 `@PropertySource` 注解导入配置。

38、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

39、Spring Boot 的目录结构是怎样的？

```
cn
+- javastack
  +- MyApplication.java
  |
```

```
+-- customer
|   +- Customer.java
|   +- CustomerController.java
|   +- CustomerService.java
|   +- CustomerRepository.java
|
+-- order
    +- Order.java
    +- OrderController.java
    +- OrderService.java
    +- OrderRepository.java
```

这个目录结构是主流及推荐的做法，而在主入口类上加上 @SpringBootApplication 注解来开启 Spring Boot 的各项能力，如自动配置、组件扫描等。

40、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

41、Spring Boot 自动配置原理是什么？

注解 @EnableAutoConfiguration, @Configuration, @ConditionalOnClass 就是自动配置的核心，首先它得是一个配置文件，其次根据类路径下是否有这个类去自动配置。

42、如何在 Spring Boot 启动的时候运行一些特定的代码？

可以实现接口 ApplicationRunner 或者 CommandLineRunner，这两个接口实现方式一样，它们都只提供了一个 run 方法

43、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

44、Spring Boot 支持哪些日志框架？推荐和默认的日志框架是哪个？

Spring Boot 支持 Java Util Logging, Log4j2, Logback 作为日志框架，如果你使用 Starters 启动器，Spring Boot 将使用 Logback 作为默认日志框架

45、Spring Boot 如何定义多套不同环境配置？

提供多套配置文件，如：

```
application.properties
application-dev.properties
application-test.properties
application-prod.properties
```

运行时指定具体的配置文件

46、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。

47、保护 Spring Boot 应用有哪些方法？

- 在生产中使用 HTTPS
- 使用 Snyk 检查你的依赖关系
- 升级到最新版本
- 启用 CSRF 保护
- 使用内容安全策略防止 XSS 攻击
- ...

48、Spring Boot 2.X 有什么新特性？与 1.X 有什么区别？

- 配置变更
- JDK 版本升级
- 第三方类库升级
- 响应式 Spring 编程支持
- HTTP/2 支持
- 配置属性绑定
- 更多改进与加强...

49、如何重新加载Spring Boot上的更改，而无需重新启动服务器？

这可以使用DEV工具来实现。通过这种依赖关系，您可以节省任何更改，嵌入式tomcat将重新启动。

Spring Boot有一个开发工具（DevTools）模块，它有助于提高开发人员的生产力。Java开发人员面临的一个主要挑战是将文件更改自动部署到服务器并自动重启服务器。

开发人员可以重新加载Spring Boot上的更改，而无需重新启动服务器。这将消除每次手动部署更改的需要。Spring Boot在发布它的第一个版本时没有这个功能。

这是开发人员最需要的功能。DevTools模块完全满足开发人员的需求。该模块将在生产环境中被禁用。它还提供H2数据库控制台以更好地测试应用程序。

```
org.springframework.boot
spring-boot-devtools
true
```

50、springboot集成mybatis的过程

添加mybatis的starter maven依赖

```
org.mybatis.spring.boot
mybatis-spring-boot-starter
1.2.0
```

在mybatis的接口中 添加@Mapper注解

在application.yml配置数据源信息

51、Spring Boot、Spring MVC 和 Spring 有什么区别？

SpringFrame

SpringFramework 最重要的特征是依赖注入。所有 SpringModules 不是依赖注入就是 IOC 控制反转。

当我们恰当的使用 DI 或者是 IOC 的时候，我们可以开发松耦合应用。松耦合应用的单元测试可以很容易的进行。

SpringMVC

Spring MVC 提供了一种分离式的方法来开发 Web 应用。通过运用像 DispatcherServlet，MoudlAndView 和 ViewResolver 等一些简单的概念，开发 Web 应用将会变得非常简单。

SpringBoot

Spring 和 SpringMVC 的问题在于需要配置大量的参数。

```
<bean
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix">
    <value>/WEB-INF/views/</value>
  </property>
  <property name="suffix">
    <value>.jsp</value>
  </property>
</bean>

<mvc:resources mapping="/webjars/**" location="/webjars/" />
```

Spring Boot 通过一个自动配置和启动的项来解决这个问题。为了更快的构建产品就绪应用程序，Spring Boot 提供了一些非功能性特征。

52、什么是 Spring Boot Stater ?

启动器是一套方便的依赖描述符，它可以放在自己的程序中。你可以一站式的获取你所需要的 Spring 和相关技术，而不需要依赖描述符的通过示例代码搜索和复制粘贴的负载。

例如，如果你想使用 Spring 和 JPA 访问数据库，只需要你的项目包含 spring-boot-starter-data-jpa 依赖项，你就可以完美进行。

问题四 你能否举一个例子来解释更多 Staters 的内容？

让我们来思考一个 Stater 的例子 -Spring Boot Stater Web。

如果你想开发一个 web 应用程序或者是公开 REST 服务的应用程序。Spring Boot Start Web 是首选。让我们使用 Spring Initializr 创建一个 Spring Boot Start Web 的快速项目。

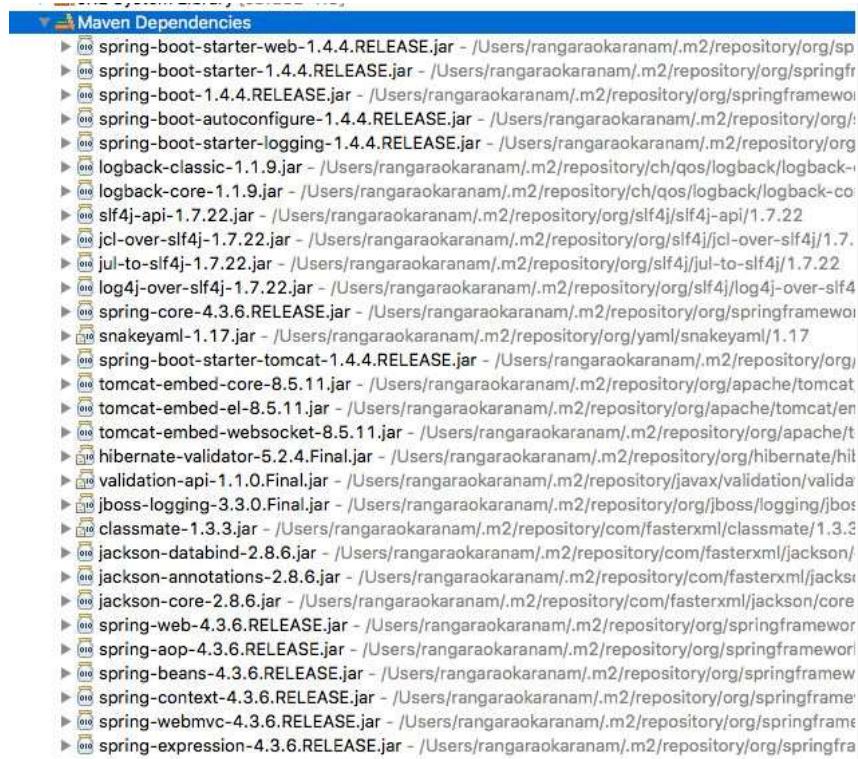
Spring Boot Start Web 的依赖项

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

```

下面的截图是添加进我们应用程序的不同的依赖项



依赖项可以被分为

```

Spring - core, beans, context, aop
Web MVC - (Spring MVC)
Jackson - for JSON Binding
Validation - Hibernate, validation API
Embedded Servlet Container - Tomcat
Logging - logback, slf4j

```

任何经典的 Web 应用程序都会使用所有这些依赖项。Spring Boot Starter Web 预先打包了这些依赖项。

作为一个开发者，我不需要再担心这些依赖项和它们的兼容版本。

53、Spring Boot 还提供了其它的哪些 Starter Project Options ?

Spring Boot 也提供了其它的启动器项目包括，包括用于开发特定类型应用程序的典型依赖项。

spring-boot-starter-web-services - SOAP Web Services

spring-boot-starter-web - Web 和 RESTful 应用程序

spring-boot-starter-test - 单元测试和集成测试

spring-boot-starter-jdbc - 传统的 JDBC

spring-boot-starter-hateoas - 为服务添加 HATEOAS 功能

spring-boot-starter-security - 使用 SpringSecurity 进行身份验证和授权

spring-boot-starter-data-jpa - 带有 Hibernat 的 Spring Data JPA

spring-boot-starter-data-rest - 使用 Spring Data REST 公布简单的 REST 服务

54、Spring 是如何快速创建产品就绪应用程序的？

Spring Boot 致力于快速产品就绪应用程序。为此，它提供了一些譬如高速缓存，日志记录，监控和嵌入式服务器等开箱即用的非功能性特征。

spring-boot-starter-actuator - 使用一些如监控和跟踪应用的高级功能

spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat - 选择您的特定嵌入式 Servlet 容器

spring-boot-starter-logging - 使用 logback 进行日志记录

spring-boot-starter-cache - 启用 Spring Framework 的缓存支持

###Spring2 和 Spring5 所需要的最低 Java 版本是什么？

Spring Boot 2.0 需要 Java8 或者更新的版本。Java6 和 Java7 已经不再支持。

推荐阅读：

<https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0.0-M1-Release-Notes>

55、创建一个 Spring Boot Project 的最简单的方法是什么？

Spring Initializr 是启动 Spring Boot Projects 的一个很好的工具。

The screenshot shows the Spring Initializr web interface at start.spring.io. The title bar says "SPRING INITIALIZR bootstrap your application now". Below it, there's a button "Generate a [Maven Project] with Spring Boot 2.0.0 (SNAPSHOT)". The left side has a "Project Metadata" section with fields for "Group" and "Artifact". The right side has a "Dependencies" section with a search bar and a list of selected dependencies: "Web", "Actuator", and "DevTools". At the bottom is a large green "Generate Project" button.

- 就像上图中所展示的一样，我们需要做一下几步：
- 登录 Spring Initializr，按照以下方式进行选择：
- 选择 com.in28minutes.springboot 为组
- 选择 student-services 为组件
- 选择下面的依赖项
- Web
- Actuator
- DevTools
- 点击生 GenerateProject
- 将项目导入 Eclipse。文件 - 导入 - 现有的 Maven 项目

56、Spring Initializr 是创建 Spring Boot Projects 的唯一方法吗？

不是的。

Spring Initializr 让创建 Spring Boot 项目变的很容易，但是，你也可以通过设置一个 maven 项目并添加正确的依赖项来开始一个项目。

在我们的 Spring 课程中，我们使用两种方法来创建项目。

第一种方法是 start.spring.io。

另外一种方法是在项目的标题为“Basic Web Application”处进行手动设置。

手动设置一个 maven 项目

这里有几个重要的步骤：

- 在 Eclipse 中，使用文件 - 新建 Maven 项目来创建一个新项目
- 添加依赖项。
- 添加 maven 插件。
- 添加 Spring Boot 应用程序类。

到这里，准备工作已经做好！

57、如何使用 SpringBoot 自动重装我的应用程序？

使用 Spring Boot 开发工具。

把 Spring Boot 开发工具添加进入你的项目是简单的。

把下面的依赖项添加至你的 Spring Boot Project pom.xml 中

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
</dependency>
```

重启应用程序，然后就可以了。

同样的，如果你想自动装载页面，有可以看看 FiveReload

- <http://www.logicbig.com/tutorials/spring-framework/spring-boot/boot-live-reload/>.

在我测试的时候，发现了 LiveReload 漏洞，如果你测试时也发现了，请一定要告诉我们。

58、什么是嵌入式服务器？我们为什么要使用嵌入式服务器呢？

思考一下在你的虚拟机上部署应用程序需要些什么。

第一步：安装 Java

第二部：安装 Web 或者是应用程序的服务器（Tomcat/Wesphere/Weblogic 等等）

第三部：部署应用程序 war 包

如果我们想简化这些步骤，应该如何做呢？

让我们来思考如何使服务器成为应用程序的一部分？

你只需要一个安装了 Java 的虚拟机，就可以直接在上面部署应用程序了，
是不是很爽？

这个想法是嵌入式服务器的起源。

当我们创建一个可以部署的应用程序的时候，我们将会把服务器（例如，tomcat）嵌入到可部署的服务器中。

例如，对于一个 Spring Boot 应用程序来说，你可以生成一个包含 Embedded Tomcat 的应用程序 jar。你就可以像运行正常 Java 应用程序一样来运行 web 应用程序了。

嵌入式服务器就是我们的可执行单元包含服务器的二进制文件（例如，tomcat.jar）。

59、如何在 Spring Boot 中添加通用的 JS 代码？

在源文件夹下，创建一个名为 static 的文件夹。然后，你可以把你的静态的内容放在这里面。

例如，myapp.js 的路径是 resources\static\js\myapp.js

你可以参考它在 jsp 中的使用方法

```
<script src="/js/myapp.js"></script>
```

错误：HAL browser gives me unauthorized error - Full authentication is required to access this resource.

该如何来修复这个错误呢？

```
{
  "timestamp": 1488656019562,
  "status": 401,
  "error": "Unauthorized",
  "message": "Full authentication is required to access this resource.",
  "path": "/beans"
}
```

两种方法：

方法 1：关闭安全验证

application.properties

```
management.security.enabled:FALSE
```

方法二：在日志中搜索密码并传递至请求标头中

60、什么是 Spring Data？

来自：<http://projects.spring.io/spring-data/>

Spring Data 的使命是在保证底层数据存储特殊性的前提下，为数据访问提供一个熟悉的，一致性的，基于 Spring 的编程模型。这使得使用数据访问技术，关系数据库和非关系数据库，map-reduce 框架以及基于云的数据服务变得很容易。

为了让它更简单一些，Spring Data 提供了不受底层数据源限制的 Abstractions 接口。

下面来举一个例子

```
interface TodoRepository extends CrudRepository<Todo, Long> {
```

你可以定义一个简单的库，用来插入，更新，删除和检索代办事项，而不需要编写大量的代码。

61、什么是 Spring Data REST?

Spring Data REST 可以用来发布关于 Spring 数据库的 HATEOAS RESTful 资源。

下面是一个使用 JPA 的例子

```
@RepositoryRestResource(collectionResourceRel = "todos", path = "todos")
public interface TodoRepository
    extends PagingAndSortingRepository<Todo, Long> {
```

不需要写太多代码，我们可以发布关于 Spring 数据库的 RESTful API。

下面展示的是一些关于 TEST 服务器的例子

POST

- URL: http://localhost:8080/todos
- Use Header: Content-Type: application/json
- Request Content

代码如下

```
{  
    "user": "jill",  
    "desc": "Learn Hibernate",  
    "done": false  
}
```

响应内容

```
{  
    "user": "jill",  
    "desc": "Learn Hibernate",  
    "done": false,  
    "_links": {  
        "self": {  
            "href": "http://localhost:8080/todos/1"  
        },  
        "todo": {  
            "href": "http://localhost:8080/todos/1"  
        }  
    }  
}
```

响应包含新创建资源的 href。

62、path="users", collectionResourceRel="users" 如何与 Spring Data Rest 一起使用？

```
@RepositoryRestResource(collectionResourceRel = "users", path = "users")
public interface UserRestRepository extends
PagingAndSortingRepository<User, Long>
```

- path- 这个资源要导出的路径段。
- collectionResourceRel- 生成指向集合资源的链接时使用的 rel 值。在生成 HATEOAS 链接时使用。

63、当 Spring Boot 应用程序作为 Java 应用程序运行时，后台会发生什么？

如果你使用 Eclipse IDE，Eclipse maven 插件确保依赖项或者类文件的改变一经添加，就会被编译并在目标文件中准备好！在这之后，就和其他的 Java 应用程序一样了。

当你启动 java 应用程序的时候，spring boot 自动配置文件就会魔法般的启用了。

- 当 Spring Boot 应用程序检测到你正在开发一个 web 应用程序的时候，它就会启动 tomcat。

64、我们能否在 spring-boot-starter-web 中用 jetty 替代 tomcat？

在 spring-boot-starter-web 移除现有的依赖项，并把下面这些添加进去。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
```

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-tomcat</artifactId>
</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

65、如何使用 Spring Boot 生成一个 WAR 文件？

推荐阅读:

- <https://spring.io/guides/gs/convert-jar-to-war/>

下面有 spring 说明文档直接的链接地址 :

- <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#build-tool-plugins-maven-packaging>

66、如何使用 Spring Boot 部署到不同的服务器？

你需要做下面两个步骤 :

- 在一个项目中生成一个 war 文件。
- 将它部署到你最喜欢的服务器 (websphere 或者 Weblogic 或者 Tomcat and so on)。

第一步 : 这本入门指南应该有所帮助 :

<https://spring.io/guides/gs/convert-jar-to-war/>

第二步 : 取决于你的服务器。

67、RequestMapping 和 GetMapping 的不同之处在哪里？

- RequestMapping 具有类属性的，可以进行 GET,POST,PUT 或者其它的注释中具有的请求方法。
- GetMapping 是 GET 请求方法中的一个特例。它只是 RequestMapping 的一个延伸，目的是为了提高清晰度。

68、为什么我们不建议在实际的应用程序中使用 Spring Data Rest?

我们认为 Spring Data Rest 很适合快速原型制造！在大型应用程序中使用需要谨慎。

通过 Spring Data REST 你可以把你的数据实体作为 RESTful 服务直接发布。

当你设计 RESTful 服务器的时候，最佳实践表明，你的接口应该考虑到两件重要的事情：

- 你的模型范围。
- 你的客户。

通过 With Spring Data REST，你不需要再考虑这两个方面，只需要作为 TEST 服务发布实体。

这就是为什么我们建议使用 Spring Data Rest 在快速原型构造上面，或者作为项目的初始解决方法。对于完整演变项目来说，这并不是一个好的注意。

69、在 Spring Initializer 中，如何改变一个项目的包名字？

好消息是你可以定制它。点击链接“转到完整版本”。你可以配置你想要修改的包名称！

70、可以配置 application.properties 的完整的属性列表在哪里可以找到？

这里是完整的指南：

- <https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

71、JPA 和 Hibernate 有哪些区别？

简而言之

- JPA 是一个规范或者接口
- Hibernate 是 JPA 的一个实现

当我们使用 JPA 的时候，我们使用 javax.persistence 包中的注释和接口时，不需要使用 hibernate 的导入包。

我们建议使用 JPA 注释，因为哦我们没有将其绑定到 Hibernate 作为实现。后来（我知道 - 小于百分之一的几率），我们可以使用另一种 JPA 实现。

72、使用 Spring Boot 启动连接到内存数据库 H2 的 JPA 应用程序需要哪些依赖项？

在 Spring Boot 项目中，当你确保下面的依赖项都在类路里面的时候，你可以加载 H2 控制台。

- web 启动器
- h2
- jpa 数据启动器

其它的依赖项在下面：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

需要注意的一些地方：

- 一个内部数据内存只在应用程序执行期间存在。这是学习框架的有效方式。
- 这不是你希望的真是世界应用程序的方式。
- 在问题“如何连接一个外部数据库？”中，我们解释了如何连接一个你所选择的数据库。

73、如何不通过任何配置来选择 Hibernate 作为 JPA 的默认实现？

因为 Spring Boot 是自动配置的。

下面是我们添加的依赖项

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

spring-boot-starter-data-jpa 对于 Hibernate 和 JPA 有过渡依赖性。

当 Spring Boot 在类路径中检测到 Hibernate 中，将会自动配置它为默认的 JPA 实现。

74、指定的数据连接信息在哪里？它是如何知道自动连接至 H2 的？

这就是 Spring Boot 自动配置的魔力。

来自：

<https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-auto-configuration.html>

Spring Boot auto-configuration 试图自动配置你已经添加的基于 jar 依赖项的 Spring 应用程序。比如说，如果 HSQLDBis 存在你的类路径中，并且，数据库连接 bean 还没有手动配置，那么我们可以自动配置一个内存数据库。

进一步的阅读：

<http://www.springbootutorial.com/spring-boot-auto-configuration>

75、我们如何连接一个像 MSSQL 或者 oracle 一样的外部数据库？

让我们以 MySQL 为例来思考这个问题：

第一步 - 把 mysql 连接器的依赖项添加至 pom.xml

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

第二步 - 从 pom.xml 中移除 H2 的依赖项

或者至少把它作为测试的范围。

```
<!--
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
<scope>test</scope>
</dependency>
-->
```

第三步 - 安装你的 MySQL 数据库

更多的来看看这里 -<https://github.com/in28minutes/jpa-with-hibernate#installing-and-setting-up-mysql>

第四步 - 配置你的 MySQL 数据库连接

配置 application.properties

```
spring.jpa.hibernate.ddl-auto=none  
spring.datasource.url=jdbc://localhost:3306/todo_example  
spring.datasource.username=todouser  
spring.datasource.password=YOUR_PASSWORD
```

第五步 - 重新启动，你就准备好了！

就这么简单！

76、Spring Boot 配置的默认 H2 数据库的名字是上面？为什么默认的数据库名字是 testdb ？

在 application.properties 里面，列出了所有的默认值

- <https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

找到下面的属性

```
spring.datasource.name=testdb # Name of the datasource.
```

如果你使用了 H2 内部存储数据库，它里面确定了 Spring Boot 用来安装你的 H2 数据库的名字。

77、如果 H2 不在类路径里面，会出现上面情况？

将会报下面的错误

```
Cannot determine embedded database driver class for database type NONE
```

把 H2 添加至 pom.xml 中，然后重启你的服务器

```
<dependency>  
    <groupId>com.h2database</groupId>  
    <artifactId>h2</artifactId>  
    <scope>runtime</scope>  
</dependency>
```

78、你能否举一个以 ReadOnly 为事务管理的例子？

当你从数据库读取内容的时候，你想把事物中的用户描述或者是其它描述设置为只读模式，以便于 Hebernate 不需要再次检查实体的变化。这是非常高效的。

79、发布 Spring Boot 用户应用程序自定义配置的最好方法是什么？

@Value 的问题在于，你可以通过应用程序分配你配置值。更好的操作是采取集中的方法。

你可以使用 @ConfigurationProperties 定义一个配置组件。

```
@Component  
@ConfigurationProperties("basic")  
public class BasicConfiguration {  
    private boolean value;  
    private String message;  
    private int number;
```

你可以在 application.properties 中配置参数。

```
basic.value: true  
basic.message: Dynamic Message  
basic.number: 100
```

80、配置文件的需求是什么？

企业应用程序的开发是复杂的，你需要混合的环境：

- Dev
- QA

- Stage
- Production

在每个环境中，你想要不同的应用程序配置。

配置文件有助于在不同的环境中进行不同的应用程序配置。

Spring 和 Spring Boot 提供了你可以制定的功能。

- 不同配置文件中，不同环境的配置是什么？
- 为一个指定的环境设置活动的配置文件。

Spring Boot 将会根据特定环境中设置的活动配置文件来选择应用程序的配置。

81、如何使用配置文件通过 Spring Boot 配置特定环境的配置？

配置文件不是识别环境的关键。

在下面的例子中，我们将会用到两个配置文件

- dev
- prod

缺省的应用程序配置在 application.properties 中。让我们来看下面的例子：

application.properties

```
basic.value= true
basic.message= Dynamic Message
basic.number= 100
```

我们想要为 dev 文件自定义 application.properties 属性。我们需要创建一个名为 application-dev.properties 的文件，并且重写我们想要自定义的属性。

application-dev.properties

```
basic.message: Dynamic Message in DEV
```

一旦你特定配置了配置文件，你需要在环境中设定一个活动的配置文件。

有多种方法可以做到这一点：

- 在 VM 参数中使用 Dspring.profiles.active=prod
- 在 application.properties 中使用 spring.profiles.active=prod

文章来源：<http://www.3xmq.com/article/1522809264295>

82、我们如何使用Maven设置Spring Boot应用程序？

我们可以像在任何其他库中一样在 Maven 项目中包含 Spring Boot。但是，最好的方法是从 spring-boot-starter-parent 项目继承并声明依赖于 Spring Boot 启动器。这样做可以让我们的项目重用 Spring Boot 的默认设置。继承 spring-boot-starter-parent 项目非常简单 - 我们只需要在 pom.xml 中指定一个 parent 元素：

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.6.RELEASE</version>
</parent>
```

我们可以在 Maven 中央仓库找到最新版本的 spring-boot-starter-parent。上面的方式很方便但是并不一定符合实际需要。例如公司要求所有项目依赖构建从一个标准 BOM 开始，我们就不能按上面的方式进行。在这种情况下，我们可以进行如下引用：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dependencies</artifactId>
      <version>2.1.6.RELEASE</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

83、如何禁用特定的自动配置？

如果我们要禁用特定的自动配置，我们可以使用 `@EnableAutoConfiguration` 注解的 `exclude` 属性来指示它。如下禁用了数据源自动配置 `DataSourceAutoConfiguration`：

```
// other annotations
@EnableAutoConfiguration(exclude = DataSourceAutoConfiguration.class)
public class MyConfiguration { }
```

如果我们使用 `@SpringBootApplication` 注解。它具有 `@EnableAutoConfiguration` 作为元注解 - 我们同样可以配置 `exclude` 属性来禁用自动配置：

```
// other annotations
@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
public class MyConfiguration { }
```

我们还可以使用 `spring.autoconfigure.exclude` 环境属性禁用自动配置。在 `application.properties` (也可以是 `application.yml`) 配置文件设置如下也可以达到同样的目的：

```
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration
```

84、Spring boot 支持哪些外部配置？

Spring Boot 支持外部配置，允许我们在各种环境中运行相同的应用程序。我们可以使用 `properties` 文件，`YAML` 文件，环境变量，系统属性和命令行选项参数来指定配置属性。然后，我们可以访问使用这些属性 @Value 注释，经由绑定对象的 `@ConfigurationProperties` 注释或 `Environment` 环境抽象类注入。以下是最常见的外部配置来源：

- 命令行属性：命令行选项参数是以双连字符开头的程序参数，例如 `-server.port = 8080`。Spring Boot 将所有参数转换为属性，并将它们添加到环境属性集中。
- 应用程序属性：应用程序属性是从 `application.properties` 文件或其 `YAML` 对应文件加载的属性。默认情况下，Spring Boot 会在当前目录，类路径根或其 `config` 子目录中搜索此文件。特定于配置文件的属性：特定于配置文件的属性从 `application-{profile}.properties` 文件或其 `YAML` 对应文件加载。`{profile}` 占位符是指活性轮廓。这些文件与非特定属性文件位于相同位置，并且优先于非特定属性文件。

85、如何对 Spring Boot 应用进行测试？

在为 Spring 应用程序运行集成测试时，我们必须有一个 `ApplicationContext`。为了简化测试，Spring Boot 为测试提供了一个特殊的注释 `@SpringBootTest`。此注释从其 `classes` 属性指示的配置类创建 `ApplicationContext`。如果未设置 `classes` 属性，Spring Boot 将搜索主配置类。搜索从包含测试的包开始，直到找到使用 `@SpringBootApplication` 或 `@SpringBootConfiguration` 注释的类。请注意，如果我们使用 `JUnit 4`，我们必须用 `@RunWith(SpringRunner.class)` 装饰测试类。

86、Spring Boot Actuator 有什么用？

Spring Boot Actuator 可以帮助你监控和管理 Spring Boot 应用，比如健康检查、审计、统计和 HTTP 追踪等。所有的这些特性可以通过 JMX 或者 HTTP endpoints 来获得。Actuator 同时还可以与外部应用监控系统整合，比如 Prometheus，Graphite，DataDog，Influx，Wavefront，New Relic 等。这些系统提供了非常好的仪表盘、图标、分析和告警等功能，使得你可以通过统一的接口轻松的监控和管理你的应用。Actuator 使用 Micrometer 来整合上面提到的外部应用监控系统。这使得只要通过非常小的配置就可以集成任何应用监控系统。将 Spring Boot Actuator 集成到一个项目中非常简单。我们需要做的就是在 `pom.xml` 文件中包含 `spring-boot-starter-actuator` 启动器：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Spring Boot Actuator 可以使用 HTTP 或 JMX 端点公开操作信息。但是，大多数应用程序都使用 HTTP，其中端点的标识和/执行器前缀形成 URL 路径。以下是 Actuator 提供的一些最常见的内置端点：

- `auditevents`：公开审计事件信息
- `env`：公开环境属性
- `health`：显示应用程序运行状况信息
- `httptrace`：显示 HTTP 跟踪信息

- `info` : 显示任意应用程序信息
- `metrics` : 显示指标信息
- `mappings` : 显示所有@RequestMapping路径的列表
- `scheduledtasks` : 显示应用程序中的计划任务
- `threaddump` : 执行线程转储
- `beans` : 所有加载的spring bean

更多关于 Spring Boot Actuator 的信息可查看[Spring Boot 2.x 中的 Actuator](#)。请注意：生产使用Actuator务必保护好这些端点，避免未授权的访问请求。

87、SpringBoot 中静态首页默认位置可以放在哪里？

当我们应用根目录时，可以直接映射，将 index.html 放入下面的位置：

```
classpath:/META-INF/resources/index.html
classpath:/resources/index.html
classpath:/static/index.html
classpath:/public/index.html
```

89、SpringBoot 中静态资源直接映射的优先级是怎样的？

SpringBoot 静态资源直接映射为/**，可以通过根目录来访问。/META-INF/resources/webjars/映射为/webjars/，通过访问 /webjar 访问。优先级顺序为：META-INF/resources > resources > static > public。

90、继承 WebMvcConfigurerAdapter 抽象类，常用的重写方法列举几个？

WebMvcConfigurerAdapter 实现 WebMvcConfigurer 接口，常用的可能需要重写的方法有以下几个：

```
/** 解决跨域问题 */
public void addCorsMappings(CorsRegistry registry) ;
/** 添加拦截器 */
void addInterceptors(InterceptorRegistry registry);
/** 这里配置视图解析器 */
void configureViewResolvers(ViewResolverRegistry registry);
/** 配置内容裁决的一些选项 */
void configureContentNegotiation(ContentNegotiationConfigurer configurer);
/** 视图跳转控制器 */
void addViewControllers(ViewControllerRegistry registry);
/** 静态资源处理 */
void addResourceHandlers(ResourceHandlerRegistry registry);
/** 默认静态资源处理器 */
void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer);
```

91、@SpringBootApplication 引入了哪3个重要的注解？

@SpringBootConfiguration、@EnableAutoConfiguration、@ComponentScan。其它的 4 个 @Target、@Retention、@Documented、@Inherited，也重要，但应该不是本题想问的知识点。

92、@SpringBootApplication 注解中的属性相当于哪几个注解？

等价于以默认属性使用 @Configuration，@EnableAutoConfiguration 和 @ComponentScan。

Spring Cloud面试题

1、什么是 Spring Cloud ?

Spring cloud 流应用程序启动器是基于 Spring Boot 的 Spring 集成应用程序，提供与外部系统的集成。Spring cloud Task，一个生命周期短暂的微服务框架，用于快速构建执行有限数据处理的应用程序。

2、使用 Spring Cloud 有什么优势？

使用 Spring Boot 开发分布式微服务时，我们面临以下问题

- 1、与分布式系统相关的复杂性-这种开销包括网络问题，延迟开销，带宽问题，安全问题。
- 2、服务发现-服务发现工具管理群集中的流程和服务如何查找和互相交谈。它涉及一个服务目录，在该目录中注册服务，然后能够查找并连接到该目录中的服务。
- 3、冗余-分布式系统中的冗余问题。
- 4、负载平衡 --负载平衡改善跨多个计算资源的工作负荷，诸如计算机，计算机集群，网络链路，中央处理单元，或磁盘驱动器的分布。
- 5、性能-问题 由于各种运营开销导致的性能问题。
- 6、部署复杂性-Devops 技能的要求。

3、服务注册和发现是什么意思？Spring Cloud 如何实现？

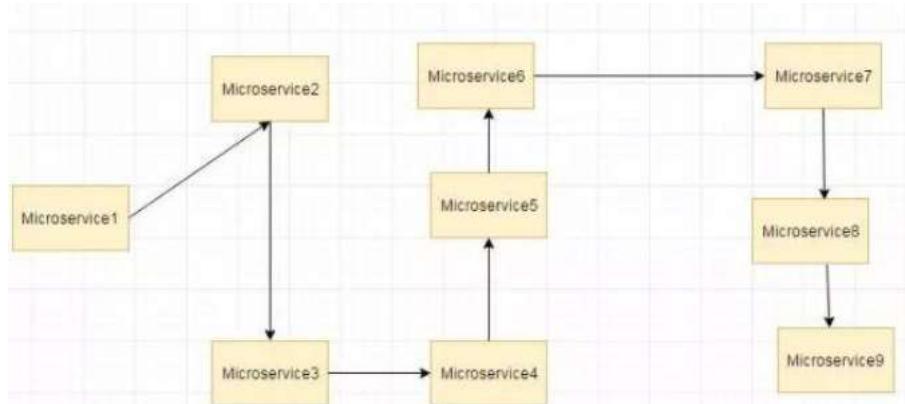
当我们开始一个项目时，我们通常在属性文件中进行所有的配置。随着越来越多的服务开发和部署，添加和修改这些属性变得更加复杂。有些服务可能会下降，而某些位置可能会发生变化。手动更改属性可能会产生问题。Eureka 服务注册和发现可以在这种情况下提供帮助。由于所有服务都在 Eureka 服务器上注册并通过调用 Eureka 服务器完成查找，因此无需处理服务地点的任何更改和处理。

4、负载平衡的意义什么？

在计算中，负载平衡可以改善跨计算机，计算机集群，网络链接，中央处理单元或磁盘驱动器等多种计算资源的工作负载分布。负载平衡旨在优化资源使用，最大化吞吐量，最小化响应时间并避免任何单一资源的过载。使用多个组件进行负载平衡而不是单个组件可能会通过冗余来提高可靠性和可用性。负载平衡通常涉及专用软件或硬件，例如多层交换机或域名系统服务器进程。

5、什么是 Hystrix？它如何实现容错？

Hystrix 是一个延迟和容错库，旨在隔离远程系统，服务和第三方库的访问点，当出现故障是不可避免的故障时，停止级联故障并在复杂的分布式系统中实现弹性。通常对于使用微服务架构开发的系统，涉及到许多微服务。这些微服务彼此协作。思考以下微服务



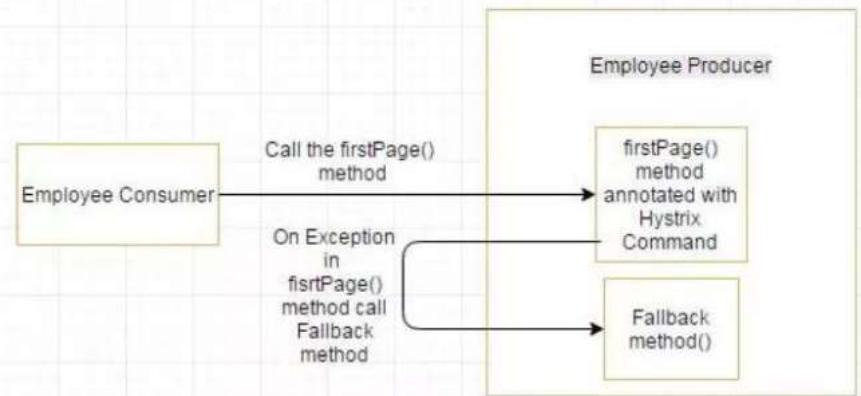
假设如果上图中的微服务 9 失败了，那么使用传统方法我们将传播一个异常。但这仍然会导致整个系统崩溃。随着微服务数量的增加，这个问题变得更加复杂。微服务的数量可以高达 1000.这是 hystrix 出现的地方 我们将使用 Hystrix 在这种情况下的 Fallback 方法功能。我们有两个服务 employee-consumer 使用由 employee-consumer 公开的服务。简化图如下所示



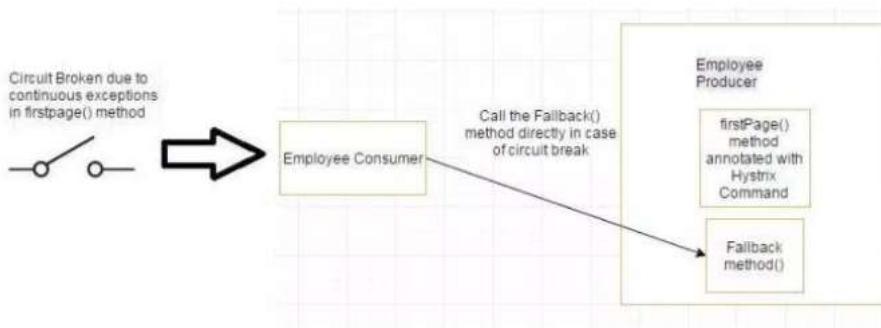
现在假设由于某种原因，employee-producer 公开的服务会抛出异常。我们在这种情况下使用 Hystrix 定义了一个回退方法。这种后备方法应该具有与公开服务相同的返回类型。如果暴露服务中出现异常，则回退方法将返回一些值。

6、什么是 Hystrix 断路器？我们需要它吗？

由于某些原因，employee-consumer 公开服务会引发异常。在这种情况下使用Hystrix 我们定义了一个回退方法。如果在公开服务中发生异常，则回退方法返回一些默认值。



如果 firstPage method() 中的异常继续发生，则 Hystrix 电路将中断，并且员工使用者将一起跳过 firtsPage 方法，并直接调用回退方法。断路器的目的是给第一页方法或第一页方法可能调用的其他方法留出时间，并导致异常恢复。可能发生的情况是，在负载较小的情况下，导致异常的问题有更好的恢复机会。



7、什么是 Netflix Feign ? 它的优点是什么 ?

Feign 是受到 Retrofit , JAXRS-2.0 和 WebSocket 启发的 java 客户端联编程序。Feign 的第一个目标是将约束分母的复杂性统一到 http apis , 而不考虑其稳定性。在 employee-consumer 的例子中 , 我们使用了 employee-producer 使用 REST 模板公开的 REST 服务。

但是我们必须编写大量代码才能执行以下步骤

- 1、使用功能区进行负载平衡。
- 2、获取服务实例 , 然后获取基本 URL。
- 3、利用 REST 模板来使用服务。前面的代码如下

```

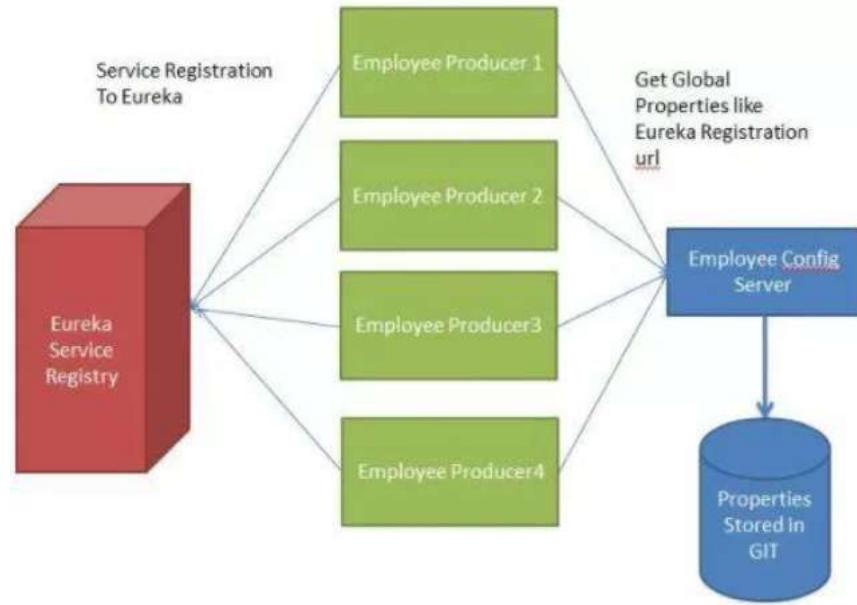
@Controller
public class ConsumerControllerClient {
    @Autowired
    private LoadBalancerClient loadBalancer;
    public void getEmployee() throws RestClientException, IOException {
        ServiceInstance serviceInstance=loadBalancer.choose("employee-producer");
        System.out.println(serviceInstance.getUri());
        String baseUrl=serviceInstance.getUri().toString();
        baseUrl=baseUrl+"/employee";
        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<String> response=null;
        try{
            response=restTemplate.exchange(baseUrl,
                HttpMethod.GET, getHeaders(),String.class);
        }catch (Exception ex)
        {
            System.out.println(ex);
        }
        System.out.println(response.getBody());
    }
}

```

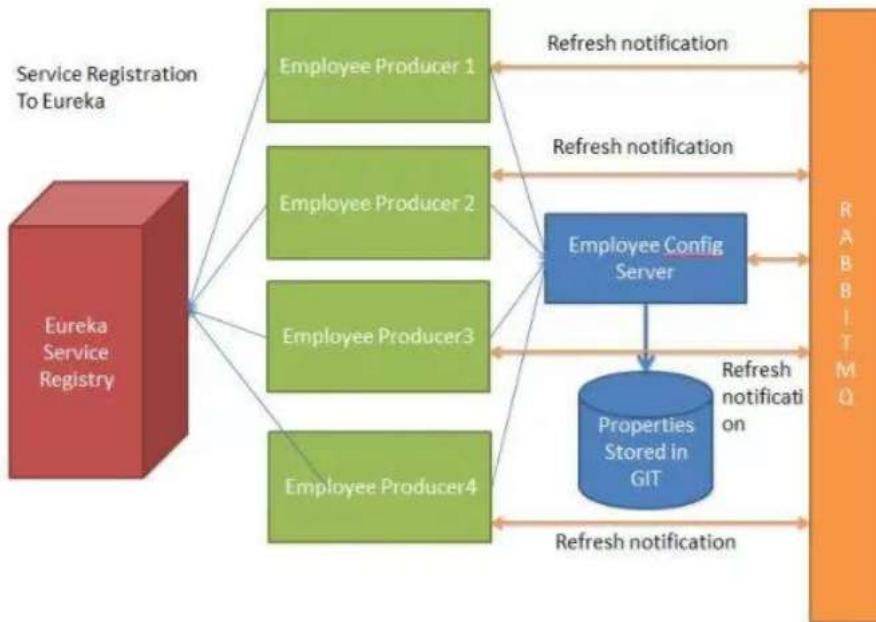
之前的代码 , 有像 NullPointerException 这样的例外的机会 , 并不是最优的。我们将看到如何使用 Netflix Feign 使呼叫变得更加轻松和清洁。如果 Netflix Ribbon 依赖关系也在类路径中 , 那么 Feign 默认也会负责负载平衡。

8、什么是 Spring Cloud Bus ? 我们需要它吗 ?

考虑以下情况 : 我们有多个应用程序使用 Spring Cloud Config 读取属性 , 而 Spring Cloud Config 从 GIT 读取这些属性。下面的例子中多个员工生产者模块从 Employee Config Module 获取 Eureka 注册的财产



如果假设 GIT 中的 Eureka 注册属性更改为指向另一台 Eureka 服务器，会发生什么情况。在这种情况下，我们将不得不重新启动服务以获取更新的属性。还有另一种使用执行器端点/刷新的方式。但是我们将不得不为每个模块单独调用这个 url。例如，如果 Employee Producer1 部署在端口 8080 上，则调用 <http://localhost:8080/refresh>。同样对于 Employee Producer2 <http://localhost:8081/refresh> 等等。这又很麻烦。这就是 Spring Cloud Bus 发挥作用的地方。



Spring Cloud Bus 提供了跨多个实例刷新配置的功能。因此，在上面的示例中，如果我们刷新 Employee Producer1，则会自动刷新所有其他必需的模块。如果我们有多个微服务启动并运行，这特别有用。这是通过将所有微服务连接到单个消息代理来实现的。无论何时刷新实例，此事件都会订阅到侦听此代理的所有微服务，并且它们也会刷新。可以通过使用端点/总线/刷新来实现对任何单个实例的刷新。

9、什么是微服务

微服务架构是一种架构模式或者说是一种架构风格，它提倡将单一应用程序划分为一组小的服务，每个服务运行在其独立的自己的进程中，服务之间相互协调、互相配合，为用户提供最终价值。服务之间采用轻量级的通信机制互相沟通（通常是基于HTTP的RESTful API），每个服务都围绕着具体的业务进行构建，并且能够被独立的构建在生产环境、类生产环境等。另外，应避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据业务上下文，选择合适的语言、工具对其进行构建，可以有一个非常轻量级的集中式管理来协调这些服务，可以使用不同的语言来编写服务，也可以使用不同的数据存储。

10、什么是服务熔断？什么是服务降级

熔断机制是应对雪崩效应的一种微服务链路保护机制。当某个微服务不可用或者响应时间太长时，会进行服务降级，进而熔断该节点微服务的调用，快速返回“错误”的响应信息。当检测到该节点微服务调用响应正常后恢复调用链路。在SpringCloud框架里熔断机制通过Hystrix实现，Hystrix会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是5秒内调用20次，如果失败，就会启动熔断机制。

服务降级，一般是从整体负荷考虑。就是当某个服务熔断之后，服务器将不再被调用，此时客户端可以自己准备一个本地的fallback回调，返回一个缺省值。这样做，虽然水平下降，但好歹可用，比直接挂掉强。

Hystrix相关注解

@EnableHystrix : 开启熔断
@HystrixCommand(fallbackMethod="XXX") : 声明一个失败回滚处理函数XXX , 当被注解的方法执行超时 (默认是1000毫秒) , 就会执行fallback函数 , 返回错误提示

微服务条目	落地的技术
服务开发	SpringBoot、Spring、SpringMVC
服务配置管理	Netflix公司的Archaius、阿里的Diamond等
服务注册与发现	Eureka、Consul、Zookeeper
服务调用	RPC、Rest、gRPC
服务熔断器	Hystrix、Envoy等
负载均衡	Nginx、Ribbon
服务接口调用 (客户端调用服务的简化工具)	Feign
消息队列	Kafka、RabbitMQ、ActiveMQ等
服务配置中心配置管理	SpringCloudConfig、Chef等
服务路由 (API网关)	Zuul
服务监控	Zabbix、Nagios、Metrics、Spectator等
全链路追踪	Zipkin、Brave、Dapper等
服务部署	Docker、OpenStack、Kubernetes等
数据流操作开发包	SpringCloud Stream
事件消息总线	Spring Cloud Bus

11、Eureka和zookeeper都可以提供服务注册与发现的功能，请说说两个的区别？

Zookeeper保证了CP (C : 一致性 , P : 分区容错性) , Eureka保证了AP (A : 高可用) 1.当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前的信息，但不能容忍直接down掉不可用。也就是说，服务注册功能对高可用性要求比较高，但zk会出现这样一种情况，当master节点因为网络故障与其他节点失去联系时，剩余节点会重新选leader。问题在于，选取leader时间过长，30 ~ 120s，且选取期间zk集群都不可用，这样就会导致选取期间注册服务瘫痪。在云部署的环境下，因网络问题使得zk集群失去master节点是较大概率会发生的事，虽然服务能够恢复，但是漫长的选取时间导致的注册长期不可用是不能容忍的。

2.Eureka保证了可用性，Eureka各个节点是平等的，几个节点挂掉不会影响正常节点的工作，剩余的节点仍然可以提供注册和查询服务。而Eureka的客户端向某个Eureka注册或发现时发生连接失败，则会自动切换到其他节点，只要有一台Eureka还在，就能保证注册服务可用，只是查到的信息可能不是最新的。除此之外，Eureka还有自我保护机制，如果在15分钟内超过85%的节点没有正常的心跳，那么Eureka就认为客户端与注册中心发生了网络故障，此时会出现以下几种情况：

- ①、Eureka不在从注册列表中移除因为长时间没有收到心跳而应该过期的服务。
- ②、Eureka仍然能够接受新服务的注册和查询请求，但是不会被同步到其他节点上（即保证当前节点仍然可用）
- ③、当网络稳定时，当前实例新的注册信息会被同步到其他节点。因此，Eureka可以很好的应对因网络故障导致部分节点失去联系的情况，而不会像Zookeeper那样使整个微服务瘫痪

12、SpringBoot和SpringCloud的区别？

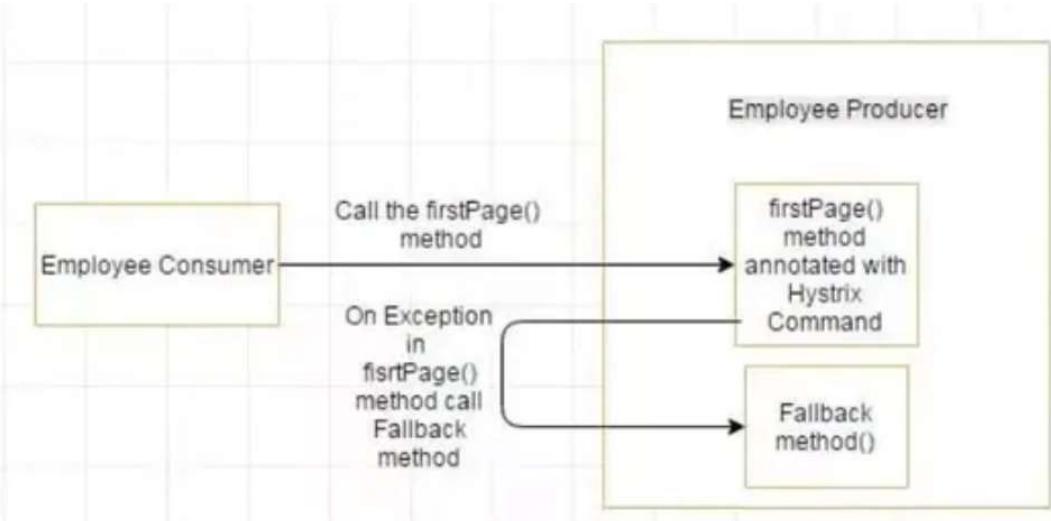
SpringBoot专注于快速方便的开发单个个体微服务。

SpringCloud是关注全局的微服务协调治理框架，它将SpringBoot开发的一个个单体微服务整合并管理起来，为各个微服务之间提供，配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等集成服务SpringBoot可以离开SpringCloud独立使用开发项目，但是SpringCloud离不开SpringBoot，属于依赖的关系。

SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架

13、什么是Hystrix断路器？我们需要它吗

由于某些原因，employee-consumer公开服务会引发异常。在这种情况下使用Hystrix我们定义了一个回退方法。如果在公开服务中发生异常，则回退方法返回一些默认值。



如果firstPage method() 中的异常继续发生，则Hystrix电路将中断，并且员工使用者将一起跳过firtsPage方法，并直接调用回退方法。断路器的目的是给第一页方法或第一页方法可能调用的其他方法留出时间，并导致异常恢复。可能发生的情况是，在负载较小的情况下，导致异常的问题有更好的恢复机会。



14、说说 RPC 的实现原理

首先需要有处理网络连接通讯的模块，负责连接建立、管理和消息的传输。其次需要有编解码的模块，因为网络通讯都是传输的字节码，需要将我们使用的对象序列化和反序列化。剩下的就是客户端和服务端的部分，服务器端暴露要开放的服务接口，客户调用服务接口的一个代理实现，这个代理实现负责收集数据、编码并传输给服务器然后等待结果返回。

15、微服务的优点缺点?说下开发项目中遇到的坑?

优点:

- 1.每个服务直接足够内聚，代码容易理解
- 2.开发效率高，一个服务只做一件事，适合小团队开发
- 3.松耦合，有功能意义的服务。
- 4.可以用不同语言开发，面向接口编程。
- 5.易于第三方集成
- 6.微服务只是业务逻辑的代码，不会和HTML,CSS或其他界面结合.
- 7.可以灵活搭配，连接公共库/连接独立库

缺点:

- 1.分布式系统的责任性
- 2.多服务运维难度加大。
- 3.系统部署依赖，服务间通信成本，数据一致性，系统集成测试，性能监控。

16、spring cloud 和dubbo区别?

- 1.服务调用方式 dubbo是RPC springcloud Rest Api
- 2.注册中心,dubbo 是zookeeper springcloud是eureka，也可以是zookeeper
- 3.服务网关,dubbo本身没有实现，只能通过其他第三方技术整合，springcloud有Zuul路由网关，作为路由器，进行消费者的请求分发,springcloud支持断路器，与git完美集成配置文件支持版本控制，事物总线实现配置文件的更新与服务自动装配等等一系列的微服务架构要素。

17、REST 和RPC对比

- 1.RPC主要的缺陷是服务提供方和调用方式之间的依赖太强，需要对每一个微服务进行接口的定义，并通过持续继承发布，严格版本控制才不会出现冲突。
- 2.REST是轻量级的接口，服务的提供和调用不存在代码之间的耦合，只需要一个约定进行规范。

18、你所知道的微服务技术栈 ?

维度(springcloud)
服务开发：springboot spring springmvc
服务配置与管理:Netflix公司的Archaius ,阿里的Diamond
服务注册与发现:Eureka,Zookeeper
服务调用:Rest RPC gRpc
服务熔断器:Hystrix
服务负载均衡:Ribbon Nginx
服务接口调用:Fegin
消息队列:Kafka Rabbitmq activemq
服务配置中心管理:SpringCloudConfig
服务路由 (API网关) Zuul
事件消息总线:SpringCloud Bus

19、微服务之间是如何独立通讯的?

- 1.远程调用，比如feign调用，直接通过远程过程调用来访问别的service。
- 2.消息中间件

20、springcloud如何实现服务的注册?

- 1.服务发布时，指定对应的服务名,将服务注册到 注册中心(eureka zookeeper)
- 2.注册中心加@EnableEurekaServer,服务用@EnableDiscoveryClient，然后用ribbon或feign进行服务直接的调用发现。

21、Eureka和Zookeeper区别

- 1.Eureka取CAP的AP，注重可用性，Zookeeper取CAP的CP注重一致性。
- 2.Zookeeper在选举期间注册服务瘫痪，虽然服务最终会恢复，但选举期间不可用。
- 3.eureka的自我保护机制，会导致一个结果就是不会再从注册列表移除因长时间没收到心跳而过期的服务。依然能接受新服务的注册和查询请求，但不会被同步到其他节点。不会服务瘫痪。
- 4.Zookeeper有Leader和Follower角色，Eureka各个节点平等。
- 5.Zookeeper采用过半数存活原则，Eureka采用自我保护机制解决分区问题。
- 6.eureka本质是一个工程，Zookeeper只是一个进程。

22、eureka自我保护机制是什么?

- 1.当Eureka Server 节点在短时间内丢失了过多实例的连接时(比如网络故障或频繁启动关闭客户端)节点会进入自我保护模式，保护注册信息，不再删除注册数据，故障恢复时，自动退出自我保护模式。

23、什么是Ribbon ?

ribbon是一个负载均衡客户端，可以很好的控制http和tcp的一些行为。feign默认集成了ribbon。

24、什么是feign ? 它的优点是什么 ?

- 1.feign采用的是基于接口的注解
- 2.feign整合了ribbon，具有负载均衡的能力
- 3.整合了Hystrix，具有熔断的能力

使用:

- 1.添加pom依赖。
- 2.启动类添加@EnableFeignClients
- 3.定义一个接口@FeignClient(name="xxx")指定调用哪个服务

25、Ribbon和Feign的区别 ?

- 1.Ribbon都是调用其他服务的，但方式不同。
- 2.启动类注解不同，Ribbon是@RibbonClient feign的是@EnableFeignClients
- 3.服务指定的位置不同，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。
- 4.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需要将调用的方法定义成抽象方法即可。

26、什么是Spring Cloud Bus?

spring cloud bus 将分布式的节点用轻量的消息代理连接起来，它可用于广播配置文件的更改或者服务直接的通讯，也可用于监控。
如果修改了配置文件，发送一次请求，所有的客户端便会重新读取配置文件。

使用:

- 1.添加依赖
- 2.配置rabbitmq

27、springcloud断路器作用？

当一个服务调用另一个服务由于网络原因或自身原因出现问题，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导致更多的请求等待，发生连锁效应（雪崩效应）。

断路器有完全打开状态：一段时间内达到一定的次数无法调用，并且多次监测没有恢复的迹象。断路器完全打开，那么下次请求就不会请求到该服务。

半开：短时间内有恢复迹象，断路器会将部分请求发给该服务，正常调用时，断路器关闭。

关闭：当服务一直处于正常状态，能正常调用。

28、Spring Cloud Gateway？

Spring Cloud Gateway是Spring Cloud官方推出的第二代网关框架，取代Zuul网关。网关作为流量的，在微服务系统中有着非常作用，网关常见的功能有路由转发、权限校验、限流控制等作用。

使用了一个RouteLocatorBuilder的bean去创建路由，除了创建路由RouteLocatorBuilder可以让你添加各种predicates和filters，predicates断言的意思，顾名思义就是根据具体的请求的规则，由具体的route去处理，filters是各种过滤器，用来对请求做各种判断和修改。

29、作为服务注册中心，Eureka比Zookeeper好在哪里？

1.Eureka保证的是可用性和分区容错性，Zookeeper保证的是一致性和分区容错性。

2.Eureka还有一种自我保护机制，如果在15分钟内超过85%的节点都没有正常的心跳，那么Eureka就认为客户端与注册中心出现了网络故障。而不会像zookeeper那样使整个注册服务瘫痪。

30、什么是 Ribbon负载均衡？

1.Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端负载均衡的工具。

2. Ribbon客户端组件提供一系列完善的配置项如连接超时，重试等。简单的说，就是在配置文件中列出Load Balancer（简称LB）后面所有的机器，Ribbon会自动的帮助你基于某种规则（如简单轮询，随机连接等）去连接这些机器。我们也很容易使用Ribbon实现自定义的负载均衡算法。

31、Ribbon负载均衡能干什么？

1. 将用户的请求平均的分配到多个服务上。

2. 集中式LB即在服务的消费方和提供方之间使用独立的LB设施（可以是硬件，如F5，也可以是软件，如nginx），由该设施负责把访问请求通过某种策略转发至服务的提供方；

3. 进程内LB将LB逻辑集成到消费方，消费方从服务注册中心获知有哪些地址可用，然后自己再从这些地址中选择出一个合适的服务器。

注意：Ribbon就属于进程内LB，它只是一个类库，集成于消费方进程，消费方通过它来获取到服务提供方的地址。

32、什么是 zuul路由网关

1.Zuul 包含了对请求的路由和过滤两个最主要的功能，其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础；而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础。

2.Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即以后的访问微服务都是通过Zuul跳转后获得。

注意：Zuul服务最终还是会注册进Eureka 提供=代理+路由+过滤 三大功能。

33、分布式配置中心能干嘛？

1. 集中管理配置文件不同环境不同配置，动态化的配置更新，分环境部署比如dev/test/prod/beta/release

2. 运行期间动态调整配置，不再需要在每个服务部署的机器上编写配置文件，服务会向配置中心统一拉取配置自己的信息。

3. 当配置发生变动时，服务不需要重启即可感知到配置的变化并应用新的配置将配置信息以REST接口的形式暴露。

34、Hystrix相关注解

@EnableHystrix：开启熔断。

@HystrixCommand(fallbackMethod="XXX")：声明一个失败回滚处理函数XXX，当被注解的方法执行超时（默认是1000毫秒），就会执行fallback函数，返回错误提示。