

简历制作

今年从提前批到秋招前后投递了有**50**家左右的公司，投递的公司有（按照记忆路线，并非投递时间路线）：

西安：华为、荣耀、中兴、广联达、绿盟、大华、海康、阿里云、腾讯云、奇安信、诺瓦科技

北京：美团、字节、百度、快手、微博、爱奇艺、猿辅导、好未来、贝壳、最右、用友、高途、网易、知乎、京东

上海：拼多多、小红书、携程、b站、去哪儿

深圳：腾讯、顺丰科技、货拉拉、oppo、TCL

杭州：vivo、TPLink

合肥：zoom、蔚来汽车

其他：龙湖、虎牙、陌陌、商汤、旷世

我印象最深刻的简历挂的两家公司，一家是**TPLink**，另外一家就是虾皮（shopee）。另外就是商汤和旷世他们招开发招的不多，所以也是没给机会。其余的公司全部都给笔试机会了！

下面进入正题部分

不断更新

文件	修改日期	大小
Java开发-...	(第二... 2021/3/22 星期...	DOCX 文件 32 KB
Java开发-...	(第二... 2021/3/22 星期...	WPS PDF 文档 362 KB
Java开发-...	(第三... 2021/3/28 星期...	DOCX 文件 28 KB
Java开发-...	(第三... 2021/3/28 星期...	WPS PDF 文档 353 KB
Java开发-...	(第四... 2021/7/5 星期一 ...	DOCX 文件 47 KB
Java开发-...	第四.... 2021/5/14 星期...	WPS PDF 文档 327 KB
Java开发-...	第五... 2021/7/5 星期一 ...	DOCX 文件 47 KB
Java开发...	.docx 2021/9/11 星期...	DOCX 文件 40 KB
Java开发-...	.pdf 2021/9/11 星期...	WPS PDF 文档 329 KB
算法机器学习岗...	2021/3/5 星期五 ...	DOCX 文件 45 KB
算法机器学习岗...	2021/3/5 星期五 ...	WPS PDF 文档 350 KB
研究生成绩单 (中文) .pdf	2021/5/28 星期...	WPS PDF 文档 256 KB

这个是我的简历更新版本，总共有6个版本，但是第一个版本不知道被放哪了。这些版本里面基本都是有word和pdf两个文档形式，我投递简历都是选择pdf投递，当然有的公司需要自己填写个人信息。

word版方便自己修改，pdf方便保持格式不变，建议大家也这么去整理自己的简历。

个人信息

基本信息里要把自己最重要的手机号和邮箱填写正确，否则可能会错过笔试或面试通知，包括后期的offer沟通等！

其次就是自己的教育背景，把学历和对应的专业以及排名情况写一下。见下图！

The screenshot shows a resume template with the following sections and highlighted fields:

- 基本情况**: Includes fields for Name (姓名), Intended Position (意向岗位: Java开发工程师), Job Position (求职岗位), and ID Photo (证件照).
- 教育背景**: Includes fields for Birth Month (出生年月: 1997.01), Location (所在地: 陕西西安), Political Status (政治面貌), Contact Number (联系电话), Email (E-mail),籍贯 (place of origin), and Education Details.
 - Education Details:
 - 2019.09—2022.06: Computer Technology (Full-time Master's degree)
 - 2015.09—2019.06: Electronic Information Engineering (Engineering Bachelor's degree)
 - Major Ranks (专业排名): listed as "专业排名" (rank) for both degrees.

个人技能

看看第二版的简历、第四版和最后一版的简历对比

第二版

个人技能

● **基本技能**：熟悉 Java , Matlab 和 Python 等编程语言，掌握常用数据结构和算法，掌握 Eclipse、Anaconda 等工具。

专业技能：熟练掌握 Java 基本语法知识、了解 JVM 相关知识、掌握 SSM (Mybatis+Spring+SpringMVC) 框架和 MySQL 的基本知识，了解 Html、CSS、JavaScript 和 Ajax 等前端知识，了解 SpringBoot 的基本知识。掌握机器学习相关知识，掌握机器学习常用算法，掌握分类，回归，特征构造等基本方法。

第四版

个人技能

● 专业技能：

- 计算机基础：掌握常用的数据结构和算法，了解计算机网络和操作系统基本知识
- Java：熟悉 Java 语法知识、集合等基础框架，**了解多线程与 JVM 相关知识**
- 数据库：了解 MySQL 数据库、Redis 和 MongoDB
- 框架：了解 SSM 框架整合、SpringBoot 和 SpringCloud 的常用组件
- 中间件：了解 RabbitMQ

第六版

个人技能

● 专业技能：

- 计算机基础：掌握常用的数据结构和算法，了解计算机网络和操作系统基本知识
- Java：熟悉 Java 语法知识、集合等基础框架，了解多线程与 JVM 相关知识
- 数据库：了解 MySQL 数据库、Redis 和 MongoDB
- 框架：了解 SSM 框架整合、SpringBoot 和 SpringCloud 的常用组件
- 中间件：了解 RabbitMQ

从第二版到第四版变化还是很大的，但是后序变化的基本不大了，个人技能要注意以下几点。

1：可以分模块写，比如计算机基础、数据库相关、框架等

2：注意用词，掌握、熟悉、了解、精通...一定要用的恰到好处！

3：与自己应聘岗位无关的建议不要写，比如你找 Java 开发工程师，关于算法相关的不太建议写，就比如我的第二版，就写了熟悉一些机器学习相关的知识等。

4：注意细节，比如 Java 别写成 java，多线程写成多线性、MySQL 别写成 mysql... 等

项目经历

在介绍自己的项目经历的时候，首先要说明项目来源（实验室、比赛或自学项目），其次写清楚项目主要做什么或者有什么功能，然后自己负责哪些模块。最后就是用到了哪些技术，自己收获了什么。

第二版

项目和科研经历

- 2020.07—至今 基于xx的xx分析平台（实验室项目）
- **项目描述** 基于xx框架，搭建设备的健康管理数据分析平台。
- **主要职责**：（1）对历史的故障数据进行处理与分析，完成特征的选择，供预测使用，完成初步的故障预测算法构建。（2）负责项目平台的整体设计，完成数据库的构建工作，涉及到多表的查询，以及表之间的关系设计，主要是解决多对多的问题，页面的动态数据请求，这里使用到了Ajax技术，完成后台数据库操作代码编写，负责部分前端代码的编写。用到的相关技术：xx、Ajax、MySQL和Html。

第六版

项目和科研经历

- 2021.3-2021.5 医院预约挂号平台
- 【项目描述】基于SpringBoot+SpringCloud+MySQL+Vue构建的医院预约挂号微服务系统，后台包括医院设置管理、数据管理、用户管理、订单管理和统计管理等功能，前台实现了用户登录、就诊人管理、实名认证和预约挂号等功能。具体结合了Redis、MongoDB、RabbitMQ、MyBatisPlus和wxPay等技术。 所用技术
- 【主要职责】完成了后台代码的编写，解决了预约挂号、订单支付和取消预约的功能，从中学习了SpringCloud的常用微服务组件、Redis缓存和消息队列等中间件的应用，对分布式系统和微服务有了更深的了解。
- 2020.9—2021.4 基于xx的xx分析平台（实验室项目）
- 【项目描述】基于xx框架，搭建设备的健康管理数据分析平台。具体结合了xx、Ajax、MySQL和Html等技术。
- 【主要职责】（1）对历史的故障数据进行处理与分析，完成特征的选择，供预测使用，完成初步的故障预测算法构建。（2）负责项目平台的整体设计，完成数据库的构建工作，涉及到多表的查询，以及表之间的关系设计，主要是解决多对多的问题，页面的动态数据请求，这里使用到了Ajax技术，完成后台数据库操作代码编写，负责部分前端代码的编写。

有些同学可能在前期制作简历的时候，自己的项目还没做完或者实习还没结束，那么在后期正式秋招的时候就可以把自己的一些实习项目或者比赛项目或者是自学项目写上去。

务必要记住，写到简历上的项目，一定一定要好好的再看看，有些细节的地方面试可能会被问到，自己没做的或者自己没搞太懂的尽量不要写，否则就是给自己挖坑！

还有就是可以自己多反思反思，面试官针对这个项目可能会问哪些问题。

竞赛经历

有些同学可能找算法岗，或者是算法转开发的，在平时都会参加一些相关的比赛，如果你是找算法岗，这些比赛肯定是要写上去的，但如果你是找开发岗位的话，建议选一个成绩比较好的写上去。

第二版

竞赛经历

- 2020.12--2021.01 Riiid答案正确性预测 (铜牌：top6%)
- 竞赛描述：该比赛是在Kaggle平台进行的，本次比赛中，挑战是创建“知识跟踪”算法，即随时间推移对学生知识进行建模。目标是准确预测学生在未来互动中的表现。
 - 职责描述：对已给的大数据量（1亿数据）训练集进行处理和数据筛选，完成对特征的理解。挖掘与预测结果重要性比较高的时间特征，完成特征的构造与处理，并利用机器学习算法Lightbgm和Catboost进行预测。完成模型的融合，将机器学习和深度学习模型的预测结果进行融合。
- 2020.10--2020.12 企业风险非法集资预测 ([REDACTED] , 总队伍3403)
- 项目描述：根据大量的企业信息建立预测模型并判断企业是否存在非法集资风险。
 - 职责描述：对多表进行数据分析与处理，充分理解赛题含义，选择出对预测结果有用的特征，挖掘与标签关联性比较强的特征，挖掘潜在特征以及特征直接的关系，通过交叉方法构造新特征，并验证新特征的效果，并利用Lightbgm算法训练一个预测模型。

第六版

竞赛经历

- 2020.12--2021.01 Riiid答案正确性预测 (铜牌：top6%)
- 竞赛描述：该比赛是在Kaggle平台进行的，本次比赛目标是准确预测学生在未来互动中的表现。
 - 职责描述：对已给的大数据量（1亿数据）训练集进行处理和数据筛选，完成对特征的理解。挖掘与预测结果重要性比较高的时间特征，完成特征的构造与处理，并利用机器学习算法Lightbgm和Catboost进行预测。完成模型的融合，将机器学习和深度学习模型的预测结果进行融合。

获奖荣誉

这块内容主要写自己一些学业奖学金、评优、优秀大学生或者优秀研究生，优秀毕业生、国家奖学金之类的，或者一些比赛获奖的情况等。

第二版

科研成果

- 计算机软件著作权1项 发明专利1项

奖项荣誉

- 获得 [REDACTED] 研究生一等学业奖学金 1 次，获得 [REDACTED] 研究生二等学业奖学金 1 次；
- 获得 2020CCF 大数据与计算智能大赛的“室内用户运动时序数据分类”赛题第 1 名；
- 被评为 [REDACTED] “校三好学生” 1 次，“校优秀大学生” 1 次，获得“校文体活动先进个人” 1 次；
- 被评为 [REDACTED] 校二等奖学金 5 次，获得校三等奖学金 2 次；获得 2016 年全国大学生创新创业项目省奖；

第六版

奖项荣誉

- 获得[校]研究生一等学业奖学金1次，获得[校]研究生二等学业奖学金1次；获得[校]“优秀研究生”1次。
- 获得2020CCF大数据与计算智能大赛的“室内用户运动时序数据分类”赛题第1名；
- 被评为[校]“校三好学生”1次，“校优秀大学生”1次，获得“校文体活动先进个人”1次；
- 被评为[校]二等奖学金5次，获得校三等奖学金2次；获得2016年全国大学生创新创业项目省奖；

简历总结

个人建议，简历模板尽量朴素一点，不需要太花里胡哨的，主要体现自己的一些特点和技能即可，另外就是将简历制作的更加精细，尽量浓缩到一页即可，以上这些是个人建议。

| 更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜

Q 代码界的小白

Java基础

基础知识高频面试问题

1. Java语言的三大特性是什么？

回答：Java语言的三大特性分别是封装、继承和多态。

封装是指将对象的属性私有化，提供一些可以访问属性的方法，我们通过访问这些方法得到对象的属性。

继承是指某新类继承已经存在的类，该新类拥有被继承的类的所有属性和方法，并且新类可以根据自己的情况拓展属性或方法。其中新类称为子类，原存在的类被称为父类。

1. 子类拥有父类对象所有的属性和方法（包括私有属性和私有方法），但是父类中的私有属性和方法子类是无法访问，只是拥有。
2. 子类可以拥有自己属性和方法，即子类可以对父类进行扩展。
3. 子类可以用自己的方式实现父类的方法。

注意：Java不支持多继承

多态是同一个行为具有多个不同表现形式或形态的能力。多态就是同一个接口，使用不同的实例而执行不同操作

多态是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量到底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。【摘自JavaGuide】

在Java中有两种形式可以实现多态：继承（多个子类对同一方法的重写）和接口（实现接口并覆盖接口中同一方法）。

2. 重载与重写的区别

回答：

重载是发生在同一个类中，具有相同的方法名，但是有不同的参数，参数的个数不一样、参数的位置不一样，这就叫重载，常见的就比如构造方法，有有参构造和无参构造。

重写是发生在当子类继承父类时，对父类中的一些方法根据自己的需求进行重写操作。

3. 接口和抽象类的区别是什么？

接口(interface)和抽象类(abstract class)是支持抽象类定义的两种机制。

接口是公开的，不能有私有的方法或变量，接口中的所有方法都没有方法体，通过关键字 `interface` 实现。

抽象类是可以有私有方法或私有变量的，通过把类或者类中的方法声明为 `abstract` 来表示一个类是抽象类，被声明为抽象的方法不能包含方法体。子类实现方法必须含有相同的或者更低的访问级别(`public->protected->private`)。抽象类的子类为父类中所有抽象方法的具体实现，否则也是抽象类。

相同点：

- (1) 都不能被实例化
- (2) 接口的实现类或抽象类的子类都只有实现了接口或抽象类中的方法后才能实例化。

不同点：

- (1) 接口只有定义，不能有方法的实现，但java 1.8中可以定义 `default` 方法体，而抽象类可以有定义与实现，方法可在抽象类中实现。
- (2) 实现接口的关键字为 `implements`，继承抽象类的关键字为 `extends`。一个类可以实现多个接口，但一个类只能继承一个抽象类。所以，使用接口可以间接地实现多重继承。
- (3) 接口强调特定功能的实现，而抽象类强调所属关系。
- (4) 接口方法默认修饰符是 `public`，抽象方法可以有 `public`、`protected` 和 `default` 这些修饰符（抽象方法就是为了被重写所以不能使用 `private`` 关键字修饰！）。
- (5) 接口被用于常用的功能，便于日后维护和添加删除，而抽象类更倾向于充当公共类的角色，不适用于日后重新对立面的代码修改。从设计层面来说，抽象是对类的抽象，是一种模板设计，而接口是对行为的抽象，是一种行为的规范。

4. Java中的内部类说一下

回答：内部类有四种，分别是静态内部类、局部内部类、匿名内部和成员内部类

静态内部类：常见的 `main` 函数就是静态内部类，调用静态内部类通过“外部类.静态内部类”

局部内部类：定义在方法中的类叫做局部内部类。

匿名内部类：是指继承一个父类或者实现一个接口的方式直接定义并使用的类，匿名内部类没有 `class` 关键字，因为匿名内部类直接使用 `new` 生成一个对象

5. 说一下 final 关键字的作用

回答： final 关键字可以修饰类、方法和属性。

当 final 修饰类的时候，表明这个类不能被继承。 final 类中的所有成员方法都会被隐式地指定为 final 方法。

当 final 修饰方法的时候，表明这个方法不能被重写。

当 final 修饰属性的时候，如果是基本数据类型的变量，则其数值一旦在初始化之后便不能更改；如果是引用类型的变量，则在对其初始化之后便不能再让其指向另一个对象。

6. 说一下 String, StringBuilder 和 StringBuffer 的区别

String 类中使用 final 关键字修饰字符数组来保存字符串， private final char value[], 所以 String 对象是不可变的。

StringBuilder 与 StringBuffer 都继承自 AbstractStringBuilder 类，在 AbstractStringBuilder 中也是使用字符数组保存字符串 char[] value 但是没有用 final` 关键字修饰，所以这两种对象都是可变的。 StringBuffer 对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全。 StringBuilder 并没有对方法进行加同步锁，所以是非线程安全的。

总结：

1. 操作少量的数据：适用 String
2. 单线程操作字符串缓冲区下操作大量数据：适用 StringBuilder
3. 多线程操作字符串缓冲区下操作大量数据：适用 StringBuffer

7. 说一下 Java 中的 == 与 equals 的区别

==：它的作用是判断两个对象的地址是不是相等。即，判断两个对象是不是同一个对象(基本数据类型 == 比较的是值，引用数据类型 == 比较的是内存地址)。

equals：

- 情况 1：类没有重写 equals() 方法。则通过 equals() 比较该类的两个对象时，等价于通过“==”比较这两个对象。
- 情况 2：类重写了 equals() 方法。一般，我们都重写 equals() 方法来比较两个对象的内容是否相等；若它们的内容相等，则返回 true (即，认为这两个对象相等)。

8. Java访问修饰符有哪些？都有什么区别？

回答：Java中的修饰符有public private protected

1、public： public表明该数据成员、成员函数是对所有用户开放的，所有用户都可以直接进行调用

2、private： private表示私有，私有的意思就是除了class自己之外，任何人都不可以直接使用，私有财产神圣不可侵犯嘛，即便是子女，朋友，都不可以使用。

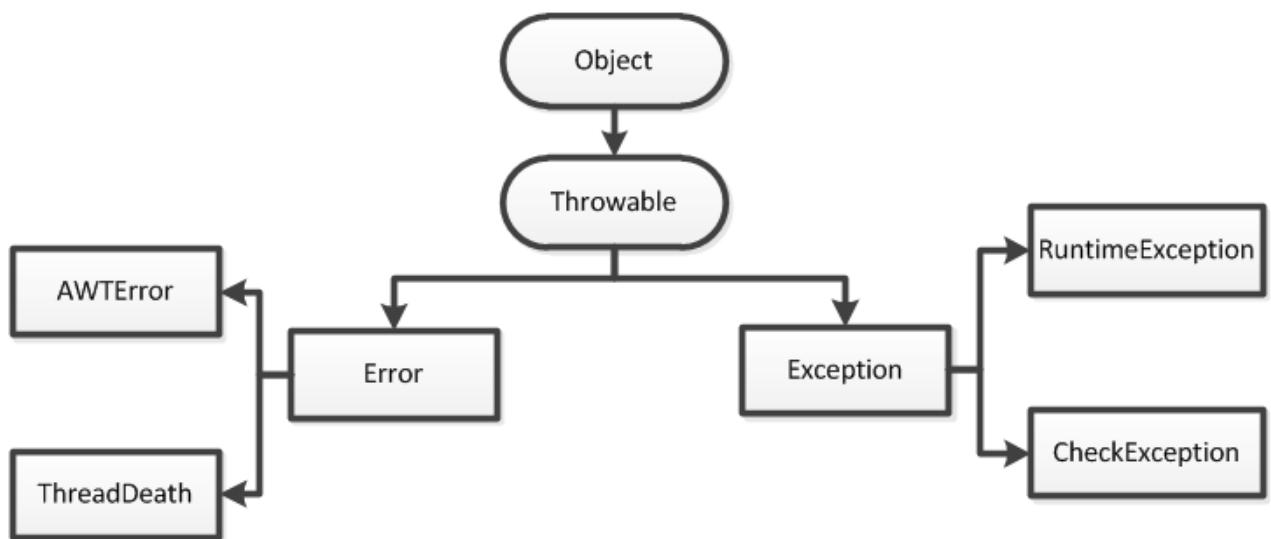
3、protected： protected对于子女、朋友来说，就是public的，可以自由使用，没有任何限制，而对于其他的外部class，protected就变成private。

追问1：怎么获取**private**修饰的变量

回答：private通过反射获取，可以设置setAccessible为true实现

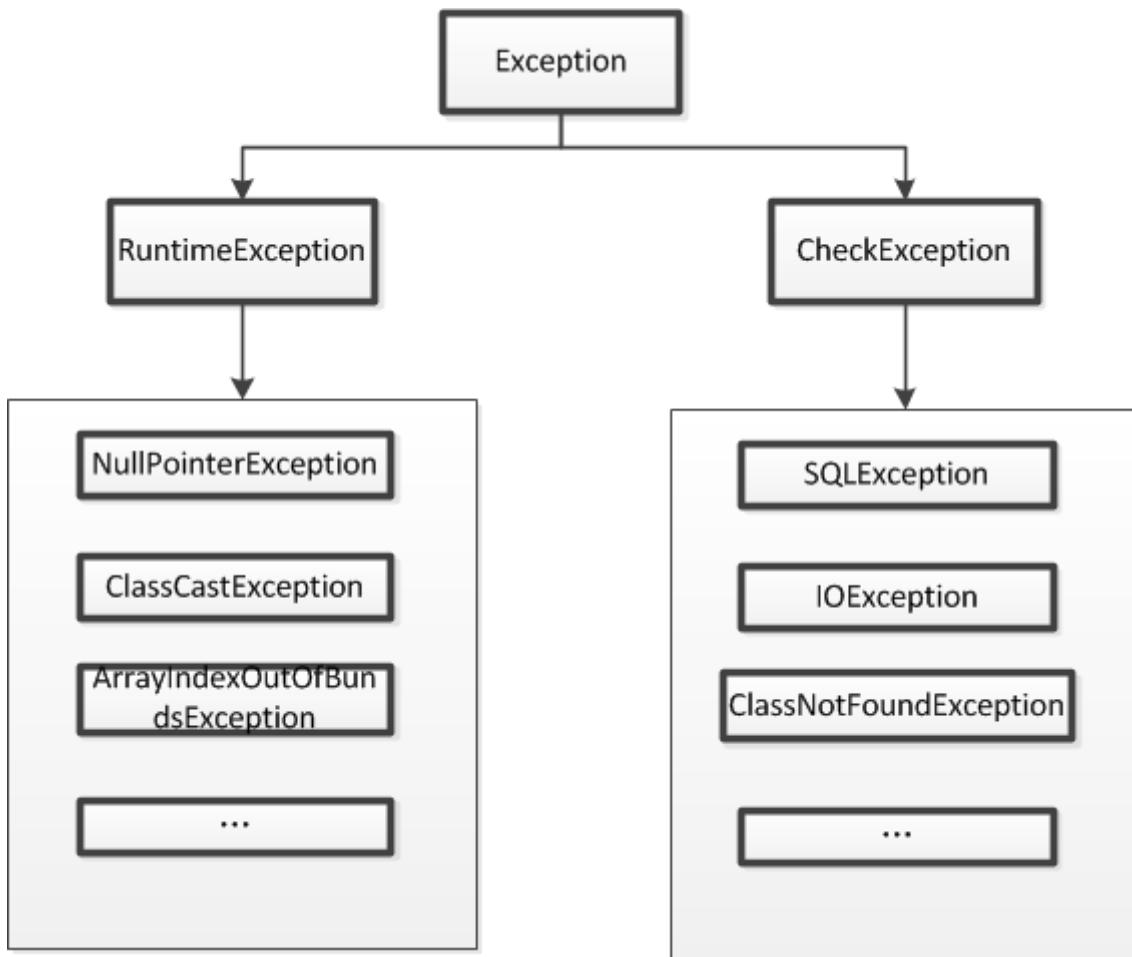
9. Java中的异常体系说一下？

回答：Java中的异常主要分为Error和Exception



Error 指Java程序运行错误，如果程序在启动时出现Error，则启动失败；如果程序运行过程中出现Error，则系统将退出程序。出现Error是系统的内部错误或资源耗尽，Error不能在程序运行过程中被动态处理，一旦出现Error，系统能做的只有记录错误的原因和安全终止。

Exception 指 Java程序运行异常，在运行中的程序发生了程序员不期望发生的事情，可以被Java异常处理机制处理。**Exception**也是程序开发中异常处理的核心，可分为 **RuntimeException**（运行时异常）和 **CheckedException**（检查异常），如下图所示



- **RuntimeException**（运行时异常）：指在Java虚拟机正常运行期间抛出的异常，**RuntimeException**可以被捕获并处理，如果出现此情况，我们需要抛出异常或者捕获并处理异常。常见的有**NullPointerException**、**ClassCastException**、**ArrayIndexOutOfBoundsException**等
- **CheckedException**（检查异常）：指在编译阶段**Java**编译器检查**CheckedException**异常，并强制程序捕获和处理此类异常，要求程序在可能出现异常的地方通过**try catch**语句块捕获异常并处理异常。常见的有由于I/O错误导致的**IOException**、**SQLException**、**ClassNotFoundException**等。该类异常通常由于打开错误的文件、**SQL**语法错误、类不存等引起。

追问1：异常的处理方式？

回答：异常处理方式有抛出异常和使用**try catch**语句块捕获异常两种方式。

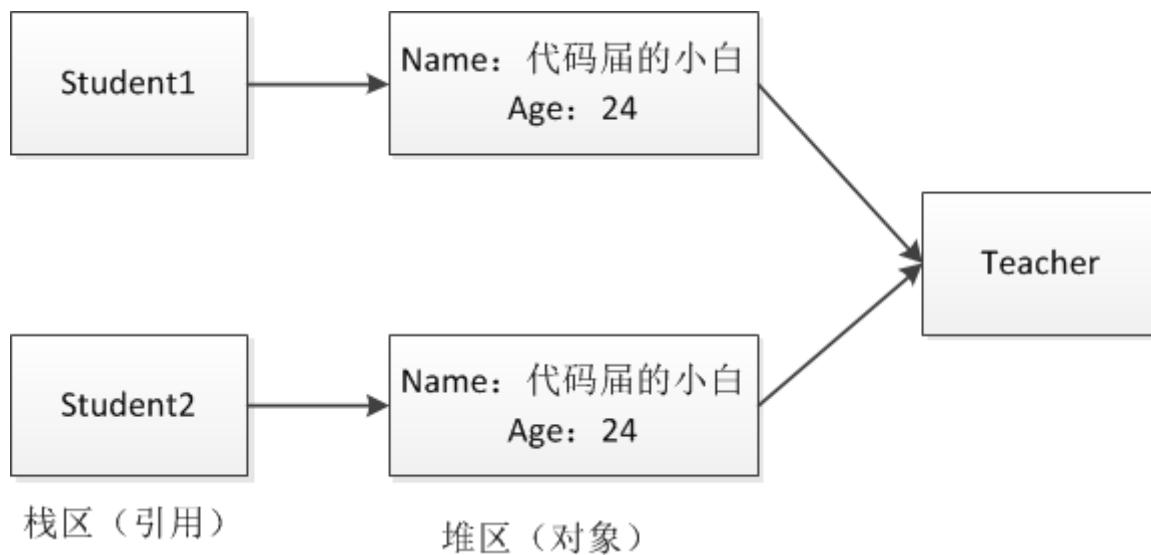
(1) 抛出异常：遇到异常时不进行具体的处理，直接将异常抛给调用者，让调用者自己根据情况处理。抛出异常的三种形式：**throws**、**throw**和系统自动抛出异常。其中**throws**作用在方法上，用于定义方法可能抛出的异常；**throw**作用在方法内，表示明确抛出一个异常。

(2) 使用**try catch**捕获并处理异常：使用费**try catch**捕获异常能够有针对性的处理每种可能出现的异常，并在捕获到异常后根据不同的情况做不同的处理。其使用过程比较简单：用**try catch**语句块将可能出现异常的代码抱起来即可。

10. Java中的深拷贝和浅拷贝说一下？

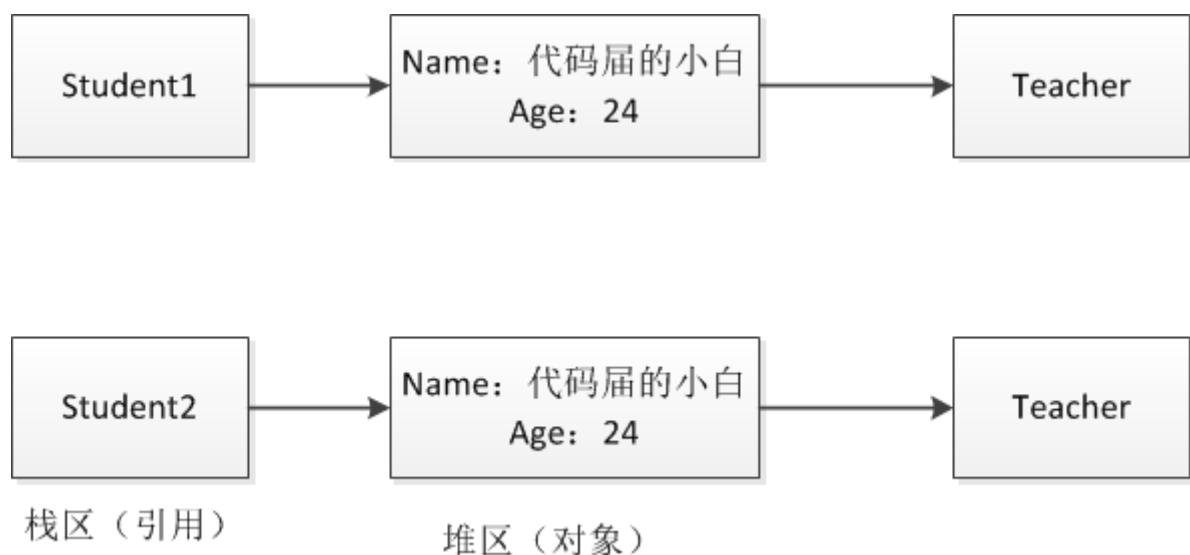
回答：深拷贝和浅拷贝都是对象拷贝

浅拷贝：按位拷贝对象，它会创建一个新对象，这个对象有着原始对象属性值的一份精确拷贝。如果属性是基本类型，拷贝的就是基本类型的值；如果属性是内存地址（引用类型），拷贝的就是内存地址，因此如果其中一个对象改变了这个地址，就会影响到另一个对象。（浅拷贝仅仅复制所考虑的对象，而不复制它所引用的对象。）



上图：两个引用 `student1` 和 `student2` 指向不同的两个对象，但是两个引用 `student1` 和 `student2` 中的两个 `teacher` 引用指向的是同一个对象，所以说明是浅拷贝。

深拷贝：在拷贝引用类型成员变量时，为引用类型的数据成员另辟了一个独立的内存空间，实现真正内容上的拷贝。（深拷贝把要复制的对象所引用的对象都复制了一遍。）



上图：两个引用 `student1` 和 `student2` 指向不同的两个对象，两个引用 `student1` 和 `student2` 中的两个 `teacher` 引用指向的是两个对象，但对 `teacher` 对象的修改只能影响 `student1` 对象，所以说是深拷贝。

摘自：*JavaGuide*

浅拷贝：对基本数据类型进行值传递，对引用数据类型进行引用传递般的拷贝，此为浅拷贝。

深拷贝：对基本数据类型进行值传递，对引用数据类型，创建一个新的对象，并复制其内容，此为深拷贝。

追问1：浅拷贝与深拷贝的特点是什么？

浅拷贝特点

- (1) 对于基本数据类型的成员对象，因为基础数据类型是值传递的，所以是直接将属性值赋值给新的对象。基础类型的拷贝，其中一个对象修改该值，不会影响另外一个。
- (2) 对于引用类型，比如数组或者类对象，因为引用类型是引用传递，所以浅拷贝只是把内存地址赋值给了成员变量，它们指向了同一内存空间。改变其中一个，会对另外一个也产生影响。

深拷贝特点

- (1) 对于基本数据类型的成员对象，因为基础数据类型是值传递的，所以是直接将属性值赋值给新的对象。基础类型的拷贝，其中一个对象修改该值，不会影响另外一个（和浅拷贝一样）。
- (2) 对于引用类型，比如数组或者类对象，深拷贝会新建一个对象空间，然后拷贝里面的内容，所以它们指向了不同的内存空间。改变其中一个，不会对另外一个也产生影响。
- (3) 对于有多层对象的，每个对象都需要实现 `Cloneable` 并重写 `clone()` 方法，进而实现了对象的串行层层拷贝。
- (4) 深拷贝相比于浅拷贝速度较慢并且耗销较大。

更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜



代码界的小白

Java集合高频面试问题

本章节主要介绍一些Java中的集合在面试中常问的高频问题，通过本章节，让大家对Java中的集合有一个更深的理解与掌握！

1. Java中的集合框架有哪些？

回答：Java集合框架主要包括两种类型的容器，一种是集合（Collection），存储一个元素集合，另一种是图（Map），存储键/值对映射。

Collection接口又有3种子类型，List、Set和Queue，再下面是一些抽象类，最后是具体实现类，常用的有ArrayList、LinkedList、HashSet、LinkedHashSet、HashMap、TreeMap、LinkedHashMap等等。



2. ArrayList和LinkedList的底层实现和区别？

回答：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。

ArrayList:增删慢、查询快，线程不安全，对元素必须连续存储。

LinkedList:增删快，查询慢，线程不安全。

追问1：说说**ArrayList**的扩容机制？

回答：通过阅读**ArrayList**的源码我们可以发现当以无参数构造方法创建 **ArrayList** 时，实际上初始化赋值的是一个空数组。当真正对数组进行添加元素操作时，才真正分配容量。即向数组中添加第一个元素时，数组容量扩为 **10**。当插入的元素个数大于当前容量时，就需要进行扩容了，**ArrayList** 每次扩容之后容量都会变为原来的 **1.5** 倍左右。

3.**HashMap**的底层实现？扩容？是否线程安全？

回答：在jdk1.7之前**HashMap**是基于数组和链表实现的，而且采用头插法。

而jdk1.8之后在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为 **8**）（将链表转换成红黑树前会判断，如果当前数组的长度小于 **64**，那么会选择先进行数组扩容，而不是转换为红黑树）时，将链表转化为红黑树，以减少搜索时间。采用尾插法。

HashMap默认的初始化大小为 **16**。当**HashMap**中的元素个数之和大于负载因子*当前容量的时候就要进行扩充，容量变为原来的 **2** 倍。（这里注意不是数组中的个数，而且数组中和链/树中的所有元素个数之和！）

注意：我们还可以在预知存储数据量的情况下，提前设置初始容量（初始容量 = 预知数据量 / 加载因子）。这样做的好处是可以减少 *resize()* 操作，提高 *HashMap* 的效率

美团面试的时候问到这个问题，还给出具体的值，让我算出初始值设置为多少合适？

HashMap是线程不安全的，其主要体现：

- 1.在jdk1.7中，在多线程环境下，扩容时会造成环形链或数据丢失。
- 2.在jdk1.8中，在多线程环境下，会发生数据覆盖的情况。

追问1：**HashMap**扩容的时候为什么是2的n次幂？

回答：数组下标的计算方法是 $(n - 1) \& hash$ ，取余($\%$)操作中如果除数是**2**的幂次则等价于与其除数减一的与(**&**)操作（也就是说 $hash \% length == hash \& (length - 1)$ 的前提是 **length** 是 **2** 的 **n** 次方；）。并且采用二进制位操作 **&**，相对于%能够提高运算效率，这就解释了 **HashMap** 的长度为什么是**2**的幂次方。

追问2：HashMap的put方法说一下。

回答：通过阅读源码，可以从jdk1.7和1.8两个方面来回答

1.根据key通过哈希算法与与运算得出数组下标

2.如果数组下标元素为空，则将key和value封装为Entry对象（JDK1.7是Entry对象，JDK1.8是Node对象）并放入该位置。

3.如果数组下标位置元素不为空，则要分情况

(i)如果是在JDK1.7，则首先会判断是否需要扩容，如果要扩容就进行扩容，如果不需要扩容就生成Entry对象，并使用头插法添加到当前链表中。

(ii)如果是在JDK1.8中，则会先判断当前位置上的TreeNode类型，看是红黑树还是链表Node

(a)如果是红黑树TreeNode，则将key和value封装为一个红黑树节点并添加到红黑树中去，在这个过程中会判断红黑树中是否存在当前key，如果存在则更新value。

(b)如果此位置上的Node对象是链表节点，则将key和value封装为一个Node并通过尾插法插入到链表的最后位置去，因为是尾插法，所以需要遍历链表，在遍历过程中会判断是否存在当前key，如果存在则更新其value，当遍历完链表后，将新的Node插入到链表中，插入到链表后，会看当前链表的节点个数，如果大于8，则会将链表转为红黑树

(c)将key和value封装为Node插入到链表或红黑树后，在判断是否需要扩容，如果需要扩容，就结束put方法。

追问3：HashMap源码中在计算hash值的时候为什么要右移16位？

回答：我的理解是让元素在HashMap中更加均匀的分布，具体的可以看下图，下图是《阿里调优手册》里说的。

我们先来了解下 `hash()` 方法中的算法。如果我们没有使用 `hash()` 方法计算 `hashCode`，而是直接使用对象的 `hashCode` 值，会出现什么问题呢？

假设要添加两个对象 `a` 和 `b`，如果数组长度是 16，这时对象 `a` 和 `b` 通过公式 $(n - 1) \& hash$ 运算，也就是 $(16-1) \& a.hashCode$ 和 $(16-1) \& b.hashCode$ ，15 的二进制为 0000000000000000000000001111，假设对象 A 的 `hashCode` 为 1000010001110001000001111000000，对象 B 的 `hashCode` 为 0111011100111000101000010100000，你会发现上述与运算结果都是 0。这样的哈希结果就太让人失望了，很明显不是一个好的哈希算法。

但如果我们将 `hashCode` 值右移 16 位（`h >>> 16` 代表无符号右移 16 位），也就是取 `int` 类型的一半，刚好可以将该二进制数对半切开，并且使用位异或运算（如果两个数对应的位置相反，则结果为 1，反之为 0），这样的话，就能避免上面的情况发生。这就是

`hash()` 方法的具体实现方式。**简而言之，就是尽量打乱 `hashCode` 真正参与运算的低 16 位。**

我再来解释下 $(n - 1) \& hash$ 是怎么设计的，这里的 `n` 代表哈希表的长度，哈希表习惯将长度设置为 2 的 `n` 次方，这样恰好可以保证 $(n - 1) \& hash$ 的计算得到的索引值总是位于 `table` 数组的索引之内。例如：`hash=15, n=16` 时，结果为 15；`hash=17, n=16` 时，结果为 1。

4. Java中线程安全的集合有哪些？

`Vector`: 就比`ArrayList`多了个同步化机制（线程安全）。

`Hashtable`: 就比`HashMap`多了个线程安全。

`ConcurrentHashMap`:是一种高效但是线程安全的集合。

`Stack`: 栈，也是线程安全的，继承于`Vector`。

追问1：说一下**ConcurrentHashMap**的底层实现，它为什么是线程安全的？

回答：在jdk1.7是分段的数组+链表，jdk1.8的时候跟HashMap1.8的时候一样都是基于数组+链表/红黑树。

ConcurrentHashMap是线程安全的

(1) 在jdk1.7的时候是使用分段所**segment**，每一把锁只锁容器其中一部分数据，多线程访问容器里不同数据段的数据，就不会存在锁竞争，提高并发访问率。

(2) 在jdk1.8的时候摒弃了 Segment的概念，而是直接用 Node 数组+链表+红黑树的数据结构来实现，并发控制使用 **synchronized** 和 **CAS** 来操作。**synchronized**只锁定当前链表或红黑二叉树的首节点。

5.HashMap和Hashtable的区别

回答：

(1) 线程是否安全：HashMap是非线程安全的，HashTable是线程安全的，因为HashTable内部的方法基本都经过synchronized修饰。（如果你要保证线程安全的话就使用ConcurrentHashMap吧！）；

(2) 对 **Null key** 和 **Null value** 的支持：HashMap可以存储null的key和value，但null作为键只能有一个，null作为值可以有多个；HashTable不允许有null键和null值，否则会抛出NullPointerException。

(3) 初始容量大小和每次扩充容量大小的不同：

① 创建时如果不指定容量初始值，Hashtable默认的初始大小为11，之后每次扩充，容量变为原来的 $2n+1$ 。HashMap默认的初始化大小为16。之后每次扩充，容量变为原来的2倍。

② 创建时如果给定了容量初始值，那么Hashtable会直接使用你给定的大小，而HashMap会将其扩充为2的幂次方大小（HashMap中的tableSizeFor()方法保证，下面给出了源代码）。也就是说HashMap总是使用2的幂作为哈希表的大小，后面会介绍到为什么是2的幂次方。

(4) 底层数据结构：JDK1.8以后的HashMap在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为8）（将链表转换成红黑树前会判断，如果当前数组的长度小于64，那么会选择先进行数组扩容，而不是转换为红黑树）时，将链表转化为红黑树，以减少搜索时间。Hashtable没有这样的机制。

(5) 效率：因为线程安全的问题，HashMap要比HashTable效率高一点。另外，HashTable基本被淘汰，不要在代码中使用它；

6.HashMap和TreeMap的区别？

回答：

1、HashMap是通过hash值进行快速查找的；HashMap中的元素是没有顺序的；TreeMap中所有的元素都是有某一固定顺序的，如果需要得到一个有序的结果，就应该使用TreeMap；

2、HashMap和TreeMap都是线程不安全的；

3、HashMap继承AbstractMap类；覆盖了hashCode() 和equals() 方法，以确保两个相等的映射返回相同的哈希值；

TreeMap继承SortedMap类；他保持键的有序顺序；

4、HashMap：基于hash表实现的；使用HashMap要求添加的键类明确定义了hashCode() 和equals()（可以重写该方法）；为了优化HashMap的空间使用，可以调优初始容量和负载因子；

TreeMap：基于红黑树实现的；TreeMap就没有调优选项，因为红黑树总是处于平衡的状态；

5、HashMap：适用于Map插入，删除，定位元素；

TreeMap：适用于按自然顺序或自定义顺序遍历键（key）

| 更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！

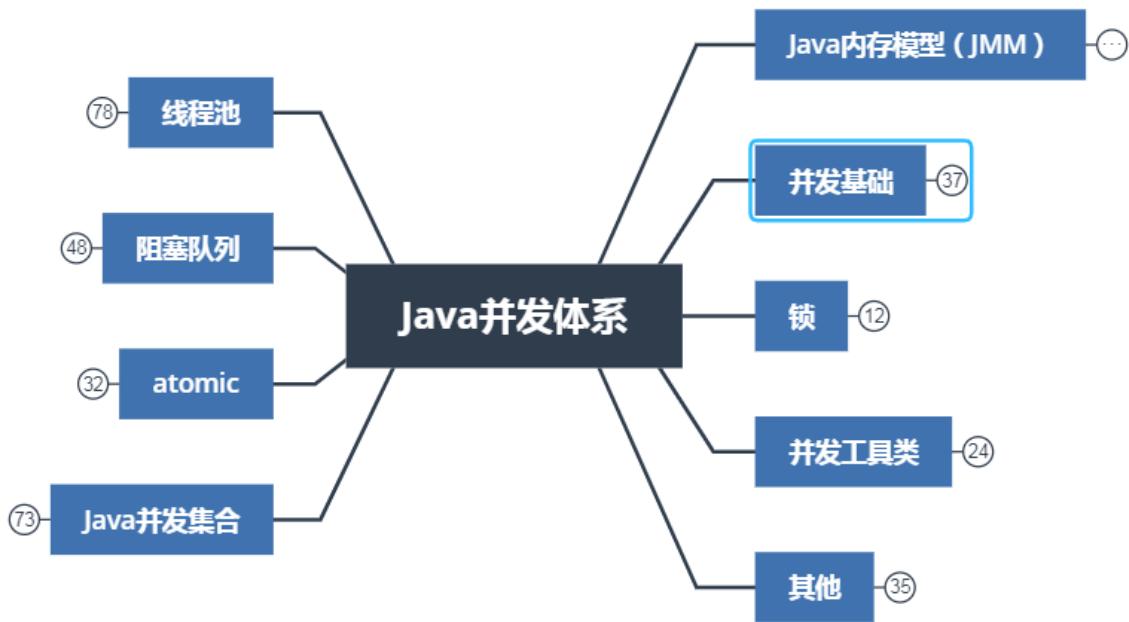


微信搜一搜



代码界的小白

多线程与并发编程高频面试问题



1. 说说什么是线程安全？如何实现线程安全？

回答：当多个线程同时访问一个对象时，如果不用考虑这些线程在运行时环境下的调度和交替执行，也不需要进行额外的同步，或者在调用方进行任何其他的协调操作，调用这个对象的行为都可以获得正确的结果，那就称这个对象是线程安全的。【摘自深入理解JVM虚拟机】

实现线程安全的方式有三种方法，分别是互斥同步、非阻塞同步和无同步方案。

互斥同步：同步是指多个线程并发访问共享数据时，保证共享数据在同一各时刻只被一条（或一些，当使用信号量的时候）线程使用。而互斥是实现同步的一种手段，临界区、互斥量和信号量都是常见的互斥实现方式。Java中实现互斥同步的手段主要有synchronized关键字或ReentrantLock等。

非阻塞同步类似是一种乐观并发的策略，比如CAS。

无同步方案，比如使用ThreadLocal。

追问1：**synchronized**和**ReentLock**的区别是什么？

相同点：

- (1) 都是可重入锁
- (2) 都保证了可见性和互斥性
- (3) 都可以用于控制多线程对共享对象的访问

不同点：

- (1) ReentrantLock等待可中断
- (2) synchronized中的锁是非公平的，ReentrantLock默认也是非公平的，但是可以通过修改参数来实现公平锁。
- (3) ReentrantLock绑定多个条件
- (4) synchronized是Java中的关键字是JVM级别的锁，而ReentrantLock是一个Lock接口下的实现类，是API层面的锁。
- (5) synchronized隐式获取锁和释放锁，ReentrantLock显示获取和释放锁，在使用时避免程序异常无法释放锁，需要在finally控制块中进行解锁操作。

追问2：**syn**和**volatile**的区别

synchronized关键字和 **volatile**关键字是两个互补的存在，而不是对立的存在！

- volatile关键字是线程同步的轻量级实现，所以volatile性能肯定比synchronized关键字要好。但是volatile关键字只能用于变量而synchronized关键字可以修饰方法以及代码块。
- volatile关键字能保证数据的可见性，但不能保证数据的原子性。synchronized关键字两者都能保证。
- volatile关键字主要用于解决变量在多个线程之间的可见性，而synchronized关键字解决的是多个线程之间访问资源的同步性。

追问3：**synchronized**锁升级的过程说一下？

回答：在jdk1.6后Java对synchronize锁进行了升级过程，主要包含偏向锁、轻量级锁和重量级锁，主要是针对对象头MarkWord的变化。

(1) 偏向锁：

为什么要引入偏向锁？

因为经过HotSpot的作者大量的研究发现，大多数时候是不存在锁竞争的，常常是一个线程多次获得同一个锁，因此如果每次都要竞争锁会增大很多没有必要付出的代价，为了降低获取锁的代价，才引入的偏向锁。

偏向锁的升级

当线程1访问代码块并获取锁对象时，会在java对象头和栈帧中记录偏向的锁的threadID，因为偏向锁不会主动释放锁，因此以后线程1再次获取锁的时候，需要比较当前线程的threadID和Java对象头中的threadID是否一致，如果一致（还是线程1获取锁对象），则无需使用CAS来加锁、解锁；如果不一致（其他线程，如线程2要竞争锁对象，而偏向锁不会主动释放因此还是存储的线程1的threadID），那么需要查看Java对象头中记录的线程1是否存活，如果没有存活，那么锁对象被重置为无锁状态，其它线程（线程2）可以竞争将其设置为偏向锁；如果存活，那么立刻查找该线程（线程1）的栈帧信息，如果还是需要继续持有这个锁对象，那么暂停当前线程1，撤销偏向锁，升级为轻量级锁，如果线程1不再使用该锁对象，那么将锁对象状态设为无锁状态，重新偏向新的线程。

(2) 轻量级锁

为什么要引入轻量级锁？

轻量级锁考虑的是竞争锁对象的线程不多，而且线程持有锁的时间也不长的情景。因为阻塞线程需要CPU从用户态转到内核态，代价较大，如果刚刚阻塞不久这个锁就被释放了，那这个代价就有点得不偿失了，因此这个时候就干脆不阻塞这个线程，让它自旋这等待锁释放。

轻量级锁什么时候升级为重量级锁？

线程1获取轻量级锁时会先把锁对象的对象头MarkWord复制一份到线程1的栈帧中创建的用于存储锁记录的空间（称为DisplacedMarkWord），然后使用CAS把对象头中的内容替换为线程1存储的锁记录（DisplacedMarkWord）的地址；

如果在线程1复制对象头的同时（在线程1CAS之前），线程2也准备获取锁，复制了对象头到线程2的锁记录空间中，但是在线程2CAS的时候，发现线程1已经把对象头换了，线程2的**CAS**失败，那么线程2就尝试使用自旋锁来等待线程1释放锁。

但是如果自旋的时间太长也不行，因为自旋是要消耗CPU的，因此自旋的次数是有限制的，比如10次或者100次，如果自旋次数到了线程1还没有释放锁，或者线程1还在执行，线程2还在自旋等待，这时又有一个线程3过来竞争这个锁对象，那么这个时候轻量级锁就会膨胀为重量级锁。重量级锁把除了拥有锁的线程都阻塞，防止CPU空转。

追问4：synchronize锁的作用范围

回答：

- (1) synchronize作用于成员变量和非静态方法时，锁住的是对象的实例，即this对象。
- (2) synchronize作用于静态方法时，锁住的是Class实例
- (3) synchronize作用于一个代码块时，锁住的是所有代码块中配置的对象。

2. Java中线程的状态有哪些？线程间的通信方式有哪些？

回答：Java中线程生命周期分为新建（New）、运行（Runnable）、阻塞（Blocked）、无限期等待（Waiting）、限期等待（Time Waiting）和结束（Terminated）这6种状态。

状态名称	说 明
NEW	初始状态，线程被构建，但是还没有调用 start() 方法
RUNNABLE	运行状态，Java 线程将操作系统中的就绪和运行两种状态笼统地称作“运行中”
BLOCKED	阻塞状态，表示线程阻塞于锁
WAITING	等待状态，表示线程进入等待状态，进入该状态表示当前线程需要等待其他线程做出一些特定动作（通知或中断）
TIME_WAITING	超时等待状态，该状态不同于 WAITING，它是可以在指定的时间自行返回的
TERMINATED	终止状态，表示当前线程已经执行完毕

Java中线程间通信方式有：

互斥量(Mutex)：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问。比如 Java 中的 synchronized 关键词和各种 Lock 都是这种机制。

信号量(Semphares)：它允许同一时刻多个线程访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量

事件(Event) :Wait/Notify：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操

追问1: sleep后进入什么状态, wait后进入什么状态?

回答: sleep后进入Time waiting超时等待状态, wait后进入等待waiting状态。

追问2: sleep和wait的区别?

回答:

(1) sleep方法属于Thread类, wait方法属于Object类

(2) sleep方法暂停执行指定的时间, 让出CPU给其他线程, 但其监控状态依然保持在指定的时间过后又会自动恢复运行状态。

(3) 在调用sleep方法的过程中, 线程不会释放对象锁, 而wait会释放对象锁。

追问3: wait为什么是数Object类下面的方法?

这个问题我被问到过两次, 第一次不会(美团), 就去百度搜了搜, 第二次遇到就会了(贝壳), 下面是网上搜到的。

所谓的释放锁资源实际是通知对象内置的monitor对象进行释放, 而只有所有对象都有内置的monitor对象才能实现任何对象的锁资源都可以释放。又因为所有类都继承自Object, 所以wait()就成了Object方法, 也就是通过wait()来通知对象内置的monitor对象释放, 而且事实上因为这涉及对硬件底层的操作, 所以wait()方法是native方法, 底层是用C写的。【来自网络】

追问4:start方法和run方法有什么区别?

(1) start方法用于启动线程, 真正实现了多线程运行。在调用了线程的start方法后, 线程会在后台执行, 无须等待run方法体的代码执行完毕。

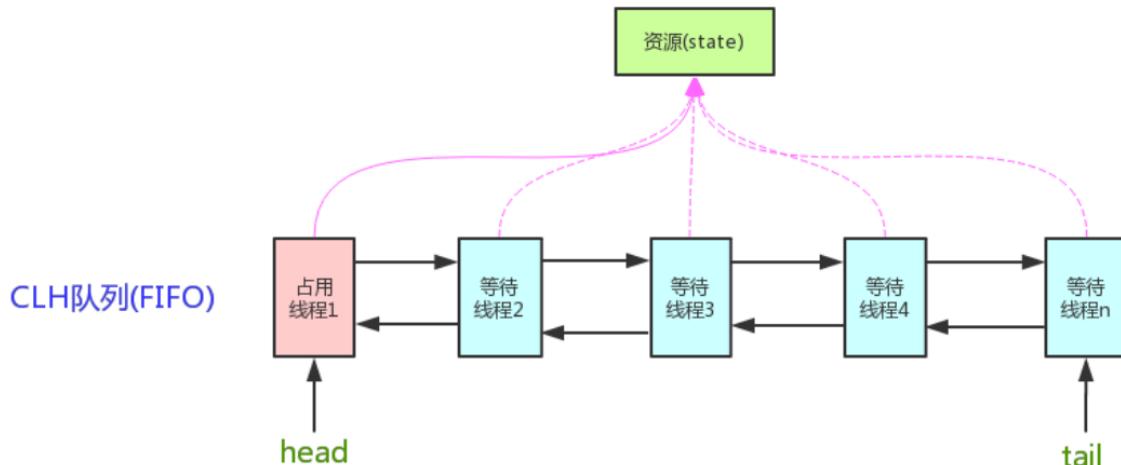
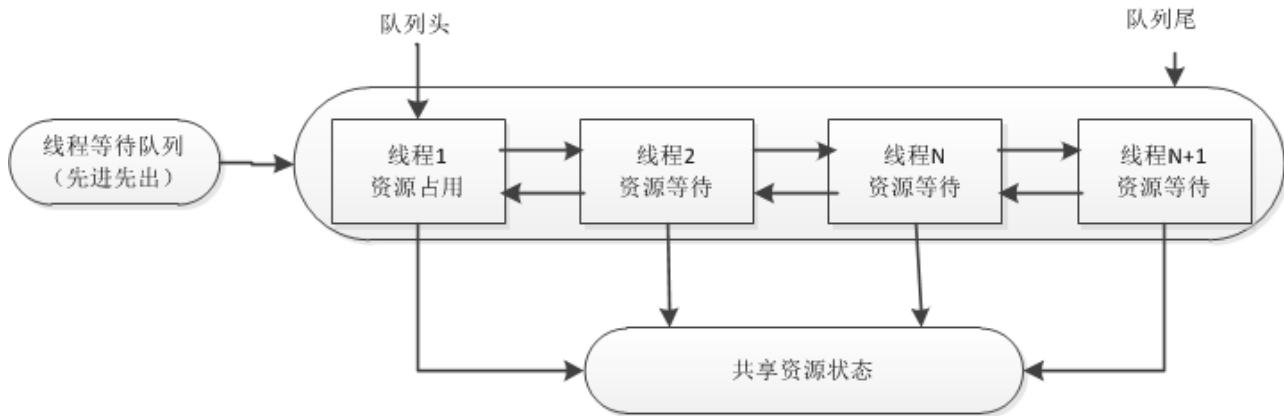
(2) 通过调用start方法启动一个线程时, 此线程处于就绪状态, 并没有运行。

(3) run方法也叫做线程体, 包含了要执行的线程的逻辑代码, 在调用run方法后, 线程就进入运行状态, 开始运行run方法中的代码, 在run方法运行结束后, 该线程终止, CPU在调度其他的线程。

3.AQS了解吗？

回答：AQS是一个抽象队列同步器，通过维护一个状态标志位state和一个先进先出的（FIFO）的线程等待队列来实现一个多线程访问共享资源的同步框架。

AQS的原理大概是这样的，给每个共享资源都设置一个共享锁，线程在需要访问共享资源时首先需要获取共享资源锁，如果获取到了共享资源锁，便可以在当前线程中使用该共享资源，如果没有获取到共享锁，该线程被放入到等待队列中，等待下一次资源调度。



AQS定义了两种资源共享方式：独占式和共享式

独占式：只有一个线程能执行，具体的Java实现有ReentrantLock。

共享式：多个线程可同时执行，具体的Java实现有Semaphore和CountDownLatch。

AQS只是一个框架（模板模式），只定义了一个接口，具体资源的获取、释放都交由自定义同步器去实现。不同的自定义同步器争取用共享资源的方式也不同，自定义同步器在实现时只需实现共享资源state的获取与释放方式即可，至于具体线程等待队列的维护，如获取资源失败入队、唤醒出队等，AQS已经在顶层实现好，不需要具体的同步器在做处理。

追问1：Java中的并发关键字

Java中常见的并发关键字有CountDownLatch、CyclicBarrier、Semaphore和volatile。

追问2：你使用过哪个AQS组件，有将其用于多线程编程吗？（给一个例题说一下思路或者直接写）

回答：这个可以自己去网上找一些，或者自己总结一些。我一般举的例子是我在笔试中遇到的一个例子。就有四个子线程分别统计四个盘的容量，然后最终通过一个主线程将四个盘的进行求和，输出总的容量。

用到了CountDownLatch，他是基于线程计数器来实现并发控制，主要用于主线程等待其他子线程都执行完毕后执行相关操作。

下面是代码：

```
class DiskMemory {  
  
    private int totalsize;  
  
    public int getSize() {  
        return (new Random().nextInt(3) + 1) * 100; // 加一为了防止获取磁盘大小为0，不符合常理  
    }  
  
    public void setSize(int size) {  
        totalsize += size;  
    }  
  
    public int getTotalsize() {  
        return totalsize;  
    }  
}  
  
public class t3 {  
  
    public static void main(String[] args) {  
  
        ExecutorService executorService =  
        Executors.newFixedThreadPool(4);  
        DiskMemory diskMemory = new DiskMemory();  
        // 设置四个子线程 main函数为主线程  
        CountDownLatch countDownLatch = new CountDownLatch(4);  
    }  
}
```

```
for (int i = 0; i < 4; i++) {  
  
    executorService.execute(() -> {  
        try {  
            int size = diskMemory.getSize();  
            diskMemory.setSize(size);  
            //Thread.sleep(1000);  
            System.out.println("线程执行，磁盘大小：" + size);  
        } catch (Exception e) { //InterruptedException e  
            e.printStackTrace();  
        }  
        countDownLatch.countDown(); //计数器减一  
        System.out.println("-----");  
    });  
}  
try {  
  
    countDownLatch.await(); //唤醒主线程  
} catch (InterruptedException e) {  
    e.printStackTrace();  
}  
System.out.println("磁盘总大小：" +  
diskMemory.getTotalSize());  
//线程池使用完需要手动关闭  
executorService.shutdown();  
  
}  
}
```

4.CAS说一下

CAS指Compare and swap比较和替换是设计并发算法时用到的一种技术，CAS指令有三个操作数，分别是内存位置（在Java中可以简单的理解为变量的内存地址，用V表示），旧的预期值（用A表示）和准备设置的新值（用B表示）。CAS指令在执行的时候，当且仅当V符合A时，处理器才会用B更新V的值，否则它就不会执行更新。

追问1：CAS带来的问题是什么？如何解决的？

回答：ABA问题、循环时间长开销很大、只能保证一个共享变量的原子操作

一般加版本号进行解决

追问2：什么是乐观锁，什么是悲观锁？

回答：悲观锁和乐观锁并不是某个具体的“锁”而是一种并发编程的基本概念。乐观锁和悲观锁最早出现在数据库的设计当中，后来逐渐被 Java 的并发包所引入。

悲观锁：认为对于同一个数据的并发操作，一定是会发生修改的，哪怕没有修改，也会认为修改。因此对于同一个数据的并发操作，悲观锁采取加锁的形式。悲观地认为，不加锁的并发操作一定会出问题。

乐观锁：正好和悲观锁相反，它获取数据的时候，并不担心数据被修改，每次获取数据的时候也不会加锁，只是在更新数据的时候，通过判断现有的数据是否和原数据一致来判断数据是否被其他线程操作，如果没被其他线程修改则进行数据更新，如果被其他线程修改则不进行数据更新。

5.Java中创建线程的方式有哪些？

回答：Java中创建线程的方式有4种，分别是

- (1) 写一个类继承子Thread类，重写run方法
- (2) 写一个类重写Runnable接口，重写run方法
- (3) 写一个类重写Callable接口，重写call方法
- (4) 使用线程池

追问1：线程池的好处？说几个Java中常见的线程池？说一下其中的参数和运行流程？

回答：使用线程池可以降低资源消耗（反复创建线程是一件很消耗资源的事，利用已创建的线程降低线程创建和销毁造成的消耗）、提供处理速度（当任务到达时，可以直接使用已有线程，不必等到线程创建完成才去执行。）、线程资源可管理性和通过控制系统的最大并发数，以保证系统高效且安全的运行。

Executors 实现了以下四种类型的 ThreadPoolExecutor：

类型	特性
newCachedThreadPool	线程池的大小不固定，可灵活回收空闲线程，若无可回收，则新建线程
newFixedThreadPool	固定大小的线程池，当有新的任务提交，线程池中如果有空闲线程，则立即执行，否则新的任务会被缓存在一个任务队列中，等待线程池释放空闲线程
newScheduledThreadPool	定时线程池，支持定时及周期性任务执行
newSingleThreadExecutor	只创建一个线程，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序 (FIFO-LIFO-优先级) 执行

线程池有7大核心参数，分别是

corePoolSize: 核心线程数

maximumPoolSize: 线程池中最大线程数

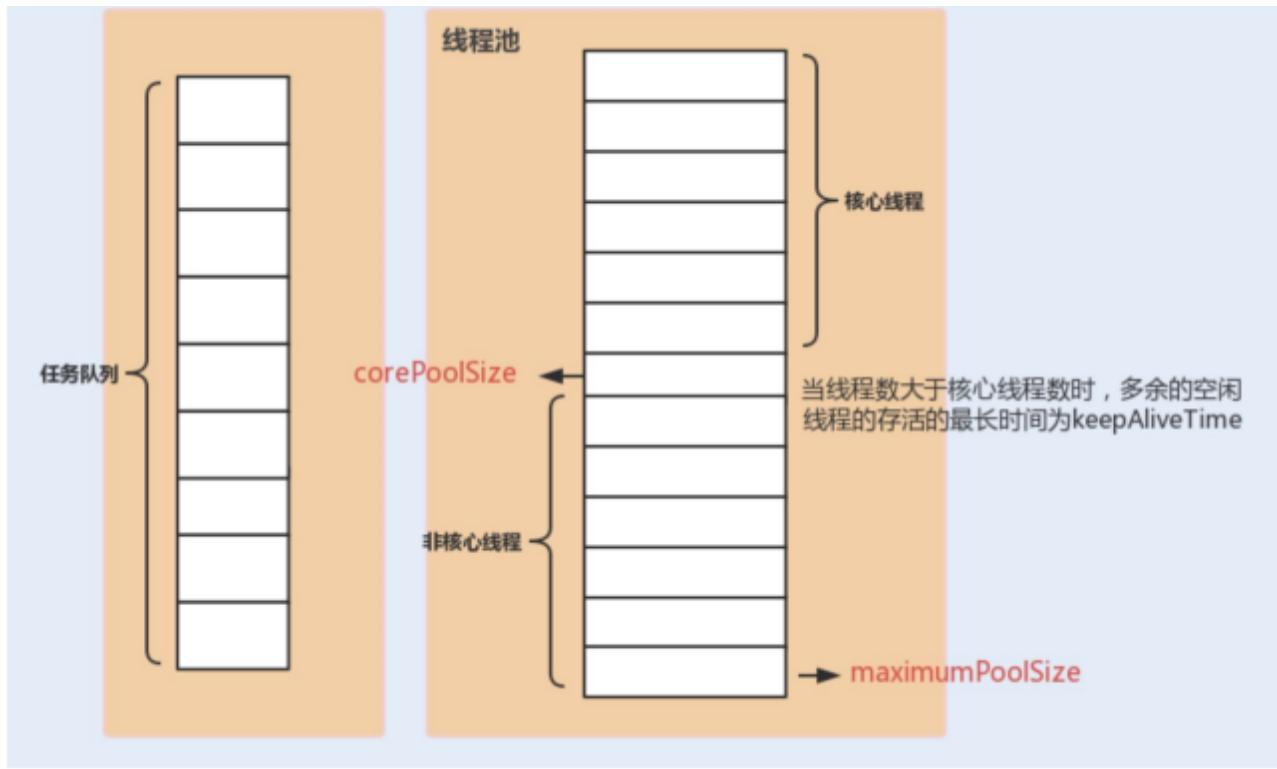
keepAliveTime: 多余空闲线程数的存活时间，当前线程数大于corePoolSize，并且等待时间大于keepAliveTime，多余线程或被销毁直到剩下corePoolSize为止。

TimeUnit unit: keepAliveTime的单位。

workQueue: 阻塞队列，被提交但未必执行的任务。

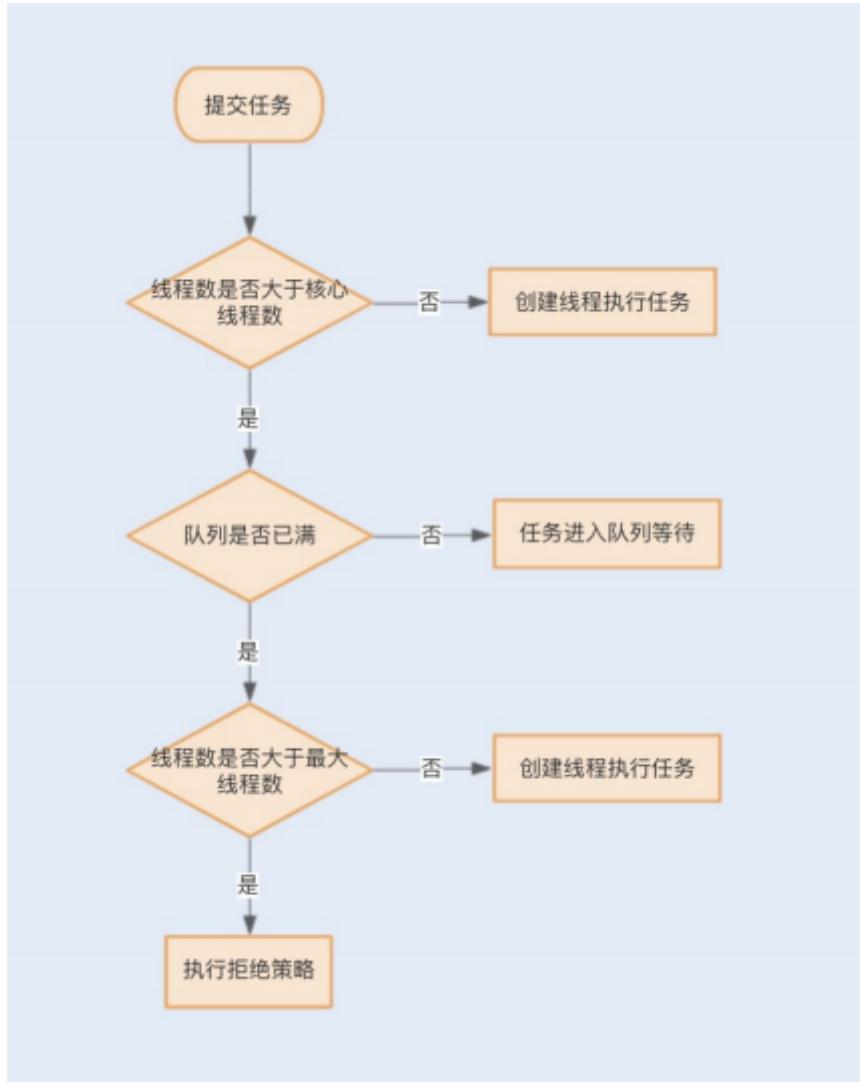
threadFactory: 用于创建线程池中工作线程的线程工厂，一般用默认的。

handler: 拒绝策略，当堵塞队列满了并且工作线程大于线程池的最大线程数(maximumPoolSize)。



线程池中的执行流程：

- (1) 当线程数小于核心线程数的时候，使用核心线程数。
- (2) 如果核心线程数小于线程数，就将多余的线程放入任务队列（阻塞队列）中
- (3) 当任务队列（阻塞队列）满的时候，就启动最大线程数.
- (4) 当最大线程数也达到后，就将启动拒绝策略。



追问2：拒绝策略有哪些？

回答：有四种拒绝策略

1.ThreadPoolExecutor.AbortPolicy

线程池的默认拒绝策略为AbortPolicy，即丢弃任务并抛出RejectedExecutionException异常（即后面提交的请求不会放入队列也不会直接消费并抛出异常）；

2.ThreadPoolExecutor.DiscardPolicy

丢弃任务，但是不抛出异常。如果线程队列已满，则后续提交的任务都会被丢弃，且是静默丢弃（也不会抛出任何异常，任务直接就丢弃了）。

3.ThreadPoolExecutor.DiscardOldestPolicy

丢弃队列最前面的任务，然后重新提交被拒绝的任务（丢弃掉了队列最前的任务，并不抛出异常，直接丢弃了）。

4.ThreadPoolExecutor.CallerRunsPolicy

由调用线程处理该任务（不会丢弃任务，最后所有的任务都执行了，并不会抛出异常）

追问3：线程池的参数如何确定呢？

回答：

一般需要确定核心线程数、最大线程数、任务队列和拒绝策略，这些需要根据实际的业务场景去设置，可以大致分为CPU密集型和IO密集型。

CPU密集型时，任务可以少配置线程数，大概和机器的cpu核数相当，这样可以使得每个线程都在执行任务。

IO密集型时，大部分线程都阻塞，故需要多配置线程数， $2 * \text{cpu核数}$ 。

详细可以看一下《阿里调优手册》，需要的请在公众号回复：阿里调优手册

追问4：Java中常见的阻塞队列有哪些？

ArrayBlockingQueue：是一个我们常用的典型的有界队列，其内部的实现是基于数组来实现的。

LinkedBlockingQueue 从它的名字我们可以知道，它是一个由链表实现的队列，这个队列的长度 `Integer.MAX_VALUE`，这个值是非常大的，几乎无法达到，对此我们可以认为这个队列基本属于一个无界队列（也认为是有界队列）。此队列按照先进先出的顺序进行排序。

SynchronousQueue 是一个不存储任何元素的阻塞队列，每一个 `put` 操作必须等待 `take` 操作，否则不能添加元素。同时它也支持公平锁和非公平锁。

PriorityBlockingQueue 是一个支持优先级排序的无界阻塞队列，可以通过自定义实现 `compareTo()` 方法来指定元素的排序规则，或者通过构造器参数 `Comparator` 来指定排序规则。但是需要注意插入队列的对象必须是可比较大小的，也就是 `Comparable` 的，否则会抛出 `ClassCastException` 异常。

DelayQueue 是一个实现 `PriorityBlockingQueue` 的延迟获取的无界队列。具有“延迟”的功能。

6.ThreaLocal知道吗？

回答：Java中每一个线程都有自己的专属本地变量，JDK中提供的`ThreadLocal`类，`ThreadLocal`类主要解决的就是让每个线程绑定自己的值，可以将`ThreadLocal`类形象的比喻成存放数据的盒子，盒子中可以存储每个线程的私有数据。

1.`ThreadLocal`是Java中所提供的线程本地存储机制，可以利用该机制将数据存在某个线程内部，该线程可以在任意时刻、任意方法中获取缓存的数据

2.`ThreadLocal`底层是通过`ThreadLocalmap`来实现的，每个`Thread`对象（注意不是`ThreadLocal`对象）中都存在一个`ThreadLocalMap`，`Map`的key为`ThreadLocal`对象，`Map`的value为需要缓存的值。

3.`ThreadLocal`经典的应用场景就是连接管理（一个线程持有一个链接，该连接对象可以在不同给的方法之间进行线程传递，线程之间不共享同一个连接）

追问1：用它可能会带来什么问题？

回答：如果在线程池中使用`ThreadLocal`会造成内存泄漏，因为当`ThreadLocal`对象使用完之后，应该要把设置的key，value，也就是Entry对象进行回收，但线程池中的线程不会回收，而线程对象是通过强引用指向`ThreadLocalmap`，`ThreadLocalmap`也是通过强引用指向Entry对象，线程不被回收，Entry对象也就不会被回收，从而出现内存泄漏，解决办法是：在使用了`ThreadLocal`对象之后，手动调用`ThreadLocal`的`remove`方法，手动清除Entry对象。

`ThreadLocalMap`中使用的key为`ThreadLocal`的弱引用，而value是强引用。所以，如果`ThreadLocal`没有被外部强引用的情况下，在垃圾回收的时候，key会被清理掉，而value不会被清理掉。这样一来，`ThreadLocalMap`中就会出现key为null的Entry。假如我们不做任何措施的话，value永远无法被GC回收，这个时候就可能会产生内存泄露。【摘自JavaGuide】

追问2：什么是强软弱虚引用？

回答：

(1) 强引用是使用最普遍的引用。只要某个对象有强引用与之关联，JVM必定不会回收这个对象，即使在内存不足的情况下，JVM宁愿抛出`OutOfMemory`错误也不会回收这种对象

(2) 软引用是用来描述一些有用但并不是必需的对象，在Java中用`java.lang.ref.SoftReference`类来表示。只有在内存不足的时候JVM才会回收该对象。

(3) 只具有弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。弱引用可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象被垃圾回收，Java虚拟机就会把这个弱引用加入到与之关联的引用队列中。

(4) 虚引用也称为幻影引用，一个对象是都有虚引用的存在都不会对生存时间都构成影响，也无法通过虚引用获取对一个对象的真实引用。唯一的用处：能在对象被GC时收到系统通知，JAVA中用PhantomReference来实现虚引用

虚引用必须和引用队列（ReferenceQueue）联合使用。当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之关联的引用队列中。

更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜

Q 代码界的小白

JVM高频面试问题

关于Java虚拟机，推荐大家阅读《深入理解Java虚拟机》这本书，主要阅读第二章、第三章、第六章、第七章、第八章、第十二和第十三章。如果有时间的同学也可以把第四章和第五章的调优部分看看！

1.介绍一下Java运行时数据区域，并说一下每个部分都存哪些内容？

回答：Java的运行时区主要包含堆、方法区、虚拟机栈、程序计数器和本地方法栈，其中堆和方法区是所有线程所共有的。而且虚拟机栈、程序计数器和本地方法栈是线程所私有的。

堆：存放对象实例

方法区：用来存储已经被虚拟机加载的类型信息、常量、静态变量、即时编译器编译后的代码缓存等数据。

虚拟机栈：（生命周期与线程相同）Java中每个方法执行的时候，Java虚拟机都会同步创建一个栈帧，用于存储局部变量表、操作数栈、动态链接、方法出口等信息。

程序计数器：保存下一条需要执行的字节码指令，是程序控制流的指示器，分支、循环、跳转、异常处理、线程恢复等基础功能都是依赖程序计数器。

本地方法栈：与虚拟机栈类似

追问1：程序计数器可以为空吗？

回答：可以为空，当执行的是本地方法时。

追问2：堆中又怎么细分的？

回答：堆中可以细分为新生代和老年代，其中新生代又分为Eden区，From Survivor和To Survivor区，比例是8:1:1。

追问3：哪些区域会造成OOM

回答：除了程序计数器不会产生OOM，其余的均可以产生OOM。

2. Java中对象的创建过程是什么样的？

回答：Java中对象的创建过程为5步

(1) 当遇到new关键字的时候，首先坚持这个指令的参数是否可以在常量池中定位到一个类的符号引用，并检查这个符号引用代表的类是否已被加载、解析和初始化。

(2) 在类加载检查后，接下来需要为新对象分配内存。

(3) 需要将分配到的内存空间都初始化为零。

(4) 需要对对象进行相关的设置，比如这个对象是哪个类的实例、如何才能找到类的元数据信息、对象的GC分代年龄等信息。

(5) 执行()方法。

追问1：内存分配的策略有哪些？

回答：Java中的内存分配策略主要有两种，分别是指针碰撞和空闲列表。

指针碰撞：假设Java堆中的内存都是规整的，所有被使用过的放在一边，未使用过的放在一边，中间有一个指针作为分界，分配内存仅仅需要把这个指针向空闲空间方向移动一段即可。

空闲列表：如果Java堆中的内存不是规整的，已使用过的和空闲的交错，虚拟机就需要维护一个列表，记录哪些内存是可用的，在分配的时候找到足够大的一块内存进行分配。

追问2：对象头包含哪些？

回答：虚拟机中对象头包含两类信息，第一类是用于存储对象自身运动时数据、如哈希码、GC分代年龄、线程持有的锁、偏向线程ID、偏向时间戳。对象的另外一部分是类型指针，即对象指向它的类型元数据的指针。

追问3：对象的访问定位方法有几种，各有什么优缺点？

回答：Java虚拟机中对象的访问方式有①使用句柄和②直接指针两种。

1. 句柄：如果使用句柄的话，那么Java堆中将会划分出一块内存来作为句柄池，reference中存储的就是对象的句柄地址，而句柄中包含了对象实例数据与类型数据各自的具体地址信息；

2. 直接指针：如果使用直接指针访问，那么 Java 堆对象的布局中就必须考虑如何放置访问类型数据的相关信息，而reference 中存储的直接就是对象的地址。

总结：使用句柄最大的好处就是reference中存储的是稳定句柄地址，在对象移动时只会改变句柄中的实例数据指针，而reference本身不需要被修改。使用直接指针访问方式最大的好处就是速度快，它节省了一次指针定位的时间开销。

3. 如何判断对象已死？

回答：Java中判断对象死亡的方法有引用计数法和可达性分析。

引用计数法：对象中添加一个引用计数器，每当有一个地方引用它，计数器就加1；当引用失效，计数器就减1；任何时候计数器为0的对象就是不可能再被使用的。

可达性分析：通过一系列的GC Roots的根对象作为起始节点，从这些节点开始，根据引用关系向下搜索，如果某个对象到GC Roots间没有任何引用链相连。

追问1：GCroot可以是哪些？

回答：在Java中可以作为GC Roots的比较多，分别有

(1)在虚拟机栈中引用的对象，比如各个线程被调用的方法堆栈中使用到的参数、局部变量、临时变量等。

(2)在方法区中类静态属性引用的对象，比如Java类的引用类型静态变量。

(3)在方法区中常量引用的对象，比如字符串常量池里的引用。

(4)在本地方法栈中JNI引用的对象。

(5)Java虚拟机内部的引用，如基本数据类型对应的Class对象，一些常驻的异常对象。

(6)所有被同步锁持有的对象。

追问2：被标志为GC的对象一定会被GC掉吗？

回答：不一定，还有逃脱的可能。真正宣告一个对象死亡至少经历两次标记的过程。

如果对象进行可达性分析后没有与GC Roots相连，那么这是第一次标记，之后会在进行一次筛选，筛选的条件是是否有必要执行finalize()方法。【详细可以看课本《深入理解Java虚拟机》】

4. 垃圾回收算法有哪些？详细叙述一下。

回答：垃圾回收算法主要有三种，分别标记清除、标记整理和标记复制。

标记清除：算法分为“标记”和“清除”两个阶段：首先标记出所有需要回收的对象，在标记完成后统一回收所有被标记的对象。它的主要不足空间问题，标记清除之后会产生大量不连续的内存碎片，空间碎片太多可能会导致以后在程序运行过程中需要分配较大对象时，无法找到足够的连续内存而不得不提前触发另一次垃圾收集动作。

标记复制：将可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当这一块的内存用完了，就将还存活着的对象复制到另外一块上面，然后再把已使用过的内存空间一次清理掉。这样使得每次都是对整个半区进行内存回收，内存分配时也就不用考虑内存碎片等复杂情况，只要按顺序分配内存即可，实现简单，运行高效。只是这种算法的代价是将内存缩小为了原来的一半。

标记整理：首先标记出所有需要回收的对象，在标记完成后，后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存。

追问1：新生代和老年代一般使用什么算法？

回答：新生代一般使用标记复制和标记整理算法，老年代一般使用标记清除算法。

追问2：为什么新生代不使用标记清除算法？

在新生代中，每次垃圾收集时都发现有大批对象死去，只有少量存活，那就选用复制算法，只需要付出少量存活对象的复制成本就可以完成收集。而老年代中因为对象存活率高、没有额外空间对它进行分配担保，就必须使用“标记—清理”或者“标记—整理”算法来进行回收。

5. 垃圾回收器有哪些？

回答：垃圾回收器可以在新生代和老年代都有，在新生代有Serial、ParNew、Parallel Scavenge；老年代有CMS、Serial Old、Parallel Old；还有不区分年的G1算法。

追问1：CMS垃圾回收器的过程是什么样的？会带来什么问题？

回答：CMS回收过程可以分为4个步骤。

(1) 初试标记：初试标记仅仅只是标记一下GC Roots能直接关联到的对象，速度很快，但需要暂停所有其他的工作线程。

(2) 并发标记：GC 和用户线程一起工作，执行GC Roots跟踪标记过程，不需要暂停工作线程。

(3) 重新标记：在并发标记过程中用户线程继续运作，导致在垃圾回收过程中部分对象的状态发生了变化，未来确保这部分对象的状态的正确性，需要对其重新标记并暂停工作线程。

(4) 并发清除：清理删除掉标记阶段判断的已经死亡的对象，这个过程用户线程和垃圾回收线程同时发生。

带来的问题：

(1) CMS收集器对处理器资源非常敏感。

(2) CMS无法处理“浮动垃圾”。

(3) CMS是基于标记-清除算法，会产生大量的空间碎片。

追问2：G1垃圾回收器的改进是什么？相比于CMS**突出的地方是什么？**

回答：G1垃圾回收器抛弃了分代的概念，将堆内存划分为大小固定的几个独立区域，并维护一个优先级列表，在垃圾回收过程中根据系统允许的最长垃圾回收时间，优先回收垃圾最多的区域。（G1算法是可控STW的一种算法，GC收集器和我们GC调优的目标就是尽可能的减少STW的时间和次数。）

G1突出的地方：

基于标记整理算法，不产生垃圾碎片。

可以精确的控制停顿时间，在不牺牲吞吐量的前提下实现短停顿垃圾回收。

追问3：现在jdk默认使用的是哪种垃圾回收器？

回答：(被问到过好几次)

jdk1.7 默认垃圾收集器Parallel Scavenge（新生代）+Parallel Old（老年代）

jdk1.8 默认垃圾收集器Parallel Scavenge（新生代）+Parallel Old（老年代）

jdk1.9 默认垃圾收集器G1

6.内存分配策略是什么样的？

对象优先在Eden分配，如果说Eden内存空间不足，就会发生Minor GC/Young GC

大对象直接进入老年代，大对象：需要大量连续内存空间的Java对象，比如很长的字符串和大型数组，1、导致内存有空间，还是需要提前进行垃圾回收获取连续空间来放他们，2、会进行大量的内存复制。

-XX:PretenureSizeThreshold 参数，大于这个数量直接在老年代分配，缺省为0，表示绝不会直接分配在老年代。

长期存活的对象将进入老年代，默认15岁，-XX:MaxTenuringThreshold调整

动态对象年龄判定，为了能更好地适应不同程序的内存状况，虚拟机并不是永远地要求对象的年龄必须达到了MaxTenuringThreshold才能晋升老年代，如果在Survivor空间中相同年龄所有对象大小的总和大于Survivor空间的一半，年龄大于或等于该年龄的对象就可以直接进入老年代，无须等到MaxTenuringThreshold中要求的年龄

空间分配担保：新生代中有大量的对象存活，survivor空间不够，当出现大量对象在MinorGC后仍然存活的情况（最极端的情况就是内存回收后新生代中所有对象都存活），就需要老年代进行分配担保，把Survivor无法容纳的对象直接进入老年代。只要老年代的连续空间大于新生代对象的总大小或者历次晋升的平均大小，就进行Minor GC，否则FullGC。

追问1：内存溢出与内存泄漏的区别？

内存溢出：实实在在的内存空间不足导致；

内存泄漏：该释放的对象没有释放，多见于自己使用容器保存元素的情况下。

7.jvm调优了解过吗？常用的命令和工具有哪些？

回答：Linux中有top、vmstat、pidstat，jdk中的jstat、jstack、jps、jmap等。（建议详细去看看这些命令的区别和作用，都可能会被问到）

追问1：内存持续上升，如何排查？

回答：CPU100%那么一定有线程在占用系统资源，找出哪个进程cpu高（top），该进程中的哪个线程cpu高（top -Hp），导出该线程的堆栈（jstack），查找哪个方法（栈帧）消耗时间（jstack）工作线程占比高 | 垃圾回收线程占比高。【详细可以到网络搜索，最好是自己清楚这个排查思路！】

- (1) 通过top找到占用率高的进程
- (2) 通过top -Hp pid找到占用CPU高的线程ID
- (3) 把线程ID转化为16进制，得到线程IDxx
- (4) 通过命令jstack 找到有问题的代码

追问2：jstack和jps的区别是什么？

jstack：（Stack Trace for Java）命令用于生成虚拟机当前时刻的线程快照。

线程快照就是当前虚拟机内每一条线程正在执行的方法堆栈的集合，生成线程快照的主要目的是定位线程出现长时间停顿的原因，如线程间死锁、死循环、请求外部资源导致的长时间等待等都是导致线程长时间停顿的常见原因。

在代码中可以用`java.lang.Thread`类的`getAllStackTraces()`方法用于获取虚拟机中所有线程的`StackTraceElement`对象。使用这个方法可以通过简单的几行代码就完成`jstack`的大部分功能，在实际项目中不妨调用这个方法做个管理员页面，可以随时使用浏览器来查看线程堆栈。

jps：列出当前机器上正在运行的虚拟机进程

-p :仅仅显示VM标示，不显示jar, class, main参数等信息。

-m:输出主函数传入的参数。下的hello就是在执行程序时从命令行输入的参数

-l:输出应用程序主类完整package名称或jar完整名称。

-v: 列出jvm参数, -Xms20m -Xmx50m是启动程序指定的jvm参数

8.虚拟机的加载机制是什么样的?

回答: JVM的类加载分为7个阶段: 分别是加载、验证、准备、解析、初始化、使用和卸载。

加载: 读取Class文件, 并根据Class文件描述创建对象的过程。

验证: 确保Class文件符合当前虚拟机的要求。

准备: 在方法区中为类变量分配内存空间并设置类中变量的初始值。

解析: JVM会将常量池中的符号引用替换为直接引用。

初始化: 执行类构造器方法为类进行初始化。

追问1: 类加载有哪些?

回答: JVM提供了三种类加载器, 分别启动类加载器(Bootstrap Classloader)、扩展类加载器(Extention Classloader)和应用类加载器(Application Classloader)

追问2: 什么叫双亲委派机制?

回答: 双亲委派机制是指一个类在收到类加载请求后不会尝试自己加载这个类, 而且把该类加载请求委派给其父类去完成, 父类在接收到该加载请求后又会将其委派给自己的父类, 以此类推, 这样所有的类加载请求都被向上委派到启动类加载器中。若父类加载器在接收到类加载请求后发现自己也无法加载该类, 则父类会将该请求反馈给子类向下委派子类加载器加载该类, 直到被加载成功, 若找不到会曝出异常。

追问3: 如何打破双亲委派机制?

回答: 重写一个类继承ClassLoader, 并重写loadClass方法。(Tomcat是不支持双亲委派机制的)

更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享,
可以上微信搜索公众号: 代码界的小白, 和小白一起学编程!

数据库相关

MySQL

首先给大家推荐一本复习MySQL的书《MySQL技术内幕InnoDB存储引擎》，可以说整个找工作的过程中，这本书我一直都在翻看，整本书有10个章节，如果你只是想突击面试的话，只需要着重看以下几个章节：第二章 InnoDB存储引擎、第五章 索引与算法、第六章 锁、第七章 事务，这四个章节的内容在面试中是比较高频出现的。如果你想详细的学习 MySQL，那么建议全文阅读学习。(MySQL实战有朋友推荐看极客视频MySQL45讲)

索引高频面试问题

1.请简述常用的索引有哪些种类？

回答：

普通索引：即针对数据库表创建索引

唯一索引：与普通索引类似，不同的就是：MySQL 数据库索引列的值必须唯一，但允许有空值

主键索引：它是一种特殊的唯一索引，不允许有空值。一般是在建表的时候同时创建主键索引

组合索引(联合索引)：为了进一步榨取 MySQL 的效率，就要考虑建立组合索引。

即将数据库表中的多个字段联合起来作为一个组合索引。

2.mysql 数据库中索引的工作机制是什么？

回答：数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用 B 树及其变种 B+树

3. 唯一索引和主键索引的区别，唯一、主键索引与聚簇、非聚簇索引的关系？

唯一索引与主键

唯一索引是在表上一个或者多个字段组合建立的索引，这个（或这几个）字段的值组合起来在表中不可以重复。一张表可以建立任意多个唯一索引，但一般只建立一个。

主键是一种特殊的唯一索引，区别在于，唯一索引列允许null值，而主键列不允许为null值。一张表最多建立一个主键，也可以不建立主键。

聚簇索引、非聚簇索引、主键

在《数据库原理》一书中是这么解释聚簇索引和非聚簇索引的区别：

聚簇索引的叶子节点就是数据节点，而非聚簇索引的叶子节点仍然是索引节点，只不过有指向对应数据块的指针。

怎么理解呢？

聚簇索引的顺序，就是数据在硬盘上的物理顺序。一般情况下主键就是默认的聚簇索引。

一张表只允许存在一个聚簇索引，因为真实数据的物理顺序只能有一种。如果一张表上还没有聚簇索引，为它新创建聚簇索引时，就需要对已有数据重新进行排序，所以对表进行修改速度较慢是聚簇索引的缺点，对于经常更新的列不宜建立聚簇索引。

聚簇索引性能最好，因为一旦具有第一个索引值的记录被找到，具有连续索引值的记录也一定物理地紧跟其后。一张表只能有一个聚簇索引，所以非常珍贵，必须慎重设置，一般要根据这个表最常用的SQL查询方式选择某个（或多个）字段作为聚簇索引（或复合聚簇索引）。

聚簇索引默认是主键，如果表中没有定义主键，InnoDB[1]会选择一个唯一的非空索引代替（“唯一的非空索引”是指列不能出现null值的唯一索引，跟主键性质一样）。如果没有这样的索引，InnoDB会隐式地定义一个主键来作为聚簇索引。

聚簇索引与唯一索引

严格来说，聚簇索引不一定是唯一索引，聚簇索引的索引值并不要求是唯一的，唯一聚簇索引才是！在一个有聚簇索引的列上是可以插入两个或多个相同值的，这些相同值在硬盘上的物理排序与聚簇索引的排序相同，仅此而已。

4.InnoDB存储引擎有哪些常见的索引？

回答：B+树索引、全文索引和哈希索引。

1,b-tree:是一颗树(二叉树,平衡二叉树,平衡树(B-TREE))

使用平衡树实现索引，是mysql中使用最多的索引类型；在innodb中，存在两种索引类型，第一种是主键索引（primary key），在索引内容中直接保存数据的地址；第二种是其他索引，在索引内容中保存的是指向主键索引的引用；所以在使用innodb的时候，要尽量的使用主键索引，速度非常快；

方法

2,hash:把索引的值做hash运算，并存放到hash表中，使用较少，一般是memory引擎使用；优点：因为使用hash表存储，按照常理，hash的性能比B-TREE效率高很多。

hash索引的缺点：

- 1，hash索引只能适用于精确的值比较，=，in，或者<>；无法使用范围查询；
- 2，无法使用索引排序；
- 3，组合hash索引无法使用部分索引；
- 4，如果大量索引hash值相同，性能较低；

5.说一下B+树与B树的区别？

1.B+树内节点不存储数据，所有 data 存储在叶节点导致查询时间复杂度固定为 $\log n$ 。而B-树查询时间复杂度不固定，与 key 在树中的位置有关，最好为O(1)。

2. B+树叶节点两两相连可大大增加区间访问性，可使用在范围查询等，而B-树每个节点 key 和 data 在一起，则无法区间查找。

3.B+树更适合外部存储。由于内节点无 data 域，每个节点能索引的范围更大更精确

6.为什么MySQL数据库使用B+树不使用B树？

回答：当存储同数量级的数据的时候，B+树的高度比B树的高度小，这样的话进程IO操作的次数就少，效果就高。因为B+树的所有非叶子节点只存索引，数据存在叶子节点，一般3层的树高度，即可存千万级别的数据，而B数不行。（具体的计算可以到网上去看看，有面试官可能会问你怎么算出来的。）

7.Hash 索引和 B+ 树索引区别是什么？你在设计索引是怎么抉择的？

- B+ 树可以进行范围查询，Hash 索引不能。
- B+ 树支持联合索引的最左侧原则，Hash 索引不支持。
- B+ 树支持 `order by` 排序，Hash 索引不支持。
- Hash 索引在等值查询上比 B+ 树效率更高。
- B+ 树使用 `like` 进行模糊查询的时候，`like` 后面（比如%开头）的话可以起到优化的作用，Hash 索引根本无法进行模糊查询。

8.如何选择在哪些列上建索引？

回答：一张表一般都要去建主键，所以主键索引几乎是每张表必备的，这个就不多说了。

选择性高的列，也就是重复度低的列。比如女子学校学生表中的性别列，所有数据的值都是女，这样的列就不适合建索引。比如学生表中的身份证号列，选择性就很高，就适合建索引。

经常用于查询的列（出现在**where**条件中的列）。不过如果不符合作上一条的条件，即便是出现在**where**条件中也不适合建索引，甚至就不应该出现在**where**条件中。

多表关联查询时作为关联条件的列。比如学生表中有班级ID的列用于和班级表关联查询时作为关联条件，这个列就适合建索引。

值会频繁变化的列不适合建索引。因为在数据发生变化时是需要针对索引做一些处理的，所以如果不是有非常必要的原因，不要值会频繁变化的列上建索引，会影响数据更新的性能。反过来也就是说索引要建在值比较固定不变的列上。

一张表上不要建太多的索引。和上一条的原因类似，如果一张表上的索引太多，会严重影响数据增删改的性能。也会耗费很大的磁盘空间。

创建

- 1 , 较频繁的作为查询条件的字段应该创建索引 ;
- 2 , 唯一性太差的字段不适合单独创建索引 , 即使频繁作为查询条件 ;
作为索引的列,如果不能有效的区分数据,那么这个列就不适合作为索引列;比如(性别,状态不多的状态列)
举例:SELECT sum(amount) FROM accountflow WHERE accountType = 0;
假如把accountType作为索引列,因为accountType只有14种,所以,如果根据accountType来创建索引,最多只能按照1/14的比例过滤掉数据;但是,如果可能出现,只按照该条件查询,那我们就要考虑到其他的提升性能的方式了;
- 3 , 更新非常频繁的字段不适合创建索引 ; 原因,索引有维护成本;
- 4 , 不会出现在WHERE 子句中的字段不该创建索引 ;
- 5, 索引不是越多越好;(只为必要的列创建索引)
 - 1,不管你有多少个索引,一次查询至多采用一个索引;(索引和索引之间是独立的)
 - 2,因为索引和索引之间是独立的,所以说每一个索引都应该是单独维护的;数据的增/改/删,会导致所有的索引都要单独维护;

9.什么是最左匹配原则?

最左前缀匹配原则和联合索引的索引存储结构和检索方式是有关系的。

在组合索引树中, 最底层的叶子节点按照第一列a列从左到右递增排列, 但是b列和c列是无序的, b列只有在a列值相等的情况下小范围内递增有序, 而c列只能在a, b两列相等的情况下小范围内递增有序。

就像上面的查询, B+树会先比较a列来确定下一步应该搜索的方向, 往左还是往右。如果a列相同再比较b列。但是如果查询条件没有a列, B+树就不知道第一步应该从哪个节点查起。

可以说创建的idx_abc(a,b,c)索引, 相当于创建了(a)、(a,b)、(a,b,c) 三个索引。、

组合索引的最左前缀匹配原则: 使用组合索引查询时, mysql会一直向右匹配直至遇到范围查询(>、<、between、like)就停止匹配。

10. 说一下覆盖索引？

覆盖索引并不是说是索引结构，覆盖索引是一种很常用的优化手段。因为在使用辅助索引的时候，我们只可以拿到主键值，相当于获取数据还需要再根据主键查询主键索引再获取到数据。但是试想下这么一种情况，在上面 *abc_innodb* 表中的组合索引查询时，如果我只需要 *abc* 字段的，那是不是意味着我们查询到组合索引的叶子节点就可以直接返回了，而不需要回表。这种情况就是覆盖索引。

举例子：

假设我们只需要查询商品的名称、价格信息，我们有什么方式来避免回表呢？我们可以建立一个组合索引，即商品编码、名称、价格作为一个组合索引。如果索引中存在这些数据，查询将不会再次检索主键索引，从而避免回表。

从辅助索引中查询得到记录，而不需要通过聚族索引查询获得，MySQL 中将其称为覆盖索引。使用覆盖索引的好处很明显，我们不需要查询出包含整行记录的所有信息，因此可以减少大量的 I/O 操作。

通常在 InnoDB 中，除了查询部分字段可以使用覆盖索引来优化查询性能之外，统计数量也会用到。例如，`SELECT COUNT(*)` 时，如果不存在辅助索引，此时会

通过查询聚族索引来统计行数，如果此时正好存在一个辅助索引，则会通过查询辅助索引来统计行数，减少 I/O 操作。

更多 Java 面试八股、Java 后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜
代码界的小白

代码界的小白

锁的高频面试问题

1.什么是锁，锁的作用是什么？

回答：锁是数据库系统区别文件系统的一个关键特性，锁机制用于管理对共享资源的并发访问，保持数据的完整性和一致性。【摘自：MySQL技术内幕InnoDB存储引擎】

2.数据库有哪些锁？**lock**和**latch**的区别

回答：数据库中有表锁和行锁等

lock锁：锁的对象是事务，用于锁定数据库中的对象，如表、页、行等，并且**lock**锁一般在commit或rollback后释放，有死锁机制。

latch锁：一般称为轻量级锁，要求锁定的时间必须非常短，在InnoDB中又可以分为**mutex**(互斥量)和**rwlock**(读写锁)。目的是用来保证并发线程操作临界资源的正确性，并且通常没有死锁检测的机制。

3.InnoDB存储引擎中的锁都有哪些类型？

回答：可以分为共享锁、排他锁、意向锁、一致性非锁定读和一致性锁定读。

其中共享锁和排他锁均属于行级锁。

共享锁(S Lock): 运行事务读一行数据。

排他锁(X Lock): 允许事务删除或更新一行数据。

行锁的三种算法：

Record Lock: 单个行记录上的锁

Gap Lock: 间隙锁，锁定一个范围，但不包含记录本身。

Next-Key Lock: *Gap+Record Lock*锁定一个范围，并且锁定记录本身。

意向锁属于表级别的锁，又可以分为意向共享锁(IS Lock)和意向排他锁(IX Lock)。

意向共享锁(IS Lock): 事务想要获得一张表中某几行的共享锁。

意向排他锁(IX Lock): 事务想要获得一张表中某几行的排他锁。

一致性非锁定读: 指InnoDB存储引擎通过多版本控制的方式来读取当前执行时间数据库中行的数据。如果读取的行正在执行DELETE或UPDATE操作，这时读取操作不会因此等待行上锁的释放，相反的，InnoDB存储引擎会读取一个快照数据。

一致性锁定读: InnoDB存储引擎对于SELECT语句支持两种一致性锁定读的操作：

select ... for update和select ... lock in share mode。

4. 什么是一致性非锁定读(MVCC)?

什么是MVCC?

MVCC实现原理【MVCC多版本并发控制，指的是一种提高并发的技术。】

Multi-Version Concurrency Control。

最早的数据库系统，只有读读之间可以并发，读写，写读，写写都要阻塞。引入多版本之后，只有写写之间相互阻塞，其他三种操作都可以并行，这样大幅度提高了InnoDB的并发度。

MVCC能解决什么问题，好处是？

数据库并发场景有三种，分别为：

读-读: 不存在任何问题，也不需要并发控制

读-写: 有线程安全问题，可能会造成事务隔离性问题，可能遇到脏读，幻读，不可重复读

写-写: 有线程安全问题，可能会存在更新丢失问题，比如第一类更新丢失，第二类更新丢失

MVCC带来的好处是？

MVCC可以为数据库解决以下问题

在并发读写数据库时，可以做到在读操作时不用阻塞写操作，写操作也不用阻塞读操作，提高了数据库并发读写的性能

同时还可以解决脏读，幻读，不可重复读等事务隔离问题，但不能解决更新丢失问题

MVCC只在读取已提交和可重复读两种隔离级别下有作用

MVCC常见的实现方式乐观锁和悲观锁

MVCC是行级锁的变种，很多情况下避免了加锁操作。

应对高并发事务，MVCC比单纯的加锁更高效；

InnoDB存储引擎在数据库每行数据的后面添加了三个字段，不是两个!!

核心概念【很重要！！！】

1.Read view一致性视图【主要是用来做可见性判断的，比较普遍的解释便是"本事务不可见的当前其他活跃事务"，】

2.read view快照的生成时机，也非常关键，正是因为生成时机的不同，造成了RC,RR两种隔离级别的不同可见性；

在innodb中(默认repeatable read级别)，事务在begin/start transaction之后的第一条select读操作后，会创建一个快照(read view)，将当前系统中活跃的其他事务记录记录起来；

在innodb中(默认repeatable committed级别)，事务中每条select语句都会创建一个快照(read view)；

3.undo-log 【回滚日志，通过undo读取之前的版本信息，以此实现非锁定读取！】是MVCC的重要组成部分！

当我们对记录做了变更操作时就会产生undo记录，Undo记录默认被记录到系统表空间(ibdata)中，但从5.6开始，也可以使用独立的Undo 表空间。

Undo记录中存储的是老版本数据，当一个旧的事务需要读取数据时，为了能读取到老版本的数据，需要顺着undo链找到满足其可见性的记录。

另外，在回滚段中的undo logs分为：insert undo log 和 update undo log
insert undo log : 事务对insert新记录时产生的undolog, 只在事务回滚时需要，并且在事务提交后就可以立即丢弃。 update undo

update undo log : 事务对记录进行delete和update操作时产生的undo log, 不仅在事务回滚时需要，

一致性读也需要，所以不能随便删除，只有当数据库所使用的快照中不涉及该日志记录，对应的回滚日志才会被purge线程删除。

4.InnoDB存储引擎在数据库每行数据的后面添加了三个字段

分别是事务ID、回滚指针和

6字节的DB_ROW_ID字段：包含一个随着新行插入而单调递增的行ID，当由innodb自动产生聚集索引时，聚集索引会包括这个行ID的值，否则这个行ID不会出现在任何索引中。

5.可见性比较算法（这里每个比较算法后面的描述是建立在rr级别下，rc级别也是使用该比较算法，此处未做描述）

设要读取的行的最后提交事务id(即当前数据行的稳定事务id)为 trx_id_current

当前新开事务id为 new_id

当前新开事务创建的快照read view 中最早的事务id为up_limit_id, 最迟的事务id为low_limit_id(注意这个low_limit_id=未开启的事务id=当前最大事务id+1)

比较：

1. `trx_id_current < up_limit_id`, 这种情况比较好理解, 表示, 新事务在读取该行记录时, 该行记录的稳定事务ID是小于, 系统当前所有活跃的事务, 所以当前行稳定数据对新事务可见, 跳到步骤5.
2. `trx_id_current >= trx_id_last`, 这种情况也比较容易理解, 表示, 该行记录的稳定事务id是在本次新事务创建之后才开启的, 但是却在本次新事务执行第二个select前就commit了, 所以该行记录的当前值不可见, 跳到步骤4
3. `trx_id_current <= trx_id_current <= trx_id_last`, 表示: 该行记录所在事务在本次新事务创建的时候处于活动状态, 从`up_limit_id`到`low_limit_id`进行遍历, 如果`trx_id_current`等于他们之中的某个事务id的话, 那么不可见, 调到步骤4, 否则表示可见。
4. 从该行记录的`DB_ROLL_PTR`指针所指向的回滚段中取出最新的undo-log的版本号, 将它赋值该`trx_id_current`, 然后跳到步骤1重新开始判断。
5. 将该可见行的值返回。

5. 锁可能会带来什么问题？

回答：通过锁机制实现了事务的隔离性，使得事务可以并发的工作，但同时也会有一些潜在的问题。锁会带来如下问题：脏读、不可重复度、丢失修改和幻读。

- **脏读（Dirty read）**：当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是“脏数据”，依据“脏数据”所做的操作可能是不正确的。
- **丢失修改（Lost to modify）**：指在一个事务读取一个数据时，另外一个事务也访问了该数据，那么在第一个事务中修改了这个数据后，第二个事务也修改了这个数据。这样第一个事务内的修改结果就被丢失，因此称为丢失修改。例如：事务1读取某表中的数据A=20，事务2也读取A=20，事务1修改A=A-1，事务2也修改A=A-1，最终结果A=19，事务1的修改被丢失。
- **不可重复读（Unrepeatable read）**：指在一个事务内多次读同一数据。在这个事务还没有结束时，另一个事务也访问该数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改导致第一个事务两次读取的数据可能不太一样。这就发生了在一个事务内两次读到的数据是不一样的情况，因此称为不可重复读。
- **幻读（Phantom read）**：幻读与不可重复读类似。它发生在一个事务（T1）读取了几行数据，接着另一个并发事务（T2）插入了一些数据时。在随后的查询中，

第一个事务（T1）就会发现多了一些原本不存在的记录，就好像发生了幻觉一样，所以称为幻读。

不可重复读和幻读区别：

不可重复读的重点是修改比如多次读取一条记录发现其中某些列的值被修改，幻读的重点在于新增或者删除比如多次读取一条记录发现记录增多或减少了。【摘自MySQL技术内幕：InnoDB存储引擎可以使用Next-Key Locking机制来避免Phantom Problem问题】

6.数据库中的死锁概念你知道吗？

回答：死锁是指两个或两个以上的事务在执行过程中，因争夺资源而造成的一种互相等待的现象。

解决死锁的办法：一种是超时回滚，一种是采用死锁检测机制（wait-for graph等待图）

如果面试官让你举例子，可以举例下面的例子：

事务A	事务B
BEGIN; SELECT id FROM `order_record` where `order_no` = 4 for update; //检查是否存在 order_no等于4的订单	BEGIN;
	SELECT id FROM `order_record` where `order_no` = 5 for update; //检查是否存在 order_no等于5的订单
INSERT INTO `order_record`(`order_no`, `status`, `create_date`) VALUES (4, 1, '2019-07-13 10:57:03'); //如果没有，则插入 信息 此时，锁等待中.....	
	INSERT INTO `order_record`(`order_no`, `status`, `create_date`) VALUES (5, 1, '2019-07-13 10:57:03'); //如果没有，则插入 信息 此时，锁等待中.....
COMMIT;(未完成)	COMMIT;(未完成)

在 MySQL 中，gap lock 默认是开启的，即innodb_locks_unsafe_for_binlog 参数值是 disable 的，且 MySQL 中默认的是 RR 事务隔离级别。

当我们执行以下查询 SQL 时，由于 `order_no` 列为非唯一索引，此时又是 RR 事务隔离级别，所以 SELECT 的加锁类型为 gap lock，这里的 gap 范围是 $(4, +\infty)$ 。

```
SELECT id FROM demo.order_record where order_no = 4 for update;
```

执行查询 SQL 语句获取的 gap lock 并不会导致阻塞，而当我们执行以下插入 SQL 时，会在插入间隙上再次获取插入意向锁。插入意向锁其实也是一种 gap 锁，它与 gap lock 是冲突的，所以当其它事务持有该间隙的 gap lock 时，需要等待其它事务释放 gap lock 之后，才能获取到插入意向锁。

以上事务 A 和事务 B 都持有间隙 $(4, +\infty)$ 的 gap 锁，而接下来的插入操作为了获取到插入意向锁，都在等待对方事务的 gap 锁释放，于是就造成了循环等待，导致死锁。

```
INSERT INTO demo.order_record(order_no, status, create_date)
VALUES (5, 1, '2019-07-13 10:57:03');
```

最后送上锁之间的兼容性表格：

	Gap	Insert Intention	Record	Next-Key
Gap	兼容	冲突	兼容	兼容
Insert Intention	冲突	兼容	兼容	冲突
Record	兼容	兼容	冲突	冲突
Next-Key	兼容	兼容	冲突	冲突
备注	横向是已经持有的锁，纵向是正在请求的锁。			

更多 Java 面试八股、Java 后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！

事务的高频面试问题

1.什么是事务？

事务是数据库区别于文件系统的重要特性之一，事务可以一条非常简单的SQL语句组成，也可以由一组复杂的SQL语句组成，事务是访问并更新数据库中各种数据项的一个程序执行单元。在事务中的操作，要么都做修改，要么都不做，这就是事务的主要目的。

2.事务的特性有哪些？分别是怎么保证的？

回答：事务的四大特性分别是原子性、一致性、隔离性和持久性。

下面这段话摘自《MySQL技术内幕-InnoDB存储引擎》：

原子性、一致性和持久性是通过数据库的redo log和undo log来完成。redo log称为重做日志，用来保证事务的原子性和持久性。undo log用来保证事务的一致性。而隔离性是通过锁实现的。【具体大家看书吧，书上比较详细。】

面试官可能会问三种日志的区别和作用。

redo log：恢复提交事务修改的页操作；通常是物理日志，记录的是页的物理修改操作。

undo log：回滚记录到某个特定版本；通常是逻辑日志，根据每行记录进行记录。

bin log：来进行Point-In-Time(PIT)的恢复及主从复制环境的建立。

这里面试官可能会接着问**binlog**和**redolog**的区别？

回答：

(1) 重做日志是在InnoDB存储引擎层产生的，而二进制日志是在MySQL数据库上层产生的，二进制日志不仅仅针对InnoDB存储引擎，任何存储引擎都会产生二进制日志。

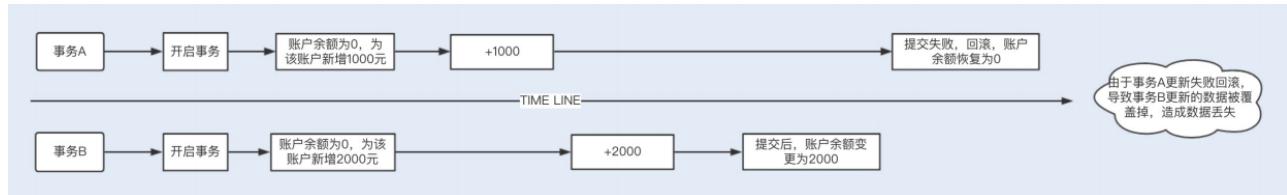
(2) 两种日志的记录内容形式不同。二进制日志是一种逻辑日志，记录的是SQL语句；而InnoDB存储引擎层面的重做日志是物理格式日志，记录的是对于每个页的修改。

(3) 写入磁盘的时间不同，二进制日志只在事务提交完成后进行一次写入，而redo log在事务进行中不断的写入。

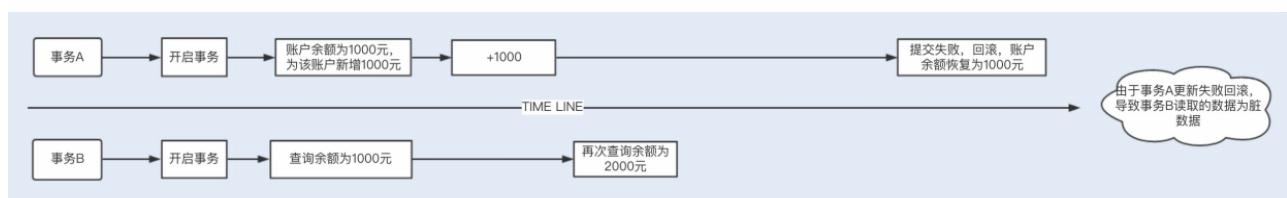
3. 事务的隔离级别有哪些？InnoDB存储引擎默认的是什么存储引擎？为什么？

回答：首先回答一些数据库中并发事务带来的问题

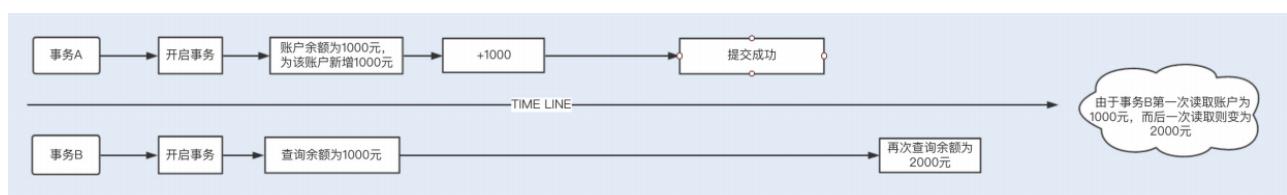
1. 数据丢失



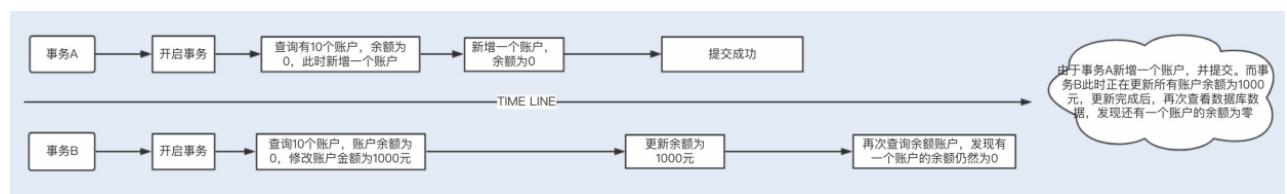
2. 脏读



3. 不可重复读



4. 幻读



不同的隔离级别

分别可以解决并发事务产生的几个问题，对应如下：

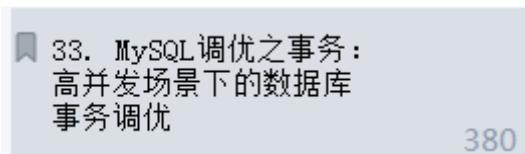
未提交读（**Read Uncommitted**）：在事务 A 读取数据时，事务 B 读取和修改数据加了共享锁。这种隔离级别，会导致脏读、不可重复读以及幻读。

已提交读（**Read Committed**）：在事务 A 读取数据时增加了共享锁，一旦读取，立即释放锁，事务 B 读取修改数据时增加了行级排他锁，直到事务结束才释放锁。也就是说，事务 A 在读取数据时，事务 B 只能读取数据，不能修改。当事务 A 读取到数据后，事务 B 才能修改。这种隔离级别，可以避免脏读，但依然存在不可重复读以及幻读的问题。

可重复读（**Repeatable Read**）：在事务 A 读取数据时增加了共享锁，事务结束，才释放锁，事务 B 读取修改数据时增加了行级排他锁，直到事务结束才释放锁。也就是说，事务 A 在没有结束事务时，事务 B 只能读取数据，不能修改。当事务 A 结束事务，事务 B 才能修改。这种隔离级别，可以避免脏读、不可重复读，但依然存在幻读的问题。

可序列化（**Serializable**）：在事务 A 读取数据时增加了共享锁，事务结束，才释放锁，事务 B 读取修改数据时增加了表级排他锁，直到事务结束才释放锁。可序列化解决了脏读、不可重复读、幻读等问题，但隔离级别越来越高的同时，并发性会越来越低。

建议：这块内容可以看一下《阿里调优手册》中的第33讲



需要阿里调优手册在微信公众号后台回复：阿里调优手册

MySQL InnoDB 存储引擎的默认支持的隔离级别是 **REPEATABLE-READ**（可重读）

隔离级别	脏读	不可重复读	幻读
READ-UNCOMMITTED	√	√	√
READ-COMMITTED	✗	√	√
REPEATABLE-READ	✗	✗	√
SERIALIZABLE	✗	✗	✗

至于InnoDB为什么选用可重复读，我的个人理解是：在InnoDB存储引擎中，使用可重复读可以解决脏读、不可重复读，而幻读也有可能发生，但是是可以避免的，通过加Next-Key Lock锁可以解决幻读问题。并且并非隔离级别越高越好，隔离级别越高的话，并发性能越低，所以在实际的开发中，需要根据业务场景进行选择事务的隔离级别。

更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！

Redis

一般关于Redis面试官可能都是从你的项目中引出的问题，或者问你有没有用过除了MySQL以外的数据库等。

1. 说说你对Redis的了解？

首先Redis是基于C语言编写的，而且是内存中的数据库，读写速度很快。在项目中也经常会使用Redis，一般会用来做缓存、或者分布式锁，也可以来设计消息队列，同时还支持事务、持久化、Lua脚本、多种集群方案。

追问1：与 Memcached 的区别是什么？

现在公司一般都是用 Redis 来实现缓存，而且 Redis 自身也越来越强大了！不过，了解 Redis 和 Memcached 的区别和共同点，有助于我们在做相应的技术选型的时候，能够做到有理有据！

共同点：

1. 都是基于内存的数据库，一般都用来当做缓存使用。
2. 都有过期策略。
3. 两者的性能都非常高。

区别：

1. **Redis** 支持更丰富的数据类型（支持更复杂的应用场景）。Redis 不仅仅支持简单的 k/v 类型的数据，同时还提供 list, set, zset, hash 等数据结构的存储。
Memcached 只支持最简单的 k/v 数据类型。
2. **Redis** 支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用，而 **Memecache** 把数据全部存在内存之中。
3. **Redis** 有灾难恢复机制。因为可以把缓存中的数据持久化到磁盘上。
4. **Redis** 在服务器内存使用完之后，可以将不用的数据放到磁盘上。但是，
Memcached 在服务器内存使用完之后，就会直接报异常。
5. **Memcached** 没有原生的集群模式，需要依靠客户端来实现往集群中分片写入数据；但是 **Redis** 目前是原生支持 **cluster** 模式的。
6. **Memcached** 是多线程，非阻塞 IO 复用的网络模型；**Redis** 使用单线程的多路 IO 复用模型。（Redis 6.0 引入了多线程 IO）
7. **Redis** 支持发布订阅模型、Lua 脚本、事务等功能，而 **Memcached** 不支持。
并且，**Redis** 支持更多的编程语言。

8. Memcached过期数据的删除策略只用了惰性删除，而 **Redis** 同时使用了惰性删除与定期删除。

2.Redis有哪些数据类型？

回答：常见的有五种基本数据类型和三种特殊数据类型，

基本数据结构：String、list、set、zset和hash，三种特殊数据类型：位图(bitmap)、计数器(hyperloglogs)和地理空间(geospatial indexes)。

String: 一般常用在需要计数的场景，比如用户的访问次数、热点文章的点赞转发数量等等。

list: 发布与订阅或者说消息队列、慢查询。

hash: 系统中对象数据的存储。

set: 需要存放的数据不能重复以及需要获取多个数据源交集和并集等场景

zset: 需要对数据根据某个权重进行排序的场景。比如在直播系统中，实时排行信息包含直播间在线用户列表，各种礼物排行榜，弹幕消息（可以理解为按消息维度的消息排行榜）等信息。

3.什么是Redis持久化？Redis有哪几种持久化方式？优缺点是什么？

持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失。Redis 提供了两种持久化方式：RDB（默认）快照方式 和 AOF 追加方式。

RDB(Redis DataBase): 通过创建快照来获取存储在内存里面的数据在某个时间点上的副本。在创建快照之后，用户可以对快照进行备份，可以将快照复制到其他服务器从而创建相同数据的服务器副本。（如果系统真的发生崩溃，用户将丢失最近一次生成快照之后更改的所有数据。）

AOF(Append Only File): 将被执行的写命令写到AOF文件的末尾。

比较：

1. AOF文件比RDB更新频率高，优先使用AOF还原数据。
2. AOF比RDB更安全也更大
3. RDB性能比AOF好
4. 如果两个都配了优先加载AOF

4.Redis数据过期后的删除策略？

常用的过期数据的删除策略就两个（重要！自己造缓存轮子的时候需要格外考虑的东西）：

1. 惰性删除：只会在取出key的时候才对数据进行过期检查。这样对CPU最友好，但是可能会造成太多过期key没有被删除。
2. 定期删除：每隔一段时间抽取一批key执行删除过期key操作。并且，Redis底层会通过限制删除操作执行的时长和频率来减少删除操作对CPU时间的影响。

定期删除对内存更加友好，惰性删除对CPU更加友好。两者各有千秋，所以Redis采用的是定期删除+惰性/懒汉式删除。

但是，仅仅通过给key设置过期时间还是有问题的。因为还是可能存在定期删除和惰性删除漏掉了很多过期key的情况。这样就导致大量过期key堆积在内存里，然后就Out of memory了。

5.Redis的数据淘汰策略有哪些

Redis提供6种数据淘汰策略：

1. **volatile-lru (least recently used)**：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰
2. **volatile-ttl**：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰
3. **volatile-random**：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰
4. **allkeys-lru (least recently used)**：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）
5. **allkeys-random**：从数据集（server.db[i].dict）中任意选择数据淘汰
6. **no-eviction**：禁止驱逐数据，也就是说当内存不足以容纳新写入数据时，新写入操作会报错。这个应该没人使用吧！

6.什么是缓存穿透？如何避免？什么是缓存击穿，如何避免？什么是缓存雪崩？何如避免？

缓存穿透

一般的缓存系统，都是按照key去缓存查询，如果不存在对应的value，就应该去后端系统查找（比如DB）。一些恶意的请求会故意查询不存在的key，请求量很大，就会对后端系统造成很大的压力。这就叫做缓存穿透。

如何避免？

1：对查询结果为空的情况也进行缓存，缓存时间设置短一点，或者该key对应的数据insert了之后清理缓存。

2: 对一定不存在的key进行过滤。可以把所有的可能存在的key放到一个大的Bitmap中，查询时通过该bitmap过滤。

缓存击穿：对于设置了过期时间的key，缓存在某个时间点过期的时候，恰好这时间点对这个Key有大量的并发请求过来，这些请求发现缓存过期一般都会从后端DB加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把DB压垮。

如何避免？：
1. 使用互斥锁：当缓存失效时，不立即去load db，先使用如Redis的setnx去设置一个互斥锁，当操作成功返回时再进行load db的操作并回设缓存，否则重试get缓存的方法。
2. 永远不过期：物理不过期，但逻辑过期（后台异步线程去刷新）。

缓存雪崩

当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，会给后端系统带来很大压力。导致系统崩溃。

如何避免？

- 1: 在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个key只允许一个线程查询数据和写缓存，其他线程等待。
- 2: 做二级缓存，A1为原始缓存，A2为拷贝缓存，A1失效时，可以访问A2，A1缓存失效时间设置为短期，A2设置为长期
- 3: 不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀

7. 使用过Redis分布式锁么，它是怎么实现的？

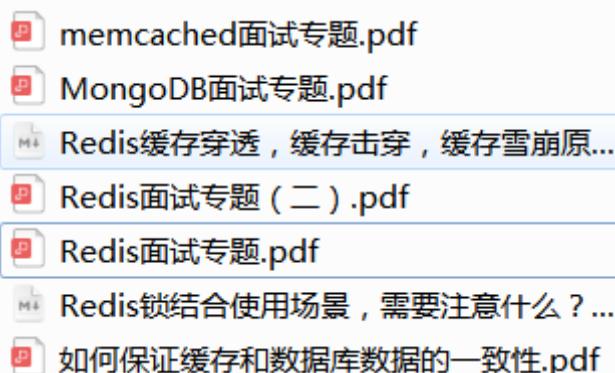
先拿setnx来争抢锁，抢到之后，再用expire给锁加一个过期时间防止锁忘记了释放。

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？

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！

8. 如果解决数据不一致问题？

这个问题的答案在文档中，需要文档的在微信公众号后台回复：Redis电子版资料，即可获取。



这里列出来一些常问的问题，还有一些其他的问题，具体的大家可以获取Redis电子版的pdf，Redis结合项目间的比较多，如果你做的项目里用到了，一定要好好的看看！尤其是分布式锁，或者是在单点登录的时候用到了Redis等。

更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜



代码界的小白

设计模式相关

1. 你了解的设计模式有哪些？

回答：总的设计模式有23种，可以分为三大类。(建议在面试的时候说几个自己熟悉的，比如单例模式、工厂模式、模板模式等)

创建型模式(共五种)：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

结构型模式(共七种)：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

行为型模式(共十一种)：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

在问Spring的时候可能会问到这样一个问题：在**Spring**框架中都用到了哪些设计模式，并举例说明？

- 工厂设计模式：**Spring**使用工厂模式通过 `BeanFactory`、`ApplicationContext` 创建 bean 对象。
- 代理设计模式：**Spring AOP** 功能的实现。
- 单例设计模式：**Spring** 中的 Bean 默认都是单例的。
- 包装器设计模式：我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。这种模式让我们可以根据客户的需求能够动态切换不同的数据源。
- 观察者模式：**Spring** 事件驱动模型就是观察者模式很经典的一个应用。
- 适配器模式：**Spring AOP** 的增强或通知(Advice)使用到了适配器模式、**spring MVC** 中也是用到了适配器模式适配 `Controller`。

2. 设计模式的原则有哪些？

回答：设计模式共有六大原则，分别是：单一职责原则、开闭原则、里氏代换原则、依赖倒转原则、接口隔离原则、迪米特法则。

单一职责：一个类只负责一个功能领域中相应的职责，或者可以定义为：就一个类而言，应该只有一个引起它变化的原因。

开闭原则：软件实体应该对扩展开放，对修改关闭。其含义是说一个软件实体应该通过扩展来实现变化，而不是通过修改已有的代码来实现变化。

里氏代换原则：通俗点讲，只要父类能出现的地方子类就可以出现，而且替换为子类也不会产生任何错误或异常，使用者可能根本就不需要知道是父类还是子类。但是，反过来就不行了，有子类出现的地方，父类未必就能适应。

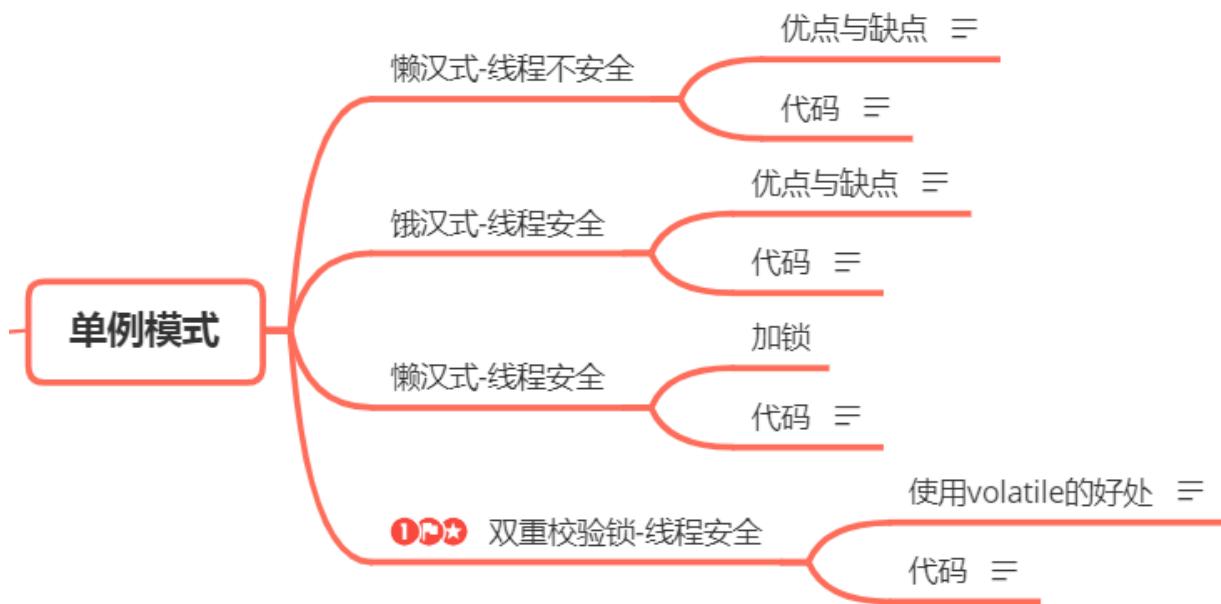
依赖倒转原则：这个是开闭原则的基础，具体内容：真对接口编程，依赖于抽象而不依赖于具体。

接口隔离原则：使用多个隔离的接口，比使用单个接口要好。还是一个降低类之间的耦合度的意思，从这儿我们看出，其实设计模式就是一个软件的设计思想，从大型软件架构出发，为了升级和维护方便。所以上文中多次出现：降低依赖，降低耦合。

迪米特法则：为什么叫最少知道原则，就是说：一个实体应当尽量少的与其他实体之间发生相互作用，使得系统功能模块相对独立。

3. 你比较了解哪些设计模式？

回答：这里大家一定要去找几个设计模式看看，比如单例、工厂和模板这些，这里就以单例模式给大家分享一下。



追问：写一下单例模式的代码？

懒汉式-线程不安全

```
public class Singleton {  
  
    private static Singleton Instance;  
  
    private Singleton() {  
        //  
    }  
  
    public static Singleton getInstance() {  
        if (Instance == null) {  
            Instance = new Singleton();  
        }  
        return Instance;  
    }  
}
```

```
}

public static Singleton GetInstance() {
    if (Instance == null) {
        Instance = new Singleton();
    }
    return Instance;
}

}
```

优缺点：

私有静态变量 `uniqueInstance` 被延迟实例化，这样做的好处是，如果没有用到该类，那么就不会实例化 `uniqueInstance`，从而节约资源。

线程不安全，多线程情况下会多次创建实例。

饿汉式-线程安全

```
public class Singleton {

    private static Singleton Instance = new Singleton();

    private Singleton() {
    }

    public static Singleton GetInstance() {
        return Instance;
    }

}
```

优缺点：

采取直接实例化 `uniqueInstance` 的方式就不会产生线程不安全问题。

但是直接实例化的方式也丢失了延迟实例化带来的节约资源的好处。

懒汉式-线程安全

```
private static Singleton Instance;

public static synchronized Singleton getInstance() {
    if (Instance == null) {
        Instance = new Singleton();
    }
    return Instance;
}
```

双重校验锁-线程安全

```
public class Singleton {

    private volatile static Singleton Instance;

    private Singleton() {
    }

    public static Singleton getInstance() {
        if (Instance == null) { //先判断实例是否存在，不存在再加锁
            synchronized (Singleton.class) { //由于Instance实例有没有被
                //创建过实例不知道，只能对其class加锁
                if (Instance == null) { //当多个线程的时候，只有一个进入，
                    //避免多次创建对象！
                    Instance = new Singleton();
                }
            }
        }
        return Instance;
    }
}
```

追问1：这里面试官可能会问使用volatile的好处？

Instance 采用 volatile 关键字修饰也是很有必要的， Instance = new Singleton();这段代码其实是分为三步执行：

1. 为 uniqueInstance 分配内存空间
2. 初始化 uniqueInstance
3. 将 uniqueInstance 指向分配的内存地址

但是由于 JVM 具有指令重排的特性，执行顺序有可能变成 1>3>2。指令重排在单线程环境下不会出现问题，但是在多线程环境下会导致一个线程获得还没有初始化的实例。例如，线程 T1 执行了 1 和 3，此时 T2 调用 `getInstance()` 后发现 `Instance` 不为空，因此返回 `Instance`，但此时 `Instance` 还未被初始化。

使用 `volatile` 可以禁止 JVM 的指令重排，保证在多线程环境下也能正常运行。

| 更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜

Q 代码界的小白

中间件等相关

Netty 高频面试问题

关于Netty这部分，如果面试被问到，那就有点难度了，不过一些基础的知识还是可以掌握一下的，如果自己做的项目里牵涉到了Netty相关的知识，那你就一定要好好准备了！

1. Netty是什么？

Netty提供异步的、事件驱动的网络应用程序框架和工具，用以快速开发高性能、高可靠的网络服务器和客户端程序。

也就是说，Netty是一个基于NIO的客户、服务器端的编程框架，使用Netty可以确保你快速和简单的开发出一个网络应用，例如实现了某种协议的客户、服务端应用。Netty相当于简化和流线化了网络应用的编程开发过程，例如：基于TCP和UDP的socket服务开发。

“快速”和“简单”并不用产生维护性或性能上的问题。Netty是一个吸收了多种协议（包括FTP、SMTP、HTTP等各种二进制文本协议）的实现经验，并经过相当精心设计的项目。最终，Netty成功的找到了一种方式，在保证易于开发的同时还保证了其应用的性能，稳定性和伸缩性。【百度百科】

总结：Netty是一个异步事件驱动的网络应用程序框架，用于快速开发可维护的高性能协议服务器和客户端。Netty是基于Nio的，封装了jdk的Nio，使用起来更加方法灵活。

追问1：Netty 特点是什么？

回答：三大特点，高并发、传输快和封装好。

高并发：Netty 是一款基于 NIO（Nonblocking IO，非阻塞IO）开发的网络通信框架，相比于 BIO（Blocking I/O，阻塞IO），他的并发性能得到了很大提高。

传输快：Netty 的传输依赖于零拷贝特性，尽量减少不必要的内存拷贝，实现了更高效率的传输。

封装好：Netty 封装了 NIO 操作的很多细节，提供了易于使用调用接口。

追问2：这里面试官可能会问你零拷贝！

零拷贝是非常重要的知识点，建议看看阿里调优手册！【后台回复：阿里调优】

零拷贝是一种避免多次内存复制的技术，用来优化读写I/O操作。

▼ Java编程性能调优

03. 字符串性能优化不容小觑，百M内存轻松有

04. 线程池使用之设计与优化
03. 字符串性能优化不容小觑，百M内存轻松有

05. ArrayList还是LinkedList？使用...

06. Stream如何提高遍历集合效率？

07. 深入浅出HashMap的设计与优化

08. 网络通信优化之I/O模型：如何解...

09. 网络通信优化之序列化：避免使用...

10. 网络通信优化之通信协议：如何优...

11. 深入了解NIO的优化实现原理

推荐几款常用的性能测试工具

追问3：Netty 的应用场景有哪些

回答：Netty可构建高性能、低延时的各种Java中间件，例如MQ、分布式服务框架、ESB消息总线等，Netty主要作为基础通信框架提供高性能、低子延时的通信服务；（阿里分布式服务框架 Dubbo，默认使用 Netty 作为基础通信组件，还有 RocketMQ 也是使用 Netty 作为通讯的基础。）

2.BIO、NIO和AIO的区别？

回答：

BIO: 一个连接一个线程，客户端有连接请求时服务器端就需要启动一个线程进行处理。

线程开销大。伪异步IO: 将请求连接放入线程池，一对多，但线程还是很宝贵的资源。

NIO: 一个请求一个线程，但客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有I/O请求时才启动一个线程进行处理。

AIO: 一个有效请求一个线程，客户端的I/O请求都是由OS先完成了再通知服务器应用去启动线程进行处理，

BIO是面向流的，**NIO**是面向缓冲区的；**BIO**的各种流是阻塞的。而**NIO**是非阻塞的；**BIO**的**Stream**是单向的，而**NIO**的**channel**是双向的。

NIO的特点：事件驱动模型、单线程处理多任务、非阻塞I/O，I/O读写不再阻塞，而是返回0、基于block的传输比基于流的传输更高效、更高级的IO函数 zero-copy、IO多路复用大大提高了Java网络应用的可伸缩性和实用性。基于 Reactor线程模型。

- **BIO (Blocking I/O):** 同步阻塞 I/O 模式，数据的读取写入必须阻塞在一个线程内等待其完成。在活动连接数不是特别高（小于单机 1000）的情况下，这种模型是比较不错的，可以让每一个连接专注于自己的 I/O 并且编程模型简单，也不用过多考虑系统的过载、限流等问题。线程池本身就是一个天然的漏斗，可以缓冲一些系统处理不了的连接或请求。但是，当面对十万甚至百万级连接的时候，传统的 BIO 模型是无能为力的。因此，我们需要一种更高效的 I/O 处理模型来应对更高的并发量。
- **NIO (Non-blocking/New I/O):** NIO 是一种同步非阻塞的 I/O 模型，在 Java 1.4 中引入了 NIO 框架，对应 `java.nio` 包，提供了 `Channel`, `Selector`, `Buffer` 等抽象。NIO 中的 N 可以理解为 *Non-blocking*, 不单纯是 *New*。它支持面向缓冲的，基于通道的 I/O 操作方法。NIO 提供了与传统 BIO 模型中的 `Socket` 和 `ServerSocket` 相对应的 `SocketChannel` 和 `ServerSocketChannel` 两种不同的套接字通道实现，两种通道都支持阻塞和非阻塞两种模式。阻塞模式使用就像传统中的支持一样，比较简单，但是性能和可靠性都不好；非阻塞模式正好与之相反。对于低负载、低并发的应用程序，可以使用同步阻塞 I/O 来提升开发速率和更好的维护性；对于高负载、高并发的（网络）应用，应使用 NIO 的非阻塞模式来开发
- **AIO (Asynchronous I/O):** AIO 也就是 NIO 2。在 Java 7 中引入了 NIO 的改进版 NIO 2，它是异步非阻塞的 IO 模型。异步 IO 是基于事件和回调机制实现的，也就是应用操作之后会直接返回，不会堵塞在那里，当后台处理完成，操作系统会通知相应的线程进行后续的操作。AIO 是异步 IO 的缩写，虽然 NIO

在网络操作中，提供了非阻塞的方法，但是 NIO 的 IO 行为还是同步的。对于 NIO 来说，我们的业务线程是在 IO 操作准备好时，得到通知，接着就由这个线程自行进行 IO 操作，IO 操作本身是同步的。查阅网上相关资料，我发现就目前来说 AIO 的应用还不是很广泛，Netty 之前也尝试使用过 AIO，不过又放弃了。【guide哥】

3.Netty线程模型知道吗？

回答：Reactor模式是基于事件驱动开发的，核心组成部分包括Reactor和线程池，其中Reactor负责监听和分配事件，线程池负责处理事件，而根据Reactor的数量和线程池的数量，又将Reactor分为三种模型：

- 单线程模型 (单Reactor单线程)
- 多线程模型 (单Reactor多线程)
- 主从多线程模型 (多Reactor多线程)

Netty通过Reactor模型基于多路复用器接收并处理用户请求，内部实现了两个线程池，boss线程池和work线程池，其中boss线程池的线程负责处理请求的accept事件，当接收到accept事件的请求时，把对应的socket封装到一个NioSocketChannel中，并交给work线程池，其中work线程池负责请求的read 和write事件，由对应的Handler处理。

三种模型详细介绍（挑重点记忆，我面试没遇到过让详细说的。）

- **单线程模型：**所有I/O操作都由一个线程完成，即多路复用、事件分发和处理都是在一个Reactor线程上完成的。既要接收客户端的连接请求，向服务端发起连接，又要发送/读取请求或应答/响应消息。一个NIO线程同时处理成百上千的链路，性能上无法支撑，速度慢，若线程进入死循环，整个程序不可用，对于高负载、大并发的应用场景不合适。
- **多线程模型：**有一个NIO线程（Acceptor）只负责监听服务端，接收客户端的TCP连接请求；NIO线程池负责网络IO的操作，即消息的读取、解码、编码和发送；1个NIO线程可以同时处理N条链路，但是1个链路只对应1个NIO线程，这是为了防止发生并发操作问题。但在并发百万客户端连接或需要安全认证时，一个Acceptor线程可能会存在性能不足问题。
- **主从多线程模型：**Acceptor线程用于绑定监听端口，接收客户端连接，将SocketChannel从主线程池的Reactor线程的多路复用器上移除，重新注册到Sub线程池的线程上，用于处理I/O的读写等操作，从而保证mainReactor只负责接入认证、握手等操作；

4.TCP 粘包/拆包的原因及解决方法?

回答：TCP是以流的方式来处理数据，一个完整的包可能会被TCP拆分成多个包进行发送，也可能把小的封装成一个大的数据包发送。

TCP粘包/分包的原因：

应用程序写入的字节大小大于套接字发送缓冲区的大小，会发生拆包现象，而应用程序写入数据小于套接字缓冲区大小，网卡将应用多次写入的数据发送到网络上，这将会发生粘包现象；

进行MSS大小的TCP分段，当TCP报文长度-TCP头部长度>MSS的时候将发生拆包；以太网帧的payload（净荷）大于MTU（1500字节）进行ip分片。

解决方法

消息定长：`FixedLengthFrameDecoder`类包尾增加特殊字符分割：

- 行分隔符类：`LineBasedFrameDecoder`
- 或自定义分隔符类：`DelimiterBasedFrameDecoder`

将消息分为消息头和消息体：`LengthFieldBasedFrameDecoder`类。分为有头部的拆包与粘包、长度字段在前且有头部的拆包与粘包、多扩展头部的拆包与粘包。

5.说说Netty的零拷贝？

回答：

- Netty的接收和发送`ByteBuffer`采用DIRECT BUFFERS，使用堆外直接内存进行Socket读写，不需要进行字节缓冲区的二次拷贝。如果使用传统的堆内存(HEAP BUFFERS)进行Socket读写，JVM会将堆内存Buffer拷贝一份到直接内存中，然后才写入Socket中。相比于堆外直接内存，消息在发送过程中多了一次缓冲区的内存拷贝。
- Netty提供了组合Buffer对象，可以聚合多个`ByteBuffer`对象，用户可以像操作一个Buffer那样方便的对组合Buffer进行操作，避免了传统通过内存拷贝的方式将几个小Buffer合并成一个大的Buffer。
- Netty的文件传输采用了`transferTo`方法，它可以直接将文件缓冲区的数据发送到

更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜



代码界的小白

消息队列高频面试问题

1. 消息队列的使用场景？

回答：消息队列主要有三大使用场景，分别是异步、流量削锋和应用解耦。另外还包含日志和消息通讯。

- 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。

2. 消息队列有什么优缺点

回答：优点可以叙述第一题的作用，都是使用消息队列的好处。

缺点有以下几个：

系统可用性降低：系统引入的外部依赖越多，越容易挂掉，本来你就是A系统调用BCD三个系统的接口就好了，人ABCD四个系统好好的，没啥问题，你偏加个MQ进来，万一MQ挂了咋整？MQ挂了，整套系统崩溃了，你不就完了么。

系统复杂性提高：硬生生加个MQ进来，你怎么保证消息没有重复消费？怎么处理消息丢失的情况？怎么保证消息传递的顺序性？头大头大，问题一大堆，痛苦不已

一致性问题：A系统处理完了直接返回成功了，人都以为你这个请求就成功了；但是问题是，要是BCD三个系统那里，BD两个系统写库成功了，结果C系统写库失败了，咋整？你这数据就不一致了。

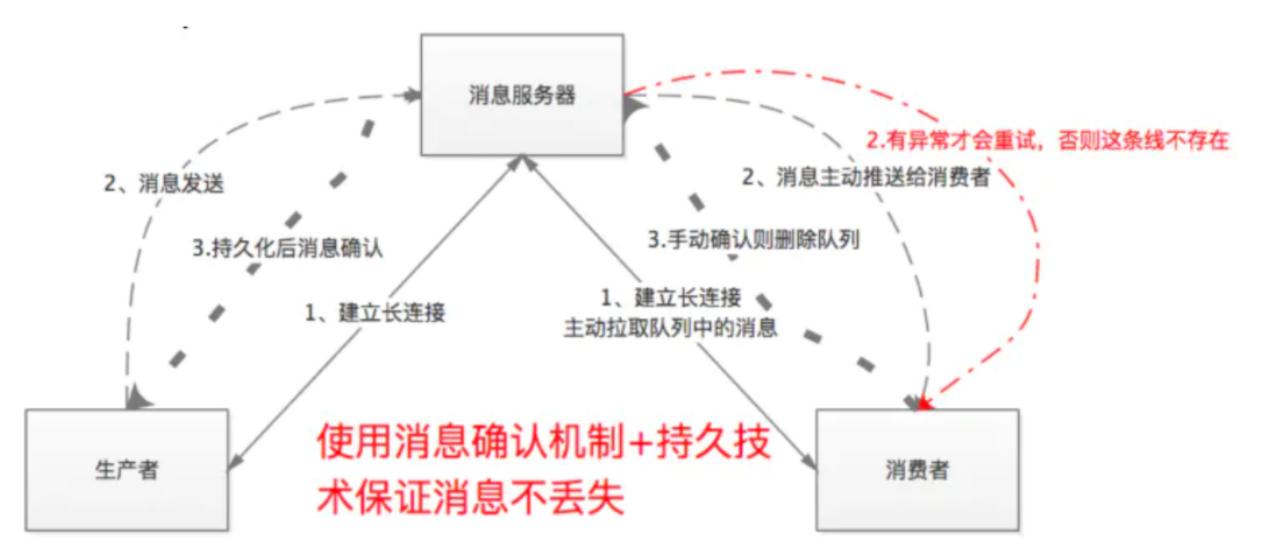
3.Kafka、ActiveMQ、RabbitMQ、RocketMQ 有什么优缺点？

ActiveMQ	RabbitMQ	RocketMQ	Kafka	ZeroMQ	
单机吞吐量	比 RabbitMQ 低	2.6w/s (消息持久化)	11.6w/s	17.3w/s	29w/s
开发语言	Java	Erlang	Java	Scala/Java	C
主要维护者	Apache	Mozilla/Spring	Alibaba	Apache	iMatix创始人已去世
成熟度	成熟	成熟	开源版本不够成熟	比较成熟	只有C、PHP等版本成熟
订阅形式	点对点(p2p)、广播(发布订阅)	提供了4种：direct, topic, Headers 和 fanout。fanout就是广播模式	基于topic/messageTag以及按照消息类型、属性进行正则匹配的发布订阅模式	基于topic以及按照topic进行正则匹配的发布订阅模式	点对点(P2P)
持久化	支持少量堆积	支持少量堆积	支持大量堆积	支持大量堆积	不支持
顺序消息	不支持	不支持	支持	支持	不支持
性能稳定性	好	好	一般	较差	很好
集群方式	支持简单集群模式，比如'主备'，对高级集群模式支持不好。	支持简单集群，'复制'模式，对高级集群模式支持不好。	常用多对'MasterSlave'模式，开源版本需手动切换Slave变成Master	天然的'LeaderSlave'无状态集群，每台服务器既是Master也是Slave	不支持
管理界面	一般	较好	一般	无	无

4.MQ如何保证消息的高可靠性？

【如何处理消息丢失的问题？】

这里以RabbitMQ为例！



从三个方面来保证，分别是生产者、消息中间件和消费者

生产者丢了数据

```
channel.confirmSelect();
.....
if (channel.waitForConfirms()) {
    System.out.println("发送消息成功");
} else {
    System.out.println("发送消息失败");
}
```

消息中间件丢了数据

消息服务器对应的队列、交换机等都持久化，保证数据的不丢失。

消息写入之后会持久化到磁盘，哪怕是 RabbitMQ 自己挂了，恢复之后会自动读取之前存储的数据，一般数据不会丢。除非极其罕见的是，RabbitMQ 还没持久化，自己就挂了，可能导致少量数据丢失，但是这个概率较小。

设置持久化有两个步骤：

- 创建 queue 的时候将其设置为持久化这样就可以保证 RabbitMQ 持久化 queue 的元数据，但是它是不会持久化 queue 里的数据的。
- 第二个是发送消息的时候将消息的 deliveryMode 设置为 2 就是将消息设置为持久化的，此时 RabbitMQ 就会将消息持久化到磁盘上去。

必须要同时设置这两个持久化才行，RabbitMQ 哪怕是挂了，再次重启，也会从磁盘上重启恢复 queue，恢复这个 queue 里的数据。

注意，哪怕是你给 RabbitMQ 开启了持久化机制，也有一种可能，就是这个消息写到了 RabbitMQ 中，但是还没来得及持久化到磁盘上，结果不巧，此时 RabbitMQ 挂了，就会导致内存里的一点点数据丢失。

所以，持久化可以跟生产者那边的 confirm 机制配合起来，只有消息被持久化到磁盘之后，才会通知生产者 ack 了，所以哪怕是在持久化到磁盘之前，RabbitMQ 挂了，数据丢了，生产者收不到 ack，你也是可以自己重发的。

消费端弄丢了数据

RabbitMQ 如果丢失了数据，主要是因为你消费的时候，刚消费到，还没处理，结果进程挂了，比如重启了，那么就尴尬了，RabbitMQ 认为你都消费了，这数据就丢了。

这个时候得用 RabbitMQ 提供的 ack 机制，简单来说，就是你必须关闭 RabbitMQ 的自动 ack，可以通过一个 api 来调用就行，然后每次你自己代码里确保处理完的时候，再在程序里 ack 一把。这样的话，如果你还没处理完，不就没有 ack 了？那 RabbitMQ 就认为你还没处理完，这个时候 RabbitMQ 会把这个消费分配给别的 consumer 去处理，消息是不会丢的。

5.MQ如何保证消息不被重复消费？

【或者说，如何保证消息消费的幂等性？】

消息重复的原因有两个：1.生产时消息重复，2.消费时消息重复。

生产时消息重复

由于生产者发送消息给MQ，在MQ确认的时候出现了网络波动，生产者没有收到确认，实际上MQ已经接收到了消息。这时候生产者就会重新发送一遍这条消息。

生产者中如果消息未被确认，或确认失败，我们可以使用定时任务+（redis/db）来进行消息重试。

消费时消息重复

消费者消费成功后，再给MQ确认的时候出现了网络波动，MQ没有接收到确认，为了保证消息被消费，MQ就会继续给消费者投递之前的消息。这时候消费者就接收到了两条一样的消息。由于重复消息是由于网络原因造成的，因此不可避免重复消息。但是我们需要保证消息的幂等性。

如何保证消息幂等性

让每个消息携带一个全局的唯一ID，即可保证消息的幂等性，具体消费过程为：

1. 消费者获取到消息后先根据id去查询redis/db是否存在该消息。
2. 如果不存在，则正常消费，消费完毕后写入redis/db。
3. 如果存在，则证明消息被消费过，直接丢弃。

6.MQ如何保证消息的顺序性？

解决方案

RabbitMQ

- 拆分为多个queue，每个queue由一个consumer消费；

- 或者就一个queue但是对应一个consumer，然后这个consumer内部用内存队列做排队，然后分发给底层不同的worker来处理

Kafka

- 一个 topic，一个 partition，一个 consumer，内部单线程消费，单线程吞吐量太低，一般不会用这个。
- 写 N 个内存 queue，具有相同 key 的数据都到同一个内存 queue；然后对于 N 个线程，每个线程分别消费一个内存 queue 即可，这样就能保证顺序性。

| 更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



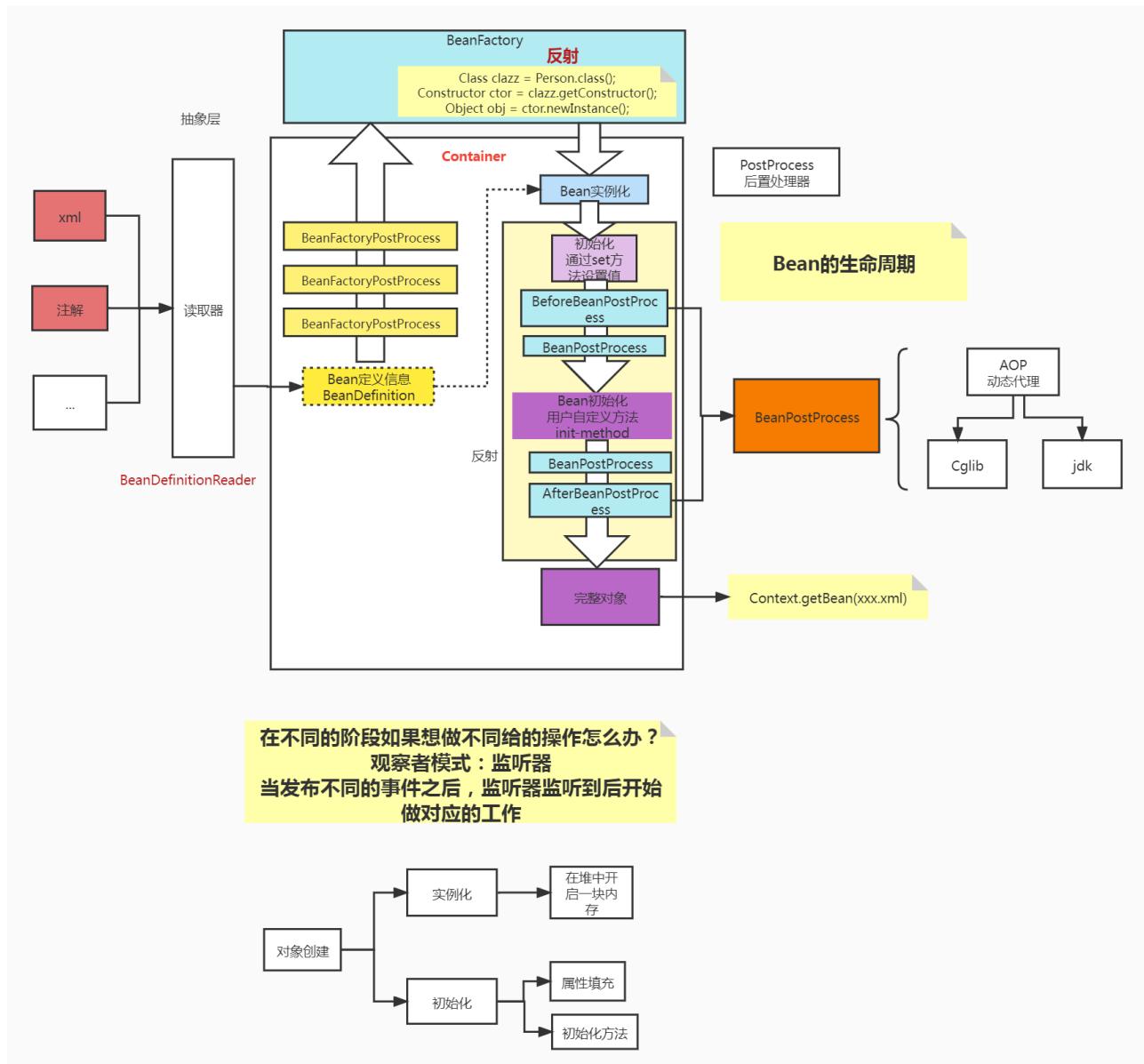
微信搜一搜

Q 代码界的小白

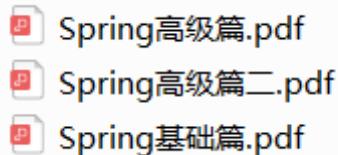
Spring相关框架

Spring高频面试问题

Spring框架是大家学习后续其他框架的基础吧，建议大家好好学习，有时间和精力的可以结合视频去看看源码，对自己的提升还是很不错的。在阅读源码的过程中可以自己画一些流程图之类的，加深自己的理解。下图就是我当时在看源码视频跟着画的图，画完后感觉印象很深，在面试的是就可以跟面试官说你看过Spring某一块的源码，这绝对是一个加分项！



需要Spring思维导图和更多**Spring**面试八股的可以公众号后台回复： **Spring** 资源



1. Spring框架了解吗？说说它的优缺点

回答：Spring 是一种轻量级开发框架，旨在提高开发人员的开发效率以及系统的可维护性。

一般说 Spring 框架指的都是 Spring Framework，它是很多模块的集合，使用这些模块可以很方便地协助我们进行开发。这些模块是：核心容器、数据访问/集成、Web、AOP（面向切面编程）、工具、消息和测试模块。比如：Core Container 中的 Core 组件是 Spring 所有组件的核心，Beans 组件和 Context 组件是实现 IOC 和 依赖注入的基础，AOP 组件用来实现面向切面编程。

Spring 官网列出的 Spring 的 6 个特征：

- 核心技术：依赖注入(DI), AOP, 事件(events), 资源, i18n, 验证, 数据绑定, 类型转换, SpEL。
- 测试：模拟对象, TestContext 框架, Spring MVC 测试, WebTestClient。
- 数据访问：事务, DAO 支持, JDBC, ORM, 编组 XML。
- Web 支持：Spring MVC 和 Spring WebFlux Web 框架。
- 集成：远程处理, JMS, JCA, JMX, 电子邮件, 任务, 调度, 缓存。
- 语言：Kotlin, Groovy, 动态语言。

Spring框架的好处？

- 轻量级： Spring框架是轻量级的，最基础的版本大约只有2MB。
- 控制反转（IOC）：通过控制反转技术，实现了解耦合。对象给出它们的依赖，而不是创建或查找依赖的对象。
- 面向切面（AOP）：Spring支持面向切面的编程，并将应用程序业务逻辑与系统服务分离。
- MVC框架： Spring的WEB框架是一个设计良好的web MVC框架，它为web框架提供了一个很棒的替代方案。
- 容器： Spring包含并管理对象的生命周期和配置。
- 事务管理： Spring提供了一个一致性的事务管理接口，可以收缩到本地事务，也可以扩展到全局事务（JTA）。
- 异常处理： Spring提供了方便的API来将具体技术的异常（由JDBC、Hibernate或JDO抛出）转换为一致的unchecked异常。

Spring框架的缺点？

Spring能够给我们带来很多方便之处，但是同样也存在很多的问题：

使用了大量的反射机制，反射机制非常占用内存。

（一）重量级框架

我们看到 Spring 架构图时会发现 Spring 里面包含有很多其他组件，比如数据访问、MVC、事务管理、面向切点、WebSocket 功能等，因此这么复杂的组件集中到一起就会提高初学者的学习成本。还有一方面随着你的服务越多，那么 Spring 的启动就会变得越慢。

（二）集成复杂

比如我们想要使用 MyBatis 或者 MongoDB 的时候，我们要做很多工作不管使用配置方式也好还是使用注解方式。

（三）配置复杂

在使用 Spring 的时候，我们更多可能是选择 XML 进行配置，但目前这种配置方式已不在流行。

（四）构建和部署复杂

启动 Spring 的 IOC 容器，是完全要依赖于第三方的 web 服务器。自身不能启动的。

2.Spring中有两个重要特性是什么？

回答： Spring中有两个非常重要的特性IOC和AOP，其中AOP是对IOC功能的拓展，

IOC: IOC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。负责创建对象，使用依赖注入（dependency injection, DI）管理它们，将对象集中起来，配置对象，管理对象的整个生命周期。

AOP: AOP模块用于为支持Spring应用程序面向切面的开发。AOP联盟提供了很多支持，这样就确保了Spring和其他AOP框架的共通性。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。

追问1：IOC的好处有哪些？

- IOC或依赖注入最小化应用程序代码量。
- 它使测试应用程序变得容易，因为单元测试中不需要单例或JNDI查找机制。
- 以最小的代价和最少的干扰来促进松耦合。
- IOC容器支持快速实例化和懒加载。

追问2：AOP是怎么实现的？

回答：**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

追问3：JDK代理和Cglib代理的区别？

回答：

1、CGLib所创建的动态代理对象在实际运行时候的性能要比JDK动态代理高(1.6和1.7的时候，CGLib更快;1.8的时候，jdk更快)

2、CGLib在创建对象的时候所花费的时间却比JDK动态代理多

3、singleton的代理对象或者具有实例池的代理，因为无需频繁的创建代理对象，所以比较适合采用CGLib动态代理，反之，则适合用JDK动态代理

4、JDK生成的代理类类型是Proxy(因为继承的是Proxy)，CGLIB生成的代理类类型是Enhancer类型

5、JDK动态代理是面向接口的，CGLib动态代理是通过字节码底层继承代理类来实现（如果被代理类被final关键字所修饰，那么会失败）

6、如果要被代理的对象是个实现类，那么Spring会使用JDK动态代理来完成操作（Spring默认采用JDK动态代理实现机制）；
如果要被代理的对象不是实现类，那么Spring会强制使用CGLib来实现动态代理。

3.Spring中bean有哪些作用域？

- **singleton**: Spring将bean定义的范围限定为每个Spring IOC容器只有一个单实例。
- **prototype**: 单个bean定义有任意数量的对象实例。
- **request**: 作用域为一次http请求，该作用域仅在基于web的Spring ApplicationContext情形下有效。
- **session**: 作用域为HTTP Session，该作用域仅在基于web的Spring ApplicationContext情形下有效。
- **global-session**: 作用域为全局的HTTP session。该作用域也是仅在基于web的Spring ApplicationContext情形下有效。

默认的作用域是**singleton**。

追问1：单例模式是线程安全的吗？

（这个问题阿里二面的时候被问到过）

若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若多个线程同时执行写操作，一般都需要考虑线程同步，否则就可能影响线程安全。（这个大家可以自行去网上搜一下，顺便学习一下。）

4.Spring中有哪些常见的注解？

回答：这个问题可以选择几个常见的说一下就行，下面给大家列出比较全面的，主要说几个自己项目里用到的即可。

- **@Component**: 用于指示类是组件。这些类用于自动注入，并在使用基于注解的配置时配置为bean。
- **@Controller**: 是一种特定类型的组件，用于MVC应用程序，主要与@RequestMapping注解一起使用。
- **@Repository**: 用于表示组件用作存储库和存储/检索/搜索数据的操作。我们可以将此注解应用于DAO实现类。
- **@Service**: 用于指示类是服务层。

- **@Required**:此注解简单地说明作用的bean属性必须在配置时通过bean定义中的显式属性值或通过自动注入填充。如果作用的bean属性未填充，容器将抛出 BeanInitializationException。
- **@ResponseBody**: 用于将对象作为response，通常用于将XML或JSON数据作为 response发送。
- **@PathVariable**:用于将动态值从URI映射到处理方法参数。
- **@Autowired**:对自动注入的位置和方式提供了更细粒度的控制。它可以用于在 setter方法上自动注入bean。就像@Required注解一样，修饰setter方法、构造器、属性或者具有任意名称和/或多个参数的PN方法。
- **@Qualifier**: 当有多个相同类型的bean并且只需要将一个bean自动注入时，@Qualifier注解与@Autowired注释一起使用，通过指定将连接哪个bean来消除歧义。
- **@Scope**:配置Spring bean的作用域。
- **@Configuration**: 表示Spring IOC容器可以将该类用作bean定义的源。
- **@ComponentScan**:应用此注解时，将扫描包下的所有可用类。
- **@Bean**: 对于基于java的配置，用@Bean注解修饰的方法将返回一个在Spring应用程序上下文中注册为Bean的对象。
- 用于配置切面和通知、@Aspect、@Before、@After、@Around、@Pointcut等的AspectJ注解。

5.XML配置和注解之间有什么区别？

注解的优点：

- 所有信息都在一个文件中
- 当类更改了，不用修改xml配置文件

xml配置的优点：

- POJO及其行为之间更清晰地分离
- 当你不知道哪个POJO负责该行为时，更容易找到该POJO

6. Spring支持的事务管理类型？

回答：

- 编程式事务，在代码中硬编码。(不推荐使用)
- 声明式事务，在配置文件中配置（推荐使用）

声明式事务又分为两种：

1. 基于XML的声明式事务

2. 基于注解的声明式事务

追问1：Spring 事务中的隔离级别有哪几种？

TransactionDefinition 接口中定义了五个表示隔离级别的常量：

- **TransactionDefinition.ISOLATION_DEFAULT:** 使用后端数据库默认的隔离级别，Mysql 默认采用的 REPEATABLE_READ 隔离级别 Oracle 默认采用的 READ_COMMITTED 隔离级别。
- **TransactionDefinition.ISOLATION_READ_UNCOMMITTED:** 最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读
- **TransactionDefinition.ISOLATION_READ_COMMITTED:** 允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生
- **TransactionDefinition.ISOLATION_REPEATABLE_READ:** 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生。
- **TransactionDefinition.ISOLATION_SERIALIZABLE:** 最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读。但是这将严重影响程序的性能。通常情况下也不会用到该级别。

更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜



代码界的小白

SpringBoot高频面试问题

1.什么是SpringBoot?

随着动态语言的流行,Java的开发显得格外的笨重:繁多的配置,低下的开发效率,复杂的部署流程以及第三方技术集成难度大.

SpringBoot应运而生.

它使用"习惯优于配置"(项目中存在大量的配置,此外还内置一个习惯性配置,让你无需手动进行配置)的理念让你的项目快速运行起来.使用SpringBoot很容易创建一个独立运行(运行Jar内嵌Servlet容器)准生产级别的基于Spring框架的项目,使用SpringBoot你可以不用或者只需要很少的Spring配置。

追问： SpringBoot的优缺点？

优点：

- 快速构建项目
- 对主流开发框架的无配置集成
- 项目可独立运行,无需外部依赖Servlet容器
- 提供运行时的应用监控
- 极大地提高了开发,部署效率
- 与云计算的天然集成

缺点：

- 书籍文档较少,且不够深入
- 版本迭代速度很快,一些模块改动很大。
- 由于不用自己做配置,报错时很难定位。
- 网上现成的解决方案比较少

2.介绍一下@SpringBootApplication注解，Spring Boot 的自动配置是如何实现的？

@SpringBootApplication: 包含了@Configuration（打开是@Configuration），@EnableAutoConfiguration，@ComponentScan注解。

JavaConfig形式的Spring Ioc容器的配置类使用的那个@Configuration，SpringBoot社区推荐使用基于JavaConfig的配置形式，所以，这里的启动类标注了@Configuration之后，本身其实也是一个IoC容器的配置类。任何一个标注了@Configuration的Java类定义都是一个JavaConfig配置类。任何一个标注了@Bean的方法，其返回值将作为一个bean定义注册到

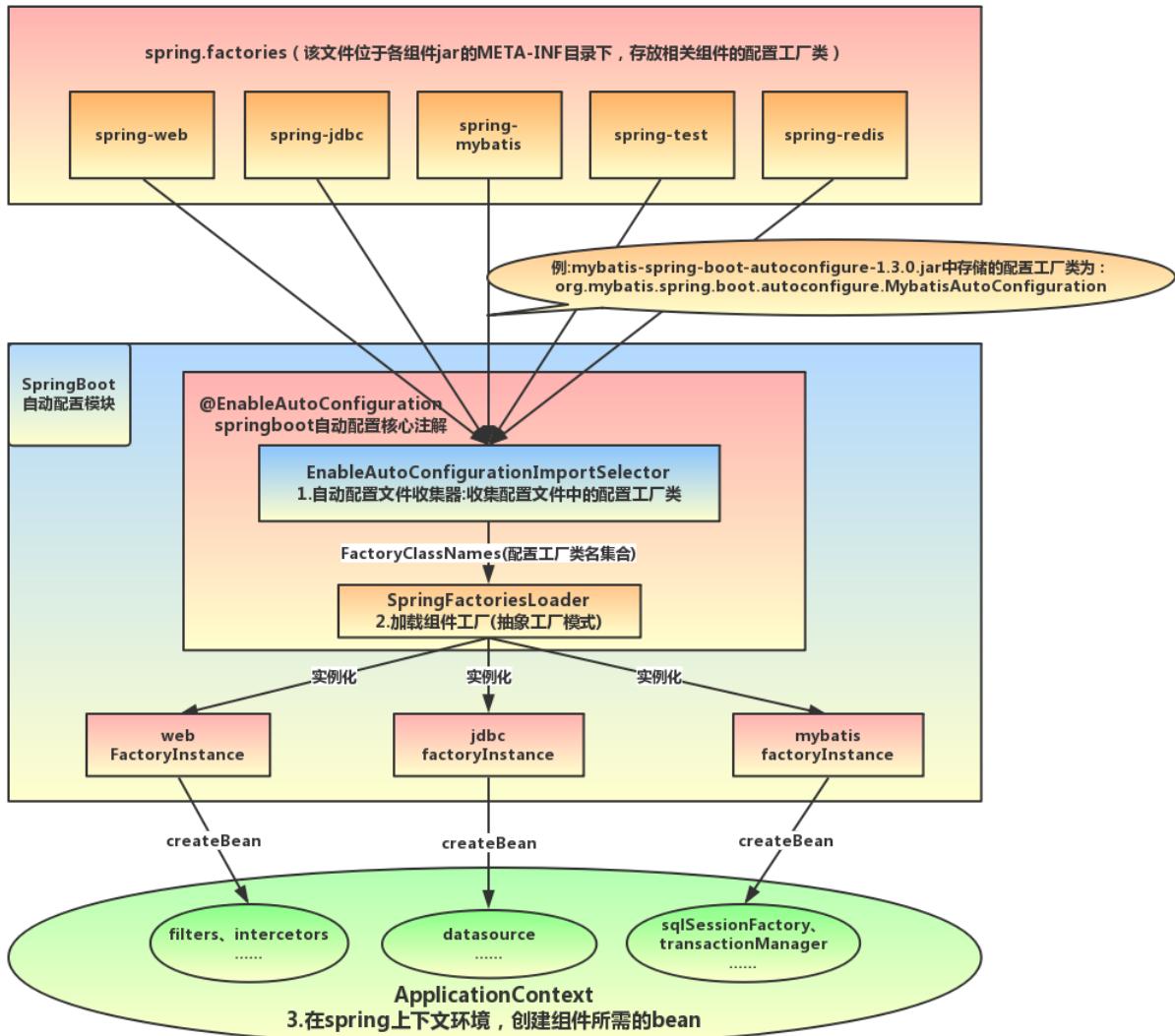
Spring的IoC容器，方法名将默认成该bean定义的id。

@ComponentScan对应XML配置中的元素，@ComponentScan的功能其实就是自动扫描并加载符合条件的组件（比如@Component和@Repository等）或者bean定义，最终将这些bean定义加载到IoC容器中。我们可以通过basePackages等属性来细粒度的定制@ComponentScan自动扫描的范围，如果不指定，则默认Spring框架实现会从声明@ComponentScan所在类的package进行扫描。

英文意思就是自动配置，概括一下就是，借助@Import的帮助，将所有符合自动配置条件的bean定义加载到IoC容器。

（里面最关键的是@Import(EnableAutoConfigurationImportSelector.class)，借助EnableAutoConfigurationImportSelector，@EnableAutoConfiguration可以帮助SpringBoot应用将所有符合条件的@Configuration配置都加载到当前SpringBoot创建并使用的IoC容器。该配置模块的主要使用到了SpringFactoriesLoader。

SpringFactoriesLoader为Spring工厂加载器，该对象提供了loadFactoryNames方法，入参为factoryClass和classLoader即需要传入工厂类名称和对应的类加载器，方法会根据指定的classLoader，加载该类加载器搜索路径下的指定文件，即spring.factories文件，传入的工厂类为接口，而文件中对应的类则是接口的实现类，或最终作为实现类。）



3.什么是SpringBoot Starters?

和自动配置一样， Spring Boot Starter的目的也是简化配置，而Spring Boot Starter解决的是依赖管理配置复杂的问题，有了它，当我需要构建一个Web应用程序时，不必再遍历所有的依赖包，一个一个地添加到项目的依赖管理中，而是只需要一个配置 `spring-boot-starter-web`，同理，如果想引入持久化功能，可以配置 `spring-boot-starter-data-jpa`

4.Spirng Boot 常用的两种配置文件

回答：一个是properties,另一个是yaml。

追问：什么是 YAML? YAML 配置的优势在哪里？

YAML现在可以算是非常流行的一种配置文件格式，无论是前端还是后端，都可以见到 YAML配置。那么 YAML配置和传统的properties配置相比到底有哪些优势呢？

配置有序。在一些特殊场景下，配置有序很关键。

支持数组，数组中的元素可以是基本数据类型也可以是对象。

简洁。

相比properties配置文件， YAML还有一个缺点，就是不支持@PropertySource注解导入自定义的YAML配置。

5.Spring Boot 加载配置文件的优先级了解么？ 3

SpringBoot加载配置文件的优先级由高到低如下：

```
file: ./config/  
file: ./  
classpath: /config/  
classpath: /
```

SpringBoot会从这四个位置全部加载主配置文件，高优先级的配置会覆盖低优先级的配置，并且，互补配置。

说明：file就代表主目录下， classpath代表类路径下，若不懂见下图说明。

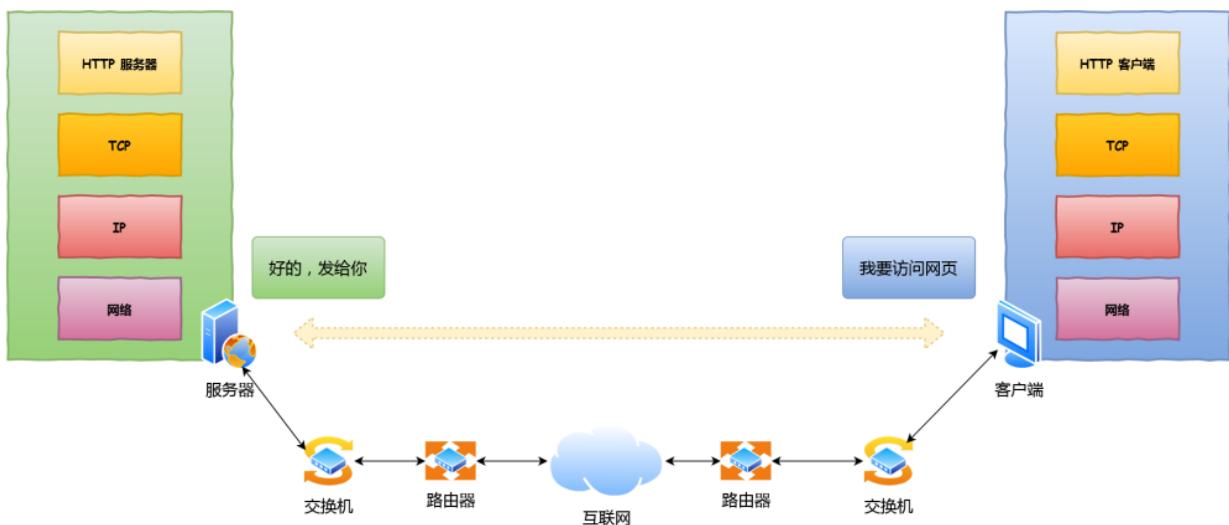


计算机基础知识

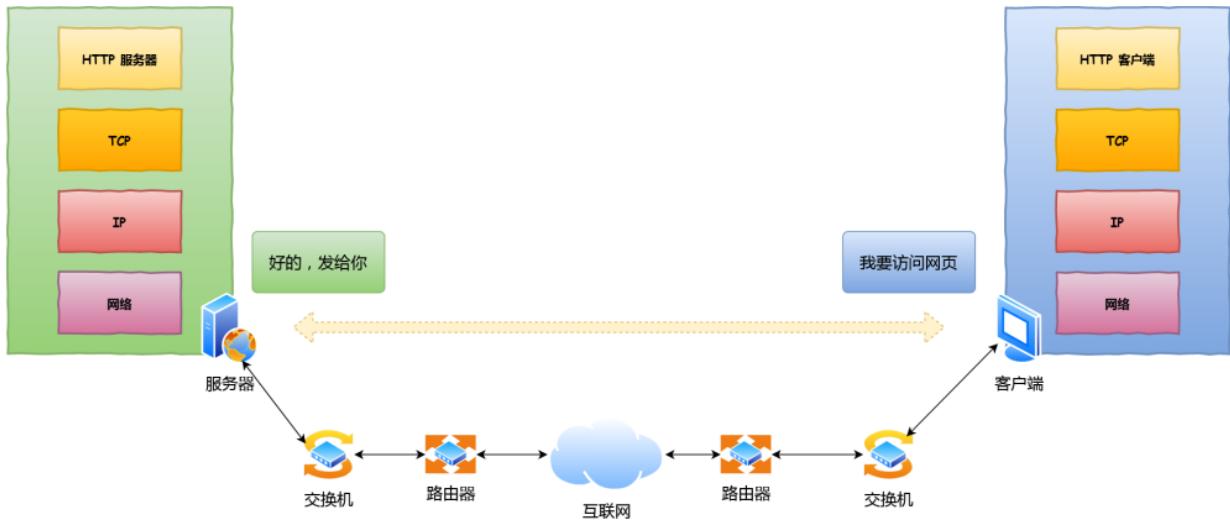
在面试中的计算机基础知识主要包含计算机网络和操作系统，这两个是被问到的频率比较高的，建议大家在背诵八股之前自己先把课本的知识过一遍，另外这两块知识推荐小林哥(公众号：小林coding)的《图解网络》和《图解系统》。

网络超高频面试问题

1. 浏览器键入一个网址都经历了哪些过程？



- 1、解析URL得到发送给web的信息，并生产HTTP请求信息
- 2、查询服务器域名对应的** IP 地址，这个过程中涉及到DNS解析。
- 3、通过 DNS 获取到 IP 后，就可以把 HTTP 的传输□作交给操作系统中的协议栈。
- 4、经过TCP三次握手建立连接进行数据传输。
- 5、TCP 模块在执行连接、收发、断开等各阶段操作时，都需要委托 IP 模块将数据封装成网络包发送给通信对象。
- 6、生成了 IP 头部之后，接下来网络包还需要在 IP 头部的前面加上 MAC 头部。
- 7、后续还会经过网卡、交换机和路由器到对端，然后就是一个反向的过程。



追问：DNS解析过程

- 1、首先会在本地的hosts文件中查找是否有这个网址的映射关系，如果有则直接调用这个IP的映射进行访问；
- 2、如果没有则会去本地DNS解析缓存查找是否有这个网址的映射关系，如果有则返回；
- 3、如果本地DNS解析缓存没有映射关系，首先找本地的TCP/IP设置的DNS服务器，这里我们叫做本地DNS服务器，如果所找的网址在本地DNS的资源范围内则返回解析给主机，此解析具有权威性；
- 4、如果不在本地DNS的资源范围内，但该服务器已经存储了网址的映射关系，那么调用这个IP的映射关系，完成地址解析，此解析不具有权威性。
- 5、如果本地服务器的解析失败并且缓存中没有对应的映射关系。那么就有两种情况
 - 1) 本地DNS服务器开启转发模式，则向上一级请求，若上一级也不能解析就找上一级依次类推，最终把解析返回给本地DNS，本地DNS返回给主机。
 - 2) 如果本地DNS未采用转发模式。就回去找13组根DNS，根DNS收到后就会去判断这个域名由谁来管理将负责此域名的IP返给本地DNS，本地DNS再去找负责的根DNS，如果根DNS不能解析就去找下一级的DNS给本地DNS，然后一直重复这个过程直到找到xx的主机。

2.Ping的工作原理

回答：Ping是基于ICMP协议栈【ICMP可以网上去看看关于他的介绍】，常常使用 **ping** 某一个 IP 地址或者某个域名看下基本连接是否正常；是否有丢包；是否有网络延迟。

*Ping*是一种计算机网络工具，用来测试数据包能否透过IP协议到达特定主机。*ping*的运作原理是向目标主机传出一个ICMP echo@要求数据包，并等待接收echo回应数据包。程序会按时间和成功响应的次数估算丢失数据包率（丢包率）和数据包往返时间（网络时延，*Round-trip delay time*）。——维基百科

ping 命令执行的时候，源主机先会构建一个 **ICMP** 回送请求消息数据包。

ICMP 数据包内包含多个字段，最重要的是两个：

第一个是类型，对于回送请求消息而言该字段为 8；另外一个是序号，主要用于区分连续 ping 的时候发出的多个数据包。

每发出一个请求数据包，序号会自动加 1。为了能够计算往返时间 RTT，它会在报文的数据部分插口发送时间。

在规定的时候间内，源主机如果没有接到 ICMP 的应答包，则说明目标主机不可达；如果接收到 ICMP 回送响应消息，则说明目标主机可达。此时，源主机会检查，用当前时刻减去该数据包最初从源主机上发出的时刻，就是 ICMP 数据包的时间延迟

3.Cookie的作用是什么?和Session有什么区别?

回答：

- session 在服务器端，cookie 在客户端（浏览器）
- session 默认被存储在服务器的一个文件里（不是内存）
- session 的运行依赖 session id，而 session id 是存在 cookie 中的，也就是说，如果浏览器禁用了 cookie，同时 session 也会失效（但是可以通过其它方式实现，比如在 url 中传递 session_id）
- session 可以放在文件、数据库、或内存中都可以。
- 用户验证这种场合一般会用 session

Cookie 和 *Session*都是用来跟踪浏览器用户身份的会话方式，但是两者的应用场景不太一样。

Cookie 一般用来保存用户信息 比如①我们在 *Cookie* 中保存已经登录过得用户信息，下次访问网站的时候页面可以自动帮你登录的一些基本信息给填了；②一般的网站都会有保持登录也就是说下次你再访问网站的时候就不需要重新登录了，这是因为用户登录的时候我们可以存放了一个 *Token* 在 *Cookie* 中，下次登录的时候只需要根据 *Token* 值来查找用户即可(为了安全考虑，重新登录一般要将 *Token* 重写)；③登录一次网站后访问网站其他页面不需要重新登录。**Session** 的主要作用就是通过服务端记录用户的状态。典型的场景是购物车，当你要添加商品到购物车的时

候，系统不知道是哪个用户操作的，因为 *HTTP* 协议是无状态的。服务端给特定的用户创建特定的 *Session* 之后就可以标识这个用户并且跟踪这个用户了。

Cookie 数据保存在客户端(浏览器端)，*Session* 数据保存在服务器端。

Cookie 存储在客户端中，而 *Session* 存储在服务器上，相对来说 *Session* 安全性更高。如果要在 *Cookie* 中存储一些敏感信息，不要直接写入 *Cookie* 中，最好能将 *Cookie* 信息加密然后使用到的时候再去服务器端解密。【摘自 *Guide* 哥】

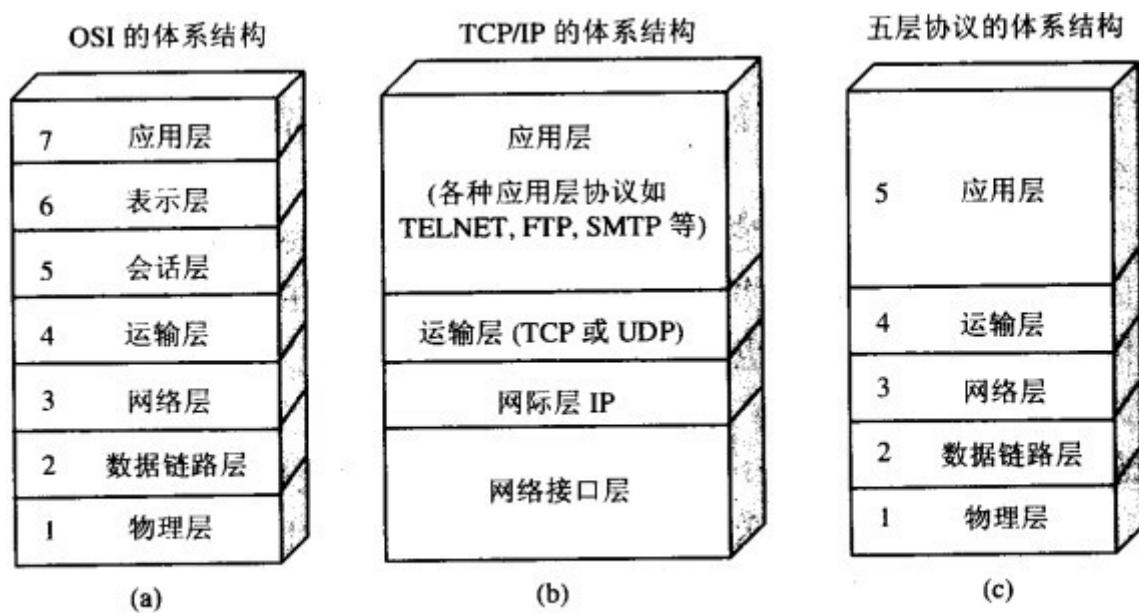
计算机网络-HTTP



这里给大家准备了电子版，需要的同学可以后台回复：图解网络

1.OSI与TCP/IP各层的结构与功能,都有哪些协议?

回答：OSI共七层协议，分别是物理层、数据链路层、网络层、运输层、会话层、表示层和应用层。



计算机网络体系结构: (a) OSI 的七层协议; (b) TCP/IP 的四层协议; (c) 五层协议

追问1: HTTP属于那一层? TCP/UDP属于哪一层? IP属于哪一层?

回答:

IP: 网络层

TCP/UDP: 传输层

HTTP、RTSP、FTP: 应用层协议

2.HTTP是什么? HTTP常见的状态码, 有哪些? GET 和 POST 的区别?

回答: HTTP是超文本传输协议(HTTP 是在计算机世界的协议。它使计算机能够理解的语言确立了一种计算机之间交流通信的规范(两个以上的参与者), 以及相关的各种控制和错误处理方式(行为约定和规范)。)

HTTP常见状态码

五大类 HTTP 状态码

		具体含义	常见的状态码
1××	提示信息，表示目前是协议处理的中间状态，还需要后续的操作；		
2××	成功，报文已经收到并被正确处理；	200、204、206	
3××	重定向，资源位置发生变动，需要客户端重新发送请求；	301、302、304	
4××	客户端错误，请求报文有误，服务器无法处理；	400、403、404	
5××	服务器错误，服务器在处理请求时内部发生了错误。	500、501、502、503	

100 CONTINUE	继续。客户端应继续其请求
101 Switching Protocols	切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议
200 OK	请求成功。一般用于GET与POST请求
201 Created	已创建。成功请求并创建了新的资源
202 Accepted	已接受。已经接受请求，但未处理完成
203 Non-Authoritative Information	非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本
204 No Content	无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
205 Reset Content	重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
206 Partial Content	部分内容。服务器成功处理了部分GET请求
300 Multiple Choices	多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
301 Moved Permanently	永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替
302 Found	临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI

100 CONTINUE	继续。客户端应继续其请求
303 See Other	查看其它地址。与301类似。使用GET和POST请求查看
304 Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
305 Use Proxy	使用代理。所请求的资源必须通过代理访问
306 Unused	已经被废弃的HTTP状态码
307 Temporary Redirect	临时重定向。与302类似。使用GET请求重定向
400 Bad Request	客户端请求的语法错误，服务器无法理解
401 Unauthorized	请求要求用户的身份认证
402 Payment Required	保留，将来使用
403 Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求
404 Not Found	服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置“您所请求的资源无法找到”的个性页面
405 Method Not Allowed	客户端请求中的方法被禁止
406 Not Acceptable	服务器无法根据客户端请求的内容特性完成请求
407 Proxy Authentication Required	请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权
408 Request Time-out	服务器等待客户端发送的请求时间过长，超时
409 Conflict	服务器完成客户端的PUT请求时可能返回此代码，服务器处理请求时发生了冲突
410 Gone	客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置
411 Length Required	服务器无法处理客户端发送的不带Content-Length的请求信息
412 Precondition Failed	客户端请求信息的先决条件错误

100 CONTINUE	继续。客户端应继续其请求
413 Request Entity Too Large	由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息
414 Request-URI Too Large	请求的URI过长（URI通常为网址），服务器无法处理
415 Unsupported Media Type	服务器无法处理请求附带的媒体格式
416 Requested range not satisfiable	客户端请求的范围无效
417 Expectation Failed	服务器无法满足Expect的请求头信息
500 Internal Server Error	服务器内部错误，无法完成请求
501 Not Implemented	服务器不支持请求的功能，无法完成请求
502 Bad Gateway	作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到一个无效的响应
503 Service Unavailable	由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中
504 Gateway Timeout	充当网关或代理的服务器，未及时从远端服务器获取请求
505 HTTP Version not supported	服务器不支持请求的HTTP协议的版本，无法完成处理

GET和POST的区别：

1. get是获取数据的，而post是提交数据的。
2. GET 用于获取信息，是无副作用的，是幂等的，且可缓存，而POST 用于修改服务器上的数据，有副作用，非幂等，不可缓存。

3.说一下HTTP的优缺点？

回答：

优点：简单、灵活和易于拓展、应用广泛和跨平台。

缺点：无状态、明文传输、不安全

追问1：HTTP1.1相对于HTTP1.0的优化是什么？

HTTP1.0最早在网页中使用是在1996年，那个时候只是使用一些较为简单的网页上和网络请求上，而HTTP1.1则在1999年才开始广泛应用于现在的各大浏览器网络请求中，同时HTTP1.1也是当前使用最为广泛的HTTP协议。主要区别主要体现在：

1. 长连接：在**HTTP/1.0**中，默认使用的是短连接，也就是说每次请求都要重新建立一次连接。HTTP是基于TCP/IP协议的，每一次建立或者断开连接都需要三次握手四次挥手的开销，如果每次请求都要这样的话，开销会比较大。因此最好能维持一个长连接，可以用个长连接来发多个请求。**HTTP 1.1**起，默认使用长连接，默认开启**Connection: keep-alive**。**HTTP/1.1**的持续连接有非流水线方式和流水线方式。流水线方式是客户在收到HTTP的响应报文之前就能接着发送新的请求报文。与之相对应的非流水线方式是客户在收到前一个响应后才能发送下一个请求。
2. 错误状态响应码：在**HTTP1.1**中新增了24个错误状态响应码，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。
3. 缓存处理：在**HTTP1.0**中主要使用header里的**If-Modified-Since**,**Expires**来做为缓存判断的标准，**HTTP1.1**则引入了更多的缓存控制策略例如**Entity tag**, **If-Unmodified-Since**, **If-Match**, **If-None-Match**等更多可供选择的缓存头来控制缓存策略。
4. 带宽优化及网络连接的使用：**HTTP1.0**中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，**HTTP1.1**则在请求头引入了**range**头域，它允许只请求资源的某个部分，即返回码是206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。

追问2：HTTP与HTTPS的区别？

1. HTTP是超文本传输协议，信息是明文传输，存在安全隐患的问题。HTTPS则解决HTTP不安全的缺陷，在TCP和HTTP网络层之间加了SSL/TLS安全协议，使得报文能够加密传输。
2. HTTP连接建立相对简单，TCP三次握手之后便可进行HTTP的报文传输。HTTPS在TCP三次握手之后，还需SSL/TLS的握手过程，才可进行加密报文传输。
3. HTTP的端口号是80，HTTPS的端口号是443。
4. HTTPS协议需要向CA（证书权威机构）申请数字证书，来保证服务器的身份是可信的。

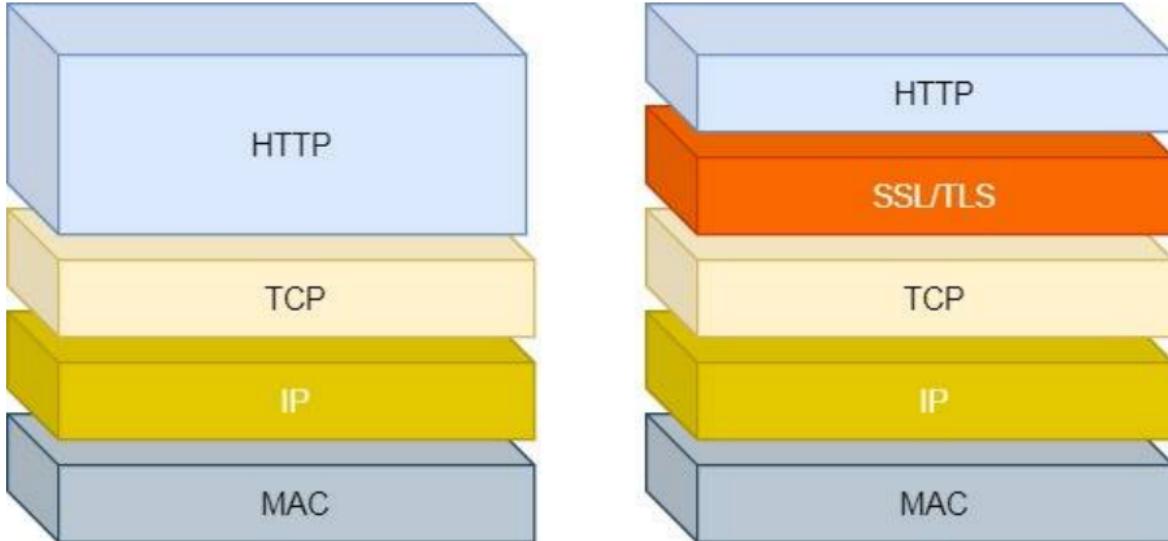
追问3：HTTPS 解决了 HTTP 的哪些问题？

HTTP 由于是明文传输，所以安全上存在以下三个风险：

窃听风险，如通信链路上可以获取通信内容，号容易没。

篡改风险，如强制植入垃圾广告，视觉污染，用户眼容易瞎。

冒充风险，冒充淘宝网站，用户钱容易没。



HTTPS 在 HTTP 与 TCP 层之间加了 SSL/TLS 协议，可以很好的解决了上述的风险：

- 信息加密：交互信息无法被窃取，但你的号会因为「自身忘记」账号而没。
- 校验机制：无法篡改通信内容，篡改了就不能正常显示，但百度「竞价排名」依然可以搜索垃圾广告。
- 身份证书：证明淘宝是真的淘宝网，但你的钱还是会因为「剁手」而没。

SSL/TLS 协议是能保证通信是安全的。

HTTPS 是如何解决上面的三个风险的？

- 混合加密的方式实现信息的机密性，解决了窃听的风险。
- 摘要算法的方式来实现完整性，它能够为数据生成独一无二的「指纹」，指纹用于校验数据的完整性，解决了篡改的风险。
- 将服务器公钥放入到数字证书中，解决了冒充的风险

4. 说一下HTTP1.1、HTTP2、HTTP3的演变？

HTTP1.1存在的性能瓶颈：

请求/响应头部未经压缩就发送，首部信息越多延迟越大，只能压缩Body部分

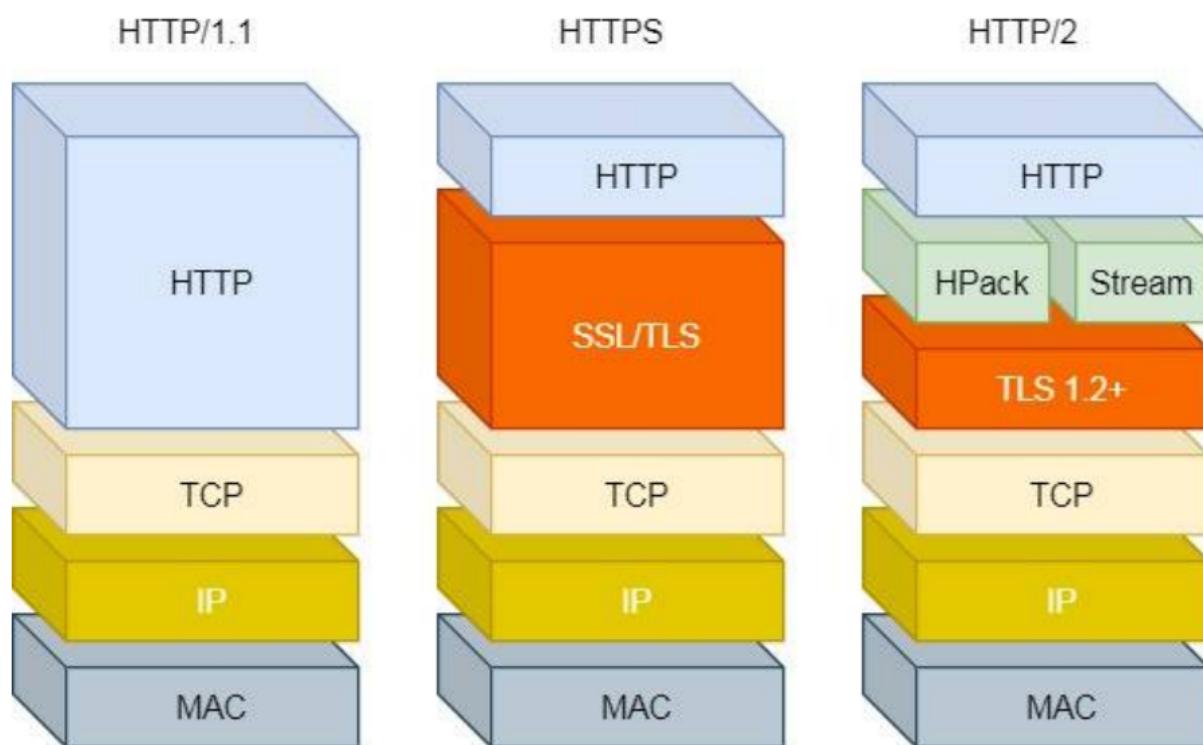
首部过长，发送时造成浪费。

服务器是按请求的顺序响应的，如果服务器响应慢，会招致客户端一直请求不到数据，也就是队头阻塞；

没有请求优先级控制；

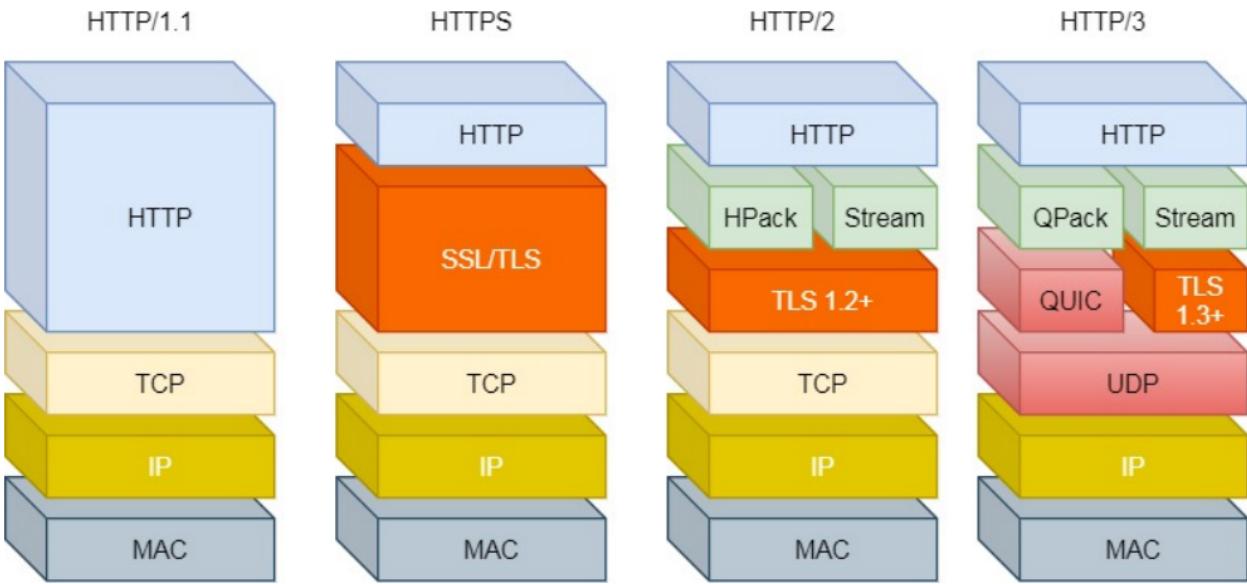
请求只能从客户端开始，服务器只能被动响应。

HTTP2改进了几个地方，分别是头部压缩、二进制格式、数据流、多路复用和服务器推送



HTTP/2 主要的问题在于，多个 HTTP 请求在复用一个 TCP 连接，下层的 TCP 协议是不知道有多少个 HTTP 请求的。所以一旦发生了丢包现象，就会触发 TCP 的重传机制，这样在一个 TCP 连接中的所有的 **HTTP** 请求都必须等待这个丢了的包被重传回来。

- HTTP/1.1 中的管道（pipeline）传输中如果有 1 个请求阻塞了，那么队列后请求也统统被阻塞住了
- HTTP/2 多个请求复一个TCP连接，一旦发生丢包，就会阻塞住所有的 HTTP 请求。这都是基于 TCP 传输层的问题，所以 **HTTP/3** 把 **HTTP** 下层的 **TCP** 协议改成了 **UDP**。



UDP发生是不管顺序，也不管丢包的，所以不会出现HTTP/1.1的队头阻塞和HTTP/2的丢包全部重传问题。

UDP是不可靠传输的，但基于UDP的**QUIC**协议可以实现类似TCP的可靠性传输。

- QUIC有自己的□套机制可以保证传输的可靠性的。当某个流丢包时，只会阻塞这个流，其他流不会受到影响。
- TLS3升级成了最新的1.3版本，头部压缩算法也升级成了QPack。
- HTTPS要建立一个连接，要花费6次交互，先是建立三次握手，然后是TLS/1.3的三次握手。QUIC直接把以往的TCP和TLS/1.3的6次交互合并成了3次，减少了交互次数。

计算机网络-TCP和UDP

1. TCP和UDP的区别是什么？

回答:TCP 和 UDP都是属于运输层的

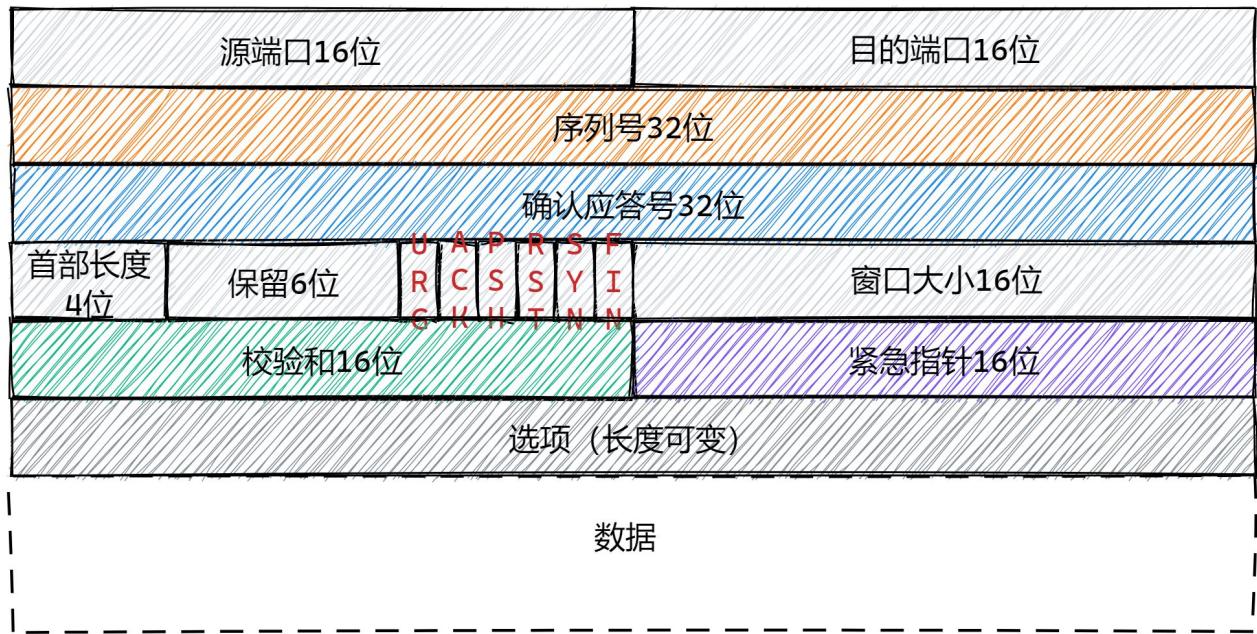
1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接

2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付

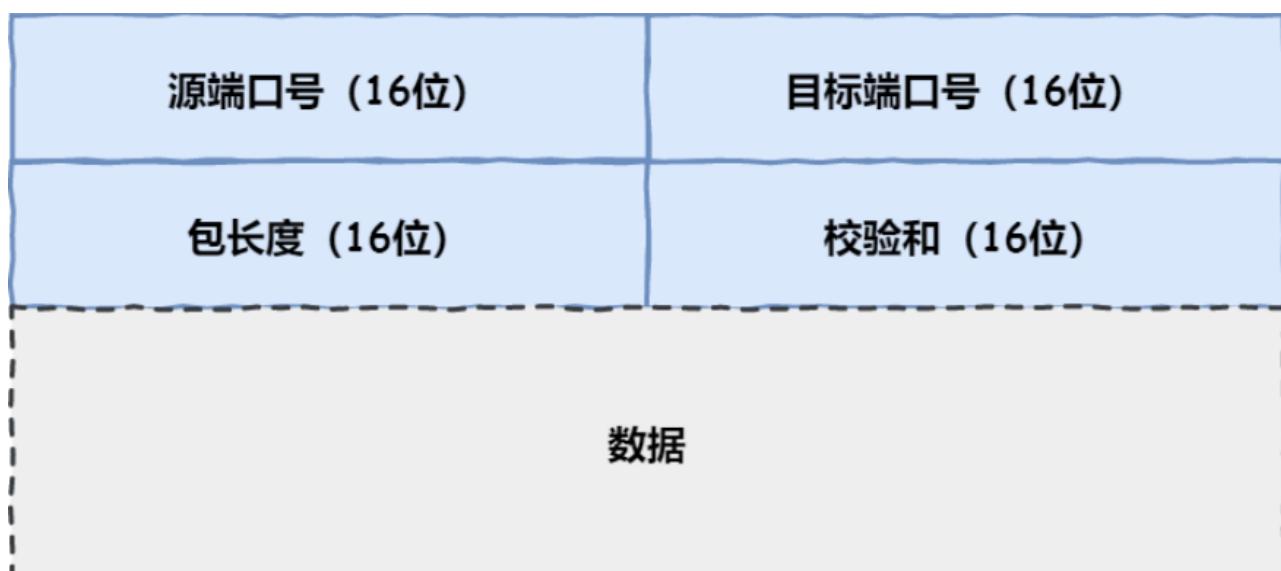
3、TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的；UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）

4、每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信

5、TCP首部开销20字节;UDP的首部开销小，只有8个字节



<https://zhidao.baidu.com/question/45210157.html>



6、TCP的逻辑通信信道是全双工的可靠信道， UDP则是不可靠信道

类型	特点			性能		应用场景	首部字节
	是否面向连接	传输可靠性	传输形式	传输效率	所需资源		
TCP	面向连接	可靠	字节流	慢	多	要求通信数据可靠 (如文件传输、邮件传输)	20-60
UDP	无连接	不可靠	数据报文段	快	少	要求通信速度高 (如域名转换)	8个字节 (由4个字段组成)

追问1：TCP和UDP的使用场景？

某些情况下 UDP 确是一种最有效的工作方式（一般用于即时通信），比如：QQ 语音、QQ 视频、直播等等。

TCP 一般用于文件传输、发送和接收邮件、远程登录等场景。

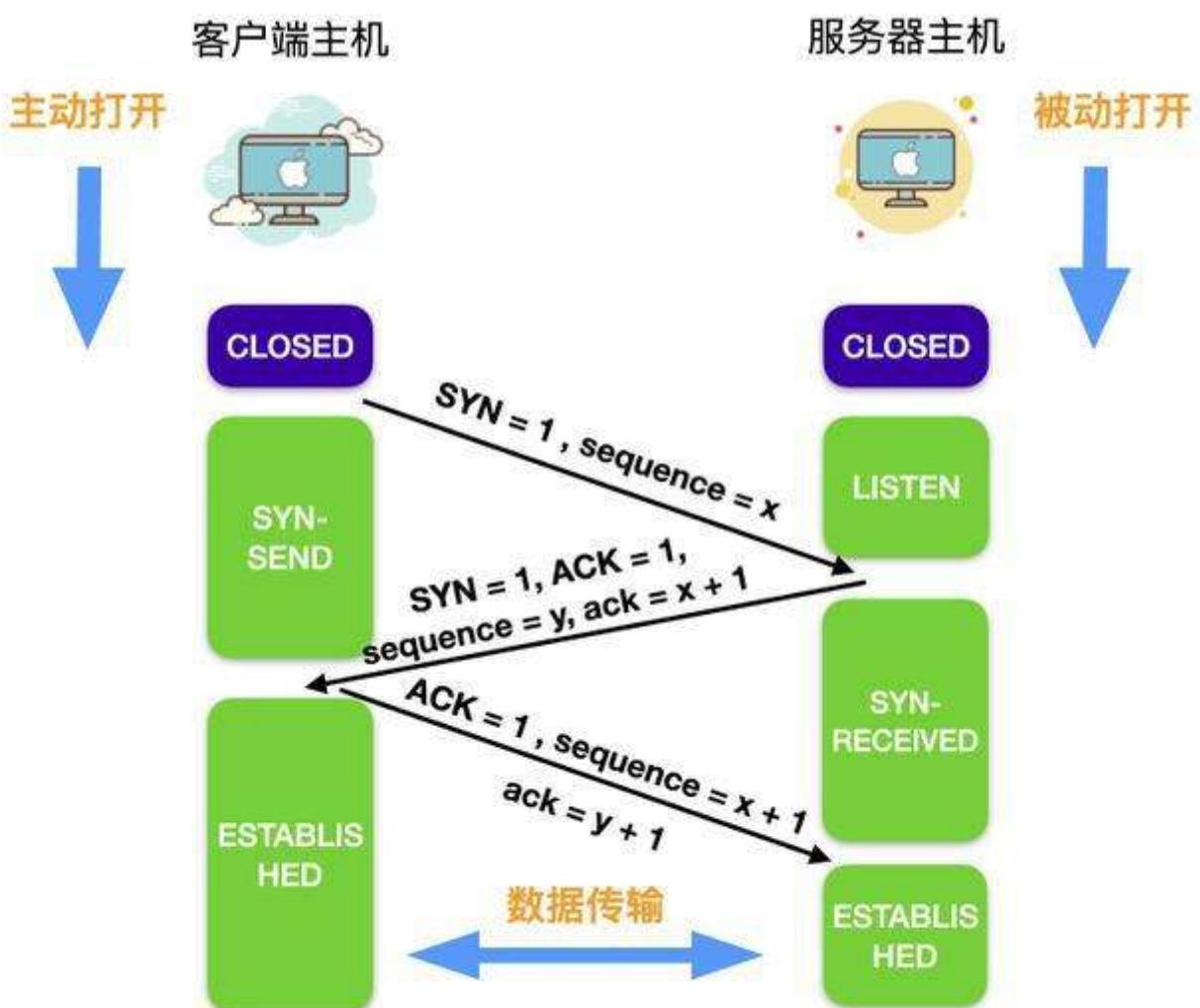
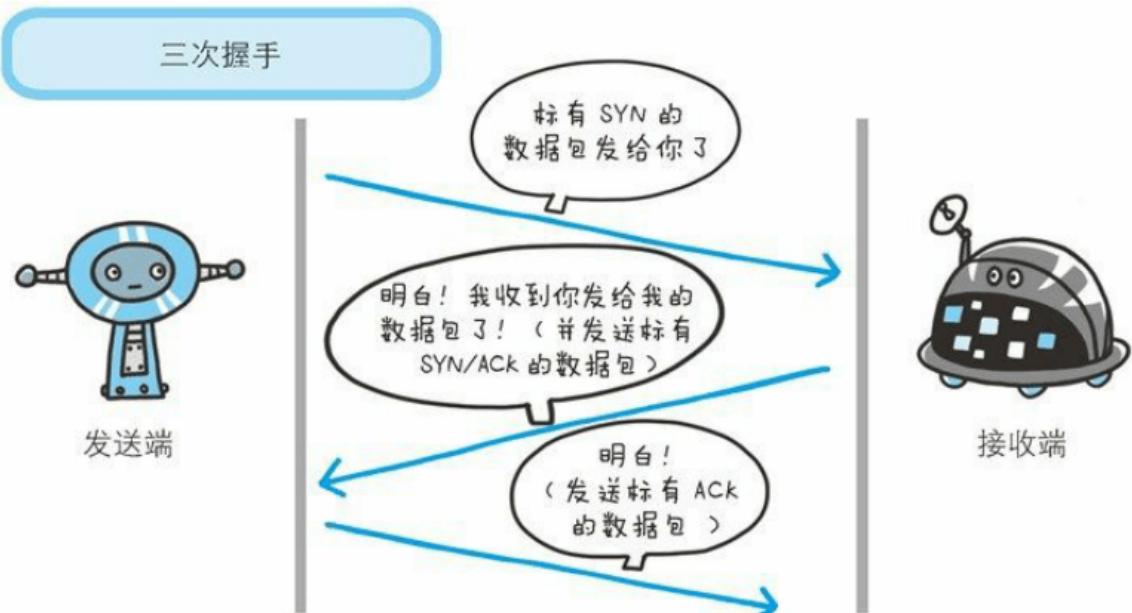
2. TCP是如何保证可靠传输的？

1. 应用数据被分割成 TCP 认为最适合发送的数据块。
2. TCP 给发送的每一个包进行编号，接收方对数据包进行排序，把有序数据传送给应用层。
3. 校验和：TCP 将保持它首部和数据的检验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错，TCP 将丢弃这个报文段和不确认收到此报文段。
4. TCP 的接收端会丢弃重复的数据。
5. 流量控制：TCP 连接的每一方都有固定大小的缓冲空间，TCP 的接收端只允许发送端发送接收端缓冲区能接纳的数据。当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。TCP 使用的流量控制协议是可变大小的滑动窗口协议。（TCP 利用滑动窗口实现流量控制）
6. 拥塞控制：当网络拥塞时，减少数据的发送。
7. ARQ 协议：也是为了实现可靠传输的，它的基本原理就是每发完一个分组就停止发送，等待对方确认。在收到确认后再发下一个分组。
8. 超时重传：当 TCP 发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。

2.TCP的三次握手和四次挥手

回答：这里可以根据图解HTTP上的图例进行总结

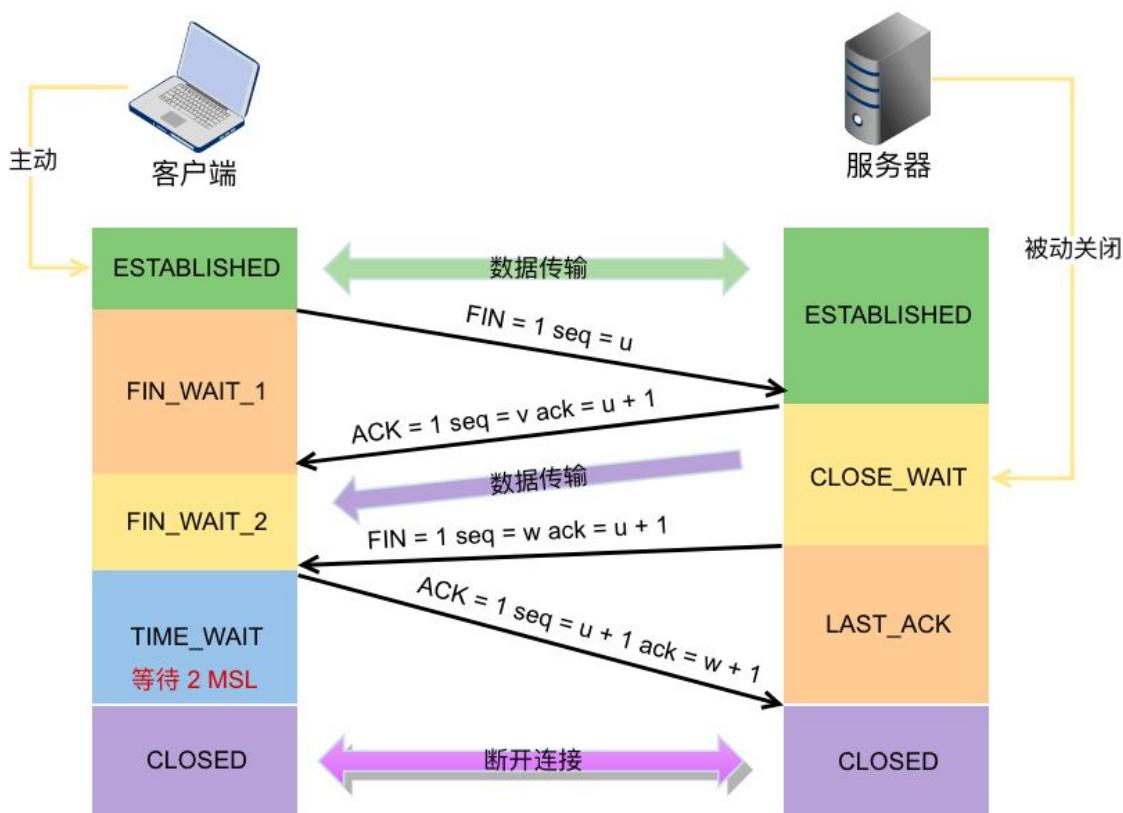
三次握手



- 开始，客户端和服务端都处于 CLOSED 状态。先是服务端主动监听某个端口，处于 LISTEN 状态
- 客户端会随机初始化序号（client_isn），将此序号置于 TCP 首部的「序号」字段中，同时把 SYN 标志位置为 1，表示 SYN 报文。接着把第一个 SYN 报文发送给服务端，表示向服务端发起连接，该报文不包含应用层数据，之后客户端处于 SYN-SENT 状态。

- 服务端收到客户端的 SYN 报文后，首先服务端也随机初始化自己的序号（`server_isn`），将此序号填入 TCP 首部的「序号」字段中，其次把 TCP 首部的「确认应答号」字段填入 `client_isn + 1`，接着把 SYN 和 ACK 标志位置为 1。最后把该报文发给客户端，该报文也不包含应用层数据，之后服务端处于 SYN-RCVD 状态。
- 客户端收到服务端报文后，还要向服务端回应最后一个应答报文，首先该应答报文 TCP 首部 ACK 标志位置为 1，其次「确认应答号」字段填入 `server_isn + 1`，最后把报文发送给服务端，这次报文可以携带客户到服务器的数据，之后客户端处于 ESTABLISHED 状态。
- 服务器收到客户端的应答报文后，也进入 ESTABLISHED 状态。

四次挥手



- 客户端打算关闭连接，此时会发送一个 TCP 首部 FIN 标志位被置为 1 的报文，也即 FIN 报文，之后客户端进 FIN_WAIT_1 状态。
- 服务端收到该报文后，就向客户端发送 ACK 应答报文，接着服务端进入 CLOSED_WAIT 状态。
- 客户端收到服务端的 ACK 应答报文后，之后进入 FIN_WAIT_2 状态。
- 等待服务端处理完数据后，也向客户端发送 FIN 报文，之后服务端进入 LAST_ACK 状态。
- 客户端收到服务端的 FIN 报文后，回一个 ACK 应答报文，之后进入 TIME_WAIT 状态
- 服务器收到了 ACK 应答报文后，就进入了 CLOSED 状态，至此服务端已经完成连接的关闭。

- 客户端在经过 2MSL 一段时间后，自动进入 CLOSED 状态，至此客户端也完成连接的关闭。

【注意是：主动关闭连接的，才有 **TIME_WAIT** 状态。】

追问1：为什么是三次？不是两次、四次？

回答：“因为三次才能保证双方具有接收和发送的能力。”，这样回答或许有些片面，可以在回答以下几点。

TCP 建立连接时，通过三次握手能防止历史连接的建立，能减少双方不必要的资源开销，能帮助双方同步初始化序列号。序列号能够保证数据包不重複、不丢弃和按序传输。

不使用两次和四次的原因：

两次：无法防止历史连接的建立，会造成双方资源的浪费，也无法可靠的同步双方序列号；

四次：三次握手就已经理论上最少可靠连接建立，所以不需要使用更多的通信次数。

追问2：为什么 **TIME_WAIT** 等待的时间是 **2MSL**？

MSL 是 Maximum Segment Lifetime, **报文最大生存时间**, 它是任何报文在网络上存在的最长时间, 超过这个时间报文将被丢弃。因为 TCP 报文基于是 IP 协议的, 而 IP 头中有一个 **TTL** 字段, 是 IP 数据报可以经过的最大路由数, 每经过一个处理他的路由器此值就减 1, 当此值为 0 则数据报将被丢弃, 同时发送 ICMP 报文通知源主机。

MSL 与 TTL 的区别: MSL 的单位是时间, 而 TTL 是经过路由跳数。所以 **MSL 应该要大于等于 TTL 消耗为 0 的时间**, 以确保报文已被自然消亡。

TIME_WAIT 等待 2 倍的 MSL, 比较合理的解释是: 网络中可能存在来自发送方的数据包, 当这些发送方的数据包被接收方处理后又会向对方发送响应, 所以**一来一回需要等待 2 倍的时间**。

比如如果被动关闭方没有收到断开连接的最后的 ACK 报文, 就会触发超时重发 Fin 报文, 另一方接收到 FIN 后, 会重发 ACK 给被动关闭方, 一来一去正好 2 个 MSL。

2MSL 的时间是从**客户端接收到 FIN 后发送 ACK 开始计时的**。如果在 TIME-WAIT 时间内, 因为客户端的 ACK 没有传输到服务端, 客户端又接收到了服务端重发的 FIN 报文, 那么 **2MSL 时间将重新计时**。

在 Linux 系统里 **2MSL** 默认是 **60** 秒, 那么一个 **MSL** 也就是 **30** 秒。**Linux 系统停留在 TIME_WAIT 的时间为固定的 60 秒**。

其定义在 Linux 内核代码里的名称为 **TCP_TIMEWAIT_LEN**:

```
#define TCP_TIMEWAIT_LEN (60*HZ) /* how long to wait to destroy TIME-WAIT
                                state, about 60 seconds */
```

如果要修改 TIME_WAIT 的时间长度, 只能修改 Linux 内核代码里 **TCP_TIMEWAIT_LEN** 的值, 并重新编译 Linux 内核。

追问3: 为什么需要 **TIME_WAIT** 状态?

主要是两个原因:

防止具有相同「四元组」的「旧」数据包被收到;

保证「被动关闭连接」的一方能被正确的关闭, 即保证最后的 ACK 能让被动关闭口接收, 从而帮助其正常关闭;

追问4: **TIME_WAIT** 过多有什么危害?

过多的 TIME-WAIT 状态主要的危害有两种:

第一是内存资源占用;

第二是对端口资源的占用, 一个 TCP 连接口少消耗口个本地端口;

3.拥塞控制有哪些控制算法？

回答：拥塞控制主要是四个算法：

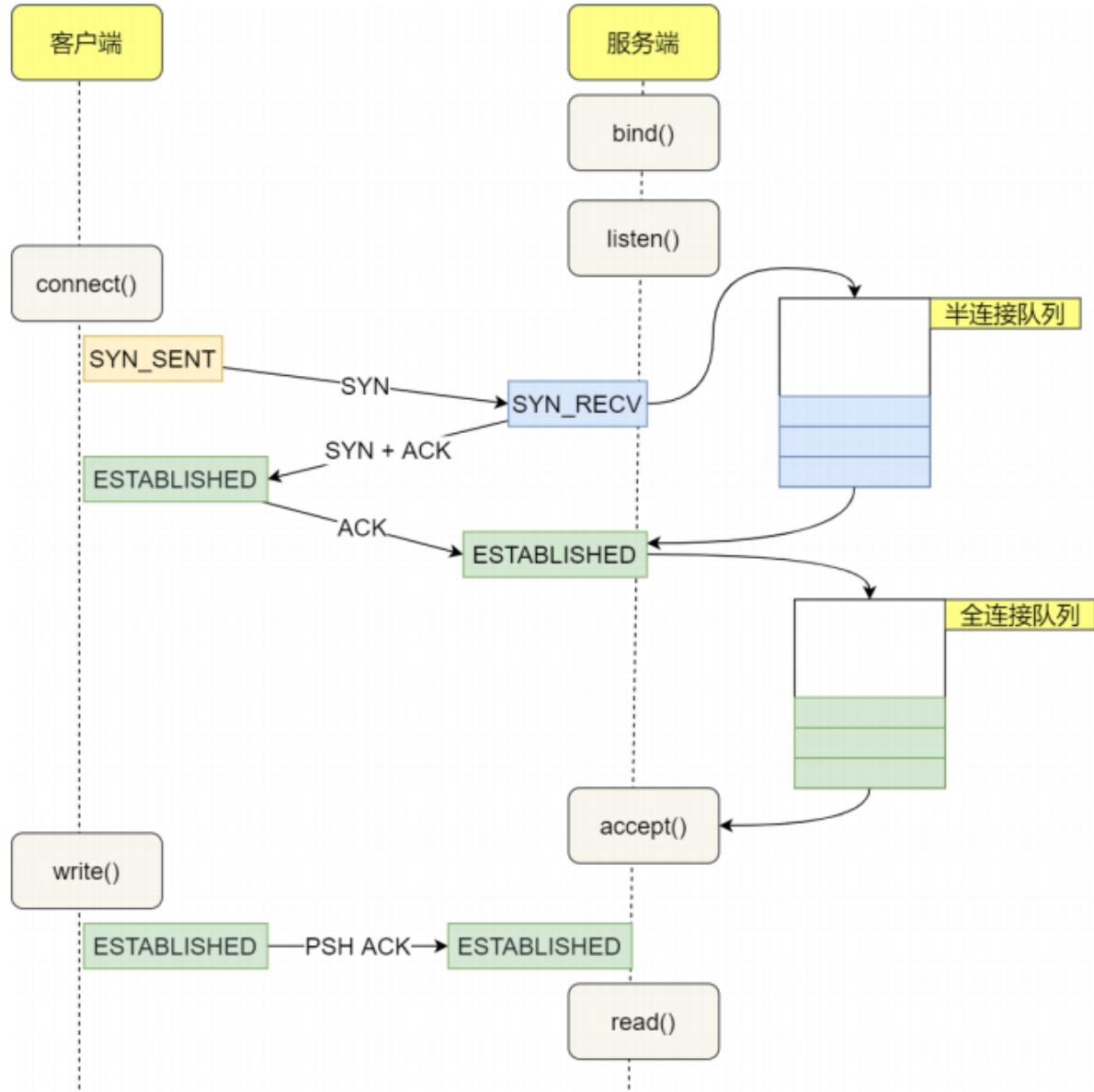
- 慢启动
- 拥塞避免
- 拥塞发生
- 快速恢复

4.TCP 半连接队列和全连接队列

在 TCP 三次握手的时候，Linux 内核会维护两个队列，分别是：

- 半连接队列，也称 SYN 队列；
- 全连接队列，也称 accept 队列

服务端收到客户端发起的 SYN 请求后，内核会把该连接存储到半连接队列，并向客户端响应 SYN+ACK，接着客户端会返回 ACK，服务端收到第三次握手的 ACK 后，内核会把连接从半连接队列移除，然后创建新的完全的连接，并将其添加到 **accept** 队列，等待进程调用 **accept** 函数时把连接取出来。



不管是半连接队列还是全连接队列，都有最大长度限制，超过限制时，内核会直接丢弃，或返回 RST 包。

更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜

代码界的小白

操作系统

1.什么是线程什么是进程，有什么区别？

回答：线程是进程当中的一条执行流程。同一个进程中多个线程之间可以共享代码段、数据段、打开的文件等资源，但每个线程各自都有一套独自的寄存器和栈，这样可以确保线程的控制流是相对独立的。

进程：编写的代码只是一个存储在硬盘的静态文件，通过编译后就会生成二进制可执文件，当我们运行这个可执文件后，它会被装载到内存中，接着 CPU 会执行程序中的每一条指令，那么这个运口中的程序，就被称为「进程（**Process**）」。

2.并发和并行有什么区别？

- 并发：同一时间段，多个任务都在执行（单位时间内不一定同时执行）；
- 并行：单位时间内，多个任务同时执行。

3.使用线程的优缺点？

线程的优点：

- 一个进程中可以同时存在多个线程；
- 各个线程之间可以并发执行；
- 各个线程之间可以共享地址空间和文件等资源；

线程的缺点：

- 当进程中的一个线程崩溃时，会导致其所属进程的所有线程崩溃。举个例口，对于游戏的用户设计，则不应该使用多线程的方式，否则一个用户挂了，会影响其他同一个进程的线程。

4.线程与进程的比较

线程与进程的比较如下：

- 进程是资源（包括内存、打开的文件等）分配的单位，线程是 CPU 调度的单位；
- 进程拥有一个完整的资源平台，而线程只独享必不可少的资源，如寄存器和栈；
- 线程同样具有就绪、阻塞、执行三种基本状态，同样具有状态之间的转换关系；
- 线程能减少并发执行的时间和空间开销；

对于，线程相比进程能减少开销，体现在：

- 线程的创建时间比进程快，因为进程在创建的过程中，还需要资源管理信息，如内存管理信息、文件管理信息，在线程在创建的过程中，不会涉及这些资源管理信息，而是共享它们；
- 线程的终止时间比进程快，因为线程释放的资源相比进程少很多；
- 同一个进程内的线程切换比进程切换快，因为线程具有相同的地址空间（虚拟内存共享），这意味着同一个进程的线程都具有同一个页表，那么在切换的时候不需要切换页表。而对于进程之间的切换，切换的时候要把页表给切换掉，而页表的切换过程开销是比较大的；
- 由于同一进程的各线程间共享内存和文件资源，那么在线程之间数据传递的时候，就不需要经过内核了，这就使得线程之间的数据交互效率更高了；

5.什么是线程的上下文切换？

线程与进程最大的区别在于：线程是调度的基本单位，而进程则是资源拥有的基本单位。

所谓操作系统的任务调度，实际上的调度对象是线程，进程只是给线程提供了虚拟内存、全局变量等资源。

对于线程和进程，我们可以这么理解：

- 当进程只有一个线程时，可以认为进程就等于线程；
- 当进程拥有多个线程时，这些线程会共享相同的虚拟内存和全局变量等资源，这些资源在上下文切换时是不需要修改的；

另外，线程也有自己的私有数据，比如栈和寄存器等，这些在上下文切换时也是需要保存的。

6.线程上下文切换的是什么？

这还得看线程是不是属于同一个进程：

- 当两个线程不是属于同一个进程，则切换的过程就跟进程上下文切换同样；
- 当两个线程是属于同一个进程，因为虚拟内存是共享的，所以在切换时，虚拟内存这些资源就保持不动，只需要切换线程的私有数据、寄存器等不共享的数据；

7. 进程的状态有哪些？

进程的三种基本状态：

运行态：占有cpu，并且就在cpu上执行。

就绪态：已经具备运行条件，但由于没有空闲cpu，而暂时不能运行。（也就是cpu没有调度到它）

阻塞态：因等待某一事件而不能运行。

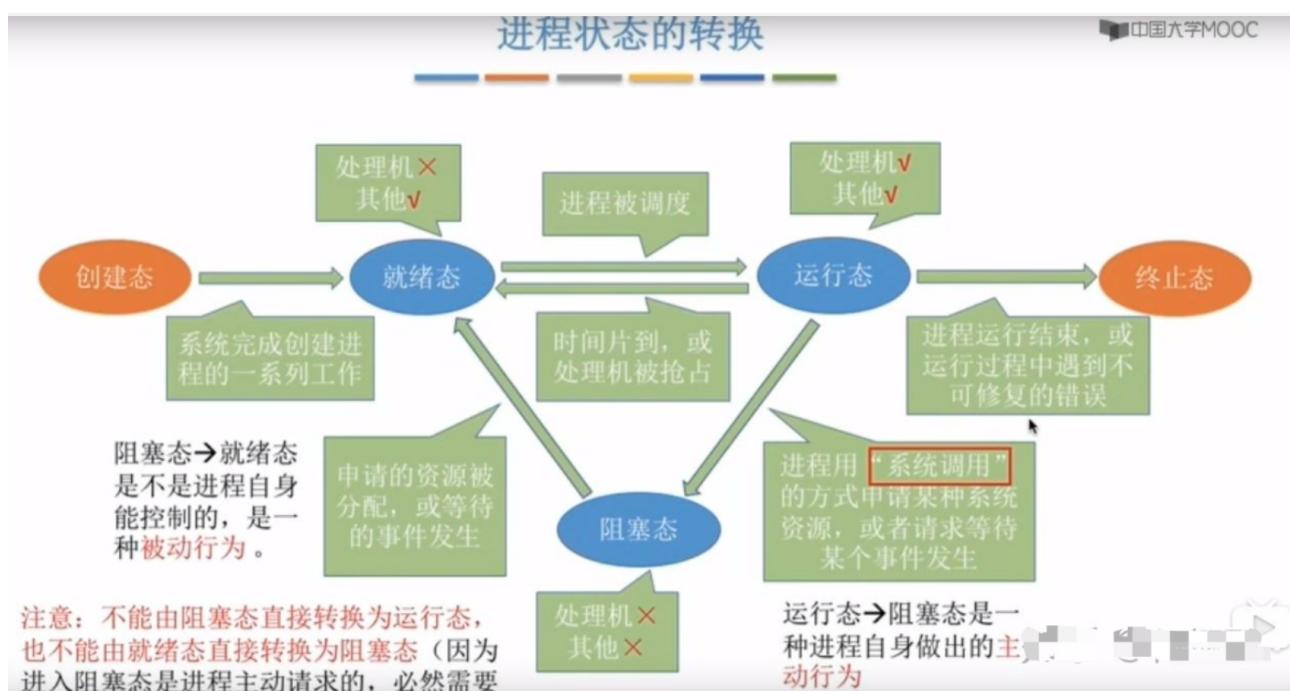
其实这个很好理解，如果看过我之前关于线程的讨论的话应该很容易理解。注意：单核处理器同一时间只有一个进程处于运行态，双核则是两个。

还有两个状态：

创建态：进程正在被创建，系统为其初始化PCB，分配资源。

终止态：进程正在从系统中撤销，回收进程的资源，撤销其PCB。

下面一张图可看出这五个状态的关系：



8. 进程间的通信方式有哪些？

1. 管道/匿名管道(**Pipes**)：用于具有亲缘关系的父子进程间或者兄弟进程之间的通信。

2. 有名管道(**Names Pipes**) : 匿名管道由于没有名字, 只能用于亲缘关系的进程间通信。为了克服这个缺点, 提出了有名管道。有名管道严格遵循先进先出(**first in first out**)。有名管道以磁盘文件的方式存在, 可以实现本机任意两个进程通信。
3. 信号(**Signal**) : 信号是一种比较复杂的通信方式, 用于通知接收进程某个事件已经发生;
4. 消息队列(**Message Queuing**) : 消息队列是消息的链表, 具有特定的格式, 存放在内存中并由消息队列标识符标识。管道和消息队列的通信数据都是先进先出的原则。与管道 (无名管道: 只存在于内存中的文件; 命名管道: 存在于实际的磁盘介质或者文件系统) 不同的是消息队列存放在内核中, 只有在内核重启(即, 操作系统重启)或者显示地删除一个消息队列时, 该消息队列才会被真正的删除。消息队列可以实现消息的随机查询, 消息不一定要以先进先出的次序读取, 也可以按消息的类型读取.比 FIFO 更有优势。消息队列克服了信号承载信息量少, 管道只能承载无格式字节流以及缓冲区大小受限等缺。
5. 信号量(**Semaphores**) : 信号量是一个计数器, 用于多进程对共享数据的访问, 信号量的意图在于进程间同步。这种通信方式主要用于解决与同步相关的问题并避免竞争条件。
6. 共享内存(**Shared memory**) : 使得多个进程可以访问同一块内存空间, 不同进程可以及时看到对方进程中对共享内存中数据的更新。这种方式需要依靠某种同步操作, 如互斥锁和信号量等。可以说这是最有用的进程间通信方式。
7. 套接字(**Sockets**) : 此方法主要用于在客户端和服务器之间通过网络进行通信。套接字是支持 TCP/IP 的网络通信的基本操作单元, 可以看做是不同主机之间的进程进行双向通信的端点, 简单的说就是通信的两方的一种约定, 用套接字中的相关函数来完成通信过程。

9. 进程间的调度方式有哪些?

- 先到先服务(**FCFS**)调度算法 : 从就绪队列中选择一个最先进入该队列的进程为之分配资源, 使它立即执行并一直执行到完成或发生某事件而被阻塞放弃占用 CPU 时再重新调度。
- 短作业优先(**SJF**)的调度算法 : 从就绪队列中选出一个估计运行时间最短的进程为之分配资源, 使它立即执行并一直执行到完成或发生某事件而被阻塞放弃占用 CPU 时再重新调度。
- 时间片轮转调度算法 : 时间片轮转调度是一种最古老, 最简单, 最公平且使用最广的算法, 又称 RR(Round robin)调度。每个进程被分配一个时间段, 称作它的时间片, 即该进程允许运行的时间。
- 多级反馈队列调度算法 : 前面介绍的几种进程调度的算法都有一定的局限性。如短进程优先的调度算法, 仅照顾了短进程而忽略了长进程。多级反馈队列调度算法既能使高优先级的作业得到响应又能使短作业(进程)迅速完成。, 因而它是目前被公认的一种较好的进程调度算法, UNIX 操作系统采取的便是这种调度算法。
- 优先级调度 : 为每个流程分配优先级, 首先执行具有最高优先级的进程, 依此类推。具有相同优先级的进程以 FCFS 方式执行。可以根据内存要求, 时间要求或任

何其他资源要求来确定优先级。

10.PCB都包含什么内容？

进程描述信息：

- 进程标识符：标识各个进程，每个进程都有一个并且唯一的标识符；
- 用户标识符：进程归属的用户，用户标识符主要为共享和保护服务；

进程控制和管理信息：

- 进程当前状态，如 new、ready、running、waiting 或 blocked 等；
- 进程优先级：进程抢占 CPU 时的优先级；

资源分配清单：

- 有关内存地址空间或虚拟地址空间的信息，所打开文件的列表和所使用的 I/O 设备信息。

CPU 相关信息：

- CPU 中各个寄存器的值，当进程被切换时，CPU 的状态信息都会被保存在相应的 PCB 中，以便进程
- 重新执行时，能从断点处继续执行。

11.进程的上下文切换到底是什么呢？

回答：进程是由内核管理和调度的，所以进程的切换只能发生在内核态。所以，进程的上下文切换不仅包含了虚拟内存、栈、全局变量等用户空间的资源，还包括了内核堆栈、寄存器等内核空间的资源。

12.产生死锁条件？

回答：

(1)互斥使用(资源独占)：一个资源每次只能给一个进程使用

(2)占有且等待(请求和保持，部分分配)：进程在申请新的资源的同时保持对原有资源的占有

(3)不可抢占(不可剥夺)：资源申请者不能强行的从资源占有者手中夺取资源，资源只能由占有者自愿释放

(4)循环等待：存在一个进程等待队列 {P1 , P2 , ... , Pn}，其中P1等待P2占有的资源， P2等待P3占有的资源， ..., Pn等待P1占有的资源，形成一个进程等待环路。

当死锁产生的时候一定会有这四个条件，有一个条件不成立都不会造成死锁。

追问1：如何避免死锁问题的发生？

回答：产生死锁的四个必要条件是：互斥条件、持有并等待条件、不可剥夺条件、环路等待条件。

那么避免死锁问题就只需要破坏其中一个条件就可以，最常用的并且可行的就是使用资源有序分配法，来破坏环路等待条件。

更多Java面试八股、Java后端实战项目、更多互联网春招、实习、秋招等经验分享，
可以上微信搜索公众号：代码界的小白，和小白一起学编程！



微信搜一搜

代码界的小白