# JAVA 面笔试 题库

日期：2022.04.22

# 《JAVA 面笔试题库》

提示：

1、 题目类型包括：单选题、多选题、简答题、填空题、编程题等

2、 题目前缀【类型】【知识点】

| 知识点目录 | | | |
|---|---|---|---|
| 数据结构与算法 | Java 语言基础 | Java 常见类库 | Java 异常与处理 |
| Java 文件与流 | Java 多线程与并发 | Java 网络编程 | JDBC 编程 |
| Swing | Servlet | JSP | JavaWeb 应用 |
| SpringCore | SpringMVC | MyBatis | SSM 框架 |
| Java 安全框架 | Spring Boot | Spring Cloud | |

## -业务场景

### 1、智慧医疗系统

**面试官**：你写的项目中，哪个项目你觉得让你成长最大，介绍一下你做的这个项目。

**应聘者**：

智慧医院系统主要是为了解决看病难，提高看病效率，该项目主要有财务系统、药房系统、住院门诊医生站、住院护士站等系统组成。整个项目是 C/S 和 B/S 结合的模式。我所处的药品组开发的药房药库系统是 B/S 架构。其中前端使用了 Bootstrap+ MyBatis+ JavaScript + jQuery +CSS+AJAX 等技术，后端使用 SpringBoot+MyBatis 实现。

项目中本人负责药房系统的门诊住院发药模块，住院药房摆药发药模块，门诊住院代配模块，备用药发药模块等模块的开发工作。负责药库系统中部分定制化模块的开发工作和维护工作。负责药房药库系统共用工具类的部分完善与维护工作。

（明确说出项目名称、项目开发背景和功能，重点介绍自己完成的内容和使用的技术技能。）

==========模拟问题==========

**面试官**：你在项目中使用了 SpringBoot，那你谈谈 SpringBoot 与 Spring 对比有什么优势。

**应聘者**：SpringBoot 是基于约定大于配置的思想，做到了自动装配，开箱即用。避免了 Spring 中繁琐的配置文件，简化了开发流程，我们不需要在配置和业务逻辑之间进行思维切换，也就是说 SpringBoot 是实现 Spring 更快速的一种方式。

**面试官**：你怎么理解 SpringBoot 的约定大于配置。

**应聘者**：约定大于配置就是在 SpringBoot 中大量减少了配置文件的使用。也可以说不需要做某些配置也可以进行项目运行，尽可能的使用 SpringBoot 中自带的配置。如果你配置了就用你配置的，没配置就用默认的，并且这些默认的是可以让项目运行起来的。

**面试官**：项目中某些表数据量比较大时，你是怎么提升这个数据的查询速度。

**应聘者**：这边是通过添加索引的方式。

**面试官**：你说一说索引的作用是什么。

**应聘者**：索引是用来快速的寻找那些具有特定值的记录。如果没有索引，一般来说执行查询时要遍历整个表。

**面试官**：一般索引可以分为几种。

**应聘者**：索引一般分为聚簇索引和非聚簇索引。聚簇索引是将数据和索引放在了一起，并且按顺序组织的，找到了索引也就找到了数据。非聚簇索引是索引和数据不在一起，找到了索引后还需要根据索引去找数据。

**面试官**：你在项目中使用了 MyBatis，你可以说一说 MyBatis 和 Hibernate 的区别吗？

**应聘者**：MyBatis 是半自动映射，而 Hibernate 是全自动映射工具。MyBatis 它是属于半自动的，我们需要自己对于 SQL 查询自己编写 SQL 语句，这就是为什么叫它半自动，MyBatis 中还提供了将数据自动转换成实体类对象的功能，通过 queryForList 和 queryForObject 来获取查询的集合或单个对象。Hibernate 中，它会自动帮我们写 SQL，这也是他被称为全自动的原因，而且 Hibernate 相对 MyBatis 提供了更加负责的机制以及更加强大的功能，但是，Hibernate 的全自动机制，对于一些较为简单的 SQL 查询确实能够简化我们的操作，但涉及到一些较为复杂的查询时，Hibernate 就显得力不从心，这个时候我们就选择 MyBatis。

## 2、智能交通智慧项目

**面试官**：你写的项目中，哪个项目你觉得让你成长最大，介绍一下你做的这个项目。

**应聘者**：

智能交通智慧项目的功能模块主要有指挥调度、诱导发布、视频监控、预案管理、接处警管理、非现场管理、缉查布控管理、信号管理、系统管理、设施管理。是协助交警办公的良好软件。交警可以通过系统实时观看路况信息；了解警力分布；发布诱导信息；了解各处警情信息等主要功能。页面采用 MyBatis ,CSS, JQuery,EasyUI，AJAX 等技术，后台使用 SpringMVC+Spring 架构。在该项目负责指挥调度的警力资源功能，完成了警力轨迹新版开发工作。负责诱导发布模块。负责诱导发布模块新版的功能设计和数据库设计，完成了新版功能的开发工作。

==========模拟问题==========

**面试官**：这个项目中这些页面的跳转是怎么实现的。

**应聘者**：页面跳转可以分为直接前端跳转和经过 controller 跳转，这个项目中使用了 SpringMVC 框架，基本上

页面跳转都是使用经过 controller 层这个方式，通过 SpringMVC 前置控制器 dispatcherServlet 接收浏览器请求，经过映射器处理请求，再通过适配器找到对应的请求 controller，执行相应的方法返回页面信息，然后根据的视图解析器解析信息获取具体的页面后渲染跳转。

**面试官**：一般页面跳转都哪些方式。

**应聘者**：一般页面跳转可以分为重定向和转发。

**面试官**：你说一说这两种方式有些什么区别。

**应聘者**：转发是用户通过浏览器发送 HTTP 请求，Web 服务器接收请求后调用内部方法完成请求并来一个转发动作把页面给前端，所以转发只有一次请求，浏览器上地址不会改变。重定向是用户发送请求后，Web 服务器会回应给浏览器一个新的请求。重定向是有两次请求，浏览器上地址会发生改变。

**面试官**：你在整个项目开发中遇到了什么困难，你怎么解决的。

**应聘者**：比如说在开发诱导屏功能时，制作一个诱导屏节目是需要多张表共同完成的，所以在制作的时候就要要求我们保证多张表数据增删改的一致性。这个我通过 Spring 中事务控制保证了数据的一致性。

（根据自己项目的真实情况说明，并且保解决问题技术的相关知识是自己可以回答上来的）

**面试官**：你说一说 Spring 事务的实现方式；

**应聘者**：Spring 的实现方式可分为编程式方式和声明式事务。

**面试官**：那你怎么理解 Spring 事务的脏读。

**应聘者**：一个事务修改了一行数据但没有提交，第二个事务可以读取到这行被修改的数据，如果第一个事务回滚，第二个事务获取到的数据将是无效的。

**面试官**：你负责过数据库表设计，设计下部门、用户、角色表。

**应聘者**：比如部门表有主键、编号、名称。用户表有主键、用户登录账号、用户名称、密码和一起其他基本信息，角色表有主键、角色编号、角色名称，部门和用户的关联表，字段有：主键、用户 id、部门 id。这个关系表为多对多的一个关系。用户和角色关系表，有主键、用户 id、角色 id。关系表也可为多对多的关系。

## 3、智能美食系统

**面试官**：你在学校的项目你觉得哪个项目写的最好，简单阐述一下

**应聘者**：

在学校的项目中觉得智能美食系统写得最好，其中包含用户端和商家端。用户端包含用户模块，订单模块，支付模块；商家端包含用户模块，产品模块，统计模块；用户端的用户模块包含登录注册，个人中心，忘记密码，退出登录；用户端的订单模块包含查看自己的历史订单信息，评价订单，投诉商家；用户端的支付模块采用支付宝的

沙箱环境。商家端的用户模块包含店铺的注册登录，开始营业和暂停营业；商家端的产品模块包含产品的上架下架，产品价格的修改，商家端的统计模块包含对指定时间段的订单数和金额的统计，以图表的形式展示出来。

**面试官**：你在这个项目中主要采用了什么框架

**应聘者**：后端主要采用了 SpringBoot+MyBatis，前端页面采用的是 HTML+CSS+jQuery

**面试官**：在使用 MyBatis 框架时做条件查询时，参数的符号用的是哪个？

**应聘者**：#

**面试官**：那如果用$可不可以

**应聘者**：也可以实现，但是有 SQL 注入的风险（如果说不可以了那基本上就被淘汰了！）

**面试官**：刚刚你的项目中用到了支付宝的沙箱环境，简单阐述一下集成支付宝沙箱环境做支付的流程

**应聘者**：支付宝沙箱环境有一个对应的支付宝的沙箱环境的 APP，在做订单支付时请求阿里的支付接口，在支付成功前的订单状态是未支付的，支付完成后会对应有一个回调接口，对参数进行解密解签，获取支付的相关信息（支付的状态，金额等）

**面试官**：刚刚后台的框架的用到了 SpringBoot，简单阐述一下 SpringBoot 与 Spring 的区别

**应聘者**：SpringBoot 是基于约定大于配置的思想，做到了自动装配，开箱即用。避免了 Spring 中繁琐的配置文件，简化了开发流程，我们不需要在配置和业务逻辑之间进行思维切换，也就是说 SpringBoot 是实现 Spring 更快速的一种方式。

**面试官**：你对未来有什么职业规划么

**应聘者**：由于刚出校门步入社会，在未来一年内成长为能够独立进行项目的开发，未来三年内能够负责项目的开发上线，未来 5 年内能够独立负责一个部门的运转。

## 4、智慧商场

**面试官**：看你的简历上面写了在学校里面做了一个智能商城的项目，简单叙述一下

**应聘者**：智能商城主要给为了方便用户能够更快更好的买到自己中意的商品，同时也方便商家能够更好了解到自己商品的销售情况，其中包含用户端和商家端。用户端包含用户模块，订单模块，支付模块；商家端包含用户模块，产品模块，统计模块；用户端的用户模块包含登录注册，个人中心，忘记密码，退出登录；用户端的订单模块包含查看自己的历史订单信息，评价订单，投诉商家；用户端的支付模块采用支付宝的沙箱环境。商家端的用户模块包含店铺的注册登录，开始营业和暂停营业；商家端的产品模块包含产品的上架下架，产品价格的修改，商家端的统计模块包含对指定时间段的订单数和金额的统计，以图表的形式展示出来。

**面试官**：你在这个项目中主要运用了什么技术

**应聘者**：后端主要采用了 Spring+SpringMVC+MyBatis，前端页面采用的是 HTML+CSS+jQuery

**面试官**：在使用 MyBatis 框架时做条件查询时，参数的符号用的是哪个？

**应聘者**：#

**面试官**：那如果用$可不可以

**应聘者**：也可以实现，但是有 SQL 注入的风险（如果说不可以了那基本上就被淘汰了！）

**面试官**：刚刚你的项目中用到了支付宝的沙箱环境，简单阐述一下集成支付宝沙箱环境做支付的流程

**应聘者**：支付宝沙箱环境有一个对应的支付宝的沙箱环境的 APP，在做订单支付时请求阿里的支付接口，在支付成功前的订单状态是未支付的，支付完成后会对应有一个回调接口，对参数进行解密解签，获取支付的相关信息（支付的状态，金额等）

**面试官**：刚刚后台的框架的用到了 SpringMVC，简单阐述一下 SpringMVC 的执行流程

**应聘者**：

第一步：发起请求到前端控制器(DispatcherServlet)；

第二步：前端控制器请求 HandlerMapping 查找 Handler，可以根据 XML 配置、注解进行查找；

第三步：处理器映射器 HandlerMapping 向前端控制器返回 Handler；

第四步：前端控制器调用处理器适配器去执行 Handler；

第五步：处理器适配器去执行 Handler；

第六步：Handler 执行完成给适配器返回 ModelAndView；

第七步：处理器适配器向前端控制器返回 ModelAndView，ModelAndView 是 Springmvc 框架的一个底层对象，包括 Model 和 view；

第八步：前端控制器请求视图解析器去进行视图解析，根据逻辑视图名解析成真正的视图(jsp)；

第九步：视图解析器向前端控制器返回 View；

第十步：前端控制器进行视图渲染视图渲染将模型数据(在 ModelAndView 对象中)填充到 request 域。

**面试官**：你对未来有什么职业规划么

**应聘者**：由于刚出校门步入社会，在未来一年内成长为能够独立进行项目的开发，未来三年内能够负责项目的开发上线，未来 5 年内能够独立负责一个部门的运转。

# -面试/笔试题

## 1.【简答题】【数据结构与算法】二分法查找的原理

1、只要低位下标不大于高位下标，就进行二分查找（步骤 1-3）；

2、先在有序的数组中对半查找中间的坐标，如果中标和要查找的下标相等时，找到目标数，那二分结束；

3、如果步骤 2 没有找到，那就会出现先 2 种情况：a、中标大于 find 值;b、中标小于 find 值；

3.1、如果中标大于 find 值，说明 find 值在中标的左边，那么高位就是此时的中标，然后继续二分；

3.2、如果中标小于 find 值，说明 find 值在中标的右边，那么低位就是此时的中标，然后继续二分；

4、如果低位下标大于高位下标：那就是没有这个想要查找的 find 值，且低位和高位一定是相挨着（类似：low[4]，high[3]，返回-5）。返回此 find 值本应插入的 负下标-1 即：-low – 1。

## 2.【简答题】【数据结构与算法】数据结构的定义。

数据结构是指相互之间存在着一种或多种关系的数据元素的集合和该集合中数据元素之间的关系组成。

## 3.【简答题】【数据结构与算法】存储结构由哪两种基本的存储方法实现？

主要分顺序存储结构和链式存储结构两种。

## 4.【简答题】【数据结构与算法】头指针和头结点的区别

头指针是指在第一个结点之前的指针，它是一个链表存在的标志，是必须存在必不可少的。

头结点是第一个结点之前的结点，它是为了方面在第一个结点之前进行元素的插入和删除操作，它不是必须的，并且数据域也可以不存放信息。

## 5.【简答题】【数据结构与算法】栈和队列的区别

栈是只能在一端进行插入和删除的线性表，插入和删除都在栈顶进行，它的特点是“先进后出”。常用于浏览器的回退或者是括号的匹配问题，递归问题，但是递归问题要注意堆栈的溢出现象；

队列是在一端插入在另一端删除的线性表，插入的那端是队尾，删除的那端是队首，特点是“先进先出”，在层次遍历和 BFS 算法、狄杰斯特拉算法中使用到。

**6.【简答题】【数据结构与算法】给定一组数据（6,2,7,10,3,12），以它构造一棵哈夫曼树，请求出树高及树的带权路径长度 wpl 的值**

wpl=（2+3）×4+6×3+（7+10+12）×2=100

**7.【简答题】【数据结构与算法】树中节点的关系**

二叉树叶子节点总比度为 2 的节点数多 1；

树中节点的总数 = 度数*各度节点个数+1。

**8.【简答题】【数据结构与算法】顺序队的“假溢出”是怎样产生的？如何知道循环队列是空还是满？**

一般的一维数组队列的尾指针已经到了数组的上界，不能再有入队操作，但其实数组中还有空位置，这就叫“假溢出”。采用循环队列是解决假溢出的途径。

**9.【简答题】【数据结构与算法】众所周知数据结构中非常基本的树结构包括二叉查找树（BST）。当我们把如下序列：10，5，19，4，13，7，6，3，1 按顺序建立一棵 BST 时，树的最大深度是？**

提示：令根节点深度为 0，执行不进行平衡的基本插入。

**10.【简答题】【数据结构与算法】说下你对索引的理解，以及数据库索引的数据结构，为什么会被设计为 b 树或者 B+树**

索引是为了更快的查找效率；

b 树 b+树的平均查找长度小，树的左右子树均衡，可以有效的减少磁盘访问次数提高查找速度 。

**11.【简答题】【数据结构与算法】简述下列术语：数据，数据元素、数据对象、数据结构、存储结构、数据类型和抽象数据类型。**

数据：客观事物的符号表示。能输入到计算机并能被计算机程序处理的符号总称。

数据元素：数据的基本单位。

数据对象：具有相同性质的数据元素的集合，是数据的一个子集。

数据结构：相互之间存在一种或多种特定关系的数据元素的组合。

存储结构：数据结构在计算机中的表示。

数据类型：一个值的集合和操作的集合。

抽象数据类型：对一般数据类型的扩展，一个数学模型以及定义在这个模型上的一组操作。

**12.【简答题】【数据结构与算法】什么是数据的逻辑结构?基本的逻辑结构有哪些?什么是数据的物理结构?主要的物理存储方法有哪几种?**

逻辑结构就是数据在逻辑上的一种结构方式，就是用数学模型去描述的，一般分为集合、线性、树形、图形

物理结构又称存储结构，指在存储器中的存放的方式，一般分为顺序，链式、索引、散列（哈希表）等四类

**13.【简答题】【数据结构与算法】数据结构中，在逻辑上可以把数据结构分成**

从逻辑上可以将数据结构分为线性结构、非线性结构。

**14.【简答题】【数据结构与算法】从权值分别为 9,2,5,7 的四个叶子结点构造一棵哈夫曼树（Huffman）树，则该树的带权路径长度 WPL 为多少**

（2+5）*3+7*2+9*1=44

**15.【简答题】【数据结构与算法】在设计快速排序法的非递归算法时，通常利用了一个堆栈来记录待排序区间的首、尾两个端点的位置，而实际上也可以利用其他数据结构（如队列）来代替这个堆栈。请说明其中的理由。**

栈与队列的不同在于：栈是先进后出，队列是先进先出。 要想让队列实现栈的功能，则让队列模拟出先进后出就可以了。 使用两个队列 q1 和 q2 数据先进入 q1 除队尾元素外，把数据从 q1（q2）送入 q2（q1）； 反复进行，直到 q1 和 q2 为空 。

**16.【选择题】【数据结构与算法】在“快速排序”、“堆排序”和“归并排序”三个排序算法中，当只需获取前几个最小关键字记录时，选取何种排序算法为最佳？**

堆排序

**17.【简答题】【数据结构与算法】假定一个初始堆为(1, 5, 3, 9, 12, 7, 15, 10)，则进行第一趟堆排序后得到的结果为什么**

3 5 7 9 12 10 15 1

**18.【简答题】【数据结构与算法】已知待排序列以下,利用二路归并排序进行按小到大排序,除了最终结果外,要求写出每一趟排序的结果.初始序列为：[8] [4] [5] [6] [2] [1] [7] [3]**

1、4 8 5 6 1 2 3 7

2、4 5 6 8 1 2 3 7

3、1 2 3 4 5 6 7 8

**19.【简答题】【数据结构与算法】一组记录的排序码为(25，48，16，35，79，82，23，40)，其中含有 4 个长度为 2 的有序表，按归并排序的方法对该序列进行一趟归并后的结果为什么**

16 25 35 48 23 40 79 82 36 72

**20.【简答题】【数据结构与算法】一棵度为 2 的有序树与一棵二叉树的区别**

并非在任何情况下折半查找都比顺序查找快。例如，若待查元素是该顺序表的第一个元素，则顺序查找顺序表会更快。对有序顺序表采用顺序查找，若元素存在表中，则在任一位置，查找都可能成功。同样，若元素不在表中，则在任一位置， 查找都可能结束。折半查找必须经过一系列计算，方知查找成功还是失败。尽管如此，一般说来，在大多数情况下，折半查找还是比顺序查找快。

**21.【编程题】【数据结构与算法】说一下什么是二分法？使用二分法时需要注意什么？如何用代码实现？**

二分法查找（Binary Search）也称折半查找，是指当每次查询时，将数据分为前后两部分，再用中值和待搜索的值进行比较，如果搜索的值大于中值，则使用同样的方式（二分法）向后搜索，反之则向前搜索，直到搜索结束为止。

二分法使用的时候需要注意：二分法只适用于有序的数据，也就是说，数据必须是从小到大，或是从大到小排序的。

```
public class Lesson7_4 {
    public static void main(String[] args) {
        // 二分法查找
        int[] binaryNums = {1, 6, 15, 18, 27, 50};
        int findValue = 27;
        int binaryResult = binarySearch(binaryNums, 0, binaryNums.length - 1, findValue);
        System.out.println("元素第一次出现的位置（从 0 开始）：" + binaryResult);
    }
    /**
     * 二分查找，返回该值第一次出现的位置（下标从 0 开始）
     * @param nums     查询数组
     * @param start    开始下标
     * @param end      结束下标
     * @param findValue 要查找的值
     * @return int
     */
    private static int binarySearch(int[] nums, int start, int end, int findValue) {
        if (start <= end) {
            // 中间位置
            int middle = (start + end) / 2;
            // 中间的值
            int middleValue = nums[middle];
            if (findValue == middleValue) {
                // 等于中值直接返回
                return middle;
            } else if (findValue < middleValue) {
                // 小于中值，在中值之前的数据中查找
                return binarySearch(nums, start, middle - 1, findValue);
            } else {
                // 大于中值，在中值之后的数据中查找
                return binarySearch(nums, middle + 1, end, findValue);
            }
        }
        return -1;
    }
}
```

执行结果如下：

元素第一次出现的位置（从 0 开始）：4

## 22.【编程题】【数据结构与算法】什么是斐波那契数列？用代码如何实现？

斐波那契数列（Fibonacci Sequence），又称黄金分割数列、因数学家列昂纳多·斐波那契（Leonardoda Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233，377，610，987，1597，2584，4181，6765，10946，17711…… 在数学上，斐波那契数列以如下被以递推的方法定义：

F(1)=1，F(2)=1, F(n)=F(n-1)+F(n-2)（n>=3，n∈N*）在现代物理、准晶体结构、化学等领域，斐波纳契数列都有直接的应用。

斐波那契数列之所以又称黄金分割数列，是因为随着数列项数的增加，前一项与后一项之比越来越逼近黄金分割的数值 0.6180339887……

斐波那契数列指的是这样一个数列：0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233，377，610，987，1597，2584，4181，6765，10946，17711……

斐波那契数列的特征：第三项开始（含第三项）它的值等于前两项之和。

斐波那契数列代码实现示例，如下所示：

```
public class Lesson7_4 {
    public static void main(String[] args) {
        // 斐波那契数列
        int fibonacciIndex = 7;
        int fibonacciResult = fibonacci(fibonacciIndex);
        System.out.println("下标(从 0 开始)" + fibonacciIndex + "的值为：" + fibonacciResult);
    }
    /**
     * 斐波那契数列
     * @param index 斐波那契数列的下标（从 0 开始）
     * @return int
     */
    private static int fibonacci(int index) {
        if (index == 0 || index == 1) {
            return index;
        } else {
            return fibonacci(index - 1) + fibonacci(index - 2);
        }
    }
```

```
}
```

执行结果如下：

下标(从 0 开始)7 的值为：13

## 23.【简答题】【数据结构与算法】冒泡排序，冒泡排序的优化方案。

原理：比较两个相邻的元素，将值大的元素交换至右端。

思路：设数组的长度为 N：

比较前后相邻的二个数据，如果前面数据大于后面的数据，就将这二个数据交换。

这样对数组的第 0 个数据到 N-1 个数据进行一次遍历后，最大的一个数据就“沉”到数组第 N-1 个位置。

N=N-1，如果 N 不为 0 就重复前面二步，否则排序完成。

优化一：如果对于一个本身有序的序列，或则序列后面一大部分都是有序的序列，上面的算法就会浪费很多的时间开销，这里设置一个标志 flag，如果这一趟发生了交换，则为 true，否则为 false。明显如果有一趟没有发生交换，说明排序已经完成。

优化二：比如，现在有一个包含 1000 个数的数组，仅前面 100 个无序，后面 900 个都已排好序且都大于前面 100 个数字，那么在第一趟遍历后，最后发生交换的位置必定小于 100，且这个位置之后的数据必定已经有序了，也就是这个位置以后的数据不需要再排序了，于是记录下这位置，第二次只要从数组头部遍历到这个位置就可以了。如果是对于上面的冒泡排序算法 2 来说，虽然也只排序 100 次，但是前面的 100 次排序每次都要对后面的 900 个数据进行比较，而对于现在的排序算法 3，只需要有一次比较后面的 900 个数据，之后就会设置尾边界，保证后面的 900 个数据不再被排序。

## 24.【简答题】【数据结构与算法】解释算法的时间复杂度？

算法的时间复杂度表示程序运行完成所需的总时间，它通常用大 O 表示法来表示。

## 25.【简答题】【数据结构与算法】解释是否可以使用二分法检索链表？

由于随机访问在链表中是不可接受的，所以不可能到达 O（1）时间的中间元素。因此，对于链表来说，二分法检索是不可以的（对顺序链表或排序后的链表是可以用的）。

## 26.【简答题】【数据结构与算法】解释什么是“哈希算法”，它们用于什么？

“哈希算法”是一个哈希函数，它使用任意长度的字符串，并将其减少为唯一的固定长度字符串。它用于密码有效性、消息和数据完整性以及许多其他加密系统。

## 27.【简答题】【数据结构与算法】解释加密算法的工作原理？

加密是将明文转换为称为“密文”的密码格式的过程。要转换文本，算法使用一系列被称为“键”的位来进行计算。密钥越大，创建密文的潜在模式数越多。大多数加密算法使用长度约为 64 到 128 位的固定输入块，而有些则使用流方法。

## 28.【简答题】【数据结构与算法】解释什么是堆排序？

堆排序可以看成是选择排序的改进，它可以定义为基于比较的排序算法。它将其输入划分为未排序和排序的区域，通过不断消除最小元素并将其移动到排序区域来收缩未排序区域。

## 29.【简答题】【数据结构与算法】解释一个算法的最佳情况和最坏情况之间有什么区别？

最佳情况：算法的最佳情况解释为算法执行最佳的数据排列。例如，我们进行二分法检索，如果目标值位于正在搜索的数据中心，则这就是最佳情况，最佳情况时间复杂度为 0。

·最差情况：给定算法的最差输入参考。例如快速排序，如果选择关键值的子列表的最大或最小元素，则会导致最差情况出现，这将导致时间复杂度快速退化到 O（n2）。

## 30.【简答题】【数据结构与算法】解释什么是递归算法？

递归算法是一个解决复杂问题的方法，将问题分解成较小的子问题，直到分解的足够小，可以轻松解决问题为止。通常，它涉及一个调用自身的函数。

## 31.【简答题】【数据结构与算法】提到递归算法的三个定律是什么？

所有递归算法必须遵循三个规律

递归算法必须有一个基点

递归算法必须有一个趋向基点的状态变化过程

递归算法必须自我调用

## 32.【简答题】【数据结构与算法】试举一列，说明对相同的逻辑结构，同一种运算在不同的存储方式下实现时，其运算效率不同。

线性表既可以用顺序存储方式实现，又可以用链式存储方式实现。在顺序存储方式下，在线性表中插入和删除元素，平均要移动近一半的元素，时间复杂度为 0（n），而在链式存储方式下，插入和删除的时间复杂度都是 O(1)。

## 33.【简答题】【数据结构与算法】递归进层时需要做哪些事？

需要做三件事：

保留本层参数与返回地址；

为被调用函数的局部变量分配存储区，给下层函数赋值；

将程序转移到被调函数入口。

## 34.【简答题】【数据结构与算法】简述选择排序的核心思想。

第一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后再从剩余的未排序元素中寻找到最小（大）元素，然后放到已排序的序列的末尾。以此类推，直到全部待排序的数据元素的个数为零。

## 35.【简答题】【数据结构与算法】简要叙述 B 树和 B+树区别

（1）.B 树每个节点都存储数据，所有节点组成这棵树。B+树只有叶子节点存储数据（B+数中有两个头指针：一个指向根节点，另一个指向关键字最小的叶节点），叶子节点包含了这棵树的所有数据，所有的叶子结点使用链表相连，便于区间查找和遍历，所有非叶节点起到索引作用。

(2).B 树中叶节点包含的关键字和其他节点包含的关键字是不重复的，B+树的索引项只包含对应子树的最大关键字和指向该子树的指针，不含有该关键字对应记录的存储地址。

(3).B 树中每个节点（非根节点）关键字个数的范围为[m/2(向上取整)-1,m-1](根节点为[1,m-1])，并且具有 n 个关键字的节点包含（n+1）棵子树。B+树中每个节点（非根节点）关键字个数的范围为[m/2(向上取整),m](根节点为[1,m])，具有 n 个关键字的节点包含（n）棵子树。

(4).B+树中查找，无论查找是否成功，每次都是一条从根节点到叶节点的。

## 36.【简答题】【数据结构与算法】在单链表中，什么是头结点？什么是头指针？什么是首元结点？

头结点：在单链表的第一个结点之前附设一个结点，称为头结点

头指针：指向链表中第一个结点（单链表由一个头指针唯一确定）的指针（指针指的是存储地址）

首元结点：指链表中存储线性表中第一个数据元素 a1 的结点。为了操作方便，通常在链表的首元结点之前附设一个结点，称为头结点.

## 37.【简答题】【数据结构与算法】简述顺序表和链表的优缺点

（1）顺序表

优点：

顺序表的内存空间连续。

尾插、尾删效率较高，时间复杂度是 O(1)。

支持随机访问，可以高效的按下标进行操作，时间复杂度是 O(1)。

缺点：

在顺序表中间插入或删除元素时都涉及到元素的移动，效率较低，时间复杂度为 O(N)。

顺序表长度固定，有时需要扩容。

链表

优点

链表的内存空间不连续。

如果知道要处理节点的前一个位置，则进行插入和删除的复杂度为 O(1);

如果不知道要处理节点的前一个位置，则进行插入和删除的复杂度为 O(N)。

头插、头删的效率高，时间复杂度是 O(1)。

没有空间限制，不会溢出，可以存储很多元素。

缺点：

链表不支持随机访问，查找元素效率低，需要遍历节点，时间复杂度是 O(1)。

## 38.【简答题】【数据结构与算法】栈结构与队列的区别？

栈（stack）：限定只能在表的一端进行插入和删除操作的线性表。

队列（queue）：限定只能在表的一端插入和在另一端进行删除操作的线性表。

1）队列先进先出，栈先进后出。

2）对插入和删除操作的“限定”不同。

3）遍历数据速度不同。队列遍历数据的速度要快得多。

## 39.【简答题】【数据结构与算法】简要说明循环队列的优缺点。

循环队列的优点：可以有效的利用资源。用数组实现队列时，如果不移动，随着数据的不断读写，会出现假满队列的情况。即尾数组已满但头数组还是空的；循环队列也是一种数组，只是它在逻辑上把数组的头和尾相连，形成循环队列， 当数组尾满的时候，要判断数组头是否为空，不为空继续存放数据。

循环队列的缺点：循环队列中，由于入队时尾指针向前追赶头指针；出队时头指针向前追赶尾指针，造成队空和队满时头尾指针均相等。因此，无法通过条件 front==rear 来判别队列是"空"是"满"。

解决这个问题有两个办法：一是增加一个参数，用来记录数组中当前元素的个数；第二个办法是，少用一个存储空间，也就是数组的最后一个存数空间不用，当（rear+1）%maxsiz=front 时，队列满。

## 40.【简答题】【数据结构与算法】谈谈对 KMP 算法的理解？

KMP 算法是一种用于模式匹配的算法，其主要的特点是以空间换取时间，在提前得到 next 数组信息后，利用该信息减少指针回溯，提高了匹配效率。主要的原理是利用了提前计算好匹配串内 next 数组的信息，该数组记录了匹配串内每位字符内存在的最长的匹配前后缀。（谈到算法的原理即可）

## 41.【简答题】【Java 语言基础】String 是最基本的数据类型吗?

基本数据类型包括 byte、int、char、long、float、double、boolean 和 short。

java.lang.String 类是 final 类型的，因此不可以继承这个类、不能修改这个类。为了提高效率节省空间，我们应该用 StringBuffer 类。

## 42.【简答题】【Java 语言基础】int 和 Integer 有什么区别

Java 提供两种不同的类型：引用类型和原始类型（或内置类型）。Int 是 java 的原始数据类型，Integer 是 java 为 int 提供的封装类。Java 为每个原始类型提供了封装类。

引用类型和原始类型的行为完全不同，并且它们具有不同的语义。引用类型和原始类型具有不同的特征和用法，它们包括：大小和速度问题，这种类型以哪种类型的数据结构存储，当引用类型和原始类型用作某个类的实例数据时所指定的缺省值。对象引用实例变量的缺省值为 null，而原始类型实例变量的缺省值与它们的类型有关。

## 43.【简答题】【Java 语言基础】Collection 和 Collections 的区别

Collection 是集合类的上级接口，继承与他的接口主要有 Set 和 List.

Collections 是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

## 44.【简答题】【Java 语言基础】&和&&的区别

&是位运算符，表示按位与运算，&&是逻辑运算符，表示逻辑与（and）。

## 45.【简答题】【Java 语言基础】List、Map、Set 三个接口，存取元素时，各有什么特点？

List 以特定次序来持有元素，可有重复元素。Set 无法拥有重复元素,内部排序。Map 保存 key-value 值，value 可多值。

## 46.【简答题】【Java 异常与处理】error 和 exception 有什么区别?

error 表示恢复不是不可能但很困难的情况下的一种严重问题。比如说内存溢出。不可能指望程序能处理这样的情况。

exception 表示一种设计或实现问题。也就是说，它表示如果程序运行正常，从不会发生的情况

## 47.【简答题】【Java 多线程与并发】同步和异步有何异同，在什么情况下分别使用他们？举例说明。

如果数据将在线程间共享。例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行同步存取。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。

## 48.【简答题】【Java 基础语言】heap 和 stack 有什么区别？

栈是一种线形集合，其添加和删除元素的操作应在同一段完成。栈按照后进先出的方式进行处理。堆是栈的一个组成元素

## 49.【简答题】【Java 基础语言】Math.round(11.5)等於多少？Math.round(-11.5)等於多少?

Math.round(11.5)==12

Math.round(-11.5)==-11

round 方法返回与参数最接近的长整数，参数加 1/2 后求其 floor。

## 50.【简答题】【Java 基础语言】Java 有没有 goto？

java 中的保留字，现在没有在 java 中使用。

## 51.【简答题】【Java 多线程与并发】启动一个线程是用 run()还是 start()?

启动一个线程是调用 start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由 JVM 调度并执行。这并不意味着线程就会立即运行。run()方法可以产生必须退出的标志来停止一个线程。

## 52.【简答题】【Java 基础语言】构造器 Constructor 是否可被 override?

构造器 Constructor 不能被继承，因此不能重写 Overriding，但可以被重载 Overloading。

## 53.【简答题】【Java 面向对象】当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递?

是值传递。Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的。

## 54.【简答题】【Java 网络编程】socket 通信（tcp/udp 区别及 JAVA 的实现方式）

TCP——传输控制协议，具有极高的可靠性，保证数据包按照顺序准确到达，但其也有着很高的额外负担。
UDP——使用者数据元协议，并不能保证数据包会被成功的送达，也不保证数据包到达的顺序，但传输速度很快。
大多数我们会使用 TCP，偶尔才会动用 UDP，如声音讯号，即使少量遗失，也无关紧要。

## 55.【简答题】【Java 基础语言】JAVA 的四种基本权限的定义

public private protected 默认。

## 56.【简答题】【JDBC 编程】Java 中访问数据库的步骤，Statement 和 PreparedStatement 之间的区别。

访问数据库的步骤是：
1、首先加载驱动
2、提供 JDBC 连接的 URL
3、创建数据库的连接
4、创建一个 statement 执行者
5、执行 SQL 语句
6、处理返回结果
7、关闭 JDBC 对象
PreparedStatement 对象与 Statement 对象的不同点在于它的 SQL 语句是预编译过的，并且可以有占位符使用运行时参数。

## 57.【简答题】【Java 基础语言】 String s = new String("xyz");创建了几个 String Object?

两个对象，一个是“xyx”,一个是指向“xyx”的引用对象 s。

## 58.【简答题】【Java 基础语言】一个“.java”源文件中是否可以包括多个类（不是内部类）？有什么限制？

可以，必须只有一个类名与文件名相同。

## 59.【简答题】【Java 基础语言】什么时候用 assert

assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。在实现中，assertion 就是在程序中的一条语句，它对一个 boolean 表达式进行检查，一个正确程序必须保证这个 boolean 表达式的值为 true；如果该值为 false，说明程序已经处于不正确的状态下，系统将给出警告或退出。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion 检查通常是关闭的。

## 60.【简答题】【Java 语言基础】char 型变量中能不能存贮一个中文汉字?为什么?

char 型变量是用来存储 Unicode 编码的字符的，unicode 编码字符集中包含了汉字，所以， char 型变量中当然可以存储汉字啦。不过，如果某个特殊的汉字没有被包含在 unicode 编码字符集中，那么，这个 char 型变量中就不能存储这个特殊汉字。补充说明：unicode 编码占用两个字节，所以，char 类型的变量也是占用两个字节。

备注：后面一部分回答虽然不是在正面回答题目，但是，为了展现自己的学识和表现自己对问题理解的透彻深入，可以回答一些相关的知识，做到知无不言，言无不尽。

## 61.【简答题】【Java 文件与流】字节流和字符流哪个好？怎么选择？

1. 大多数情况下使用字节流会更好，因为大多数时候 IO 操作都是直接操作磁盘文件，所以这些流在传输时都是以字节的方式进行的（图片等都是按字节存储的）

2. 如果对于操作需要通过 IO 在内存中频繁处理字符串的情况使用字符流会好些，因为字符流具备缓冲区，提高了性能。

## 62.【简答题】【Java 文件与流】什么是缓冲区？有什么作用？

1. 缓冲区就是一段特殊的内存区域，很多情况下当程序需要频繁地操作一个资源（如文件或数据库）则性能会

很低，所以为了提升性能就可以将一部分数据暂时读写到缓存区，以后直接从此区域中读写数据即可，这样就显著提升了性。

2. 对于 Java 字符流的操作都是在缓冲区操作的，所以如果我们想在字符流操作中主动将缓冲区刷新到文件则可以使用 flush() 方法操作。

## 63.【简答题】【Java 文件与流】什么是 Java 序列化，如何实现 Java 序列化？

序列化就是一种用来处理对象流的机制，将对象的内容进行流化。可以对流化后的对象进行读写操作，可以将流化后的对象传输于网络之间。序列化是为了解决在对象流读写操作时所引发的问题。

序列化的实现：将需要被序列化的类实现 Serialize 接口，没有需要实现的方法，此接口只是为了标注对象可被序列化的，然后使用一个输出流（如：FileOutputStream）来构造一个 ObjectOutputStream(对象流)对象，再使用 ObjectOutputStream 对象的 write(Object obj)方法就可以将参数 obj 的对象写出。

## 64.【简答题】【Java 文件与流】BufferedReader 属于哪种流,它主要是用来做什么的,它里面有那些经典的方法？

属于处理流中的缓冲流，可以将读取的内容存在内存里面，有 readLine()方法，它用来读取一行。

## 65.【简答题】【Java 文件与流】流一般需要不需要关闭,如果关闭的话在用什么方法，一般要在那个代码块里面关闭比较好，处理流是怎么关闭的，如果有多个流互相调用传入是怎么关闭的？

流一旦打开就必须关闭，使用 close 方法；

放入 finally 语句块中（finally 语句一定会执行）；

调用的处理流就关闭处理流；

多个流互相调用只关闭最外层的流；

## 66.【简答题】【Java 文件与流】InputStream 里的 read()返回的是什么，read(byte[] data)是什么意思，返回的是什么值？

返回的是所读取的字节的 int 型（范围 0-255）；

read（byte [ ] data）将读取的字节储存在这个数组，返回的就是传入数组参数个数。

## 67.【简答题】【Java 文件与流】OutputStream 里面的 write()是什么意思,write(byte b[], int off, int len)这个方法里面的三个参数分别是什么意思？

write 将指定字节传入数据源；

Byte b[ ]是 byte 数组；

b[off]是传入的第一个字符、b[off+len-1]是传入的最后的一个字符 、len 是实际长度。

## 68.【简答题】【JDBC 编程】execute，executeQuery，executeUpdate 的区别是什么？

1、Statement 的 execute(String query)方法用来执行任意的 SQL 查询，如果查询的结果是一个 ResultSet，这个方法就返回 true。如果结果不是 ResultSet，比如 insert 或者 update 查询，它就会返回 false。

2、Statement 的 executeQuery(String query)接口用来执行 select 查询，并且返回 ResultSet。即使查询不到记录返回的 ResultSet 也不会为 null。我们通常使用 executeQuery 来执行查询语句，这样的话如果传进来的是 insert 或者 update 语句的 话，它会抛出错误信息为 “executeQuery method can not be used for update”的 java.util.SQLException。

3、Statement 的 executeUpdate(String query)方法用来执行 insert 或者 update/delete（DML）语句。

4、只有当你不确定是什么语句的时候才应该使用 execute()方法，否则应该使用 executeQuery 或者 executeUpdate 方法。

## 69.【简答题】【JDBC 编程】JDBC 的 PreparedStatement 是什么？

PreparedStatement 对象代表的是一个预编译的 SQL 语句。用它提供的 setter 方法可以传入查询的变量。由于 PreparedStatement 是预编译的，通过它可以将对应的 SQL 语句高效的执行多次。由于 PreparedStatement 自动对特殊字符转义，避免了 SQL 注入攻击，因此应当尽量的使用它。

## 70.【简答题】【JDBC 编程】说说事务的概念，在 JDBC 编程中处理事务的步骤。

1、事务是作为单个逻辑工作单元执行的一系列操作；

2、一个逻辑工作单元必须有四个属性，称为原子性、一致性、隔离性和持久性 (ACID) 属性， 只有这样才能成为一个事务；

事务处理步骤：

3、conn.setAutoComit(false);设置提交方式为手工提交；

4、conn.commit()提交事务；

5、出现异常，回滚 conn.rollback()。

## 71.【简答题】【JDBC 编程】数据库连接池的原理。为什么要使用连接池。

1、数据库连接是一件费时的操作，连接池可以使多个操作共享一个连接。

2、数据库连接池的基本思想就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。我们可以通过设定连接池最大连接数来防止系统无尽的与数据库连接。 更为重要的是我们可以通过连接池的管理机制监视数据库的连接的数量、使用情况，为系统开发，测试及性能调整提供依据。

3、使用连接池是为了提高对数据库连接资源的管理。

## 72.【简答题】【JDBC 编程】JDBC 的脏读是什么？哪种数据库隔离级别能防止脏读？

当我们使用事务时，有可能会出现这样的情况，有一行数据刚更新，与此同时另一个查询读到了这个刚更新的值。这样就导致了脏读，因为更新的数据还没有进行持久化，更新这行数据的业务可能会进行回滚，这样这个数据就是无效的。

数据库的 TRANSACTIONREADCOMMITTED，TRANSACTIONREPEATABLEREAD，和 TRANSACTION_SERIALIZABLE 隔离级别可以防止脏读。

## 73.【简答题】【JDBC 编程】JDBC 的 DriverManager 是用来做什么的？

JDBC 的 DriverManager 是一个工厂类，我们通过它来创建数据库连接。当 JDBC 的 Driver 类被加载进来时，它会自己注册到 DriverManager 类里面,然后我们会把数据库配置信息传成 DriverManager.getConnection()方法，DriverManager 会使用注册到它里面的驱动来获取数据库连接，并返回给调用的程序。

## 74.【简答题】【JDBC 编程】JDBC 的 ResultSet 是什么？

在查询数据库后会返回一个 ResultSet，它就像是查询结果集的一张数据表。ResultSet 对象维护了一个游标，指向当前的数据行。开始的时候这个游标指向的是第一行。如果调用了 ResultSet 的 next()方法游标会下移一行，如果没有更多的数据了，next()方法会返回 false。可以在 for 循环中用它来遍历数据集。默认的 ResultSet 是不能更新的，游标也只能往下移。也就是说你只能从第一行到最后一行遍历一遍。不过也可以创建可以回滚或者可更新的 ResultSet。

当生成 ResultSet 的 Statement 对象要关闭或者重新执行或是获取下一个 ResultSet 的时候，ResultSet 对象也会自动关闭。可以通过 ResultSet 的 getter 方法，传入列名或者从 1 开始的序号来获取列数据。

## 75.【简答题】【Java 语言基础】使用 final 关键字修饰一个变量时，是引用不能变，还是引用的对象不能变？

使用 final 关键字修饰一个变量时，是指引用变量不能变，引用变量所指向的对象中的内容还是可以改变的。例如，对于如下语句：

final StringBuffer a=new StringBuffer("immutable");

执行如下语句将报告编译期错误：

a=new StringBuffer("");

但是，执行如下语句则可以通过编译：

a.append(" broken!");

有人在定义方法的参数时，可能想采用如下形式来阻止方法内部修改传进来的参数对象：

public void method(final StringBuffer param){

}

实际上，这是办不到的，在该方法内部仍然可以增加如下代码来修改参数对象：

param.append("a");

## 76.【简答题】【Java 语言基础】是否可以从一个 static 方法内部发出对非 static 方法的调用？

不可以。因为非 static 方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，而 static 方法调用时不需要创建对象，可以直接调用。也就是说，当一个 static 方法被调用时，可能还没有创建任何实例对象，如果从一个 static 方法中发出对非 static 方法的调用，那个非 static 方法是关联到哪个对象上的呢？这个逻辑无法成立，所以，一个 static 方法内部发出对非 static 方法的调用。

## 77.【简答题】【Java 语言基础】java 中实现多态的机制是什么？

是父类或接口定义的引用变量可以指向子类或具体实现类的实例对象，而程序调用的方法在运行期才动态绑定，就是引用变量所指向的具体实例对象的方法，也就是内存里正在运行的那个对象的方法，而不是引用变量的类型中定义的方法。

## 78.【简答题】【Java 语言基础】什么是内部类？Static Nested Class 和 Inner Class 的不同。

内部类就是在一个类的内部定义的类，内部类中不能定义静态成员， 内部类可以直接访问外部类中的成员变量，内部类可以定义在外部类的方法外面，也可以定义在外部类的方法体中。

在方法外部定义的内部类前面可以加上 static 关键字，从而成为 Static Nested Class，它不再具有内部类的特性，所有，从狭义上讲，它不是内部类。Static Nested Class 与普通类在运行时的行为和功能上没有什么区别，只是在编程引用时的语法上有一些差别，它可以定义成 public、protected、默认的、private 等多种类型，而普通类只能定义成 public 和默认的这两种类型。在外面引用 Static Nested Class 类的名称为“外部类名.内部类名”。在外面不需要创建外部类的实例对象，就可以直接创建 Static Nested Class。

由于 static Nested Class 不依赖于外部类的实例对象，所以，static Nested Class 能访问外部类的非 static 成员变量(不能直接访问，需要创建外部类实例才能访问非静态变量)。当在外部类中访问 Static Nested Class 时，可以直接使用 Static Nested Class 的名字，而不需要加上外部类的名字了，在 Static Nested Class 中也可以直接引用外部类的 static 的成员变量，不需要加上外部类的名字。

## 79.【简答题】【Java 语言基础】内部类可以引用它的包含类的成员吗？有没有什么限制？

完全可以。如果不是静态内部类，那没有什么限制！

如果你把静态嵌套类当作内部类的一种特例，那在这种情况下不可以访问外部类的普通成员变量，而只能访问外部类中的静态成员，例如，下面的代码：

```
class Outer
{
static int x;
static class Inner
{
 void test()
                {
 syso(x);
                }
 }
}
```

答题时，也要能察言观色，揣摩提问者的心思，显然人家希望你说的是静态内部类不能访问外部类的成员，但你一上来就顶牛，这不好，要先顺着人家，让人家满意，然后再说特殊情况，让人家吃惊。

## 80.【简答题】【Java 常见类库】List, Set, Map 是否继承自 Collection 接口？

List，Set 是，Map 不是

**91.【填空题】【Java 语言基础】写出 5 个 Object 类的常用方法 _____、_____、_____、_____、_____**

Object 类是一个特殊的类，是所有类的父类。它主要提供了以下方法： getClass()、hashCode()、clone()、toString（）、finalize()、wait（）、notify()、notifyAll（）。

**92.【简答题】【Java 语言基础】怎么理解 Java 中自动装箱和拆箱？**

装箱：将基本类型用它们对应的引用类型包装起来；
拆箱：将包装类型转换为基本数据类型；

**93.【填空题】【Java 语言基础】String s1=”ab”,String s2=”a”+”b”,String s3=”a”,String s4=”b”,s5=s3+s4 请问 s5==s2 返回结果是_____**

返回 false.在编译过程中,编译器会将 s2 直接优化为”ab”,会将其放置在常量池当中,s5 则是被创建在堆区,相当于 s5=new String(“ab”);

**94.【简答题】【Java 语言基础】switch 可以接收的数据类型有哪几种？**

byte、char、short、int、enum；以上是 JDK1.6 以前的版本。JDK1.7 时，又增加了 String。

**95.【选择题】【Java 面向对象】 不通过构造函数也能创建对象吗?**

**A：是　　B：否**

答案：A

Java 创建对象的几种方式（重要）：

(1) 用 new 语句创建对象，这是最常见的创建对象的方法。

(2) 运用反射手段,调用 java.lang.Class 或者 java.lang.reflect.Constructor 类的 newInstance()实例方法。

(3) 调用对象的 clone()方法。

(4) 运用反序列化手段，调用 java.io.ObjectInputStream 对象的 readObject()方法。

**96.【选择题】【Java 面向对象】 Java 接口的修饰符可以为（）。**

**A private　B protected　C final　D abstract**

答案：CD

（1）接口用于描述系统对外提供的所有服务,因此接口中的成员常量和方法都必须是公开(public)类型的,确保外部使用者能访问它们；

（2）接口仅仅描述系统能做什么,但不指明如何去做,所以接口中的方法都是抽象(abstract)方法；

（3）接口不涉及和任何具体实例相关的细节,因此接口没有构造方法,不能被实例化,没有实例变量，只有静态（static）变量；

（4）接口的中的变量是所有实现类共有的，既然共有，肯定是不变的东西，因为变化的东西也不能够算共有。所以变量是不可变(final)类型，也就是常量了。

（5） 接口中不可以定义变量？如果接口可以定义变量，但是接口中的方法又都是抽象的，在接口中无法通过行为来修改属性。

## 97.【简答题】【Java 多线程与并发】说一说自己对 synchronized 关键字的了解。

synchronized 关键字解决的是多个线程之间访问资源的同步性，synchronized 关键字可以保证被它修饰的方法或者 代码块在任意时刻只能有一个线程执行。

## 98.【简答题】【Java 多线程与并发】说说自己是怎么使用 synchronized 关键字。

synchronized 关键字最主要的三种使用方式:

修饰实例方法，作用于当前对象实例加锁，进入同步代码前要获得当前对象实例的锁。

修饰静态方法，作用于当前类对象加锁，进入同步代码前要获得当前类对象的锁 。也就是给当前类加锁，会作用于类的所有对象实例，因为静态成员不属于任何一个实例对象，是类成员（ static 表明这是该类的一个静态资源，不管 new 了多少个对象，只有一份，所以对该类的所有对象都加了锁）。所以如果一个线程 A 调用一个实例对象的非静态 synchronized 方法，而线程 B 需要调用这个实例对象所属类的静态 synchronized 方法，是允许的，不会发生互斥现象，因为访问静态 synchronized 方法占用的锁是当前类的锁，而访问非静态 synchronized 方法占用的锁是当前实例对象锁。

修饰代码块，指定加锁对象，对给定对象加锁，进入同步代码库前要获得给定对象的锁。 和 synchronized 方法一样，synchronized(this)代码块也是锁定当前对象的。synchronized 关键字加到 static 静态方法和 synchronized(class)代码块上都是是给 Class 类上锁。这里再提一下：synchronized 关键字加到非 static 静态方法上是给对象实例上锁。另外需要注意的是：尽量不要使用 synchronized(String a) 因为 JVM 中，字符串常量池具有缓冲功能.

## 99.【简答题】【Java 常见类库】谈一谈 ArrayList 与 Vector 区别。

Vector 类的所有方法都是同步的。可以由两个线程安全地访问一个 Vector 对象、但是一个线程访问 Vector 的话代码要在同步操作上耗费大量的时间。

Arraylist 不是同步的，所以在不需要保证线程安全时时建议使用 Arraylist。

## 100.【简答题】【Java 常见类库】谈一谈 Arraylist 与 LinkedList 异同

1. 是否保证线程安全： ArrayList 和 LinkedList 都是不同步的，也就是不保证线程安全；

2. 底层数据结构： Arraylist 底层使用的是 Object 数组；LinkedList 底层使用的是双向链表数据结构（JDK1.6 之前为循环链表，JDK1.7 取消了循环。注意双向链表和双向循环链表的区别：）； 详细可阅读 JDK1.7-LinkedList 循环链表优化；

3. 插入和删除是否受元素位置的影响： ① ArrayList 采用数组存储，所以插入和删除元素的时间复杂度受元素位置的影响。 比如：执行 add(E e) 方法的时候， ArrayList 会默认在将指定的元素追加到此列表的末尾，这种情况时间复杂度就是 O(1)。但是如果要在指定位置 i 插入和删除元素的话（ add(int index, E element) ）时间复杂度就为 O(n-i)。因为在进行上述操作的时候集合中第 i 和第 i 个元素之后的(n-i)个元素都要执行向后位/向前移一位的操作。② LinkedList 采用链表存储，所以插入，删除元素时间复杂度不受元素位置的影响，都是近似 O（1）而数组为近似 O（n）；

4. 是否支持快速随机访问： LinkedList 不支持高效的随机元素访问，而 ArrayList 支持。快速随机访问就是通过元素的序号快速获取元素对象(对应于 get(int index) 方法)；

5. 内存空间占用： ArrayList 的空 间浪费主要体现在在 list 列表的结尾会预留一定的容量空间，而 LinkedList 的空间花费则体现在它的每一个元素都需要消耗比 ArrayList 更多的空间（因为要存放直接后继和直接前驱以及数据）。

## 101.【简答题】【Java 常见类库】谈一谈 HashMap 的实现原理？

HashMap 概述： HashMap 是基于哈希表的 Map 接口的非同步实现。此实现提供所有可选的映射操作，并允许使用 null 值和 null 键。此类不保证映射的顺序，特别是它不保证该顺序恒久不变。

HashMap 的数据结构： HashMap 实际上是一个“链表散列”的数据结构，即数组和链表的结合体。

当我们往 HashMap 中 put 元素时,首先根据 key 的 hashcode 重新计算 hash 值，根据 hash 值得到这个元素在数组中的位置(下标)，如果该数组在该位置上已经存放了其他元素，那么在这个位置上的元素将以链表的形式存放,新加入的放在链头，最先加入的放入链尾。如果数组中该位置没有元素，就直接将该元素放到数组的该位置上。

Jdk1.8 中对 HashMap 的实现做了优化，当链表中的节点数据超过八个之后，该链表会转为红黑树来提高查询效率，从原来的 O(n) 到 O(logn)。

## 102.【简答题】【Java 异常与处理】列举 Throwable 类常用方法。

public string getMessage():返回异常发生时的详细信息。

public string toString():返回异常发生时的简要描述。

public string getLocalizedMessage():返回异常对象的本地化信息。使用 Throwable 的子类覆盖这个方法，可以声称本地化信息。如果子类没有覆盖该方法，则该方法返回的信息与 getMessage（）返回的结果相同。

public void printStackTrace():在控制台上打印 Throwable 对象封装的异常信息。

## 103.【简答题】【Java 异常与处理】说一说在哪些情况下 finally 块不会被执行。

1. 在 fifinally 语句块中发生了异常。
2. 在前面的代码中用了 System.exit()退出程序。
3. 程序所在的线程死亡。
4. 关闭 CPU。

## 104.【选择题】【Java 文件与流】下面哪个流类属于面向字符的输入流。

**A、 BufferedWriter**

**B、 FileInputStream**

**C、 ObjectInputStream**

**D、 InputStreamReader**

答案：D

面向字节的操作为以 8 位为单位对二进制的数据进行操作，对数据不进行转换，这些类都是 InputStream 和 OutputStream 的子类。

面向字符的操作为以字符为单位对数据进行操作，在读的时候将二进制数据转为字符，在写的时候将字符转为二进制数据，这些类都是 Reader 和 Writer 的子类。

## 105.【简答题】【Java 语言基础】HashMap 和 Hashtable 的区别。

HashMap 是 Hashtable 的轻量级实现（非线程安全的实现），他们都完成了 Map 接口，主要区别在于 HashMap 允许空（null）键值（key）,由于非线程安全，效率上可能高于 Hashtable。

最大的不同是，Hashtable 的方法是 Synchronize 的，而 HashMap 不是，在多个线程访问 Hashtable 时，不需要自己为它的方法实现同步，而 HashMap 就必须为之提供外同步。 Hashtable 和 HashMap 采用的 hash/rehash 算法都大概一样，所以性能不会有很大的差异。

## 106.【简答题】【Java 语言基础】 Overload 和 Override 的区别。Overloaded 的方法是否可以改变返回值的类型?

方法的重写 Overriding 和重载 Overloading 是 Java 多态性的不同表现。重写 Overriding 是父类与子类之间多态性的一种表现，重载 Overloading 是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被"屏蔽"了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。Overloaded 的方法是可以改变返回值的类型。

## 107.【简答题】【Java 语言基础】 synchronized、volatile 区别

1) volatile 主要应用在多个线程对实例变量更改的场合，刷新主内存共享变量的值从而使得各个 线程可以获得最新的值，线程读取变量的值需要从主存中读取；synchronized 则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。另外，synchronized 还会创建一个内存屏障，内存屏障指令保证了所有 CPU 操作结果都会直接刷到主存中（即释放锁前），从而保证 了操作的内存可见性，同时也使得先获得这个锁的线程的所有操作

2) volatile 仅能使用在变量级别；synchronized 则可以使用在变量、方法、和类级别的。 volatile 不会造成线程的阻塞；synchronized 可能会造成线程的阻塞，比如多个线程争抢 synchronized 锁对象时，会出现阻塞。

3) volatile 仅能实现变量的修改可见性，不能保证原子性；而 synchronized 则可以保证变量的修改可见性和原子性，因为线程获得锁才能进入临界区，从而保证临界区中的所有语句全部得到执行。

4) volatile 标记的变量不会被编译器优化，可以禁止进行指令重排；synchronized 标记的变量可以被编译器优化。

## 108.【简答题】【Java 语言基础】 谈一谈多态的好处？

允许不同类对象对同一消息做出响应,即同一消息可以根据发送对象的不同而采用多种不同的行为方式(发送消息就是函数调用).主要有以下优点:

可替换性:多态对已存在代码具有可替换性；

可扩充性:增加新的子类不影响已经存在的类结构；

接口性:多态是超类通过方法签名,向子类提供一个公共接口,由子类来完善或者重写它来实现的；灵活性；简化性。

## 109.【简答题】【Java 语言基础】 说说反射的用途及实现，反射是不是很慢，我们在项目中是否要避免使用反射；

一、用途

反射被广泛地用于那些需要在运行时检测或修改程序行为的程序中。

二、实现方式

Foo foo = new Foo();

第一种：通过 Object 类的 getClass 方法

Class cla = foo.getClass();

第二种：通过对象实例方法获取对象

Class cla = foo.class;

第三种：通过 Class.forName 方式

Class cla = Class.forName("xx.xx.Foo");

三、缺点

1）影响性能

反射包括了一些动态类型，所以 JVM 无法对这些代码进行优化。因此，反射操作的效

率要比那些非反射操作低得多。我们应该避免在经常被执行的代码或对性能要求很高的程

序中使用反射。

2）安全限制

使用反射技术要求程序必须在一个没有安全限制的环境中运行。

3）内部暴露

由于反射允许代码执行一些在正常情况下不被允许的操作（比如访问私有的属性和方 法），所以 使用反射可能会导致意料之外的副作用－－代码有功能上的错误，降低可移植性。 反射代码破坏了抽象性，因此当平台发生改变的时候，代码的行为就有可能也随着变化。

## 110.【简答题】【Java 语言基础】 List 和 Map 区别

一、概述

List 是存储单列数据的集合，Map 是存储键和值这样的双列数据的集合，

List 中存储的数据是有顺序，并且允许重复，值允许有多个 null；

Map 中存储的数据是没有顺序的，键不能重复，值是可以有重复的，key 最多有一个 null。

二、明细

List

1）可以允许重复的对象。

2）可以插入多个 null 元素。

3）是一个有序容器，保持了每个元素的插入顺序，输出的顺序就是插入的顺序。

4）常用的实现类有 ArrayList、LinkedList 和 Vector。ArrayList 最为流行，它提供了使用索引的随意访问，而 LinkedList 则对于经常需要从 List 中添加或删除元素的场合更为合适。

Map

1）Map 不是 collection 的子接口或者实现类。Map 是一个接口。

2）Map 的 每个 Entry 都持有两个对象，也就是一个键一个值，Map 可能会持有相同的值对象但键对象必须是唯一的。

3）TreeMap 也通过 Comparator 或者 Comparable 维护了一个排序顺序。

4）Map 里你可以拥有随意个 null 值但最多只能有一个 null 键。

5）Map 接口最流行的几个实现类是 HashMap、LinkedHashMap、Hashtable 和 TreeMap。

（HashMap、TreeMap 最常用）

Set（问题扩展）

1）不允许重复对象

2）无序容器，你无法保证每个元素的存储顺序，TreeSet 通过 Comparator 或 Comparable 维护了一个排序顺序。

3）只允许一个 null 元素

4）Set 接口最流行的几个实现类是 HashSet、LinkedHashSet 以及 TreeSet。最流行的是基于 HashMap 实现的 HashSet；TreeSet 还实现了 SortedSet 接口，因此 TreeSet 是一个根据其 compare() 和 compareTo() 的定义进行排序的有序容器。

三、场景（问题扩展）1）如果你经常会使用索引来对容器中的元素进行访问，那么 List 是你的正确的选择。如果你已经知道索引了的话，那么 List 的实现类比如 ArrayList 可以提供更快速的访问,如果经常添加删除元素的，那么肯定要选择 LinkedList。

2）如果你想容器中的元素能够按照它们插入的次序进行有序存储，那么还是 List，因为 List 是一个有序容器，它按照插入顺序进行存储。

3）如果你想保证插入元素的唯一性，也就是你不想有重复值的出现，那么可以选择一个 Set 的实现类，比如 HashSet、LinkedHashSet 或者 TreeSet。所有 Set 的实现类都遵循了统一约束比如唯一性，而且还提供了额外的特性比如 TreeSet 还是一个 SortedSet，所有存储于 TreeSet 中的元素可以使用 Java 里的 Comparator 或者 Comparable 进行排序。LinkedHashSet 也按照元素的插入顺序对它们进行存储。

4）如果你以键和值的形式进行数据存储那么 Map 是你正确的选择。你可以根据你的后续需要从 Hashtable、HashMap、TreeMap 中进行选择。

## 111.【简答题】【Java 语言基础】 Arraylist 与 LinkedList 区别，ArrayList 与 Vector 区别；

1）数据结构

Vector、ArrayList 内部使用数组，而 LinkedList 内部使用双向链表，由数组和链表的特性知：

LinkedList 适合指定位置插入、删除操作，不适合查找；

ArrayList、Vector 适合查找，不适合指定位置的插入删除操作。

但是 ArrayList 越靠近尾部的元素进行增删时，其实效率比 LinkedList 要高

2)线程安全

Vector 线程安全，ArrayList、LinkedList 线程不安全。

3）空间

ArrayList 在元素填满容器时会自动扩充容器大小的 50%，而 Vector 则是 100%，因此 ArrayList 更节省空间。

## 112.【简答题】【Java 语言基础】GC 分哪两种，Minor GC 和 Full GC 有什么区别？什么时候会触发 Full GC？分别采用什么算法？

参考答案:


对象从新生代区域消失的过程，我们称之为 "minor GC"

对象从老年代区域消失的过程，我们称之为 "major GC"

Minor GC

清理整个 YouGen 的过程，eden 的清理，S0\S1 的清理都会由于 MinorGC Allocation

Failure(YoungGen 区内存不足），而触发 minorGC

Major GC

OldGen 区内存不足，触发 Major GC

Full GC

Full GC 是清理整个堆空间—包括年轻代和永久代

Full GC 触发的场景

1）System.gc

2）promotion failed (年代晋升失败,比如 eden 区的存活对象晋升到 S 区放不下，又尝试直接晋升到 Old 区又放不下，那么 Promotion Failed,会触发 FullGC)

3）CMS 的 Concurrent-Mode-Failure

由于 CMS 回收过程中主要分为四步: 1.CMS initial mark 2.CMS Concurrent mark 3.CMS remark 4.CMS Concurrent sweep。在 2 中 gc 线程与用户线程同时执行，那么用户线程依旧可能同时产生垃圾， 如果这个垃圾较多无法放入预留的空间就会产生 CMS-Mode-Failure， 切换为 SerialOld 单线程做 mark-sweep-compact。

4）新生代晋升的平均大小大于老年代的剩余空间 （为了避免新生代晋升到老年代失败）

当使用 G1,CMS 时，FullGC 发生的时候 是 Serial+SerialOld。

当使用 ParalOld 时，FullGC 发生的时候是 ParallNew +ParallOld。

## 113.【简答题】【Java 语言基础】Interger 常量池 ：

缓存了 -128 ～ 127 的值

## 114.【简答题】【Java 语言基础】HashMap,HashTable,LinkedHashMap 区别；

（1）HashMap 是一个最常用的 Map，它根据键的 hashCode 值存储数据，根据键可以直接获取它的值，具有很快的访问速度。HashMap 最多只允许一条记录的键为 null，不允许多条记录的值为 null。HashMap 不支持线程的同步，即任一时刻可以有多个线程同时写 HashMap，可能会导致数据的不一致。如果需要同步，可以用 Collections.synchronizedMap(HashMap map)方法使 HashMap 具有同步的能力。

（2）Hashtable 与 HashMap 类似，不同的是：它不允许记录的键或者值为空；它支持线程的同步，即任一时刻只有一个线程能写 Hashtable，然而，这也导致了 Hashtable 在写入时会比较慢。

（3）LinkedHashMap 保存了记录的插入顺序，在用 Iteraor 遍历 LinkedHashMap 时，先得到的记录肯定是先插入的。在遍历的时候会比 HashMap 慢。有 HashMap 的全部特性。

（4）TreeMap 能够把它保存的记录根据键排序，默认是按升序排序，也可以指定排序的比较器。当用 Iteraor 遍历 TreeMap 时，得到的记录是排过序的。TreeMap 的键和值都不能为空。

## 115.【简答题】【Java 语言基础】在一个.java 文件里面，可以有多个类吗，有什么限制？

Java 可以包含多个类 ，但 public 修饰的类只有一个，且类名必须与文件名一致

## 116.【简答题】【Java 语言基础】hashcode() 与 equals()区别，简单说明。

equal()相等的两个对象他们的 hashCode()肯定相等，也就是用 equal()对比是绝对可靠的。

hashCode()相等的两个对象他们的 equal()不一定相等，也就是 hashCode()不是绝对可靠的。

对于需要大量并且快速的对比的话如果都用 equal()去做显然效率太低，所以解决方式是，每当需要对比的时候，首先用 hashCode()去对比，如果 hashCode()不一样，则表示这两个对象肯定不相等（也就是不必再用 equal()去再对比了）,如果 hashCode()相同，此时再对比他们的 equal()，如果 equal()也相同，则表示这两个对象是真的相同了，这样既能大大提高了效率也保证了对比的绝对正确性！

## 117.【简答题】【Java 面向对象】 请简述面向对象和面向过程的区别。

1）出发点不同

面向对象方法是用符合常规思维的方式来处理客观世界的问题，强调把问题域的要领直接映射到对象及对象之间的接口上。而面向过程方法强调的则是过程的抽象化与模块化，它是以过程为中心构造或处理客观世界问题的。

2）层次逻辑关系不同

面向对象方法则是用计算机逻辑来模拟客观世界中的物理存在， 以对象的集合类作为处理问题的基本单位，尽可能地使计算机世界向客观世界靠拢，以使问题的处理更清晰直接，面向对象方法是用类的层次结构来体现类之间的继承和发展。而面向过程方法处理问题的基本单位是能清晰准确地表达过程的模块，用模块的层次结构概括模块或模块间的关系与功能，把客观世界的问题抽象成计算机可以处理的过程。

3）数据处理方式与控制程序方式不同

面向对象方法将数据与对应的代码封装成一个整体，原则上其他对象不能直接修改其数据，即对象的修改只能由自身的成员函数完成，控制程序方式上是通过“事件驱动”来激活和运行程序。而面向过程方法是直接通过程序来处理数据，处理完毕后即可显示处理结果，在控制程序方式上是按照设计调用或返回程序，不能自由导航，各模块之间存在着控制与被控制、调用与被调用的关系。

4）分析设计与编码转换方式不同

分析设计与编码转换方式不同。而向对象方法贯穿于软件生命周期的分析、设计及编码中，是一种平滑过程，从分析到设计再到编码是采用一致性的模型表示，即实现的是一种无缝连接。而向过程方法强调分析、设计及编码之间按规则进行转换，贯穿于软件生命周期的分析、设计及编码中，实现的是一种有缝的连接。

## 118.【简答题】【Java 反射】反射获取一个对象的方式有哪些

1、new Object().getClass 2、Object.class 3、 Class.forName("java.util.String")

通过 Object.class 的方法获取对象的 Class 对象，根本不会调用对象中任何的代码块或代码。而 Class.forName() 会调用静态代码块的内容。

而 new Object().getClas 打印所有内容的原因很显然，就因为要先实例化对象。

## 119.【简答题】【Java 面向对象】 构造函数啥啥啥的加载顺序？

顺序：静态代码块 - 代码块 - 构造函数

父类静态变量、 父类静态代码块、 子类静态变量、 子类静态代码块、父类非静态变量（父类实例成员变量）、 父类构造函数、 子类非静态变量（子类实例成员变量）、子类构造函数。

## 120.【简答题】【Java 语言基础】数组有没有 length()这个方法? String 有没有 length()这个方法？

数组没有 length（）方法，有 length 属性，String 类有 length（）方法。

## 121.【简答题】【JavaJDBC】JDBC 的流程：

第一步： 加载驱动类， 注册驱动

第二步：通过 DriverManager,使用 url，用户名和密码建立连接(Connection)；

第三步：通过 Connection，使用 SQL 语句打开 Statement 对象；

第四步：执行语句，将结果返回 resultSet；

第五步：对结果 resultSet 进行处理；

第六步：释放资源

## 122.【简答题】【Java 常用类】File 类常用方法

1）String getName()  返回 File 对象所表示的文件名或文件路径

2）String getPath()  返回 File 对象所对应的相对路径名。

3）File getAbsoluteFile() 返回 File 对象的绝对路径文件

4）String getAbsolutePath() 返回 File 对象所对应的绝对路径名

## 123.【简答题】【Java 多线程与并发】sleep()和 wait()有什么区别?

1、每个对象都有一个锁来控制同步访问，Synchronized 关键字可以和对象的锁交互，来实现同步方法或同步块。sleep()方法正在执行的线程主动让出 CPU（然后 CPU 就可以去执行其他任务），在 sleep 指定时间后 CPU 再回到该线程继续往下执行(注意：sleep 方法只让出了 CPU，而并不会释放同步资源锁!!! )；wait()方法则是指当前线程让自己暂时退让出同步资源锁，以便其他正在等待该资源的线程得到该资源进而运行，只有调用了 notify()方法，之前调用 wait()的线程才会解除 wait 状态，可以去参与竞争同步资源锁，进而得到执行。（注意：notify 的作用相当于叫醒睡着的人，而并不会给他分配任务，就是说 notify 只是让之前调用 wait 的线程有权利重新参与线程的调度）；

2、sleep()方法可以在任何地方使用；wait()方法则只能在同步方法或同步块中使用；

3、sleep()是线程线程类（Thread）的方法，调用会暂停此线程指定的时间，但监控依然保持，不会释放对象锁，到时间自动恢复；wait()是 Object 的方法，调用会放弃对象锁，进入等待队列，待调用 notify()/notifyAll()唤醒指定的线程或者所有线程，才会进入锁池，不再次获得对象锁才会进入运行状态；

## 124.【简答题】【Java 语言基础】两个对象的 hashCode()相同，则 equals()也一定为 true，对吗？

不对，两个对象的 hashCode()相同，equals()不一定 true。，因为在散列表中，hashCode()相等即两个键值对的哈希值相等，然而哈希值相等，并不一定能得出键值对相等。

## 125.【简答题】【Java 语言基础】请简要说明静态变量有哪些特点

静态变量可以使用“类名.变量名”的方式调用;静态变量会被类的实例对象所共享;静态变量随着类的加载而加载到内存静态区;静态变量随着类的消失而消失。

## 126.【简答题】【Java 语言基础】String, Stringbuffer, StringBuilder 的区别

StringBuffer 是线程安全，可以不需要额外的同步用于多线程中;

StringBuilder 是非同步,运行于多线程中就需要使用着单独同步处理，但是速度就比 StringBuffer 快多了;

StringBuffer 与 StringBuilder 两者共同之处:可以通过 append、indert 进行字符串的操作。

String 实现了三个接口:Serializable、Comparable<String>、CarSequence

StringBuilder 只实现了两个接口 Serializable、CharSequence，相比之下 String 的实例可以通过 compareTo 方法进行比较，其他两个不可以。

## 127.【简答题】【Java 多线程与并发】多线程有哪些实现方式?

1.继承 Thread 类，重写 run 方法；

2.实现 Runnable 接口，重写 run 方法，实现 Runnable 接口的实现类的实例对象作为 Thread 构造函数的 target；

3.通过 Callable 和 FutureTask 创建线程；

4.通过线程池创建线程。

## 128.【简答题】【Java 多线程与并发】线程冲突有哪些解决方法

第一种 同步方法

第二种 同步代码块

第三种 使用特殊成员变量（volatile 成员变量）实现线程同步（前提是对成员变量的操作是原子操作）

第四种 使用 Lock 接口（java.util.concurrent.locks 包）

第五种 使用线程局部变量（thread-local）解决多线程对同一变量的访问冲突，而不能实现同步（ThreadLocal 类）

第六种 使用阻塞队列实现线程同步（java.util.concurrent 包）

第七种 使用原子变量实现线程同步 （java.util.concurrent.atomic 包）

## 129.【简答题】【Java 多线程与并发】线程池 ThreadPoolExecutor 有哪些常用的方法？

ThreadPoolExecutor 有如下常用方法：

submit()/execute()：执行线程池

shutdown()/shutdownNow()：终止线程池

isShutdown()：判断线程是否终止

getActiveCount()：正在运行的线程数

getCorePoolSize()：获取核心线程数

getMaximumPoolSize()：获取最大线程数

getQueue()：获取线程池中的任务队列

allowCoreThreadTimeOut(boolean)：设置空闲时是否回收核心线程

这些方法可以用来终止线程池、线程池监控等。

## 130.【简答题】【Java 多线程与并发】说说 submit(和 execute 两个方法有什么区别？

submit() 和 execute() 都是用来执行线程池的，只不过使用 execute() 执行线程池不能有返回方法，而使用 submit() 可以使用 Future 接收线程池执行的返回值。

## 131.【简答题】【Java 多线程与并发】线程池为什么要使用阻塞队列而不使用非阻塞队列？

线程池是采用生产者-消费者模式设计的

线程池为消费者。

在线程池中活跃线程数达到 corePoolSize 时，线程池将会将后续的 task 提交到 BlockingQueue 中,（每个 task 都是单独的生产者线程）进入到堵塞对列中的 task 线程会 wait（） 从而释放 cpu，从而提高 cpu 利用率。

## 132.【简答题】【Java 多线程与并发】同步和异步有何异同，在什么情况下分别使用他们，举例说明。

如果数据将在线程间共享。例如正在写的数据以后可能被另一个线程读到，或者正在读的数 据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行同步存取。

当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。

## 133.【简答题】【Java 语言基础】什么是哈希冲突，发生哈希冲突会有什么影响，怎么解决。

哈希冲突：对应不同的关键字可能获得相同的 hash 地址，即 key1≠key2，但是 f(key1)=f(key2)

哈希函数是从关键字集合和地址集合的映像，通常关键字集合比较大，而地址集合的元素仅为哈希表中的地址值。因此总会存在不同的数据经过计算后得到的值相同，这就是哈希冲突。

解决方法:

1.开放地址法

这种方法也称再散列法，其基本思想是：当关键字 key 的哈希地址 p=H（key）出现冲突时，以 p 为基础，产生另一个哈希地址 p1，如果 p1 仍然冲突，再以 p 为基础，产生另一个哈希地址 p2，…，直到找出一个不冲突的哈希地址 pi ，将相应元素存入其中。

就是说当发生冲突时，就去寻找下一个空的地址把数据存入其中，只要哈希表足够大，就总能找到这样一个空的地址。

2.拉链法

将所有关键字为同义字的记录存储在一个单链表中 。

3.再哈希法

在发生冲突的时候再用另外一个哈希函数算出哈希值，直到算出的哈希值不同为止。

4.建立公共溢出区

在创建哈希表的同时，再额外创建一个公共溢出区，专门用来存放发生哈希冲突的元素。查找时，先从哈希表查，查不到再去公共溢出区查。

## 134.【简答题】【Java 语言基础】Java 中 equals 与==的区别

==

基本类型：对比它们的值是否相等

引用类型：对比它们的内存地址是否相等

equals()

引用类型：默认情况下，对比它们的地址是否相等；如果 equals()方法被重写，则根据重写过程来比较

## 135.【简答题】【Servlet】说一说 Session 和 cookie 的原理和区别。

cookie 采用的是客户端的会话状态的一种储存机制。它是服务器在本地机器上存储的小段文本或者是内存中的一段数据，并随每一个请求发送至同一个服务器。

session 是一种服务器端的信息管理机制，它把这些文件信息以文件的形式存放在服务器的硬盘空间上（这是默认情况，可以用 memcache 把这种数据放到内存里面）当客户端向服务器发出请求时，要求服务器端产生一个 session 时，服务器端会先检查一下，客户端的 cookie 里面有没有 session_id，是否过期。如果有这样的 session_id 的话，服务器端会根据 cookie 里的 session_id 把服务器的 session 检索出来。如果没有这样的 session_id 的话，服务器端会重新建立一个。

区别：Cookie 保存在客户端浏览器中，而 Session 保存在服务器上。Cookie 机制是通过检查客户身上的“通行证”来确定客户身份的话，那么 Session 机制就是通过检查服务器上的“客户明细表”来确认客户身份。Session 相当于程序在服务器上建立的一份客户档案，客户来访的时候只需要查询客户档案表就可以了。

## 136.【简答题】【JavaWeb】Forward 和 Redirect 区别，简单说明

重定向是两次请求，转发是一次请求，因此转发的速度要快于重定向

转发是服务器行为，重定向是客户端行为。重定向时浏览器上的网址改变 ,转发是浏览器上的网址不变

## 137.【简答题】【JavaScript】document 操作节点的方法有哪些

document.getElementById(); ID 选择

document.getElementsByTagName(); 标签选择

document.getELementsByClassName(); 类名选择

## 138【简答题】【JavaScript】var,let 与 const 区别？

const 用来声明常量，一旦声明，其值就不能更改；在同一作用域，const 不能重复声明同一常量；

let 不允许在同一作用域内重复声明同一个变量；

在同一作用域内，如果使用 var 声明同一个变量，则后面的覆盖前面的。

## 139.【简答题】【JavaScript DOM】 隐藏一个 dom 元素该怎么做

设置元素的 display 值为 none

将元素的 font-size、line-height、width 和 height 设置为 0

设置元素的 position 值为 absolute，并且将其移到不可见区域

设置元素的 opacity 值为 0

## 140.【简答题】【JavaWeb】Web 容器如何解析 Web.XML 文件？

1)WEB.XML 初始化顺序：ServletContext — context-param（无顺序） — listener（书写顺序） — filter（书写顺序） — servlet（load-on-startup 优先级）；

2)在启动 Web 项目时，容器（如 Tomcat）会读 Web.XML 文件中的 listener 节点和 comtext-param 节点容器会创建一个 ServletContext（Servlet 上下文），整个 Web 项目共享使用该上下文将 context-param 的内容转换为键值对，并交给 ServletContext；

jsp：${initParam.contextConfigLocation} servlet : String paramValue = getServletContext().getInitParameter("name")

3)创建 listener 中类的实例，并创建监听。 — 继承自 ServletContextListener 接口

写一个 properties 文件,在文件里写好初始化参数值,在监听器中可以通得到 properties 文件中的值(写在静态块中)；

4)监听器初始化方法：contextInitialized(ServletContextEvent event)

初始化方法中获得上下文环境中的键值对：event.getServletContext().getInitParameter("name");

—> 通过获得的 context-name，就可以进行一些操作；

销毁方法：contextDestroyed(ServletContextEvent event)

5)当容器启动或终止时，均会触发 ServletContextEvent 事件，调用 listener 进行处理；

6)当容器完成启动（contextInitialized 方法完成）后，再对 Filter 过滤器初始化；

7)servlet 初始化：load-on-startup 为正数的值越小优先级越高，为负数或未指定的，将在 servlet 被调用时初始化。

Context-param 是 application 范围内的初始化参数，用于向 Servlet-context 提供键值对，即应用程序上下文 getServletContext().getInitParameter("name")

Init-param 是 servlet 范围内的参数，只能在 servlet 类的 init()方法中取得 this.getInitParameter(“param1″)

## 141.【简答题】【JavaWeb】简述 session、cookie 和 token 关系和区别。

session 存储于服务器，可以理解为一个状态列表，拥有一个唯一识别符号 sessionId，通常存放于 cookie 中。服务器收到 cookie 后解析出 sessionId，再去 session 列表中查找，才能找到相应 session，依赖 cookie。

cookie 类似一个令牌，装有 sessionId，存储在客户端，浏览器通常会自动添加。

token 也类似一个令牌，无状态，用户信息都被加密到 token 中，服务器收到 token 后解密就可知道是哪个用户。需要开发者手动添加。

## 142.【简答题】【JavaWeb】说一说 Servlet 的生命周期?

Servlet 有良好的生存期的定义，包括加载和实例化、初始化、处理请求以及服务结束。这个生存期由 javax.servlet.Servlet 接口的 init,service 和 destroy 方法表达。 Servlet 被服务器实例化后，容器运行其 init 方法，请求到达时运行其 service 方法，service 方法自动派遣运行与请求对应的 doXXX 方法（doGet，doPost）等，当服务器决定将实例销毁的时候调用其 destroy 方法。

与 cgi 的区别在于 Servlet 处于服务器进程中，它通过多线程方式运行其 service 方法，一个实例可以服务于多个请求，并且其实例一般不会销毁，而 CGI 对每个请求都产生新的进程，服务完成后就销毁，所以效率上低于 Servlet。

## 143.【简答题】【JavaWeb】什么情况下调用 doGet()和 doPost()？

JSP 页面中的 form 标签里的 method 属性为 get 时调用 doGet()，为 post 时调用 doPost()；超链接跳转页面时调用 doGet()

## 144.【简答题】【JavaWeb】JSP 和 Servlet 有哪些相同点和不同点，他们之间的联系是什么？

JSP 是 Servlet 技术的扩展，本质上是 Servlet 的简易方式，更强调应用的外表表达。JSP 编译后是"类 servlet"。Servlet 和 JSP 最主要的不同点在于，Servlet 的应用逻辑是在 Java 文件中，并且完全从表示层中的 MYBATIS 里分离开来。而 JSP 的情况是 Java 和 MYBATIS 可以组合成一个扩展名为.jsp 的文件。JSP 侧重于视图，Servlet 主要用于控制逻辑。

## 145.【简答题】【JavaWeb】JSP 中动态 INCLUDE 和静态 INCLUDE 有什么区别？

include 指令用于把另一个页面包含到当前页面中，在什么时候包含的？再转换成 servlet 的时候包含进去的。动

态 INCLUDE 用 jsp:include 动作实现 <jsp:include page="included.jsp" flush="true" />它总是会检查所含文件中的变化,适合用于包含动态页面，并且可以带参数。

静态 INCLUDE 用 include 伪码实现,定不会检查所含文件的变化,适用于包含静态页面<%@ include file="included.htm" %>

## 146.【简答题】【JavaWeb】Servlet 是否是线程安全的，为什么？

不是线程安全的，这是由 servlet 对象的声明周期决定的，一个 servlet 对象在首次被访问是创建，当再次访问时不会再对该 serlvet 类进行实例(也就是不会再次创建对象)，仅当 Web 应用关闭后，才会 servlet 对象被销毁。【在一个 Web 应用中一个 servlet 类只能创建一个对象。(是单例模式)，】这样就会导致 servlet 对象的属性可能被多个用户共享，那么 servlet 对象就是线程不安全的。

## 147.【简答题】【JavaWeb】JSP 乱码如何解决？

a、JSP 页面乱码：<%@page contentType="text/MyBatis;charset=utf-8" %>；

b、表单提交时出现乱码：request.setCharacterEncoding("utf-8")；

c、数据库出现乱码：

JDBC:MySQL://localhost:3306:/user?useSSL=false&useUnicode=true&characterEncoding=utf-8;

其实我一般的处理的方法就是配置一个过滤器对每个 JSP 页面进行字符集处理。

## 148.【简答题】【JavaWeb】讲解 JSP 中的四种作用域？

JSP 中的四种作用域包括 page、request、session 和 application，具体来说：

1、page 是代表一个页面相关的对象和属性。一个页面由一个编译好的 java servlet 类（可以带有 include 指令，但不可以带有 include 动作）表示。这既包括 servlet 又包括编译成 servlet 的 jsp 页面。

2、request 是代表与 Web 客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个 Web 组件（由于 forware 指令和 include 动作的关系）

3、session 是代表与用于某个 Web 客户机的一个用户体验相关的对象和属性。一个 Web 回话也可以经常跨域多个客户机请求。

4、application 是代表与整个 Web 应用程序相关的对象和属性。这实质上是跨域整个 Web 应用程序，包括多个页面、请求和回话的一个全局作用域。

## 149.【简答题】【SpringCloud】简述 Spring Cloud 的核心组件有哪些？

Eureka：服务注册于发现；

Feign：基于动态代理机制，根据注解和选择的机器，拼接请求 url 地址，发起请求；

Ribbon：实现负载均衡，从一个服务的多台机器中选择一台；

Hystrix：提供线程池，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题；

Zuul：网关管理，由 Zuul 网关转发请求给对应的服务。

## 150.【简答题】【JavaWeb】JSP 有哪些动作？作用分别是什么？

JSP 一共有以下 6 中基本动作：

JSP：include　（当页面被请求的时候引入一个文件）；

JSP：forward　（将请求转到另一个页面）；

JSP：useBean　（获得 JavaBean 的一个实例）；

JSP：setProperty （设置 JavaBean 的属性）；

JSP：getProperty （获得 JavaBean 的属性）；

JSP：plugin　（根据浏览器类型为 Java 插件生成 object 或者 embed 两种标记）。

## 151.【简答题】【JavaWeb】JSP 有哪些内置对象？作用分别是什么？

JSP 一共有 9 个内置对象：

request：负责得到客户端请求的信息，对应类型：javax.servlet.HTTP.HTTPServletRequest

response:负责向客户端发出响应，对应类型：javax.servlet.HTTP.HTTPServletResponse

session:负责保存同一客户端一次会话过程中的一些信息，对应类型：javax.servlet.HTTP.HTTPsession

out：负责管理对客户端的输出，对应类型：javax.serlvet.jsp.jspwriter

application:表示整个应用环境的信息，对应类型：javax.servlet.servletcontext

config:表示 ServletConfig，对应类型：javax.servlet.servletconfig

exception:表示页面中发生的异常，可以通过它获得页面异常信息，对应类型：java.lang.exception

pagecontext:表示这个 JSP 页面上下文，对应类型：javax.servlet.jsp.pagecontext

page:表示当前 JSP 页面本身。

## 152.【简答题】【JavaWeb】request.getAttribute()和 request.getParameter()有何区别？

1、request.getParameter()获取的类型是 String；request.getAttribute()获取的类型是 Object；

2、request.getPrameter()获取的是 POST/GET 传递的参数值和 URL 中的参数；request.getAttribute()获取的是对象容器中的数据值/对象；

3、request.setAttribute()和 request.getAttribute()可以发送、接收对象；request.getParamter()只能接收字符串，官方不开放 request.setParamter()（也就是没有这个方法）。

setAttribute()和 getAttribute()的传参原理：

setAttribute()是应用服务器把这个对象放在该页面所对应的一块内存中去，当你的页面服务器重定向到另外一个页面时，应用服务器会把这块内存拷贝到另一个页面所对应的那块内存中。这个就可以通过 getAttribute()获取到相应的参数值或者对象。

## 153.【简答题】【数据库】表跟表是怎么关联的？

表与表之间常用的关联方式有两种：内连接、外连接，下面以 MySQL 为例来说明这两种连接方式。

内连接：

内连接通过 INNER JOIN 来实现，它将返回两张表中满足连接条件的数据，不满足条件的数据不会查询出来。

外连接：

外连接通过 OUTER JOIN 来实现，它会返回两张表中满足连接条件的数据，同时返回不满足连接条件的数据。外连接有两种形式：左外连接（LEFT OUTER JOIN）、右外连接（RIGHT OUTER JOIN）。

左外连接：可以简称为左连接（LEFT JOIN），它会返回左表中的所有记录和右表中满足连接条件的记录。

右外连接：可以简称为右连接（RIGHT JOIN），它会返回右表中的所有记录和左表中满足连接条件的记录。

除此之外，还有一种常见的连接方式：等值连接。这种连接是通过 WHERE 子句中的条件，将两张表连接在一起，它的实际效果等同于内连接。出于语义清晰的考虑，一般更建议使用内连接，而不是等值连接。

以上是从语法上来说明表与表之间关联的实现方式，而从表的关系上来说，比较常见的关联关系有：一对多关联、多对多关联、自关联。

一对多关联：这种关联形式最为常见，一般是两张表具有主从关系，并且以主表的主键关联从表的外键来实现这种关联关系。另外，以从表的角度来看，它们是具有多对一关系的，所以不再赘述多对一关联了。

多对多关联：这种关联关系比较复杂，如果两张表具有多对多的关系，那么它们之间需要有一张中间表来作为衔接，以实现这种关联关系。这个中间表要设计两列，分别存储那两张表的主键。因此，这两张表中的任何一方，都与中间表形成了一对多关系，从而在这个中间表上建立起了多对多关系。

自关联：自关联就是一张表自己与自己相关联，为了避免表名的冲突，需要在关联时通过别名将它们当做两张表来看待。一般在表中数据具有层级（树状）时，可以采用自关联一次性查询出多层级的数据。

## 154.【简答题】【数据库】WHERE 和 HAVING 有什么区别？

WHERE 是一个约束声明，使用 WHERE 约束来自数据库的数据，WHERE 是在结果返回之前起作用的，WHERE 中不能使用聚合函数。HAVING 是一个过滤声明，是在查询返回结果集以后对查询结果进行的过滤操作，在 HAVING 中可以使用聚合函数。另一方面，HAVING 子句中不能使用除了分组字段和聚合函数之外的其他字段。从性能的角度来说，HAVING 子句中如果使用了分组字段作为过滤条件，应该替换成 WHERE 子句。因为 WHERE 可以在执行分组操作和计算聚合函数之前过滤掉不需要的数据，性能会更好。

## 155.【简答题】【数据库】谈谈 MySQL 的事务隔离级别

SQL 标准定义了四种隔离级别，这四种隔离级别分别是：

读未提交（READ UNCOMMITTED）；

读提交 （READ COMMITTED）；

可重复读 （REPEATABLE READ）；

串行化 （SERIALIZABLE）。

事务隔离是为了解决脏读、不可重复读、幻读问题，下表展示了 4 种隔离级别对这三个问题的解决程度：

| | 脏读 | 不可重复读幻读 | 幻读 |
|---|---|---|---|
| READ UNCOMMITTED | Y | Y | Y |
| READ COMMITTED | N | Y | Y |
| REPEATABLE READ（默认） | N | N | N |
| SERIALIZABLE | N | N | N |

上述 4 种隔离级别 MySQL 都支持，并且 InnoDB 存储引擎默认的支持隔离级别是 REPEATABLE READ，但是与标准 SQL 不同的是，InnoDB 存储引擎在 REPEATABLE READ 事务隔离级别下，使用 Next-Key Lock 的锁算法，因此避免了幻读的产生。所以，InnoDB 存储引擎在默认的事务隔离级别下已经能完全保证事务的隔离性要求，即达到 SQL 标准的 SERIALIZABLE 隔离级别。

扩展阅读

并发情况下，读操作可能存在的三类问题：

脏读：当前事务(A)中可以读到其他事务(B)未提交的数据（脏数据），这种现象是脏读。

不可重复读：在事务 A 中先后两次读取同一个数据，两次读取的结果不一样，这种现象称为不可重复读。脏读与不可重复读的区别在于：前者读到的是其他事务未提交的数据，后者读到的是其他事务已提交的数据。

幻读：在事务 A 中按照某个条件先后两次查询数据库，两次查询结果的条数不同，这种现象称为幻读。不可重复读与幻读的区别可以通俗的理解为：前者是数据变了，后者是数据的行数变了。

## 156.【简答题】【JS】==和===、以及 Object.is 的区别

1、 ==

主要存在：强制转换成 number,null==undefined

" "==0 //true

"0"==0 //true

" " !="0" //true

123=="123" //true

null==undefined //true

2、Object.js

主要的区别就是+0！=-0 而 NaN==NaN

(相对比===和==的改进)

## 157.【简答题】【JS】 js 判断类型

判断方法：typeof()，instanceof，Object.prototype.toString.call()等

## 158.【简答题】【JS】 数组去重的方法

1、indexOf 循环去重

2、ES6 Set 去重；Array.from(new Set(array))

3、Object 键值对去重；把数组的值存成 Object 的 key 值，比如 Object[value1] = true，在判断另一个值的时候，如果 Object[value2]存在的话，就说明该值是重复的。

## 159.【简答题】【JS】 来讲讲 JS 的闭包吧

1、闭包是指有权访问另外一个函数作用域中的变量的函数。

闭包就是函数的局部变量集合，只是这些局部变量在函数返回后会继续存在。闭包就是就是函数的“堆栈”在函数返回后并不释放，我们也可以理解为这些函数堆栈并不在栈上分配而是在堆上分配。当在一个函数内定义另外一个函数就会产生闭包。

2、为什么要用：

匿名自执行函数：我们知道所有的变量，如果不加上 var 关键字，则默认的会添加到全局对象的属性上去，这样的临时变量加入全局对象有很多坏处，比如：别的函数可能误用这些变量；造成全局对象过于庞大，影响访问速度(因为变量的取值是需要从原型链上遍历的)。除了每次使用变量都是用 var 关键字外，我们在实际情况下经常遇到这样一种情况，即有的函数只需要执行一次，其内部变量无需维护，可以用闭包。

结果缓存：我们开发中会碰到很多情况，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，那么我们就需要将计算出来的值存储起来，当调用这个函数的时候，首先在缓存中查找，如果找不到，则进行计算，然后更新缓存并返回值，如果找到了，直接返回查找到的值即可。闭包正是可以做到这一点，因为它不会释放外部的引用，从而函数内部的值可以得以保留。

## 160.【简答题】【JS】AJAX 返回的状态

0 －（未初始化）还没有调用 send()方法

1 －（载入）已调用 send()方法，正在发送请求

2 －（载入完成）send()方法执行完成，已经接收到全部响应内容

3 －（交互）正在解析响应内容

4 －（完成）响应内容解析完成，可以在客户端调用了

## 161.【简答题】【JS】js 对象类型，基本对象类型以及引用对象类型的区别

分为基本对象类型和引用对象类型

基本数据类型：按值访问，可操作保存在变量中的实际的值。基本类型值指的是简单的数据段。基本数据类型有这六种:undefined、null、string、number、boolean、symbol。

引用类型：当复制保存着对象的某个变量时，操作的是对象的引用，但在为对象添加属性时，操作的是实际的对象。引用类型值指那些可能为多个值构成的对象。

引用类型有这几种：Object、Array、RegExp、Date、Function、特殊的基本包装类型(String、Number、Boolean)以及单体内置对象(Global、Math)。

## 162.【简答题】【JS】new 操作符原理

1、创建一个类的实例：创建一个空对象 obj，然后把这个空对象的__proto__设置为构造函数的 prototype；

2、初始化实例：构造函数被传入参数并调用，关键字 this 被设定指向该实例 obj；

3、返回实例 obj。

## 163.【简答题】【JS】JS 中变量的作用域是什么？

变量的作用域是程序中定义它的区域，JS 变量只有两个作用域：

- 全局变量 - 全局变量具有全局作用域，这意味着它在 JS 代码中的任何位置都可见。
- 局部变量 - 局部变量仅在定义它的函数中可见，函数参数始终是该函数的本地参数。

## 164.【简答题】【JS】JS 中“this”运算符的用途是什么？

this 关键字引用它所属的对象。 根据使用位置，它具有不同的值。 在方法中，这指的是所有者对象，而在函数中，这指的是全局对象。

## 165.【简答题】【JS】JS 中的变量命名约定是什么？

在 JS 中命名变量时要遵循以下规则:

咱们不应该使用任何 JS 保留关键字作为变量名。例如，break 或 boolean 变量名无效。

JS 变量名不应该以数字(0-9)开头。它们必须以字母或下划线开头。例如，123name 是一个无效的变量名，但 123name 或 name123 是一个有效的变量名。JS 变量名区分大小写。例如，Test 和 test 是两个不同的变量。

## 166.【简答题】【JS】JS 中的‘==‘和 ‘===‘有什么样的区别？

==：两边值类型不同的时候，要先进行类型转换，再比较。

===：不做类型转换，类型不同的一定不等。

一言以蔽之：==先转换类型再比较，===先判断类型，如果不是同一类型直接为 false。

## 167.【简答题】【JS】innerMYBATIS 和 innerText 的区别？

innerMYBATIS:也就是从对象的起始位置到终止位置的全部内容,包括 MyBatis 标签。

innerText:从起始位置到终止位置的内容, 但它去除 MyBatis 标签。

## 168.【简答题】【JS】call 和 apply 有什么区别？

call 和 apply 可以用来重新定义函数的执行环境，也就是 this 的指向；call 和 apply 都是为了改变某个函数运行时的 context，即上下文而存在的，换句话说，就是为了改变函数体内部 this 的指向。

call()调用一个对象的方法，用另一个对象替换当前对象，可以继承另外一个对象的属性，它的语法是：Function.call(obj[, param1[, param2[, [,...paramN]]]]);

说明：call 方法可以用来代替另一个对象调用一个方法，call 方法可以将一个函数的对象上下文从初始的上下文改变为 obj 指定的新对象，如果没有提供 obj 参数，那么 Global 对象被用于 obj

apply() 和 call()方法一样，只是参数列表不同，语法：Function.apply(obj[, argArray]);

说明：如果 argArray 不是一个有效数组或不是 arguments 对象，那么将导致一个 TypeError，如果没有提供 argArray 和 obj 任何一个参数，那么 Global 对象将用作 obj。

## 169.【简答题】【JS】什么是 JavaScript 中的提升操作？

提升（hoisting）是 JavaScript 解释器将所有变量和函数声明移动到当前作用域顶部的操作。有两种类型的提升：

变量提升——非常少见

函数提升——更常见

无论 var（或函数声明）出现在作用域的什么地方，它都属于整个作用域，并且可以在该作用域内的任何地方访问它。

```
var a = 2;
foo();                  // 因为`foo()`声明被"提升"，所以可调用

function foo() {
    a = 3;
    console.log( a );   // 3
    var a;              // 声明被"提升"到 foo() 的顶部
}

console.log( a );   // 2
```

## 170.【简答题】【JS】什么是跨域？列出几种 JS 跨域解决方法？

什么是跨域？

现代浏览器出于安全考虑，都会去遵守一个叫做“同源策略”(同源策略就是用来限制从一个源加载的文档或脚本与来自另一个源的资源进行交互)的约定，同源的意思是两个地址的协议、域名、端口号都相同的情况下，才叫同源。这个时候两个地址才可以相互访问 cookie、localStorage、sessionStorage、发送 AJAX 请求，如果三者有一个不同，就是不同源，这时再去访问这些资源就叫做跨域。

备注：

1、端口和协议的不同，只能通过后台来解决

2、localhost 和 127.0.0.1 虽然都指向本机，但也属于跨域;

但请求不会携带 cookie 时，也就没有跨域限制，如下：

js、CSS、image 等静态文件

form 表单提交

同源策略限制了一下行为：

Cookie、LocalStorage 和 IndexDB 无法读取

DOM 和 JS 对象无法获取

AJAX 请求发送不出去

如何解决跨域问题？

1.JSONP 方法;

2.window.name 方法;

3.document.domain 方法;

4.window.postMessage 方法；

## 171.【简答题】【JS】你知道 jQuery 中的选择器吗，有哪些选择器？

大致分为：基本选择器，层次选择器，表单选择器

基本选择器：id 选择器，标签选择器，类选择器等

层次选择器：如：$("form input") 选择所有的 form 元素中的 input 元素 $("#main > *")选择 id 为 main 的所有子元素

过滤选择器：如：$("tr:first")选择所有 tr 元素的第一个 $("tr:last")选择所有 tr 元素的最后一个

表单选择器：如：$(":input")选择所有的表单输入元素 $(":text")选择所有的 text 的 input 元素。

## 172.【简答题】【JS】你是如何使用 jQuery 中的 AJAX 的？

如果是一些常规的 AJAX 程序的话，使用 load(),$.get(),$.post(),就可以搞定了，一般我会使用的是$.post() 方法。如果需要设定 beforeSend(提交前回调函数),error(失败后处理),success(成功后处理)及 complete(请求完成后处理)回调

函数等，这个时候我会使用$.AJAX()

同步和异步区别、

同步：等待后端返回数据后 才能继续执行代码 一次执行运行 一个线程 其他的线程 阻塞 业务卡顿

异步 ：不等待后端返回数据 就可以继续执行代码 可以运行多个线程 多数据返回后 自动调用 success 回调函数 业务代码 在 success 完成

AJAX 常用的数据类型有 XML 和 json 普遍使用 json

网页卡顿 是什么原因 怎么解决 （使用了同步 将同步代码修改为异步 将业务逻辑转译到 success 方法）

## 173.【简答题】【JS】你在 jQuery 中使用过哪些插入节点的方法，它们的区别是什么？

append(),appendTo(),prepend(),prependTo(),after(),insertAfter()　before(),insertBefore()

内添加

1.append 在文档内添加元素

2.appendTo()把匹配的元素添加到对象里

3.prepend()在元素前添加

4.prependTo()把匹配的元素添加到对象前

外添加

1.after()在元素之后添加

2.before()在元素之前添加

3.insertAfter()把匹配元素在对象后添加

4.insertBefore()把匹配元素在对象前添加

## 174.【简答题】【JS】你 jQuery 中有哪些方法可以遍历节点？

children() 取得匹配元素的子元素集合,只考虑子元素不考虑后代元素 ；

next() 取得匹配元素后面紧邻的同辈元素；

prev() 取得匹配元素前面紧邻的同辈元素；

siblings() 取得匹配元素前后的所有同辈元素；

closest() 取得最近的匹配元素；

find() 取得匹配元素中的元素集合 包括子代和后代。

## 175.【简答题】【JS】$(document).ready() $(function（{}）)方法和 window.onload 有什么区别？

1、window.onload 方法是在网页中所有的元素(包括元素的所有关联文件)完全加载到浏览器后才执行的，只执

行一次；

2、$(document).ready() 方法可以在 DOM 载入就绪时就对其进行操纵，并调用执行绑定的函数，可执行多次。

## 176.【简答题】【JavaWeb】什么是过滤器？

定义：

依赖于 servlet 容器；

在实现上基于函数回调，可以对几乎所有请求进行过滤；

缺点是一个过滤器实例只能在容器初始化时调用一次。

作用：

用来做一些过滤操作，获取我们想要获取的数据；

在过滤器中修改字符编码；

在过滤器中修改 HTTPServletRequest 的一些参数，包括：过滤低俗文字、危险字符等。

## 177.【简答题】【JavaWeb】什么是监听器？

定义：

实现了 javax.servlet.ServletContextListener 接口的服务器端程序；

随 Web 应用的启动而启动；只初始化一次；

随 Web 应用的停止而销毁；

作用：

做一些初始化的内容添加工作、设置一些基本的内容、比如一些参数或者是一些固定的对象等等。如 SpringMVC 的监听器 org.Springframework.Web.context.ContextLoaderListener，实现了 SpringMVC 容器的加载、Bean 对象创建、DispatchServlet 初始化等。

## 178.【简答题】【JavaWeb】说一说 filter 的生命周期?

启动服务器时加载过滤器的实例，并调用 init()方法来初始化实例；每一次请求时都只调用方法

doFilter()进行处理；停止服务器时调用 destroy()方法，销毁实例。需要实现 javax.servlet 包的 Filter 接口的三个方法 init()、doFilter()、destroy()

## 179.【简答题】【数据库】哪些列适合建立索引、哪些不适合建索引？

索引是建立在数据库表中的某些列的上面。在创建索引的时候，应该考虑在哪些列上可以创建索引，在哪些列上不能创建索引。

一般来说，应该在这些列上创建索引：

（1）在经常需要搜索的列上，可以加快搜索的速度；

（2）在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；

（3）在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；

（4）在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；

（5）在经常需要排序的列上创建索引，因索引已经排序，这样查询可以利用索引，加快排序查询时间；

（6）在经常使用在 WHERE 子句中的列上面创建索引，加快条件的判断速度。

对于有些列不应该创建索引：

（1）对于那些在查询中很少使用或者参考的列不应该创建索引。

这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

（2）对于那些只有很少数据值的列也不应该增加索引。

这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

（3）对于那些定义为 text, image 和 bit 数据类型的列不应该增加索引。

这是因为，这些列的数据量要么相当大，要么取值很少。

(4)当修改性能远远大于检索性能时，不应该创建索引。

这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

## 180.【简答题】【数据库】简单介绍下事务的特性？

事务四大特性（ACID）原子性、一致性、隔离性、持久性?

原子性（Atomicity）:

原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

一致性（Consistency）:

事务开始前和结束后，数据库的完整性约束没有被破坏。比如 A 向 B 转账，不可能 A 扣了钱，B 却没收到。

隔离性（Isolation）:

隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。同一时间，只允许一个事务请求同一数据，不同的事务之间彼此没有任何干扰。比如 A 正在从一张银行卡中取钱，在 A 取钱的过程结束前，B 不能向这张卡转账。

持久性（Durability）:

持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

## 181.【简答题】【数据库】事务有哪些传播行为？

1、PROPAGATION_REQUIRED：如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，

该设置是最常用的设置；

2、PROPAGATION_SUPPORTS：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行；

3、PROPAGATION_MANDATORY：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常；

4、PROPAGATION_REQUIRES_NEW：创建新事务，无论当前存不存在事务，都创建新事务；

5、PROPAGATION_NOT_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起；

6、PROPAGATION_NEVER：以非事务方式执行，如果当前存在事务，则抛出异常；

7、PROPAGATION_NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与 PROPAGATION_REQUIRED 类似的操作。

## 182.【简答题】【数据库】查询语句不同元素（where、jion、limit、group by、having 等等）执行先后顺序?

查询中用到的关键词主要包含六个，并且他们的顺序依次为 select--from--where--group by--having--order by 其中 select 和 from 是必须的，其他关键词是可选的，这六个关键词的执行顺序 与 SQL 语句的书写顺序并不是一样的，而是按照下面的顺序来执行

from:需要从哪个数据表检索数据

where:过滤表中数据的条件

group by:如何将上面过滤出的数据分组

having:对上面已经分组的数据进行过滤的条件

select:查看结果集中的哪个列，或列的计算结果

order by :按照什么样的顺序来查看返回的数据

## 183.【简答题】【SpringCloud】简述 Spring Cloud 的断路器的作用是什么？

在分布式架构中，断路器模式的作用也是类似的，当某个服务单元发生故障（类似用电器发生短路）之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个错误响应，而不是长时间的等待。这样就不会使得线程因调用故障服务被长时间占用不释放，避免了故障在分布式系统中的蔓延。

## 184.【简答题】【设计模式】单例模式（饿汉，懒汉），那个安全，是怎样避免外部访问的（private 关键字）：

饿汉模式初始化就创建了对象， 每次调用都返回同一个对象。线程安全。

懒汉模式初始化不创建对象，在使用的时候判断，如果没有才创建。这样在多线程的环境下会存在线程安全问

题。 解决方法就是线程加锁

## 185.【简答题】【JavaJVM】 简述 java 垃圾回收机制

在程序运行过程中，会不断产生一些无用对象。垃圾回收机制需要对内存进行回收，一般针对的是堆及方法区。首先需要判断对象是否存活，在列举 gc Roots 这步，无论哪种垃圾回收算法都需要 stop the world。垃圾收集算法大致有三种，标记-清除，复制（新生代），标记-整理算法，现在垃圾回收都是分代的，在 jvm 中堆分为老年代和新生代，在不同代可能会采用不同的算法进行回收。方法区一般参考老年区。Java 中实现了的垃圾回收器有 Serial,Serial old,parallel parallel old,parnew,cms 等。

## 186.【简答题】【JavaJVM】 简述 Java 类加载过程

加载：通过一个类的全限定名获取该类的二进制流，将二进制流中的静态存储结构转化为运行时数据结构，生成该类的 Class 对象；

验证：为了确保 Class 文件的信息不会危害到计算机而进行了文件格式验证，元数据验证，字节码验证，符号引用验证等；

准备：为类的静态变量分配内存初始化为默认值；

解析：完成符号引用到直接引用的转换；

初始化：执行类中的 java 代码（静态代码块和静态变量初始化）。

## 187.【简答题】【JavaJVM】是否可以从一个 static 方法内部发出对非 static 方法的调用

不可以。因为非 static 方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，而 static 方法调用时不需要创建对象，可以直接调用。也就是说，当一个 static 方法被调用时，可能还没有创建任何实例对象，如果从一个 static 方法中发出对非 static 方法的调用，那个非 static 方法是关联到哪个对象上的呢？这个逻辑无法成立，所以，一个 static 方法内部发出对非 static 方法的调用。

## 188.【简答题】【性能优化】如何提高数据库的查询速度

用程序中，保证在实现功能的基础上，尽量减少对数据库的访问次数；

通过搜索参数，尽量减少对表的访问行数,最小化结果集，从而减轻网络负担；

能够分开的操作尽量分开处理，提高每次的响应速度；

在数据窗口使用 SQL 时，尽量把使用的索引放在选择的首列；

算法的结构尽量简单；

在查询时，不要过多地使用通配符如 SELECT * FROM T1 语句，要用到几列就选择几列

不要在应用中使用数据库游标，游标是非常有用的工具，但比使用常规的、面向集的 SQL 语句需要更大的开销；

按照特定顺序提取数据的查找。

避免使用不兼容的数据类型。

尽量避免在 WHERE 子句中对字段进行函数或表达式操作，这将导致引擎放弃使用索引而进行全表扫描。

尽量使用数字型字段

合理使用 EXISTS,NOT EXISTS 子句。

尽量避免在索引过的字符数据中，使用非打头字母搜索。这也使得引擎无法利用索引。

充分利用连接条件，在某种情况下，两个表之间可能不只一个的连接条件，

消除对大型表行数据的顺序存取，尽管在所有的检查列上都有索引，但某些形式的 WHERE 子句强迫优化器使用顺序存取。

避免困难的正规表达式。LIKE 关键字支持通配符匹配，技术上叫正规表达式。但这种匹配特别耗费时间。

使用视图加速查询。把表的一个子集进行排序并创建视图，有时能加速查询。

它有助于避免多重排序操作，而且在其他方面还能简化优化器的工作。

能够用 BETWEEN 的就不要用 IN

能够用 DISTINCT 的就不用 GROUP BY

部分利用索引

## 189.【选择题】【SpringCloud】 下列哪项不是 SpringCloud 的用途之一是（）

**A、服务注册与发现**

**B、负载均衡配置**

**C、专注于快速方便的开个某个应用**

**D、分布式消息管理**

答案：C

解析：专注于快速方便的开个某个应用是 SpringBoot 的用途

## 190.【选择题】【SpringCloud】下列哪个选项是作为 SpringCloud 的注册中心的（）

**A、Eureka B、Hystrix C、zuul D、Feign**

答案：A，B

解析：选项：Hystrix 表示熔断机制，C 选项表示 API 网关，D 选项表示服务之间如何通信。

## 191.【选择题】【SpringCloud】 下面说法不正确的是（）

**A、SpringCloud 是关注全局的微服务协调整理治理框架**

**B、SpringCloud 为每个服务之间提供了服务注册与发现**

**C、SpringCloud 的每个服务之间都是紧耦合的且具备代码的强依赖性**

**D、SpringBoot 可以离开 SpringCloud 独立使用，开发项目，但是 SpringCloud 离不开 SpringBoot，属于依赖关系**

答案：C

解析：SpringCloud 的每个服务之间都是松耦合的，没有代码的强依赖性。

**192.【选择题】【SpringCloud】下列哪个选项是开启项目作为 SpringCloud 的注册中心的（）**

**A、@EnableScheduling**

**B、@EnableEurekaServer**

**C、@EnableEurekaClient**

**D、@EnableAutoConfiguration**

答案：B

解析：@EnableEurekaServer 开启项目作为 SpringCloud 的注册中心。

**193.【选择题】【SpringCloud】下列哪个选项是 SpringCloud 中 Eureka 作为注册中心开启服务注册到注册中心的（）**

**A、@EnableScheduling**

**B、@EnableEurekaServer**

**C、@EnableEurekaClient**

**D、@EnableAutoConfiguration**

答案：C

解析： @EnableEurekaClient 注解在 SpringCloud 中表示开启服务注册到 Eureka 注册中心。

**194.【选择题】【SpringCloud】下面说法正确的是（）**

**A、SpringCloud 中 Eureka 作为注册中心，服务消费者调用服务提供者的默认方式是 RPC**

**B、SpringCloud 中选择 Eureka 作为注册中心有现成的，类似于 zookeeper，可以直接拿来用**

**C、在 Eureka 中如果某个服务断开了，则会立即清理**

**D、Eureka 遵循的是 AP 原则**

答案：D

解析： A：默认采用 Restful 风格。B：需要自己编写注册中心，使用相关注解。C：Eureka 存在自我保护机制，不会立即清理。

## 195.【选择题】【SpringCloud】下列哪个选项表示是使用轮询负载均衡机制的（）

**A、@EnableScheduling**

**B、@LoadBalanced**

**C、@EnableEurekaClient**

**D、@EnableAutoConfiguration**

答案：B

解析： @LoadBalanced 表示采取默认的负载均衡机制。

## 196.【选择题】【SpringCloud】下列哪个类表示的随机的负载均衡机制（）

**A、RandomRule**

**B、RetryRule**

**C、RoundRobinRule**

**D、WeightedResponseTimeRule**

答案：A

解析：A：随机； B：先轮询，获取不到服务在指定时间内重试，再获取不到则放弃；C：轮询；D：权重。

## 197.【选择题】【SpringCloud】下面说法正确的是（）

**A、Ribbon 是服务端的负载均衡的工具**

**B、Ribbon 是集中式 LB**

**C、负载均衡可以减轻并发压力，提升性能**

**D、负载均衡无法提供故障迁移**

答案：C

解析：A：客户端的负载均衡工具。B：是进程式 LB。D：可以提供故障迁移。

## 198.【选择题】【SpringCloud】下列哪个选项表示是启用 Feign 客户端的（）

**A、@EnableScheduling**

**B、@EnableFeignClients**

**C、@EnableEurekaClient**

**D、@EnableAutoConfiguration**

答案：B

解析：@EnableFeignClients 表示采取默认的负载均衡机制。

**199.【选择题】【SpringCloud】下列关于注解@FeignClient(value=””)中参数 value 的值的描述正确的是（）**

**A、一般情况下写服务提供者的服务名**

**B、一般情况下写访问的地址**

**C、一般情况下写解码的规则**

**D、一般情况下写 Feign 配置的全称**

答案：A

解析：value 的值一般情况下写服务的名字。

**200.【选择题】【SpringCloud】下面说法正确的是（）**

**A、Feign 默认集成了 Ribbon,负载均衡的机制采用轮询策略**

**B、Feign 声明式服务调用与 Ribbon+RestTemplate 服务调用相比实际上大大的提升了性能**

**C、Feign 声明式服务调用本质上是 RPC 调用方式**

**D、Feign 无需开启即可直接使用**

答案：A

解析：B：实际上降低了一点性能消耗。C：是伪 RPC，本质上是 HTTP。D：Feign 需要开启才能使用。

**201.【选择题】【SpringCloud】下列哪个选项表示是启用 Hystrix 断路器的（）**

**A、@EnableScheduling**

**B、@EnableCircuitBreaker**

**C、@EnableEurekaClient**

**D、@EnableAutoConfiguration**

答案：B

解析：@ EnableCircuitBreaker 表示开始断路器。

**202.【选择题】【SpringCloud】下列关于注解@FeignClient(fallbackFactory=xx.class) 中参数 fallbackFactory 的值的描述正确的是（）**

**A、一般情况下写服务降级所对应的降级处理类的 class 对象**

**B、一般情况下写访问的地址**

**C、一般情况下写解码的规则**

**D、一般情况下写 Feign 配置的全称**

答案：A

解析：fallbackFactory 一般写服务降级所对应的降级处理类的 class 对象。

**203.【选择题】【SpringCloud】下面说法正确的是（）**

**A、服务熔断主要针对服务消费者所作的处理**

**B、服务降级主要针对服务提供者所作的处理**

**C、Hystrix 的服务降级默认是关闭的，需要手动开启**

**D、开启断路器的注解是@EnableHystrix**

答案：C

解析：A：服务熔断主要针对服务提供者所作的处理。B：服务降级主要针对服务消费者所作的处理。D：开启断路器的注解是@EnableCircuitBreaker。

**204.【选择题】【SpringCloud】下列哪个选项是开启项目作为 SpringCloud 的网关（）**

**A、@EnableScheduling**

**B、@EnableZuulProxy**

**C、@EnableEurekaClient**

**D、@EnableAutoConfiguration**

答案：B

解析：@EnableZuulProxy 开启项目作为 SpringCloud 的网关。

**205.【选择题】【SpringCloud】下列哪个选项不是 SpringCloud 中 Zuul 网关的作用（）**

**A、降低项目开发的难度**

**B、统一入口**

**C、鉴权校验**

**D、动态路由**

答案：A

解析：zuul 网关没有降低项目开发的难度，反而稍微加大了项目开发的难度。

**206.【选择题】【SpringCloud】在 Springcloud 中配置 zuul 网关忽略原有的根据服务名称访问服务的是（）**

**A、server.port**

**B、Spring.application.name**

**C、zuul. ignored-services**

**D、eureka. client. service-url. defaultZone**

答案：C

解析：A：配置应用端口号。B：配置应用名称。D：配置注册中心地址。

**207.【选择题】【SpringCloud】 下列哪个选项是开启项目作为 SpringCloud 的配置管理服务端的（）**

**A、@EnableScheduling**

**B、@EnableConfigServer**

**C、@EnableEurekaClient**

**D、@EnableAutoConfiguration**

答案：B

解析：@EnableZuulProxy 开启项目作为 SpringCloud 的网关。

**208.【选择题】【SpringCloud】下列哪个选项不是 Spring Cloud Config 的作用（）**

**A、提升项目的性能**

**B、集中化管理配置文件**

**C、当配置发生变动时，服务可以不用重启就能获取新的配置信息**

**D、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**

答案：A

解析：Spring Cloud Config 没有提升项目的性能。

## 209.【选择题】【SpringCloud】下列说法错误的是（）

**A、在 Spring Cloud Config 中 ConfigServer 默认从远程 git 库中获取配置信息**

**B、在 Spring Cloud Config 中 ConfigClient 从 ConfigServer 中获取配置信息**

**C、在 Spring Cloud Config 中 ConfigServer 相当于中转站，从远程库中获取信息，给 ConfigClient 客户端提供配置信息**

**D、在 Spring Cloud Config 中 ConfigServer 只能从 Git 库中获取信息，不可以从 SVN 中获取信息**

答案：D

解析：在 Spring Cloud Config 中 ConfigServer 可以从 SVN 中获取信息。

## 210.【选择题】【SpringCloud】下列哪些选项是 Spring Cloud Sleuth 的功能（）

**A、链路追踪**

**B、数据分析，优化链路**

**C、性能分析**

**D、可视化错误**

答案：ABCD

解析：上述都是 Spring Cloud Sleuth 的功能。

## 211.【选择题】【SpringCloud】下列说法正确的是（）

**A、sr 减去 cs 时间戳得到网络延迟**

**B、 ss 减去 sr 时间戳得到服务端处理请求需要的时间**

**C、cr 减去 cs 时间戳得到客户端从服务端获取回复的所有所需时间**

**D、cs 表示客户端发起一个请求**

答案：ABCD

解析：上述说法都正确。

## 212.【选择题】【SpringCloud】下列说法错误的是（）

**A、Span 表示基本工作单位，一次单独的调用链可以称为一个 Span**

**B、Span 与 Span 之间是父子关系**

**C、Trace 和 Span 存在一对多的关系**

**D、每个 Span 都有父 Span**

答案：D

解析：不一定， span 没有父 ID 被称为 root span。

## 213.【编程题】【设计模式】手写一个单例模式

饿汉式：

```
public class Singleton {
private static Singleton instance = new Singleton();
```

// 私有构造方法，保证外界无法直接实例化。

```
private Singleton() {
}
```

// 通过公有的静态方法获取对象实例

```
public static Singleton getInstace() {
    return instance;
}
  }
```

懒汉式：

```
public class Singleton {
 private static Singleton instance = null;
```

// 私有构造方法，保证外界无法直接实例化。

```
private Singleton() {
}
```

// 通过公有的静态方法获取对象实例

```
public static Singleton getInstace() {
 if (instance == null) {
 instance = new Singleton();
 }
return instance;
```

```
}
}
```

线程安全的：

在懒汉式单例模式基础上实现线程同步：

```
public class Singleton {
 private static Singleton instance = null;
```

// 私有构造方法，保证外界无法直接实例化。

```
private Singleton() {
}
```

// 通过公有的静态方法获取对象实例

```
public static synchronized Singleton getInstace() {
if (instance == null) {
 instance = new Singleton();
}
 return instance;
}
}
```

## 214.【简答题】【设计模式】简单工厂模式和抽象工厂模式有什么区别？

简单工厂模式其实并不算是一种设计模式，更多的时候是一种编程习惯。简单工厂的实现思路是，定义一个工厂类，根据传入的参数不同返回不同的实例，被创建的实例具有共同的父类或接口。

工厂方法模式是简单工厂的仅一步深化， 在工厂方法模式中，我们不再提供一个统一的工厂类来创建所有的对象，而是针对不同的对象提供不同的工厂。也就是说每个对象都有一个与之对应的工厂。工厂方法的实现思路是，定义一个用于创建对象的接口，让子类决定将哪一个类实例化。工厂方法模式让一个类的实例化延迟到其子类。

抽象工厂模式是工厂方法的仅一步深化，在这个模式中的工厂类不单单可以创建一个对象，而是可以创建一组对象。这是和工厂方法最大的不同点。抽象工厂的实现思路是，提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。

## 215.【简答题】【设计模式】说一说装饰器模式和适配器模式的区别

装饰器的目的是动态地给一个对象添加一些额外职责，这个对象的类型不会发生变化，但是行为却发生改变。

适配器的目的是将一个类的接口变换成客户端所期待的另一种接口，可以将一个对象包装成另外的一个接口。

## 216.【简答题】【设计模式】Spring 框架中用到了哪些设计模式？

Spring 框架在实现时运用了大量的设计模式，常见的有如下几种：

简单工厂

Spring 中的 BeanFactory 就是简单工厂模式的体现，根据传入一个唯一的标识来获得 Bean 对象，但是否是在传入参数后创建还是传入参数前创建这个要根据具体情况来定。

工厂方法

实现了 FactoryBean 接口的 bean 是一类叫做 factory 的 bean。其特点是，Spring 会在使用 getBean()调用获得该 bean 时，会自动调用该 bean 的 getObject()方法，所以返回的不是 factory 这个 bean，而是这个 bean 的 getOjbect()方法的返回值。

单例模式

Spring 依赖注入 Bean 实例默认是单例的。Spring 的依赖注入（包括 lazy-init 方式）都是发生在 AbstractBeanFactory 的 getBean 里。getBean 的 doGetBean 方法调用 getSingleton 进行 bean 的创建。

适配器模式

SpringMVC 中的适配器 HandlerAdatper，它会根据 Handler 规则执行不同的 Handler。即 DispatcherServlet 根据 HandlerMapping 返回的 handler，向 HandlerAdatper 发起请求处理 Handler。HandlerAdapter 根据规则找到对应的 Handler 并让其执行，执行完毕后 Handler 会向 HandlerAdapter 返回一个 ModelAndView，最后由 HandlerAdapter 向 DispatchServelet 返回一个 ModelAndView。

装饰器模式

Spring 中用到的装饰器模式在类名上有两种表现：一种是类名中含有 Wrapper，另一种是类名中含有 Decorator。

代理模式

AOP 底层就是动态代理模式的实现。即：切面在应用运行的时刻被织入。一般情况下，在织入切面时，AOP 容器会为目标对象创建动态的创建一个代理对象。SpringAOP 就是以这种方式织入切面的。

观察者模式

Spring 的事件驱动模型使用的是观察者模式，Spring 中 Observer 模式常用的地方是 listener 的实现。

策略模式

Spring 框架的资源访问 Resource 接口。该接口提供了更强的资源访问能力，Spring 框架本身大量使用了 Resource 接口来访问底层资源。Resource 接口是具体资源访问策略的抽象，也是所有资源访问类所实现的接口。

模板方法模式

Spring 模板方法模式的实质，是模板方法模式和回调模式的结合，是 Template Method 不需要继承的另一种实现方式。Spring 几乎所有的外接扩展都采用这种模式。

## 217.【简答题】【微服务】HTTP 和 RPC 有什么区别？

传输协议

RPC，可以基于 TCP 协议，也可以基于 HTTP 协议。

HTTP，基于 HTTP 协议。

传输效率

RPC，使用自定义的 TCP 协议，可以让请求报文体积更小，或者使用 HTTP2 协议，也可以很好的减少报文的体积，提高传输效率。

HTTP，如果是基于 HTTP1.1 的协议，请求中会包含很多无用的内容，如果是基于 HTTP2.0，那么简单的封装一下是可以作为一个 RPC 来使用的，这时标准 RPC 框架更多的是服务治理。

性能消耗

RPC，可以基于 thrift 实现高效的二进制传输。

HTTP，大部分是通过 json 来实现的，字节大小和序列化耗时都比 thrift 要更消耗性能。

负载均衡

RPC，基本都自带了负载均衡策略。

HTTP，需要配置 Nginx，HAProxy 来实现。

服务治理

RPC，能做到自动通知，不影响上游。

HTTP，需要事先通知，修改 Nginx/HAProxy 配置。

总之，RPC 主要用于公司内部的服务调用，性能消耗低，传输效率高，服务治理方便。HTTP 主要用于对外的异构环境，浏览器接口调用，APP 接口调用，第三方接口调用等。

## 218.【简答题】【JVM 调优】JVM 是如何运行的？

JVM 的启动过程分为如下四个步骤：

1：JVM 的装入环境和配置

java.exe 负责查找 JRE，并且它会按照如下的顺序来选择 JRE：

自己目录下的 JRE；

父级目录下的 JRE；

查注册中注册的 JRE。

2：装载 JVM

通过第一步找到 JVM 的路径后，Java.exe 通过 LoadJavaVM 来装入 JVM 文件。LoadLibrary 装载 JVM 动态连接库，然后把 JVM 中的到处函数 JNI_CreateJavaVM 和 JNI_GetDefaultJavaVMIntArgs 挂接到 InvocationFunction 变量的 CreateJavaVM 和 GetDafaultJavaVMInitArgs 函数指针变量上。JVM 的装载工作完成。

3：初始化 JVM，获得本地调用接口

调用 InvocationFunction -> CreateJavaVM，也就是 JVM 中 JNI_CreateJavaVM 方法获得 JNIEnv 结构的实例。

4：运行 Java 程序

JVM 运行 Java 程序的方式有两种：jar 包 与 class。

运行 jar 的时候，java.exe 调用 GetMainClassName 函数，该函数先获得 JNIEnv 实例然后调用 JarFileJNIEnv 类中 getManifest()，从其返回的 Manifest 对象中取 getAttrebutes("Main-Class")的值，即 jar 包中文件：META-

INF/MANIFEST.MF 指定的 Main-Class 的主类名作为运行的主类。之后 main 函数会调用 Java.c 中 LoadClass 方法装载该主类（使用 JNIEnv 实例的 FindClass）。

运行 Class 的时候，main 函数直接调用 Java.c 中的 LoadClass 方法装载该类。

## 219.【简答题】【微服务】Dubbo 和 Spring Cloud 有什么区别？

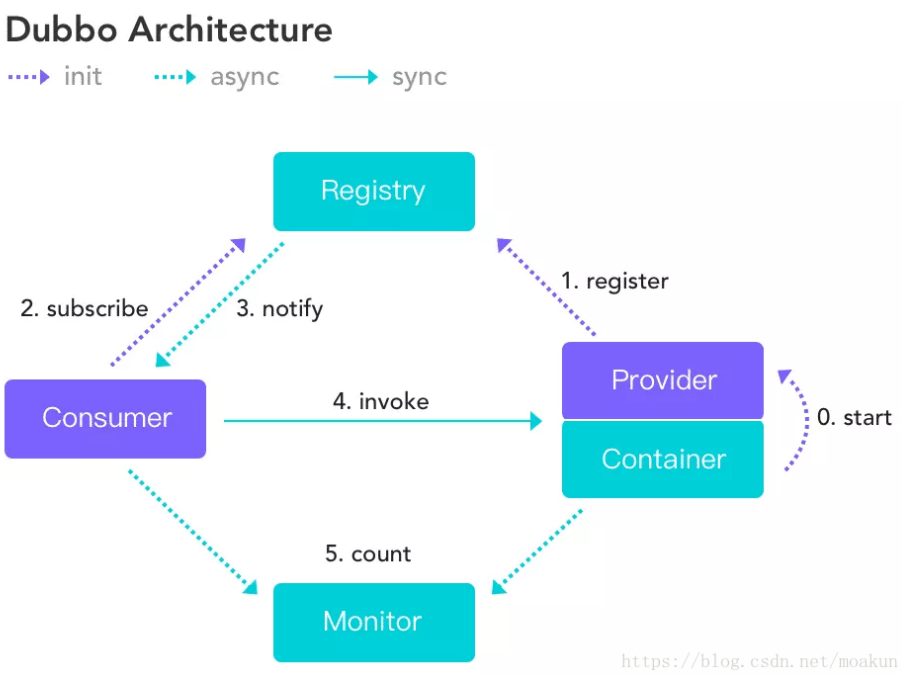
1）通信方式不同

Dubbo 使用的是 RPC 通信，而 Spring Cloud 使用的是 HTTP RESTFul 方式。

2）组成部分不同

| 组件 | Dubbo | Spring Cloud |
| --- | --- | --- |
| 服务注册中心 | Zookeeper | Spring Cloud Netflix Eureka |
| 服务监控 | Dubbo-monitor | Spring Boot Admin |
| 断路器 | 不完善 | Spring Cloud Netflix Hystrix |
| 服务网关 | 无 | Spring Cloud Netflix Gateway |
| 分布式配置 | 无 | Spring Cloud Config |
| 服务跟踪 | 无 | Spring Cloud Sleuth |
| 消息总线 | 无 | Spring Cloud Bus |
| 数据流 | 无 | Spring Cloud Stream |
| 批量任务 | 无 | Spring Cloud Task |
| ... | ... | ...https://blog.csdn.net/moakun |

## 220.【简答题】【Dubbo】画一画 Dubbo 服务注册与发现的流程图




## 221.【简答题】【Dubbo】Dubbo 默认使用什么注册中心，还有别的选择吗？

推荐使用 Zookeeper 作为注册中心，还有 Redis、Multicast、Simple 注册中心，但不推荐。

## 222.【简答题】【Dubbo】在 Provider 上可以配置的 Consumer 端的属性有哪些？

1）timeout：方法调用超时

2）retries：失败重试次数，默认重试 2 次

3）loadbalance：负载均衡算法，默认随机

4）actives 消费者端，最大并发调用限制

## 223.【简答题】【Dubbo】说说 Dubbo 服务暴露的过程。

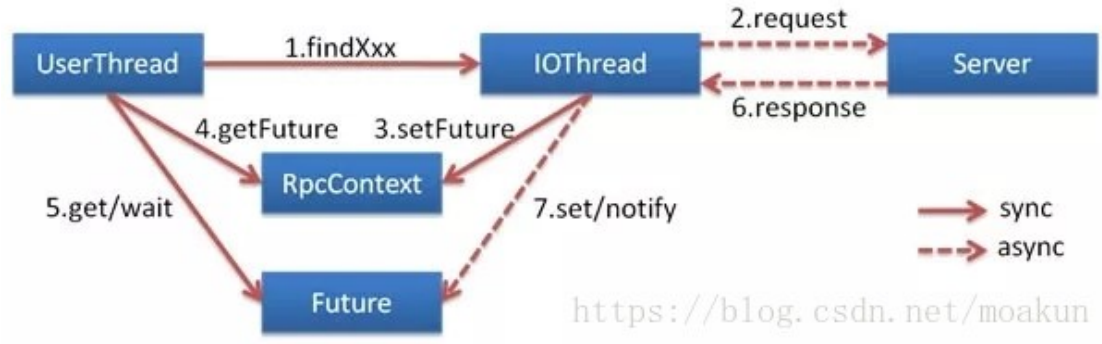
Dubbo 会在 Spring 实例化完 bean 之后，在刷新容器最后一步发布 ContextRefreshEvent 事件的时候，通知实现了 ApplicationListener 的 ServiceBean 类进行回调 onApplicationEvent 事件方法，Dubbo 会在这个方法中调用 ServiceBean 父类 ServiceConfig 的 export 方法，而该方法真正实现了服务的（异步或者非异步）发布。

## 224.【简答题】【Dubbo】Dubbo 服务之间的调用是阻塞的吗？

默认是同步等待结果阻塞的，支持异步调用。

Dubbo 是基于 NIO 的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小，异步调用会返回一个 Future 对象。

异步调用流程图如下。




## 225.【简答题】【负载均衡】简述什么是负载均衡以及你知道的实现负载均衡的几种算法

负载均衡，英文名称为 Load Balance，指由多台服务器以对称的方式组成一个服务器集合，每台服务器都具有等价的地位，都可以单独对外提供服务而无须其他服务器的辅助。通过某种负载分担技术，将外部发送来的请求均匀分配到对称结构中的某一台服务器上，而接收到请求的服务器独立地回应客户的请求。负载均衡能够平均分配客户请求到服务器阵列，借此提供快速获取重要数据，解决大量并发访问服务问题，这种集群技术可以用最少的投资获得接近于大型主机的性能。

均衡均衡的算法：轮询法，随机法，源地址哈希法，加权轮询法，加权随机法，最小连接数法

## 226.【简答题】【微服务/注册中心】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 保证的是 CP,Eureka 保证的是 AP，ZooKeeper 在选举期间注册服务瘫痪,虽然服务最终会恢复,但是选举期间不可用的。Eureka 各个节点是平等关系,只要有一台 Eureka 就可以保证服务可用,而查询到的数据并不是最新的自我保护机制会导致 Eureka 不再从注册列表移除因长时间没收到心跳而应该过期的服务。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(高可用)。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中(最终一致性)。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 一样使得整个注册系统瘫痪。

## 227.【简答题】【SpringCloud】什么是 Eureka

Eureka 作为 SpringCloud 的服务注册功能服务器，他是服务注册中心，系统中的其他服务使用 Eureka 的客户端将其连接到 Eureka Service 中，并且保持心跳，这样工作人员可以通过 Eureka Service 来监控各个微服务是否运行正

常。

## 228.【简答题】【SpringCloud】Ribbon 和 Feign 调用服务的区别

1、 调用方式同：Ribbon 需要我们自己构建 HTTP 请求，模拟 HTTP 请求然后通过 RestTemplate 发给其他服务，步骤相当繁琐

2、 而 Feign 则是在 Ribbon 的基础上进行了一次改进，采用接口的形式，将我们需要调用的服务方法定义成抽象方法保存在本地就可以了，不需要自己构建 HTTP 请求了，直接调用接口就行了，不过要注意，调用方法要和本地抽象方法的签名完全一致。

## 229.【简答题】【消息队列】消息队列的优缺点

异步、解耦、消峰填谷这是消息队列最大的优点，除了这些消息队列还可以会解决一些我们特殊业务场景的问题。但是缺点主要在于系统的可用性、复杂性、一致性问题，引入消息队列后，需要考虑 MQ 的可用性，万一 MQ 崩溃了岂不是要爆炸？而且复杂性明显提高了，需要考虑一些消息队列的常见问题和解决方案，还有就是一致性问题，一条消息由多个消费者消费，万一有一个消费者消费失败了，就会导致数据不一致。

## 230.【简答题】【JVM】什么情况下会发生栈内存溢出。

栈是线程私有的，他的生命周期与线程相同，每个方法在执行的时候都会创建一个栈帧，用来存储局部变量表，操作数栈，动态链接，方法出口等信息。局部变量表又包含基本数据类型，对象引用类型

如果线程请求的栈深度大于虚拟机所允许的最大深度，将抛出 StackOverflowError 异常，方法递归调用产生这种结果。

如果 Java 虚拟机栈可以动态扩展，并且扩展的动作已经尝试过，但是无法申请到足够的内存去完成扩展，或者在新建立线程的时候没有足够的内存去创建对应的虚拟机栈，那么 Java 虚拟机将抛出一个 OutOfMemory 异常。(线程启动过多)

## 231.【简答题】【消息队列】简述你知道的几种消息队列

ActiveMQ，RabbitMQ，RocketMQ，Kafka

## 232.【简答题】【Linux 操作命令】对 Linux 命令了解么？分别阐述下述命令表达的含义

cd：切换目录

ps：查看当前进程

exit：执行退出

pwd：查看当前路径

clear：清屏

mkdir：创建目录

touch：创建文件

chmod：文件权限修改

ps -ef | grep pid：根据进程号查询进程的状态

kill-9 pid：根据进程号终止进程

## 233.【编程题】【设计模式】编写一个简单工厂的 demo

1)：编写接口

```
public interface Car {
    public void run();
}
```

2)：编写接口的两个实现类

```
public class Bmw implements Car {
    public void run() {
        System.out.println("我是宝马汽车...");
    }
}
```

```
public class AoDi implements Car {
    public void run() {
        System.out.println("我是奥迪汽车..");
    }
}
```

3)：编写工厂类

```
public class CarFactory {

    public static Car createCar(String name) {
      if ("".equals(name)) {
           return null;
      }
      if(name.equals("奥迪")){
          return new AoDi();
      }
      if(name.equals("宝马")){
          return new Bmw();
      }
      return null;
   }
}
```

4)：编写测试类

```
public class Client01 {

    public static void main(String[] args) {
        Car aodi  =CarFactory.createCar("奥迪");
        Car bmw  =CarFactory.createCar("宝马");
        aodi.run();
        bmw.run();
    }
}
```

## 234.【编程题】【设计模式】编写一个静态代理模式的 demo

1、编写接口类

```
//接口类
public class UserDao{
    public void save() {
        System.out.println("保存数据方法");
    }
}
```

2、编写代理类

```
//代理类
public class UserDaoProxy extends UserDao {
    private UserDao userDao;

    public UserDaoProxy(UserDao userDao) {
        this.userDao = userDao;
    }

    public void save() {
        System.out.println("开启事物...");
        userDao.save();
        System.out.println("关闭事物...");
    }

}
```

3、编写测试类

```
//添加完静态代理的测试类
public class Test{
    public static void main(String[] args) {
        UserDao userDao = new UserDao();
        UserDaoProxy userDaoProxy = new UserDaoProxy(userDao);
        userDaoProxy.save();
    }
}
```

## 235.【编程题】【设计模式】编写一个观察者模式的 demo

1、定义抽象观察者，每一个实现该接口的实现类都是具体观察者。

```
//观察者的接口，用来存放观察者共有方法
public interface Observer {
    // 观察者方法
    void update(int state);
}
```

2、定义具体观察者

```
// 具体观察者
public class ObserverImpl implements Observer {

    // 具体观察者的属性
    private int myState;

    public void update(int state) {
        myState=state;
        System.out.println("收到消息,myState值改为："+state);
    }

    public int getMyState() {
        return myState;
    }
}
```

3、定义主题。主题定义观察者数组，并实现增、删及通知操作。

```
//定义主题，以及定义观察者数组，并实现增、删及通知操作。
public class Subjecct {
    //观察者的存储集合，不推荐ArrayList，线程不安全，
    private Vector<Observer> list = new Vector<>();

    // 注册观察者方法
    public void registerObserver(Observer obs) {
        list.add(obs);
    }
    // 删除观察者方法
    public void removeObserver(Observer obs) {
        list.remove(obs);
    }

    // 通知所有的观察者更新
    public void notifyAllObserver(int state) {
        for (Observer observer : list) {
            observer.update(state);
        }
    }
}
```

4、定义具体的，他继承继承 Subject 类

```java
//具体主题
public class RealObserver extends Subjecct {
    //被观察对象的属性
     private int state;
     public int getState(){
         return state;
     }
     public void  setState(int state){
         this.state=state;
         //主题对象(目标对象)值发生改变
         this.notifyAllObserver(state);
     }
}
```

5、运行测试

```java
public class Client {

    public static void main(String[] args) {
        // 目标对象
        RealObserver subject = new RealObserver();
        // 创建多个观察者
        ObserverImpl obs1 = new ObserverImpl();
        ObserverImpl obs2 = new ObserverImpl();
        ObserverImpl obs3 = new ObserverImpl();
        // 注册到观察队列中
        subject.registerObserver(obs1);
        subject.registerObserver(obs2);
        subject.registerObserver(obs3);
        // 改变State状态
        subject.setState(300);
        System.out.println("obs1观察者的MyState状态值为："+obs1.getMyState());
        System.out.println("obs2观察者的MyState状态值为："+obs2.getMyState());
        System.out.println("obs3观察者的MyState状态值为："+obs3.getMyState());
        // 改变State状态
        subject.setState(400);
        System.out.println("obs1观察者的MyState状态值为："+obs1.getMyState());
        System.out.println("obs2观察者的MyState状态值为："+obs2.getMyState());
        System.out.println("obs3观察者的MyState状态值为："+obs3.getMyState());
    }
}
```

## 236.【简答题】【SpringCloud】链路跟踪 Sleuth

当我们项目中引入 Spring Cloud Sleuth 后，每次链路请求都会添加一串追踪信息，格式是[server-name, main-traceId,sub-spanId,boolean]：

1、 server-name：服务结点名称；

2、 main-traceId：一条链路唯一的 ID，为 TraceID；

3、 sub-spanId：链路中每一环的 ID，为 SpanID；

4、 boolean：是否将信息输出到 Zipkin 等服务收集和展示；

Sleuth 的实现是基于 HTTP 的，为了在数据的收集过程中不能影响到正常业务，Sleuth 会在每个请求的 Header 上添加跟踪需求的重要信息。这样在数据收集时，只需要将 Header 上的相关信息发送给对应的图像工具即可，图像工具根据上传的数据，按照 Span 对应的逻辑进行分析、展示。

## 237.【编程题】【微服务】什么是网关，网关的作用是什么

网关相当于一个网络服务架构的入口，所有网络请求必须通过网关转发到具体的服务。

网关的作用包含统一管理微服务请求，权限控制、负载均衡、路由转发、监控、安全控制黑名单和白名单等

## 238.【编程题】【Dubbo】Zookeeper 如何 保证 CP

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注册信息，但不能接受服务直接 down 掉不可用。也就是说，服务注册功能对可用性的要求要高于一致性。但是 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，剩余节点会重新进行leader 选举。问题在于，选举 leader 的时间太长，30 ~ 120s, 且选举期间整个 zk 集群都是不可用的，这就导致在选举期间注册服务瘫痪。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，虽然服务能够最终恢复，但是漫长的选举时间导致的注册长期不可用是不能容忍的。

## 239.【编程题】【SpringCloud】什么是 Eureka 的自我保护模式

默认情况下，如果 Eureka Service 在一定时间内没有接收到某个微服务的心跳，Eureka Service 会进入自我保护模式，在该模式下 Eureka Service 会保护服务注册表中的信息，不在删除注册表中的数据，当网络故障恢复后，Eureka Servic 节点会自动退出自我保护模式

## 240.【简答题】【SpringCore】说一下 Spring 的事务隔离？

Spring 有五大隔离级别，默认值为 ISOLATION_DEFAULT（使用数据库的设置），其他四个隔离级别和数据库的隔离级别一致：

ISOLATION_DEFAULT：用底层数据库的设置隔离级别，数据库设置的是什么我就用什么；

ISOLATION_READ_UNCOMMITTED：未提交读，最低隔离级别、事务未提交前，就可被其他事务读取（会出现幻读、脏读、不可重复读）；

ISOLATION_READ_COMMITTED：提交读，一个事务提交后才能被其他事务读取到（会造成幻读、不可重复读），SQL server 的默认级别；

ISOLATION_REPEATABLE_READ：可重复读，保证多次读取同一个数据时，其值都和事务开始时候的内容是一致，禁止读取到别的事务未提交的数据（会造成幻读），MySQL 的默认级别；

ISOLATION_SERIALIZABLE：序列化，代价最高最可靠的隔离级别，该隔离级别能防止脏读、不可重复读、幻读。

脏读 ：表示一个事务能够读取另一个事务中还未提交的数据。比如，某个事务尝试插入记录 A，此时该事务还未提交，然后另一个事务尝试读取到了记录 A。

不可重复读 ：是指在一个事务内，多次读同一数据。

幻读 ：指同一个事务内多次查询返回的结果集不一样。比如同一个事务 A 第一次查询时候有 n 条记录，但是第二次同等条件下查询却有 n+1 条记录，这就好像产生了幻觉。发生幻读的原因也是另外一个事务新增或者删除或者修改了第一个事务结果集里面的数据，同一个记录的数据内容被修改了，所有数据行的记录就变多或者变少了。

## 241.【简答题】【SpringCore】解释 SpringJDBC,SpringDAO,SpringORM

Spring-DAO 并非 Spring 的一个模块，它实际上是指示你写 DAO 操作、写好 DAO 操作的一些规范。因此，对于访问你的数据它既没有提供接口也没有提供实现更没有提供模板。在写一个 DAO 的时候，你应该使用 @Repository 对其进行注解，这样底层技术(JDBC，Hibernate，JPA，等等)的相关异常才能一致性地翻译为相应的 DataAccessException 子类。

Spring-JDBC 提供了 Jdbc 模板类，它移除了连接代码以帮你专注于 SQL 查询和相关参数。Spring-JDBC 还提供了一个 JdbcDaoSupport，这样你可以对你的 DAO 进行扩展开发。它主要定义了两个属性：一个 DataSource 和一个 JdbcTemplate，它们都可以用来实现 DAO 方法。JdbcDaoSupport 还提供了一个将 SQL 异常转换为 SpringDataAccessExceptions 的异常翻译器。

Spring-ORM 是一个囊括了很多持久层技术(JPA，JDO，Hibernate，iBatis)的总括模块。对于这些技术中的每一个，Spring 都提供了集成类，这样每一种技术都能够在遵循 Spring 的配置原则下进行使用，并平稳地和 Spring 事务管理进行集成。

## 242.【简答题】【SpringCore】什么是 SpringIOC 容器？以及有什么作用？

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

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

## 243.【简答题】【SpringCore】BeanFactory 与 AppliacationContext 有什么区别？

BeanFactory

基础类型的 IOC 容器，提供完成的 IOC 服务支持。如果没有特殊指定，默认采用延迟初始化策略。相对来说，容器启动初期速度较快，所需资源有限。

ApplicationContext

ApplicationContext 是在 BeanFactory 的基础上构建，是相对比较高级的容器实现，除了 BeanFactory 的所有支持外，ApplicationContext 还提供了事件发布、国际化支持等功能。ApplicationContext 管理的对象，在容器启动后默认全部初始化并且绑定完成。

## 244.【简答题】【SpringCore】简单解释一下 Spring 的 AOP？

AOP（Aspect Oriented Programming），即面向切面编程，可以说是 OOP（Object Oriented Programming，面向对象编程）的补充和完善。OOP 引入封装、继承、多态等概念来建立一种对象层次结构，用于模拟公共行为的一个集合。不过 OOP 允许开发者定义纵向的关系，但并不适合定义横向的关系，例如日志功能。日志代码往往横向地散布在所有对象层次中，而与它对应的对象的核心功能毫无关系。对于其他类型的代码，如安全性、异常处理和透明

的持续性也都是如此。这种散布在各处的无关的代码被称为横切（cross cutting），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。

## 245.【简答题】【SpringCore】Spring 的通知是什么？有哪几种类型？

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。

## 246.【简答题】【SpringMVC】简述 Spring mvc 的执行流程？

（1）用户发送请求至前端控制器 DispatcherServlet；

（2）DispatcherServlet 收到请求后，调用 HandlerMapping 处理器映射器，请求获取 Handler；

（3）处理器映射器根据请求 url 找到具体的处理器 Handler，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，请求执行 Handler；

（5）HandlerAdapter 经过适配调用 具体处理器进行处理业务逻辑；

（6）Handler 执行完成返回 ModelAndView；

（7）HandlerAdapter 将 Handler 执行结果 ModelAndView 返回给 DispatcherServlet；

（8）DispatcherServlet 将 ModelAndView 传给 ViewResolver 视图解析器进行解析；

（9）ViewResolver 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。

## 247.【简答题】【SpringMVC】SpringMVC 怎么样设定重定向和转发的？

转发：在返回值前面加"forward:"，如"forward:user.do?name=method4"

重定向：在返回值前面加"redirect:"，如"redirect:HTTP://www.baidu.com"

## 248.【简答题】【SpringMVC】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

## 249.【简答题】【SpringMVC】什么是 Spring MVC？介绍你对 SpringMVC 的理解？

Springmvc 是一个基于 java 实现了 mvc 设计模式的轻量级 Web 框架，通过把 model，view，controller 分离，将 Web 层进行解耦，把复杂的 Web 应用分成几部分，简化开发。

## 250.【简答题】【MyBatis】#{}和${}的区别是什么？

#{}是预编译处理，${}是字符串替换。MyBatis 在处理#{}时，会将 SQL 中的#{}替换为?号，调用 PreparedStatement 的 set 方法来赋值；MyBatis 在处理${}时，就是把${}替换成变量的值。使用#{}可以有效的防止 SQL 注入，提高系统安全性。

## 251.【简答题】【MyBatis】简述 MyBatis 的优点缺点？

MyBatis 的优点：

（1）基于 SQL 语句编程，相当灵活，不会对应用程序或者数据库的现有设计造成任何影响，SQL 写在 XML 里，解除 SQL 与程序代码的耦合，便于统一管理；提供 XML 标签，支持编写动态 SQL 语句，并可重用。

（2）与 JDBC 相比，减少了 50%以上的代码量，消除了 JDBC 大量冗余的代码，不需要手动开关连接；

（3）很好的与各种数据库兼容（因为 MyBatis 使用 JDBC 来连接数据库，所以只要 JDBC 支持的数据库 MyBatis 都支持）。

（4）能够与 Spring 很好的集成；

（5）提供映射标签，支持对象与数据库的 ORM 字段关系映射；提供对象关系映射标签，支持对象关系组件维护。

MyBatis 框架的缺点：

（1）SQL 语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写 SQL 语句的功底有一定要求。

（2）SQL 语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

## 252.【简答题】【MyBatis】MyBatis 动态 SQL 是做什么的？都有哪些动态 SQL？

MyBatis 动态 SQL 可以让我们在 XML 映射文件内，以标签的形式编写动态 SQL，完成逻辑判断和动态拼接 SQL 的功能，MyBatis 提供了 9 种动态 SQL 标签 trim|where|set|foreach|if|choose|when|otherwise|bind

## 253.【简答题】【SpringBoot】SpringBoot 的核心注解有哪些？它主要有那几个注解组成？

SpringBootApplication 是 SpringBoot 的核心注解，主要包含以下注解：

@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

@ComponentScan:Spring 组件扫描

## 254.【简答题】【SpringBoot】简述 SpringBoot 优点？

减少开发、测试时间

使用 JavaConfig 有助于避免使用 XML

避免大量的 maven 导入和各种版本冲突

通过提供默认值快速开发

没有单独的 Web 服务器需要

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管理。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

## 255.【简答题】【SpringBoot】简述 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；

## 256.【简答题】【SpringCore】简述 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。

## 257.【简答题】【SpringCore】简述 Spring 中的 bean 的作用域有哪些？

1.singleton：唯一 bean 实例，Spring 中的 bean 默认都是单例的。

2.prototype：每次请求都会创建一个新的 bean 实例。

3.request：每一次 HTTP 请求都会产生一个新的 bean，该 bean 仅在当前 HTTP request 内有效。

4.session：每一次 HTTP 请求都会产生一个新的 bean，该 bean 仅在当前 HTTP session 内有效。

## 258.【简答题】【SpringCore】简述@Component 和@Bean 的区别是什么？

1.作用对象不同。@Component 注解作用于类，而@Bean 注解作用于方法。

2.@Component 注解通常是通过类路径扫描来自动侦测以及自动装配到 Spring 容器中（我们可以使用 @ComponentScan 注解定义要扫描的路径）。@Bean 注解通常是在标有该注解的方法中定义产生这个 bean，告诉 Spring 这是某个类的实例，当我需要用它的时候还给我。

3.@Bean 注解比@Component 注解的自定义性更强，而且很多地方只能通过@Bean 注解来注册 bean。比如当引用第三方库的类需要装配到 Spring 容器的时候，就只能通过@Bean 注解来实现。

## 259.【简答题】【SpringCore】说一说 Spring 依赖注入方式。

1.Set 注入

2.构造器注入

3.静态工厂的方法注入

4.实例工厂的方法注入

## 260.【简答题】【SpringCore】说一说 Spring 的事务实现方式。

编程式事务管理：你可以通过编程的方式管理事务，这种方式带来了很大的灵活性，但很难维护。

声明式事务管理：这种方式意味着你可以将事务管理和业务代码分离。你只需要通过注解或者 XML 配置管理事务。

## 261.【简答题】【SpringMVC】如何解决 POST 请求中文乱码问题，GET 的又如何处理呢？

（1）解决 post 请求乱码问题：在 Web.XML 中配置一个 CharacterEncodingFilter 过滤器，设置成 utf-8；

```
<filter>
  <filter-name>CharacterEncodingFilter</filter-name>
```

```
    <filter-class>org.Springframework.Web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>utf-8</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

（2）get 请求中文参数出现乱码解决方法有两个：

①修改 tomcat 配置文件添加编码与工程编码一致，如下：

```
<ConnectorURIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1"
redirectPort="8443"/>
```

②另外一种方法对参数进行重新编码：

```
String userName = new String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")
```

ISO8859-1 是 tomcat 默认编码，需要将 tomcat 编码后的内容按 utf-8 编码。

## 262.【简答题】【MyBatis】说一说 MyBatis 中接口绑定有几种实现方式,分别是怎么实现的?

参考答案:

接口绑定有两种实现方式：

一种是通过注解绑定,就是在接口的方法上面加上@Select@Update 等注解里面包含 SQL 语句来绑定。

另外一种就是通过 XML 里面写 SQL 来绑定,在这种情况下,要指定 XML 映射文件里面的 namespace 必须为接口的全路径名。

## 263.【简答题】【MyBatis】为什么说 MyBatis 是半自动 ORM 映射工具？它与全自动的区别在哪里？

1、Hibernate 属于全自动 ORM 映射工具，使用 Hibernate 查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而 MyBatis 在查询关联对象或关联集合对象时，需要手动编写 SQL 来完成，所以，称之为半自动 ORM 映射工具。

2、MyBatis 直接编写原生态 SQL，可以严格控制 SQL 执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，因为这类软件需求变化频繁，一但需求变化要求迅速输出成果。但是灵活的前提是 MyBatis 无法做到数据库无关性，如果需要实现支持多种数据库的软件，则需要自定义多套 SQL 映射文件，工作量大。

3、Hibernate 对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件，如果用 Hibernate 开发可以节省很多代码，提高效率。

## 264.【简答题】【SpringCore】 解释 Spring 框架中 bean 的生命周期 (重要)

要彻底搞清楚 Spring 的生命周期，首先要把这四个阶段牢牢记住。实例化和属性赋值对应构造方法和 setter 方法的注入，初始化和销毁是用户能自定义扩展的两个阶段。在这四步之间穿插的各种扩展点，稍后会讲。

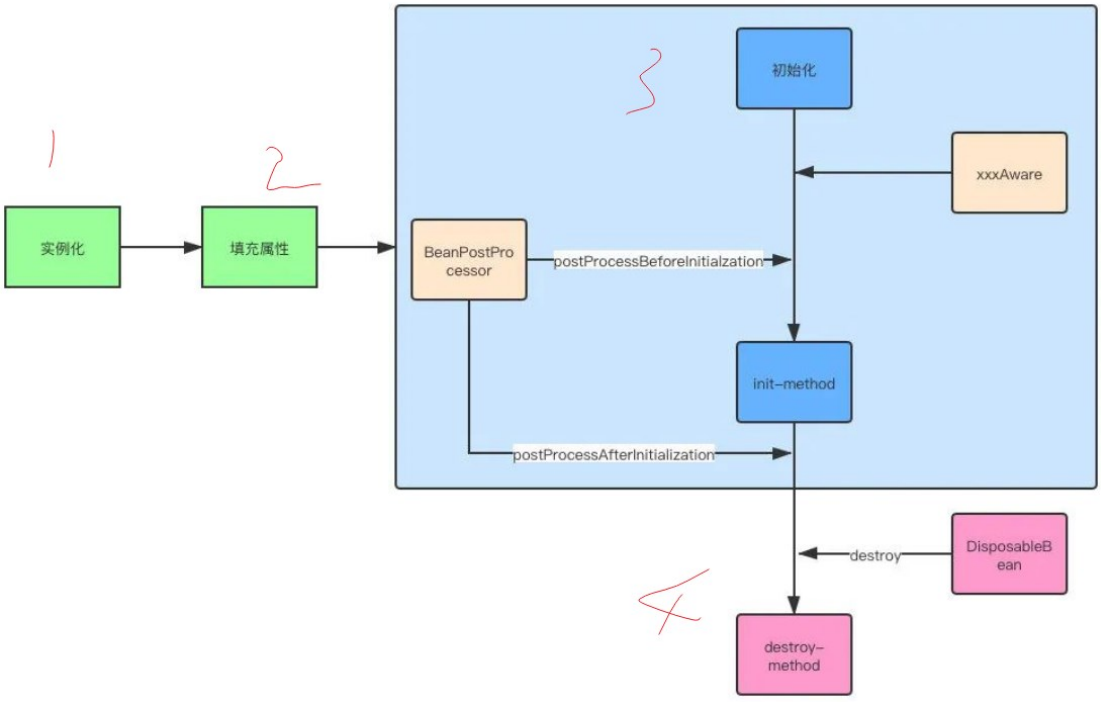
实例化 Instantiation

属性赋值 Populate

初始化 Initialization

销毁 Destruction

实例化 -> 属性赋值 -> 初始化 -> 销毁




在这里插入图片描述

各个阶段的工作:

1、实例化，创建一个 Bean 对象；

2、填充属性，为属性赋值；

3、初始化

如果实现了 xxxAware 接口，通过不同类型的 Aware 接口拿到 Spring 容器的资源

如果实现了 BeanPostProcessor 接口，则会回调该接口的 postProcessBeforeInitialzation 和 postProcessAfterInitialization 方法

如果配置了 init-method 方法，则会执行 init-method 配置的方法

4、销毁

容器关闭后，如果 Bean 实现了 DisposableBean 接口，则会回调该接口的 destroy 方法

如果配置了 destroy-method 方法，则会执行 destroy-method 配置的方法

## 265.【简答题】【SpringCore】Spring AOP 的实现原理和场景；

AOP（Aspect Orient Programming），作为面向对象编程的一种补充，广泛应用于处理一

些具有横切性质的系统级服务。

一、场景

事务管理、安全检查、权限控制、数据校验、缓存、对象池管理等

二、实现技术

AOP（这里的 AOP 指的是面向切面编程思想，而不是 Spring AOP）主要的的实现技术主要有 Spring AOP 和 AspectJ。

1、AspectJ 的底层技术。AspectJ 的底层技术是静态代理，即用一种 AspectJ 支持的特定语言编写切面，通过一个命令来

编译，生成一个新的代理类，该代理类增强了业务类，这是在编译时增强，相对于下面说的运行时增强，编译时增强的性能更好。

2、Spring AOP

Spring AOP 采用的是动态代理，在运行期间对业务方法进行增强，所以不会生成新类，对于 动态代理技术，Spring AOP 提供了对 JDK 动态代理的支持以及 CGLib 的支持。

JDK 动态代理只能为接口创建动态代理实例，而不能对类创建动态代理。需要获得被目标类的 接口信息（应用 Java 的反射技术），生成一个实现了代理接口的动态代理类（字节码），再通过反射机制获得动态代理类的构造函数，利用构造函数生成动态代理类的实例对象，在调用具体方法前调用 invokeHandler 方法来处理。

CGLib 动态代理需要依赖 asm 包，把被代理对象类的 class 文件加载进来，修改其字节码生成子类。但是 Spring AOP 基于注解配置的情况下，需要依赖于 AspectJ 包的标准注解。

## 266.【简答题】【SpringCore】Spring 的 IOC 和 AOP 介绍一下

IOC 解析

IoC（Inverse of Control:控制反转）是一种设计思想，就是 将原本在程序中手动创建对象的控制权，交由 Spring 框架来管理。 IoC 在其他语言中也有应用，并非 Spirng 特有。 IoC 容器是 Spring 用来实现 IoC 的载体， IoC 容器实际上就是个 Map（key，value）,Map 中存放的是各种对象。

将对象之间的相互依赖关系交给 IoC 容器来管理，并由 IoC 容器完成对象的注入。这样可以很大程度上简化应

用的开发，把应用从复杂的依赖关系中解放出来。 IoC 容器就像是一个工厂一样，当我们需要创建一个对象的时候，只需要配置好配置文件/注解即可，完全不用考虑对象是如何被创建出来的。

在实际项目中一个 Service 类可能有几百甚至上千个类作为它的底层，假如我们需要实例化这个 Service，你可能要每次都要搞清这个 Service 所有底层类的构造函数，这可能会把人逼疯。如果利用 IoC 的话，你只需要配置好，然后在需要的地方引用就行了，这大大增加了项目的可维护性且降低了开发难度。

Spring 时代我们一般通过 XML 文件来配置 Bean，后来开发人员觉得 XML 文件来配置不太好，于是 SpringBoot 注解配置就慢慢开始流行起来。

注入方式：setter 注入、构造方法注入、接口注入。

AOP 解析

AOP(Aspect-Oriented Programming:面向切面编程) 能够将那些与业务无关，却为业务模块所共同调用的逻辑或责任（例如事务处理、日志管理、权限控制等）封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可拓展性和可维护性。

Spring AOP 就是基于动态代理的，如果要代理的对象，实现了某个接口，那么 Spring AOP 会使用 JDK Proxy，去创建代理对象，而对于没有实现接口的对象，就无法使用 JDK Proxy 去进行代理了，这时候 Spring AOP 会使用 Cglib ，这时候 Spring AOP 会使用 Cglib 生成一个被代理对象的子类来作为代理。

使用 AOP 之后我们可以把一些通用功能抽象出来，在需要用到的地方直接使用即可，这样大大简化了代码量。我们需要增加新功能时也方便，这样也提高了系统扩展性。日志功能、事务管理等等场景都用到了 AOP 。

## 267.【简答题】【SSM 框架】 MyBatis 如何进行分页？

MyBatis 它是针对 ResultSet 结果集执行的内存分页，而非物理分页，可以在 SQL 内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用 MyBatis 提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的 SQL，然后重写 SQL，添加对应的物理分页语句和物理分页参数。

## 268.【简答题】【Spring】Spring 中的 bean 是线程安全的吗？

容器本身并没有提供 Bean 的线程安全策略，因此可以说 Spring 容器中的 Bean 本身不具备线程安全的特性，但是具体还是要结合具体 scope 的 Bean 去研究。

## 269.【简答题】【SpringMVC 方向】什么是 Spring MVC？简单介绍下你对 Spring MVC 的理解？

Spring MVC 是一个基于 Java 的实现了 MVC 设计模式的请求驱动类型的轻量级 Web 框架，通过把模型-视图-控制器分离，将 Web 层进行职责解耦，把复杂的 Web 应用分成逻辑清晰的几部分，简化开发，减少出错，方便组内开发人员之间的配合。

Spring MVC 具有以下优点：

（1）可以支持各种视图技术,而不仅仅局限于 JSP；

（2）与 Spring 框架集成（如 IoC 容器、AOP 等）；

（3）清晰的角色分配：前端控制器(dispatcherServlet) , 请求到处理器映射（handlerMapping), 处理器适配器（HandlerAdapter), 视图解析器（ViewResolver）。

（4） 支持各种请求资源的映射策略。

Spring MVC 的主要组件？

（1）前端控制器 DispatcherServlet（不需要程序员开发）

作用：接收请求、响应结果，相当于转发器，有了 DispatcherServlet 就减少了其它组件之间的耦合度。

（2）处理器映射器 HandlerMapping（不需要程序员开发）

作用：根据请求的 URL 来查找 Handler。

（3）处理器适配器 HandlerAdapter

注意：在编写 Handler 的时候要按照 HandlerAdapter 要求的规则去编写，这样适配器 HandlerAdapter 才可以正确的去执行 Handler。

（4）处理器 Handler（需要程序员开发）

（5）视图解析器 ViewResolver（不需要程序员开发）

作用：进行视图的解析，根据视图逻辑名解析成真正的视图（view）。

（6）视图 View（需要程序员开发 jsp）

View 是一个接口， 它的实现类支持不同的视图类型（jsp，freemarker，pdf 等等）。

## 270.【简答题】【SpringMVC 方向】Spring MVC 常用的注解有哪些？

@Controller： 用于标记在一个类上，使用它标记的类就是一个 SpringMVC Controller 对象。分发处理器将会扫描使用了该注解的类的方法，并检测该方法是否使用了@RequestMapping 注解。@Controller 只是定义了一个控制器类，而使用@RequestMapping 注解的方法才是真正处理请求的处理器。

@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 conreoller 方法返回对象转化为 json 对象响应给客户。

@RequestParam ：将请求的参数绑定到方法中的参数上，示例：@RequestParam(value = “name”, required = false) String name。

@PathVariable：用于对应 restful 风格 url 中的参数，示例：@RequestMapping(value="/happy/{dayid}") findPet(@PathVariable String dayid)。

## 271.【简答题】【SpringMVC 方向】Spring MVC 里面拦截器是怎么写的？

有两种写法,一种是实现 HandlerInterceptor 接口，另外一种是继承适配器类，接着在接口方法当中，实现处理逻

辑；然后在 Spring MVC 的配置文件中配置拦截器即可：

```
<!-- 配置Spring MVC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.zwp.action.MyHandlerInterceptor"></bean>
    <!-- 只针对部分请求拦截 -->
    <mvc:interceptor>
       <mvc:mapping path="/modelMap.do" />
       <bean class="com.zwp.action.MyHandlerInterceptorAdapter" />
    </mvc:interceptor>
</mvc:interceptors>
```

## 272.【简答题】【Spring】解释一下 Spring AOP 里面的几个名词？

（1）切面（Aspect）：切面是通知和切点的结合。通知和切点共同定义了切面的全部内容。 在 Spring AOP 中，切面可以使用通用类（基于模式的风格） 或者在普通类中以 @AspectJ 注解来实现。

（2）连接点（Join point）：指方法，在 Spring AOP 中，一个连接点 总是 代表一个方法的执行。 应用可能有数以千计的时机应用通知。这些时机被称为连接点。连接点是在应用执行过程中能够插入切面的一个点。这个点可以是调用方法时、抛出异常时、甚至修改一个字段时。切面代码可以利用这些点插入到应用的正常流程之中，并添加新的行为。

（3）通知（Advice）：在 AOP 术语中，切面的工作被称为通知。

（4）切入点（Pointcut）：切点的定义会匹配通知所要织入的一个或多个连接点。我们通常使用明确的类和方法名称，或是利用正则表达式定义所匹配的类和方法名称来指定这些切点。

（5）引入（Introduction）：引入允许我们向现有类添加新方法或属性。

（6）目标对象（Target Object）： 被一个或者多个切面（aspect）所通知（advise）的对象。它通常是一个代理对象。也有人把它叫做 被通知（adviced） 对象。 既然 Spring AOP 是通过运行时代理实现的，这个对象永远是一个被代理（proxied） 对象。

（7）织入（Weaving）：织入是把切面应用到目标对象并创建新的代理对象的过程。在目标对象的生命周期里有多少个点可以进行织入：

编译期：切面在目标类编译时被织入。AspectJ 的织入编译器是以这种方式织入切面的。

类加载期：切面在目标类加载到 JVM 时被织入。需要特殊的类加载器，它可以在目标类被引入应用之前增强该目标类的字节码。AspectJ5 的加载时织入就支持以这种方式织入切面。

运行期：切面在应用运行的某个时刻被织入。一般情况下，在织入切面时，AOP 容器会为目标对象动态地创建一个代理对象。SpringAOP 就是以这种方式织入切面。

## 273.【简答题】【SpringBoot 方向】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

## 274.【简答题】【SpringBoot 方向】 Spring Boot 的核心配置文件有哪几个？它们的区别是什么？

SpringBoot 的核心配置文件有 application 和 bootstarp 配置文件。

application 文件主要用于 SpringBoot 自动化配置文件。

bootstarp 文件主要有以下几种用途：

使用 Spring Cloud Config 注册中心时 需要在 bootStarp 配置文件中添加链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

## 275.【简答题】【SpringBoot 方向】 SpringBoot 的核心注解是哪个？它主要由哪几个注解组成的？

@SpringBootApplication: Spring Boot 应用标注在某个类上说明这个类是 SpringBoot 的主配置类，SpringBoot 就应该运行这个类的 main 方法来启动 SpringBoot 应用；

@SpringBootConfiguration:Spring Boot 的配置类： 标注在某个类上，表示这是一个 Spring Boot 的配置类；

@Configuration:配置类上来标注这个注解：配置类 ----- 配置文件；配置类也是容器中的一个组件；@Component

@EnableAutoConfiguration：开启自动配置功能；

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
      @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
      @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
```

## 276.【简答题】【SpringBoot 方向】Spring Boot 自动配置原理是什么？

SpringBoot 启动会加载大量的自动配置类

我们看我们需要的功能有没有在 SpringBoot 默认写好的自动配置类当中；

我们再来看这个自动配置类中到底配置了哪些组件；（只要我们要用的组件存在在其中，我们就不需要再手动配置了）

给容器中自动配置类添加组件的时候，会从 properties 类中获取某些属性。我们只需要在配置文件中指定这些属性的值即可；

xxxxAutoConfigurartion：自动配置类；给容器中添加组件

xxxxProperties:封装配置文件中相关属性；

## 277.【简答题】【SpringCore 方向】@Autowired 与@Resource 的使用和区别？

Spring 不但支持自己定义的@Autowired 注解，还支持几个由 JSR-250 规范定义的注解，它们分别是@Resource、@PostConstruct 以及@PreDestroy。

@Resource 的作用相当于@Autowired，只不过@Autowired 按 byType 自动注入，而@Resource 默认按 byName 自动注入罢了。@Resource 有两个属性是比较重要的，分是 name 和 type，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。如果既不指定 name 也不指定 type 属性，这时将通过反射机制使用 byName 自动注入策略。

@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型匹配的唯一 bean 进行装配，找不到或者找到多个，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；

@Autowired 与@Resource 的区别：

1、 @Autowired 与@Resource 都可以用来装配 bean. 都可以写在字段上,或写在 setter 方法上。

2、 @Autowired 默认按类型装配（这个注解是属于 Spring 的），默认情况下必须要求依赖对象必须存在，如果要允许 null 值，可以设置它的 required 属性为 false，如：@Autowired(required=false) ，如果我们想使用名称装配可以结合@Qualifier 注解进行使用，如下：

```
    @Autowired () @Qualifier ( "base Dao" )
    private BaseDao baseDao;
```

3、@Resource（这个注解属于 J2EE 的），默认按照名称进行装配，名称可以通过 name 属性进行指定，如果没有指定 name 属性，当注解写在字段上时，默认取字段名进行安装名称查找，如果注解写在 setter 方法上默认取属性

名进行装配。当找不到与名称匹配的 bean 时才按照类型进行装配。但是需要注意的是，如果 name 属性一旦指定，就只会按照名称进行装配。

```
    @Resource (name= "baseDao" )
    private BaseDao baseDao;
```

推荐使用：@Resource 注解在字段上，这样就不用写 setter 方法了，并且这个注解是属于 J2EE 的，减少了与 Spring 的耦合。这样代码看起就比较优雅。

Spring @Qualifier 注解

@Autowired 是根据类型进行自动装配的。如果当 Spring 上下文中存在不止一个 UserDao 类型的 bean 时，就会抛出 BeanCreationException 异常;如果 Spring 上下文中不存在 UserDao 类型的 bean，也会抛出 BeanCreationException 异常。我们可以使用@Qualifier 配合@Autowired 来解决这些问题。如下：

①可能存在多个 UserDao 实例

```
@Autowired
@Qualifier("userServiceImpl")
public IUserService userService;
```

或者

```
@Autowired
public void setUserDao(@Qualifier("userDao") UserDao userDao) {
   this.userDao = userDao;
}
```

这样 Spring 会找到 id 为 userServiceImpl 和 userDao 的 bean 进行装配。

②可能不存在 UserDao 实例

```
@Autowired(required = false)
public IUserService userService
```

个人总结：

@Autowired//默认按 type 注入
@Qualifier("cusInfoService")//一般作为@Autowired()的修饰用
@Resource(name="cusInfoService")//默认按 name 注入，可以通过 name 和 type 属性进行选择性注入

一般@Autowired 和@Qualifier 一起用，@Resource 单独用。

当然没有冲突的话@Autowired 也可以单独用

## 278.【简答题】【Redis】使用 Redis 有哪些好处？

(1) 速度快，因为数据存在内存中，类似于 HashMap，HashMap 的优势就是查找和操作的时间复杂度都是 O(1)

(2) 支持丰富数据类型，支持 string，list，set，sorted set，hash

(3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行

(4) 丰富的特性：可用于缓存，消息，按 key 设置过期时间，过期后将会自动删除

## 279.【简答题】【Redis】Redis 常见性能问题和解决方案

(1) Master 最好不要做任何持久化工作，如 RDB 内存快照和 AOF 日志文件；

(2) 如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步一次；

(3) 为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网内；

(4) 尽量避免在压力很大的主库上增加从库；

(5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3…

这样的结构方便解决单点故障问题，实现 Slave 对 Master 的替换。如果 Master 挂了，可以立刻启用 Slave1 做 Master，其他不变。

## 280.【简答题】【SSM 框架】XML 文件对应的 mapper 中的方法是否可以重载，为什么

不可以。会导致方法无法定位。

## 281.【简答题】【MQ 消息队列】为什么使用 MQ？MQ 的优点

**简答：**

异步处理 - 相比于传统的串行、并行方式，提高了系统吞吐量。

应用解耦 - 系统间通过消息通信，不用关心其他系统的处理。

流量削锋 - 可以通过消息队列长度控制请求量；可以缓解短时间内的高并发请求。

日志处理 - 解决大量日志传输。

消息通讯 - 消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等。

**详答：**

主要是：解耦、异步、削峰。

解耦：A 系统发送数据到 BCD 三个系统，通过接口调用发送。如果 E 系统也要这个数据呢？那如果 C 系统现在不需要了呢？A 系统负责人几乎崩溃…A 系统跟其它各种乱七八糟的系统严重耦合，A 系统产生一条比较关键的数据，很多系统都需要 A 系统将这个数据发送过来。如果使用 MQ，A 系统产生一条数据，发送到 MQ 里面去，哪个系统需要数据自己去 MQ 里面消费。如果新系统需要数据，直接从 MQ 里消费即可；如果某个系统不需要这条数据了，就取消对 MQ 消息的消费即可。这样下来，A 系统压根儿不需要去考虑要给谁发送数据，不需要维护这个代码，也不需要考虑人家是否调用成功、失败超时等情况。

就是一个系统或者一个模块，调用了多个系统或者模块，互相之间的调用很复杂，维护起来很麻烦。但是其实这个调用是不需要直接同步调用接口的，如果用 MQ 给它异步化解耦。

异步：A 系统接收一个请求，需要在自己本地写库，还需要在 BCD 三个系统写库，自己本地写库要 3ms，BCD 三个系统分别写库要 300ms、450ms、200ms。最终请求总延时是 3 + 300 + 450 + 200 = 953ms，接近 1s，用户

感觉搞个什么东西，慢死了慢死了。用户通过浏览器发起请求。如果使用 MQ，那么 A 系统连续发送 3 条消息到 MQ 队列中，假如耗时 5ms，A 系统从接受一个请求到返回响应给用户，总时长是 3 + 5 = 8ms。

削峰：减少高峰时期对服务器压力。

## 282.【简答题】【MQ 消息队列】说说设计 MQ 思路？

比如说这个消息队列系统，我们从以下几个角度来考虑一下：

首先这个 MQ 得支持可伸缩性吧，就是需要的时候快速扩容，就可以增加吞吐量和容量，那怎么搞？设计个分布式的系统呗，参照一下 kafka 的设计理念，broker -> topic -> partition，每个 partition 放一个机器，就存一部分数据。如果现在资源不够了，简单啊，给 topic 增加 partition，然后做数据迁移，增加机器，不就可以存放更多数据，提供更高的吞吐量了？

其次你得考虑一下这个 MQ 的数据要不要落地磁盘吧？那肯定要了，落磁盘才能保证别进程挂了数据就丢了。那落磁盘的时候怎么落啊？顺序写，这样就没有磁盘随机读写的寻址开销，磁盘顺序读写的性能是很高的，这就是 kafka 的思路。

其次你考虑一下你的 MQ 的可用性啊？这个事儿，具体参考之前可用性那个环节讲解的 kafka 的高可用保障机制。多副本 -> leader & follower -> broker 挂了重新选举 leader 即可对外服务。

能不能支持数据 0 丢失啊？可以的，参考我们之前说的那个 kafka 数据零丢失方案。

## 283.【简答题】【MQ 消息队列】RabbitMQ 的使用场景

（1）服务间异步通信

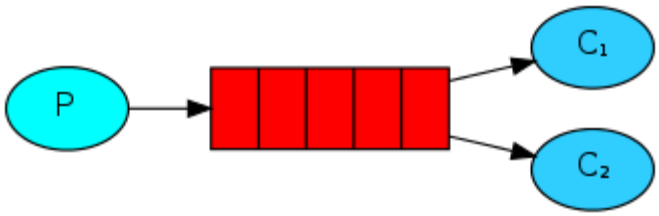
（2）顺序消费

（3）定时任务

（4）请求削峰

## 284.【简答题】【MQ 消息队列】RabbitMQ 的工作模式

一.simple 模式（即最简单的收发模式）

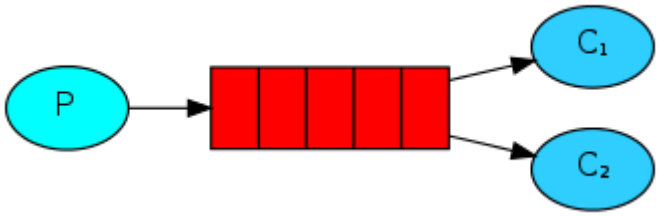



1.消息产生消息，将消息放入队列

2.消息的消费者(consumer) 监听 消息队列,如果队列中有消息,就消费掉,消息被拿走后,自动从队列中删除(隐患 消息可能没有被消费者正确处理,已经从队列中消失了,造成消息的丢失，这里可以设置成手动的 ack,但如果设置成手动

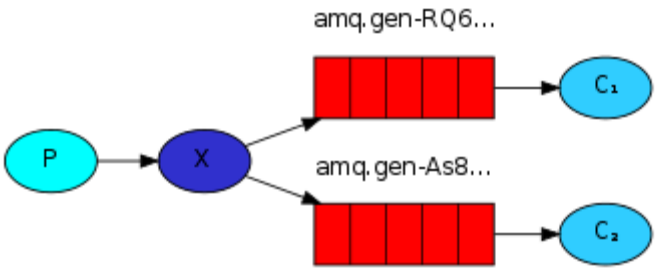
ack，处理完后要及时发送 ack 消息给队列，否则会造成内存溢出)。

二.work 工作模式(资源的竞争)




1.消息产生者将消息放入队列消费者可以有多个,消费者 1,消费者 2 同时监听同一个队列,消息被消费。C1 C2 共同争抢当前的消息队列内容,谁先拿到谁负责消费消息(隐患：高并发情况下,默认会产生某一个消息被多个消费者共同使用,可以设置一个开关(syncronize) 保证一条消息只能被一个消费者使用)。

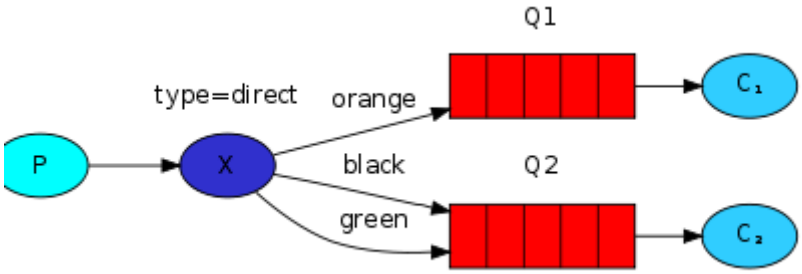
三.publish/subscribe 发布订阅(共享资源)




1、每个消费者监听自己的队列；

2、生产者将消息发给 broker，由交换机将消息转发到绑定此交换机的每个队列，每个绑定交换机的队列都将接收到消息。

四.routing 路由模式



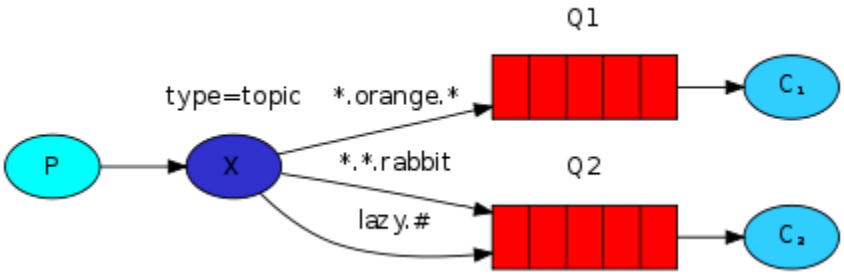

1.消息生产者将消息发送给交换机按照路由判断,路由是字符串(info) 当前产生的消息携带路由字符(对象的方法),交换机根据路由的 key,只能匹配上路由 key 对应的消息队列,对应的消费者才能消费消息;

2.根据业务功能定义路由字符串

3.从系统的代码逻辑中获取对应的功能字符串,将消息任务扔到对应的队列中。

4.业务场景:error 通知;EXCEPTION;错误通知的功能;传统意义的错误通知;客户通知;利用 key 路由,可以将程序中的错误封装成消息传入到消息队列中,开发者可以自定义消费者,实时接收错误;

五.topic 主题模式(路由模式的一种)




1.星号井号代表通配符

2.星号代表多个单词,井号代表一个单词

3.路由功能添加模糊匹配

4.消息产生者产生消息,把消息交给交换机

5.交换机根据 key 的规则模糊匹配到对应的队列,由队列的监听消费者接收消息消费

## 285.【简答题】【MQ 消息队列】消息如何分发？

若该队列至少有一个消费者订阅，消息将以循环（round-robin）的方式发送给消费者。每条消息只会分发给一个订阅的消费者（前提是消费者能够正常处理消息并进行确认）。通过路由可实现多消费的功能

## 286.【简答题】【MQ 消息队列】消息怎么路由？

消息提供方->路由->一至多个队列消息发布到交换器时，消息将拥有一个路由键（routing key），在消息创建时设定。通过队列路由键，可以把队列绑定到交换器上。消息到达交换器后，RabbitMQ 会将消息的路由键与队列的路由键进行匹配（针对不同的交换器有不同的路由规则）；

常用的交换器主要分为一下三种：

fanout：如果交换器收到消息，将会广播到所有绑定的队列上

direct：如果路由键完全匹配，消息就被投递到相应的队列

topic：可以使来自不同源头的消息能够到达同一个队列。 使用 topic 交换器时，可以使用通配符

## 287.【简答题】【MQ 消息队列】如何保证消息不被重复消费？或者说，如何保证消息消费时的幂等性？

先说为什么会重复消费：正常情况下，消费者在消费消息的时候，消费完毕后，会发送一个确认消息给消息队列，消息队列就知道该消息被消费了，就会将该消息从消息队列中删除；

但是因为网络传输等等故障，确认信息没有传送到消息队列，导致消息队列不知道自己已经消费过该消息了，再次将消息分发给其他的消费者。

针对以上问题，一个解决思路是：保证消息的唯一性，就算是多次传输，不要让消息的多次消费带来影响；保证消息等幂性；

比如：在写入消息队列的数据做唯一标示，消费消息时，根据唯一标识判断是否消费过；

假设你有个系统，消费一条消息就往数据库里插入一条数据，要是你一个消息重复两次，你不就插入了两条，这数据不就错了？但是你要是消费到第二次的时候，自己判断一下是否已经消费过了，若是就直接扔了，这样不就保留了一条数据，从而保证了数据的正确性。

## 288.【简答题】【MQ 消息队列】Apache Kafka 是什么？

Apach Kafka 是一款分布式流处理框架，用于实时构建流处理应用。它有一个核心 的功能广为人知，即作为企业级的消息引擎被广泛使用。

## 289.【简答题】【MQ 消息队列】什么是消费者组？

消费者组是 Kafka 独有的概念，如果面试官问这 个，就说明他对此是有一定了解的。我先给出标准答案：

1、定义：即消费者组是 Kafka 提供的可扩展且具有容错性的消费者机制。

2、原理：在 Kafka 中，消费者组是一个由多个消费者实例 构成的组。多个实例共同订阅若干个主题，实现共同消费。同一个组下的每个实例都配置有 相同的组 ID，被分配不同的订阅分区。当某个实例挂掉的时候，其他实例会自动地承担起 它负责消费的分区。

## 290.【简答题】【Redis】Redis 有哪些数据结构？

字符串 String、字典 Hash、列表 List、集合 Set、有序集合 SortedSet。

如果你是 Redis 中高级用户，还需要加上下面几种数据结构 HyperLogLog、Geo、Pub/Sub。

## 291.【简答题】【Redis】使用过 Redis 分布式锁么，它是什么回事？

先拿 setnx 来争抢锁，抢到之后，再用 expire 给锁加一个过期时间防止锁忘记了释放。

这时候对方会告诉你说你回答得不错，然后接着问如果在 setnx 之后执行 expire 之前进程意外 crash 或者要重启维护了，那会怎么样？

这时候你要给予惊讶的反馈：唉，是喔，这个锁就永远得不到释放了。紧接着你需要抓一抓自己得脑袋，故作思考片刻，好像接下来的结果是你主动思考出来的，然后回答：我记得 set 指令有非常复杂的参数，这个应该是可以同时把 setnx 和 expire 合成一条指令来用的！对方这时会显露笑容，心里开始默念：嗯，这小子还不错。

## 292.【简答题】【Redis】Redis 如何做持久化的？

bgsave 做镜像全量持久化，aof 做增量持久化。因为 bgsave 会耗费较长时间，不够实时，在停机的时候会导致大量丢失数据，所以需要 aof 来配合使用。在 Redis 实例重启时，优先使用 aof 来恢复内存的状态，如果没有 aof 日志，就会使用 rdb 文件来恢复。

如果再问 aof 文件过大恢复时间过长怎么办？你告诉面试官，Redis 会定期做 aof 重写，压缩 aof 文件日志大小。如果面试官不够满意，再拿出杀手锏答案，Redis4.0 之后有了混合持久化的功能，将 bgsave 的全量和 aof 的增量做了融合处理，这样既保证了恢复的效率又兼顾了数据的安全性。这个功能甚至很多面试官都不知道，他们肯定会对你刮目相看。

如果对方追问那如果突然机器掉电会怎样？取决于 aof 日志 sync 属性的配置，如果不要求性能，在每条写指令时都 sync 一下磁盘，就不会丢失数据。但是在高性能的要求下每次都 sync 是不现实的，一般都使用定时 sync，比如 1s1 次，这个时候最多就会丢失 1s 的数据。

## 293.【简答题】【Redis】Pipeline 有什么好处，为什么要用 pipeline？

可以将多次 IO 往返的时间缩减为一次，前提是 pipeline 执行的指令之间没有因果相关性。使用 Redis-benchmark 进行压测的时候可以发现影响 Redis 的 QPS 峰值的一个重要因素是 pipeline 批次指令的数目。

## 294.【简答题】【Redis】Redis 的同步机制了解么？

从从同步。第一次同步时，主节点做一次 bgsave，并同时将后续修改操作记录到内存 buffer，待完成后将 rdb 文件全量同步到复制节点，复制节点接受完成后将 rdb 镜像加载到内存。加载完成后，再通知主节点将期间修改的操作记录同步到复制节点进行重放就完成了同步过程。

## 295.【简答题】【Redis】是否使用过 Redis 集群，集群的原理是什么？

Redis Sentinal 着眼于高可用，在 master 宕机时会自动将 slave 提升为 master，继续提供服务。
Redis Cluster 着眼于扩展性，在单个 Redis 内存不足时，使用 Cluster 进行分片存储。

## 296.【简答题】【Redis】Redis 里面有 1 亿个 key，其中有 10w 个 key 是以某个固定的已知的前缀开头的，如何将它们全部找出来？

使用 keys 指令可以扫出指定模式的 key 列表。
对方接着追问：如果这个 Redis 正在给线上的业务提供服务，那使用 keys 指令会有什么问题？
这个时候你要回答 Redis 关键的一个特性：Redis 的单线程的。keys 指令会导致线程阻塞一段时间，线上服务会停顿，直到指令执行完毕，服务才能恢复。这个时候可以使用 scan 指令，scan 指令可以无阻塞的提取出指定模式的 key 列表，但是会有一定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用 keys 指令长。

## 297.【简答题】【Redis】在项目中缓存是如何使用的？为什么要用缓存？缓存使用不当会造成什么后果？

面试官心理分析

这个问题，互联网公司必问，要是一个人连缓存都不太清楚，那确实比较尴尬。

只要问到缓存，上来第一个问题，肯定是先问问你项目哪里用了缓存？为啥要用？不用行不行？如果用了以后可能会有什么不良的后果？

这就是看看你对缓存这个东西背后有没有思考，如果你就是傻乎乎的瞎用，没法给面试官一个合理的解答，那面试官对你印象肯定不太好，觉得你平时思考太少，就知道干活儿。

面试题剖析

项目中缓存是如何使用的？

这个，需要结合自己项目的业务来。

为什么要用缓存？

用缓存，主要有两个用途：高性能、高并发。

高性能

假设这么个场景，你有个操作，一个请求过来，吭哧吭哧你各种乱七八糟操作 MySQL，半天查出来一个结果，耗时 600ms。但是这个结果可能接下来几个小时都不会变了，或者变了也可以不用立即反馈给用户。那么此时咋办？

缓存啊，折腾 600ms 查出来的结果，扔缓存里，一个 key 对应一个 value，下次再有人查，别走 MySQL 折腾 600ms 了，直接从缓存里，通过一个 key 查出来一个 value，2ms 搞定。性能提升 300 倍。

就是说对于一些需要复杂操作耗时查出来的结果，且确定后面不怎么变化，但是有很多读请求，那么直接将查询出来的结果放在缓存中，后面直接读缓存就好。

高并发

所以要是你有个系统，高峰期一秒钟过来的请求有 1 万，那一个 MySQL 单机绝对会死掉。你这个时候就只能上缓存，把很多数据放缓存，别放 MySQL。缓存功能简单，说白了就是 key-value 式操作，单机支撑的并发量轻松一秒几万十几万，支撑高并发 so easy。单机承载并发量是 MySQL 单机的几十倍。

缓存是走内存的，内存天然就支撑高并发。

用了缓存之后会有什么不良后果？

常见的缓存问题有以下几个：

缓存与数据库双写不一致 、缓存雪崩、缓存穿透、缓存并发竞争后面再详细说明。

## 298.【简答题】【Redis】Redis 的过期策略都有哪些？内存淘汰机制都有哪些？

面试官心理分析

如果你连这个问题都不知道，上来就懵了，回答不出来，那线上你写代码的时候，想当然的认为写进 Redis 的数据就一定会存在，后面导致系统各种 bug，谁来负责？

常见的有两个问题：

（1）往 Redis 写入的数据怎么没了？

可能有同学会遇到，在生产环境的 Redis 经常会丢掉一些数据，写进去了，过一会儿可能就没了。我的天，同学，你问这个问题就说明 Redis 你就没用对啊。Redis 是缓存，你给当存储了是吧？

啥叫缓存？用内存当缓存。内存是无限的吗，内存是很宝贵而且是有限的，磁盘是廉价而且是大量的。可能一台机器就几十个 G 的内存，但是可以有几个 T 的硬盘空间。Redis 主要是基于内存来进行高性能、高并发的读写操作的。

那既然内存是有限的，比如 Redis 就只能用 10G，你要是往里面写了 20G 的数据，会咋办？当然会干掉 10G 的数据，然后就保留 10G 的数据了。那干掉哪些数据？保留哪些数据？当然是干掉不常用的数据，保留常用的数据了。

（2）数据明明过期了，怎么还占用着内存？

这是由 Redis 的过期策略来决定。

面试题剖析

Redis 过期策略

Redis 过期策略是：定期删除+惰性删除。

所谓定期删除，指的是 Redis 默认是每隔 100ms 就随机抽取一些设置了过期时间的 key，检查其是否过期，如果过期就删除。

假设 Redis 里放了 10w 个 key，都设置了过期时间，你每隔几百毫秒，就检查 10w 个 key，那 Redis 基本上就死了，cpu 负载会很高的，消耗在你的检查过期 key 上了。注意，这里可不是每隔 100ms 就遍历所有的设置过期时间的 key，那样就是一场性能上的灾难。实际上 Redis 是每隔 100ms 随机抽取一些 key 来检查和删除的。

但是问题是，定期删除可能会导致很多过期 key 到了时间并没有被删除掉，那咋整呢？所以就是惰性删除了。这就是说，在你获取某个 key 的时候，Redis 会检查一下 ，这个 key 如果设置了过期时间那么是否过期了？如果过期了此时就会删除，不会给你返回任何东西。

获取 key 的时候，如果此时 key 已经过期，就删除，不会返回任何东西。

答案是：走内存淘汰机制。

内存淘汰机制

Redis 内存淘汰机制有以下几个：

noeviction: 当内存不足以容纳新写入数据时，新写入操作会报错，这个一般没人用吧，实在是太恶心了。

allkeys-lru：当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的 key（这个是最常用的）。

allkeys-random：当内存不足以容纳新写入数据时，在键空间中，随机移除某个 key，这个一般没人用吧，为啥要随机，肯定是把最近最少使用的 key 给干掉啊。

volatile-lru：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，移除最近最少使用的 key（这个一般不太合适）。

volatile-random：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，随机移除某个 key。

volatile-ttl：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的 key 优先移除。

## 299.【简答题】【Redis】如何保证 Redis 的高并发和高可用？Redis 的主从复制原理能介绍一下么？Redis 的哨兵原理能介绍一下么？

面试官心理分析

其实问这个问题，主要是考考你，Redis 单机能承载多高并发？如果单机扛不住如何扩容扛更多的并发？Redis 会不会挂？既然 Redis 会挂那怎么保证 Redis 是高可用的？

其实针对的都是项目中你肯定要考虑的一些问题，如果你没考虑过，那确实你对生产系统中的问题思考太少。

面试题剖析

如果你用 Redis 缓存技术的话，肯定要考虑如何用 Redis 来加多台机器，保证 Redis 是高并发的，还有就是如何让 Redis 保证自己不是挂掉以后就直接死掉了，即 Redis 高可用。

由于此节内容较多，因此，会分为两个小节进行讲解。- Redis 主从架构 - Redis 基于哨兵实现高可用 Redis 实现高并发主要依靠主从架构，一主多从，一般来说，很多项目其实就足够了，单主用来写入数据，单机几万 QPS，多从用来查询数据，多个从实例可以提供每秒 10w 的 QPS。

如果想要在实现高并发的同时，容纳大量的数据，那么就需要 Redis 集群，使用 Redis 集群之后，可以提供每秒几十万的读写并发。

Redis 高可用，如果是做主从架构部署，那么加上哨兵就可以了，就可以实现，任何一个实例宕机，可以进行主备切换。

## 300.【简答题】【SpringBoot】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。