目录

[测试基础知识 2](#_Toc2961812)

[行业理解 2](#_Toc2961813)

[素质问答 3](#_Toc2961814)

[计算机网络 4](#_Toc2961815)

[框架工具 5](#_Toc2961816)

[项目经验 5](#_Toc2961817)

[数据结构和算法 6](#_Toc2961818)

[场景题 8](#_Toc2961819)

[编程语言 9](#_Toc2961820)

[数据库 11](#_Toc2961821)

[操作系统 12](#_Toc2961822)

[高频问题： 13](#_Toc2961823)

[计算机网络 13](#_Toc2961824)

[网络模型 13](#_Toc2961825)

[tcp和udp区别 15](#_Toc2961826)

[tcp三次握手四次挥手 19](#_Toc2961827)

[从输入url到浏览器显示页面都发生了什么 22](#_Toc2961828)

[项目经验 30](#_Toc2961829)

[详细问项目中你负责的部分，怎么实现的，碰到的问题，怎么解决的，解决问题的思路 30](#_Toc2961830)

[数据结构和算法 30](#_Toc2961831)

[查找算法、排序算法 30](#_Toc2961832)

[数组、链表对比 30](#_Toc2961833)

[hashmap、hashtable对比 30](#_Toc2961834)

[java 30](#_Toc2961835)

[java内存模型、内存分配、GC 30](#_Toc2961836)

[java重载和重写的区别，使用场景 30](#_Toc2961837)

[java接口和类、抽象类的区别，使用场景 37](#_Toc2961838)

[设计模式 48](#_Toc2961839)

[数据库 48](#_Toc2961840)

[索引的优缺点 48](#_Toc2961841)

[索引的实现原理，为什么能加快查询速度 49](#_Toc2961842)

[什么情况下建立索引 49](#_Toc2961843)

[事务、锁、sql语句 50](#_Toc2961844)

[操作系统 51](#_Toc2961845)

[linux常用命令 51](#_Toc2961846)

[进程、线程对比 51](#_Toc2961847)

[死锁，原因，解决方法 52](#_Toc2961848)

[多线程实现方式，创建方法，线程池大小 52](#_Toc2961849)

内容来自于各互联网公司测试开发工程师面试问题

# 测试基础知识

测试的分类

测试方法有哪些

测试阶段需要些什么

需求分析、测试计划、设计用例

黑盒测试、白盒测试方法

使用的测试工具

测试计划的执行步骤

测试的了解、相关概念

等价类划分在项目中的具体应用

测试用例设计方法

# 行业理解

面试每家公司都可能会问到：为什么想来XX工作，对XX有什么了解

如何看待自动化测试

对软件测试的理解

测试人员需要的品质

为什么选择测试

为什么不选择开发

为什么想来深圳/北京/上海/广州/杭州

自动化测试与手工测试的区别

对测试开发的理解

学习测试的途径/方法

对测试工程师的理解

游戏测试与普通测试的区别

为什么投递测试开发岗

# 素质问答

怎么证明你的学习能力强

最近都怎么学习/学习新知识的途径

学习开发和测试的途径

你的自信来源于哪里

让你最挫败的是

有没有考虑创新

你最自豪的一件事是

讲一下你的项目，团队合作

你认为你的优点和缺点，性格？其他？

你的职业规划，以及如何平衡生活和学习

你通常解决问题的方法

身边比你优秀的人，你觉得他们优秀的地方在哪，如何向其学习

对自己的面试打分，十分制

期待的月薪

如何处理和同事们之间的关系

有无女朋友，就业城市出现分歧如何处理

意向的就业城市

为什么选择测试而不是开发

自身做测试的优势，举例说明

是否愿意转到开发岗

为什么选择测试开发岗

参加过哪些公司面试，已拿到哪些offer

家庭情况/兴趣爱好

生活习惯，是否接受加班较多

如何看待人重复犯相同的错误

介绍下自己犯的一个错误

如何看待工作进展和预期偏差大的情况

如何看待有些人一毕业就买了房

无论怎么努力都达不到上级的满意，怎么办

如何快速地融入到公司里

列举一个生活中看到的常见缺陷现象，分析原因

说说自身最薄弱的知识部分

压力最大的阶段/最艰难的时候，解决方法

主管在会议上把项目重大事务的错误归咎于你，你怎么办

学的最难的课程和最简单的课程

为什么想去互联网公司

# 计算机网络

http

OSI七层模型

tcp/ip

常用协议

tcp/udp区别

QQ使用的是udp还是tcp，为什么

本地cookie 会影响服务器端的性能吗

网络中数据量比较大的时候，会发生什么

浏览器输入一个网址，会发生什么/从网址URL输入到显示页面的过程

输入网址返回失败，可能原因

Cookie和session的区别，session和cookie通常会产生哪些隐患

http头中的refer指的是，以及http头中的内容都有什么？

访问一个网址的整个过程？用到了哪些协议？

ARP和RARP协议在那一层？

视频采用的是哪种协议？

TCP/IP三次握手四次挥手过程

TCP拥塞控制

# 框架工具

SpringMVC的注释？分别解释注释的作用？IOC控制反转？AOP？

看过哪些开源的项目以及源码？

Git的使用，讲一下？

Spring框架:SpringMVC/SpringBoot/SpringCloud

# 项目经验

如何写好用例

在工作中做的项目如何进行了测试

在之前的工作中都做了什么

工作中遇到的最大的一个问题，怎么解决的

工作中觉得有哪些地方需要改进的

讲讲你的项目，项目里面的问题怎么解决的

工作中都学到了哪些东西

举个实际的例子，说说你为何是这样解决的

讲项目，问项目的详细实现，难点怎么解决的

项目中的高并发，怎么实现的？

针对项目中网站的测试方案

项目背景及最终成果

项目中的亮点

# 数据结构和算法

查找排序

链表和数组的区别

ArrayList和LinkedList的区别，ArrayList做了什么，能使得它自动扩容？扩容因子？时间复杂度

数组与ArrayList的异同

hashMap的底层实现

Hashmap和sortedmap的区别

B+树的特点

给定一个字符串，判断是否是括号匹配的

算法中排序的稳定和不稳定性，怎么判断

手写冒泡排序

冒泡排序的优化

全排列算法实现，如何优化？

两个栈实现一个队列

如何判断一组数中只有一个数出现了一次，其他出现两次？几种方法，时间复杂度

数组中有两个数出现了1次，其他出现了2次，找出这两个出现一次的？

上面一题，如果是字符串呢？怎么解决？

0-9，a-z，输入两个字符串类型的数，进行两个数36进制内的求和？

寻找二维数组的鞍点

堆和栈的区别，各自的存储内容

平衡二叉树

完全二叉树、大顶堆、小顶堆

手写代码：复制一个每个节点都有随机指针的单链表

算法题：统计三种颜色的球各自的个数（hashmap）

手写代码：找出str中str1和str2之间的字符串

应用题：写出上述代码的测试用例

字符编码，及之间的转化，汉字占用字符

算法题：1001个数字，包含每个1-1000的数，找出唯一一个出现两次的数（多种方法）

手写代码：Java模拟扑克牌洗牌的过程

手写代码：长度n的数组左移m位，要求空间复杂度较小

手写代码：Python统计日志文件中失败率

手写代码：二维平面点集合组成的直线中斜率最大的直线经过的点

999个数，每个都是1-1000间且均出现一次，找出唯一缺失的数

手写代码：输出"相似"的字符串对，求时间复杂度

给出上述代码的测试用例

手写代码：输出次数前10的URL，时间复杂度

给出上述代码的测试用例

手写代码:数组去重并移动（多种方法）

给出上述代码的测试用例【详细写】

手写代码:查找数组中包含不重复数字的个数

数组中找出所有和为n的子数组（复杂度尽可能低）

手写代码，100个数据，有些数字出现多次，有些数字出现1次，统计出现一次的？时间复杂度，能否优化

给定一个整型数组，和为定值，时间复杂度最少的实现，手撕

对上述的代码写测试用例，现场写

# 场景题

测试京东的网站，从哪些方面？详细说说 缓存的种类?

50W的数据量导出为1G，400个敏感词，统计每个敏感词的条数，使用8核CPU,设计一个工具，30分钟左右扫完50W的数据

详细讲讲测试京东网站中的某一功能，需要从哪些方面考虑？比如说登录？

Service容器讲讲

计算机组成原理中指令比如说运算一个a+b怎么实现，存一个数，add指令，move指令？

计算机组成原理学过吗？如何进行两个数的交换？

测试一支笔从哪些方面测试

测试一个百度网站，从哪些方面测试

你认为是bug,开发人员认为不是，应该怎么办

项目上线之后，用户反映一个问题，而在线下又难以复现，你会怎么办？需要提交一个bug，进行后续的追踪?

测试一个电梯，考虑从哪些方面测试？危险、断电

测一张A4纸，如何测

美团外卖搜索栏的测试

对于网页上出现了一片空白，主要从哪几个方面进行排查

网页访问很慢的时候，怎么排查问题？以及怎么解决？（主要考察负载均衡，以及如何实现）

项目中的客户端之间如何通信，如何保证所发的消息是与指定的对象进行通信？

对头条的搜索功能设计测试用例

对一次性水杯进行测试

实现功能：查找附近的人

Lucene全文检索引擎的实现原理

Lucene倒排序法的实现

文本中找出所有的电话号码（正则表达式）

文本中非数字的电话号码如何查找？

中文分词软件的实现原理

应用题：海量字符串数据，查找词频最高的50个单词

应用题：测试搜索框

应用题：测试网站中商品链接跳转过程

应用题：集群为外部请求提供服务，给出测试方案（负载均衡、读写）

手写代码：查询字符串中，出现最多的单词；split（）

关系单车APP测试方案

场景题：测试视频播放页面（要求回答非常全面细致）

智力题：圆桌子下象棋

智力题：数组内负数放在前，整数放在后（多种方法）

统计文本中英文单词数并优化（多线程、hadoop）

Java解析xml到数据库，设计测试此过程的方案

游戏测试：针对“庄周”的二技能进行测试，考虑所有测试用例

如何测试登录页面

什么情况需要输入验证码

# 编程语言

脚本一般用什么写，java和python的区别

java重载和重写的区别，使用场景

java接口和类、抽象类的区别，使用场景

Java内存模型

java中内存的分配

Java中对象访问的方式

Java的垃圾回收机制

Java如何保存图片

vector和list区别

string，stringbuffer继承类，及区别

Java解析xml的方法

Java连接数据库时需要的参数

Java中的io及泛型

Java多线程同步

Lamada表达式

Finally的用法

Exception类型

对Java框架的了解，写的代码量多少

判断链表中有环的方法，设快指针和慢指针，问还有其他的方法吗？

设计模式？手写代码：单例模式？使用场景？

创建线程的几种方法

使用的python包

使用的Python数据结构

tuple、list区别

Python异常处理

Python装饰器及应用场景

Python匿名函数

Python连接数据库的包

# 数据库

线性数据库和非线性数据库的区别（数组和链表）

数据库索引的分类

聚簇索引和非聚簇索引

数据库的索引？以及索引的优缺点？

数据库的连接join：左连接、右连接、内连接区别，应用场景？

项目中数据库的使用？索引的优势？

项目中数据库建立的索引有哪些？感觉比不建索引效率高了？

索引的实现原理

数据库的隔离级别？解释？

数据库线程池

MySQL空间索引的具体实现

空间索引是否支持球面

数据库事务ACID、乐观锁、悲观锁、脏读幻读不可重复读

SQL：求第10-20名的学生及对应成绩

两表join，大表和小表左右顺序应该如何选择

SQL:查询出每个员工对应的boss名字（left join）

手写代码：4个SQL题；having、join、group by、sum 、like

SQL:找出选了课程A没选课程B的学生名字

Sql：查找每班中语文成绩最高的学生信息

非关型数据库、关系型数据库对比

创建索引的语句

适合建立索引的字段

drop、truncate、delete区别

建表语句

# 操作系统

查看端口号

查看内存使用情况

死锁？造成原因？死锁的避免方案？银行家算法？

手写代码：死锁

Linux 常见的命令、查看进程、查看文件

线程和进程的异同

线程的几种状态，以及如何相互转换？

多线程实现的方式？几种创建方法？

操作系统中的多线程实现的方法，比较区别？

线程池的适宜大小

同步的控制方法

Linux熟悉吗？统计文本的行数？

批量杀死进程？

一个四核CPU的机器，同一时刻能运行多少个应用程序？为什么？

Windows开机比较慢，应该从那几个方面去排查问题

内存泄露

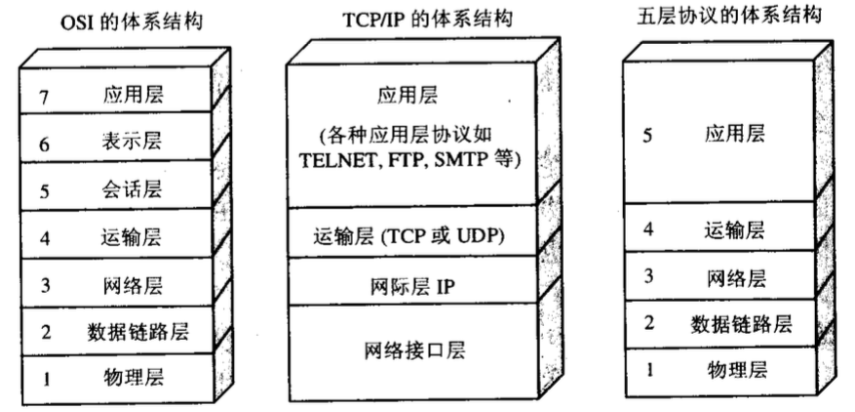
软拷贝及硬拷贝

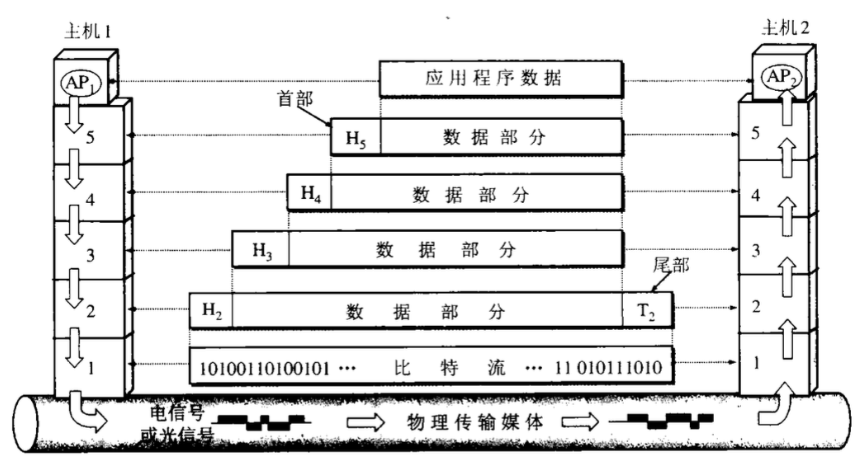
Linux：查找1.txt文件

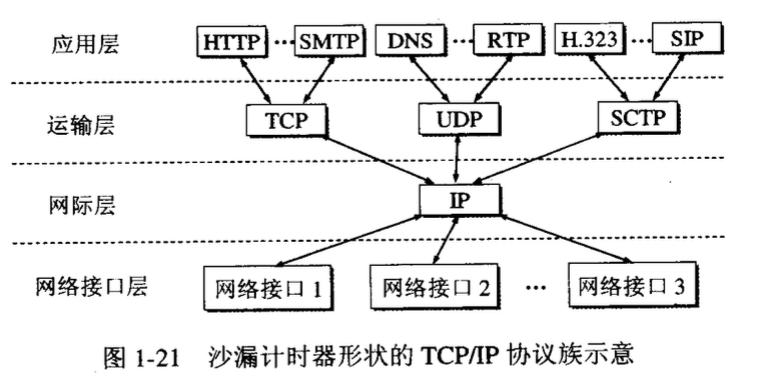
# 高频问题：

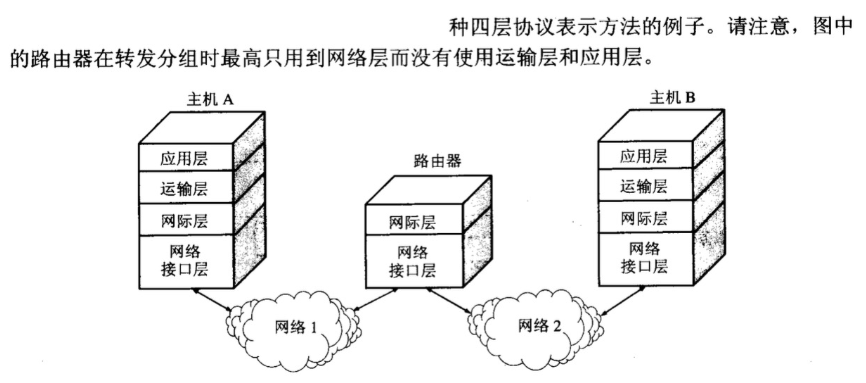
## 计算机网络

### 网络模型









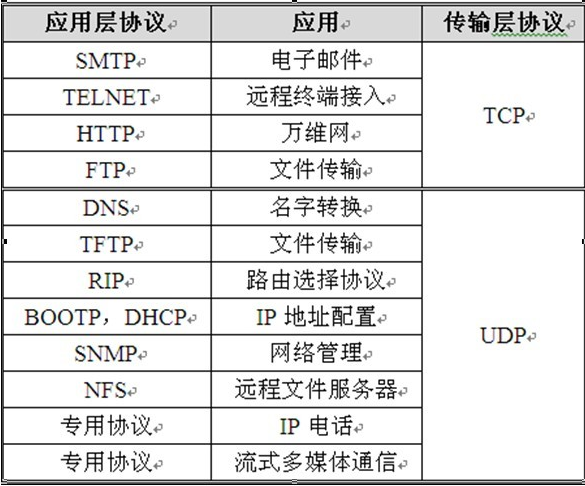


### tcp和udp区别

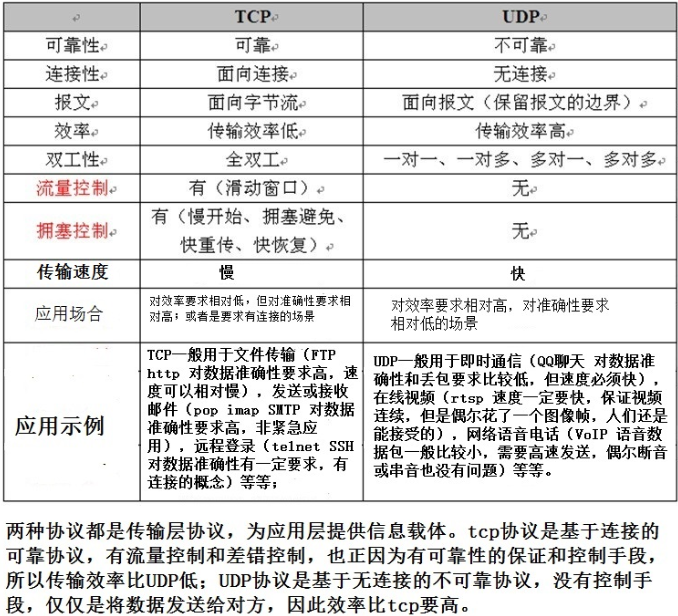
面向报文：面向报文的传输方式是应用层交给UDP多长的报文，UDP就照样发送，即一次发送一个报文。因此，应用程序必须选择合适大小的报文。若报文太长，则IP层需要分片，降低效率。若太短，会是IP太小。UDP对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。这也就是说，应用层交给UDP多长的报文，UDP就照样发送，即一次发送一个报文。

面向字节流：面向字节流的话，虽然应用程序和TCP的交互是一次一个数据块（大小不等），但TCP把应用程序看成是一连串的无结构的字节流。TCP有一个缓冲，当应用程序传送的数据块太长，TCP就可以把它划分短一些再传送。如果应用程序一次只发送一个字节，TCP也可以等待积累有足够多的字节后再构成报文段发送出去。

TCP和UDP协议的一些应用



TCP和UDP协议的比较



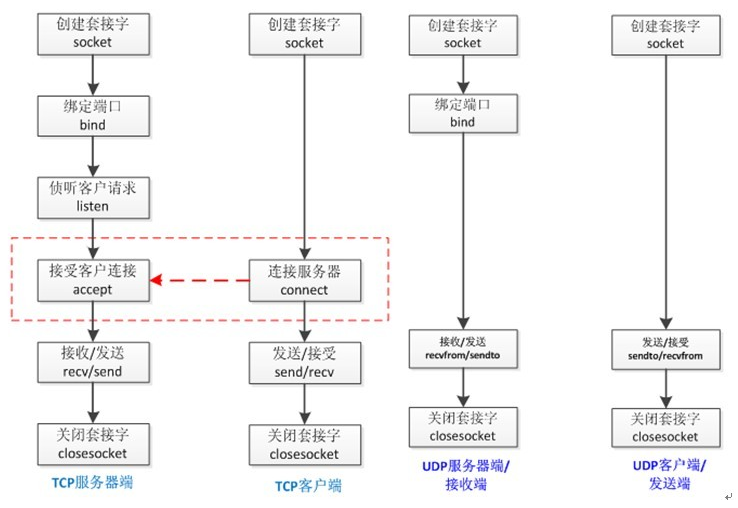
面向连接和面向无连接的区别：

面向连接举例：两个人之间通过电话进行通信;

面向无连接举例：邮政服务，用户把信函放在邮件中期待邮政处理流程来传递邮政包裹。

显然，不可达代表不可靠。

TCP/UDP编程模型



从上图也能清晰的看出，TCP通信需要服务器端侦听listen、接收客户端连接请求accept，等待客户端connect建立连接后才能进行数据包的收发（recv/send）工作。而UDP则服务器和客户端的概念不明显，服务器端即接收端需要绑定端口，等待客户端的数据的到来。后续便可以进行数据的收发（recvfrom/sendto）工作。

在前面讲解UDP时，提到了UDP保留了报文的边界，下面我们来谈谈TCP和UDP中报文的边界问题。在默认的阻塞模式下，TCP无边界，UDP有边界。

对于TCP协议，客户端连续发送数据，只要服务端的这个函数的缓冲区足够大，会一次性接收过来，即客户端是分好几次发过来，是有边界的，而服务端却一次性接收过来，所以证明是无边界的；而对于UDP协议，客户端连续发送数据，即使服务端的这个函数的缓冲区足够大，也只会一次一次的接收，发送多少次接收多少次，即客户端分几次发送过来，服务端就必须按几次接收，从而证明，这种UDP的通讯模式是有边界的。

TCP优点：可靠、稳定

TCP的可靠体现在TCP在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源。

TCP缺点：慢，效率低，占用系统资源高，易被攻击

TCP在传递数据之前，要先建连接，这会消耗时间，而且在数据传递时，确认机制、重传机制、拥塞控制机制等都会消耗大量的时间，而且要在每台设备上维护所有的传输连接，事实上，每个连接都会占用系统的CPU、内存等硬件资源。而且，因为TCP有确认机制、三次握手机制，这些也导致TCP容易被人利用，实现DOS、DDOS、CC等攻击。

UDP优点：快，比TCP稍安全

UDP没有TCP的握手、确认、窗口、重传、拥塞控制等机制，UDP是一个无状态的传输协议，所以它在传递数据时非常快。没有TCP的这些机制，UDP较TCP被攻击者利用的漏洞就要少一些。但UDP也是无法避免攻击的，比如：UDP Flood攻击

UDP缺点：不可靠，不稳定

因为UDP没有TCP那些可靠的机制，在数据传递时，如果网络质量不好，就会很容易丢包。

TCP/UDP应用场景

什么时候使用TCP

当对网络通讯质量有要求的时候，比如：整个数据要准确无误的传递给对方，这往往用于一些要求可靠的应用，比如HTTP、HTTPS、FTP等传输文件的协议，POP、SMTP等邮件传输的协议。   
在日常生活中，常见使用TCP协议的应用如下：

浏览器，用的HTTP

FlashFXP，用的FTP

Outlook，用的POP、SMTP

Putty，用的Telnet、SSH

QQ文件传输

什么时候使用UDP

当对网络通讯质量要求不高的时候，要求网络通讯速度能尽量的快，这时就可以使用UDP。   
比如，日常生活中，常见使用UDP协议的应用如下：

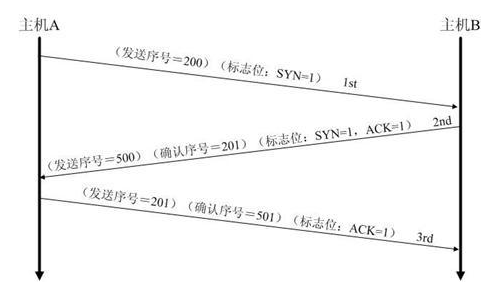
QQ语音

QQ视频

TFTP

### tcp三次握手四次挥手

TCP是主机对主机层的传输控制协议，提供可靠的连接服务，采用三次握手确认建立一个连接，如图所示。



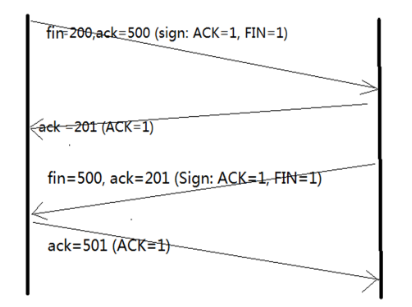
(1)第一次握手：建立连接时，客户端A发送SYN包（SYN=j）到服务器B，并进入SYN\_SEND状态，等待服务器B确认。

(2)第二次握手：服务器B收到SYN包，必须确认客户A的SYN（ACK=j+1），同时自己也发送一个SYN包（SYN=k），即SYN+ACK包，此时服务器B进入SYN\_RECV状态。

(3)第三次握手：客户端A收到服务器B的SYN＋ACK包，向服务器B发送确认包ACK（ACK=k+1），此包发送完毕，客户端A和服务器B进入ESTABLISHED状态，完成三次握手。

完成三次握手，客户端与服务器开始传送数据。

确认号：其数值等于发送方的发送序号 +1(即接收方期望接收的下一个序列号)。



由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这个原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

TCP的连接的拆除需要发送四个包，因此称为四次挥手(four-way handshake)。客户端或服务器均可主动发起挥手动作，在socket编程中，任何一方执行close()操作即可产生挥手操作。

（1）客户端A发送一个FIN，用来关闭客户A到服务器B的数据传送。

（2）服务器B收到这个FIN，它发回一个ACK，确认序号为收到的序号加1。和SYN一样，一个FIN将占用一个序号。

（3）服务器B关闭与客户端A的连接，发送一个FIN给客户端A。

（4）客户端A发回ACK报文确认，并将确认序号设置为收到序号加1。

为什么要三次握手：

在谢希仁著《计算机网络》第四版中讲“三次握手”的目的是“为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误”。在另一部经典的《计算机网络》一书中讲“三次握手”的目的是为了解决“网络中存在延迟的重复分组”的问题。

“已失效的连接请求报文段”的产生在这样一种情况下：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server。本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了。由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。但server却以为新的运输连接已经建立，并一直等待client发来数据。这样，server的很多资源就白白浪费掉了。采用“三次握手”的办法可以防止上述现象发生。例如刚才那种情况，client不会向server的确认发出确认。server由于收不到确认，就知道client并没有要求建立连接。”

**这就很明白了，防止服务器端一直等待而浪费资源。**

为什么要四次挥手

TCP协议是一种面向连接的、可靠的、基于字节流的传输层通信协议。TCP是全双工模式，这就意味着，当主机1发出FIN报文段时，只是表示主机1已经没有数据要发送了，主机1告诉主机2，它的数据已经全部发送完毕了；但是，这个时候主机1还是可以接受来自主机2的数据；当主机2返回ACK报文段时，表示它已经知道主机1没有数据发送了，但是主机2还是可以发送数据到主机1的；当主机2也发送了FIN报文段时，这个时候就表示主机2也没有数据要发送了，就会告诉主机1，我也没有数据要发送了，主机1此时发送ACK报文段表示主机1知道主机2也没有数据发送了，之后彼此就会愉快的中断这次TCP连接。

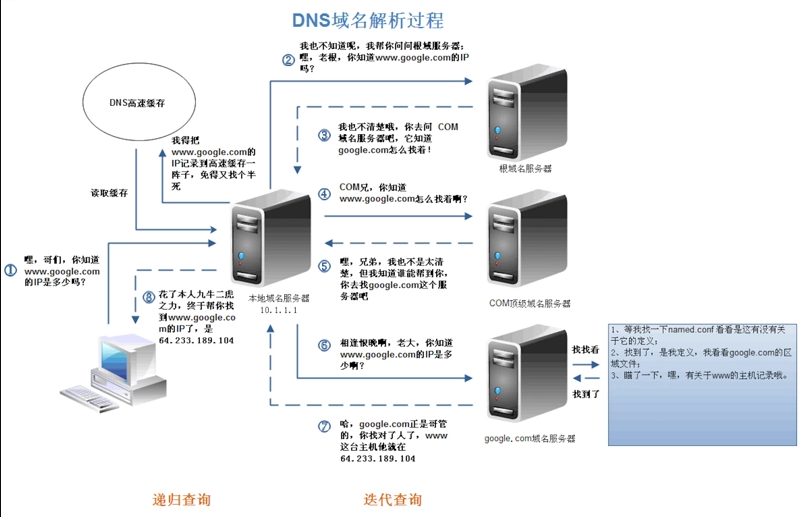
### 从输入url到浏览器显示页面都发生了什么

总体来说分为以下几个过程:

1. DNS解析
2. TCP连接
3. 发送HTTP请求
4. 服务器处理请求并返回HTTP报文
5. 浏览器解析渲染页面
6. 连接结束

DNS解析的过程就是寻找哪台机器上有你需要资源的过程。当你在浏览器中输入一个地址时，例如www.baidu.com，其实不是百度网站真正意义上的地址。互联网上每一台计算机的唯一标识是它的IP地址，但是IP地址并不方便记忆。用户更喜欢用方便记忆的网址去寻找互联网上的其它计算机，也就是上面提到的百度的网址。所以互联网设计者需要在用户的方便性与可用性方面做一个权衡，这个权衡就是一个网址到IP地址的转换，这个过程就是DNS解析。它实际上充当了一个翻译的角色，实现了网址到IP地址的转换。网址到IP地址转换的过程是如何进行的?

DNS解析是一个递归查询的过程。



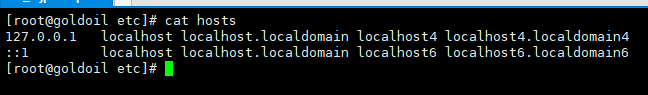
上述图片是查找www.google.com的IP地址过程。首先在本地域名服务器中查询IP地址，如果没有找到的情况下，本地域名服务器会向根域名服务器发送一个请求，如果根域名服务器也不存在该域名时，本地域名会向com顶级域名服务器发送一个请求，依次类推下去。直到最后本地域名服务器得到google的IP地址并把它缓存到本地，供下次查询使用。从上述过程中，可以看出网址的解析是一个从右向左的过程: com -> google.com -> www.google.com。但是你是否发现少了点什么，根域名服务器的解析过程呢？事实上，真正的网址是www.google.com.，并不是我多打了一个.，这个.对应的就是根域名服务器，默认情况下所有的网址的最后一位都是.，既然是默认情况下，为了方便用户，通常都会省略，浏览器在请求DNS的时候会自动加上，所有网址真正的解析过程为: . -> .com -> google.com. -> www.google.com.。

了解了DNS的过程，可以为我们带来哪些？上文中请求到google的IP地址时，经历了8个步骤，这个过程中存在多个请求(同时存在UDP和TCP请求，为什么有两种请求方式，请自行查找)。如果每次都经过这么多步骤，是否太耗时间？如何减少该过程的步骤呢？那就是DNS缓存。

DNS存在着多级缓存，从离浏览器的距离排序的话，有以下几种: 浏览器缓存，系统缓存，路由器缓存，IPS服务器缓存，根域名服务器缓存，顶级域名服务器缓存，主域名服务器缓存。

在你的chrome浏览器中输入:chrome://dns/，你可以看到chrome浏览器的DNS缓存。

系统缓存主要存在/etc/hosts(Linux系统)中:

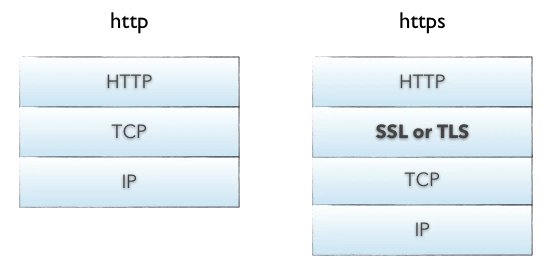


DNS负载均衡

不知道大家有没有思考过一个问题: DNS返回的IP地址是否每次都一样？如果每次都一样是否说明你请求的资源都位于同一台机器上面，那么这台机器需要多高的性能和储存才能满足亿万请求呢？其实真实的互联网世界背后存在成千上万台服务器，大型的网站甚至更多。但是在用户的眼中，它需要的只是处理他的请求，哪台机器处理请求并不重要。DNS可以返回一个合适的机器的IP给用户，例如可以根据每台机器的负载量，该机器离用户地理位置的距离等等，这种过程就是DNS负载均衡，又叫做DNS重定向。大家耳熟能详的CDN(Content Delivery Network)就是利用DNS的重定向技术，DNS服务器会返回一个跟用户最接近的点的IP地址给用户，CDN节点的服务器负责响应用户的请求，提供所需的内容。

HTTPS协议

HTTP报文是包裹在TCP报文中发送的，服务器端收到TCP报文时会解包提取出HTTP报文。但是这个过程中存在一定的风险，HTTP报文是明文，如果中间被截取的话会存在一些信息泄露的风险。那么在进入TCP报文之前对HTTP做一次加密就可以解决这个问题了。HTTPS协议的本质就是HTTP + SSL(or TLS)。在HTTP报文进入TCP报文之前，先使用SSL对HTTP报文进行加密。从网络的层级结构看它位于HTTP协议与TCP协议之间。



HTTPS过程

HTTPS在传输数据之前需要客户端与服务器进行一个握手(TLS/SSL握手)，在握手过程中将确立双方加密传输数据的密码信息。TLS/SSL使用了非对称加密，对称加密以及hash等。具体过程请参考经典的阮一峰先生的博客[TLS/SSL握手过程](http://www.ruanyifeng.com/blog/2014/09/illustration-ssl.html" \t "_blank)。HTTPS相比于HTTP，虽然提供了安全保证，但是势必会带来一些时间上的损耗，如握手和加密等过程，是否使用HTTPS需要根据具体情况在安全和性能方面做出权衡。

HTTP请求

它主要发生在客户端。发送HTTP请求的过程就是构建HTTP请求报文并通过TCP协议中发送到服务器指定端口(HTTP协议80/8080, HTTPS协议443)。HTTP请求报文是由三部分组成:请求行，请求报头和请求正文。

请求行：

格式：Method Request-URL HTTP-Version CRLF

eg: GET index.html HTTP/1.1

常用的方法有：GET, POST, PUT, DELETE, OPTIONS, HEAD

TODO:GET和POST有什么区别？

请求报头：

请求报头允许客户端向服务器传递请求的附加信息和客户端自身的信息。  
PS: 客户端不一定特指浏览器，有时候也可使用Linux下的CURL命令以及HTTP客户端测试工具等。  
常见的请求报头有: Accept, Accept-Charset, Accept-Encoding, Accept-Language, Content-Type, Authorization, Cookie, User-Agent等。

请求正文：

当使用POST, PUT等方法时，通常需要客户端向服务器传递数据。这些数据就储存在请求正文中。在请求报头中有一些与请求正文相关的信息，例如: 现在的Web应用通常采用Rest架构，请求的数据格式一般为json。这时就需要设置Content-Type: application/json。

服务器处理请求并返回HTTP报文

自然而然这部分对应的就是后端工程师眼中的HTTP。后端从在固定的端口接收到TCP报文开始，这一部分对应于编程语言中的socket。它会对TCP连接进行处理，对HTTP协议进行解析，并按照报文格式进一步封装成HTTP Request对象，供上层使用。这一部分工作一般是由Web服务器去进行，Web服务器有Tomcat, Jetty和Netty等等。

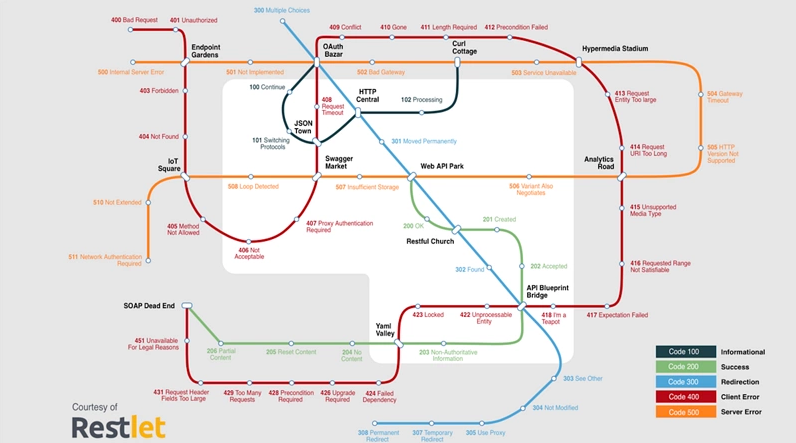
HTTP响应报文也是由三部分组成:状态码，响应报头和响应报文。

状态码是由3位数组成，第一个数字定义了响应的类别，且有五种可能取值:

* 1xx：指示信息–表示请求已接收，继续处理。
* 2xx：成功–表示请求已被成功接收、理解、接受。
* 3xx：重定向–要完成请求必须进行更进一步的操作。
* 4xx：客户端错误–请求有语法错误或请求无法实现。
* 5xx：服务器端错误–服务器未能实现合法的请求。

平时遇到比较常见的状态码有:200, 204, 301, 302, 304, 400, 401, 403, 404, 422, 500(分别表示什么请自行查找)。

TODO：301和302有什么区别？HTTP缓存？

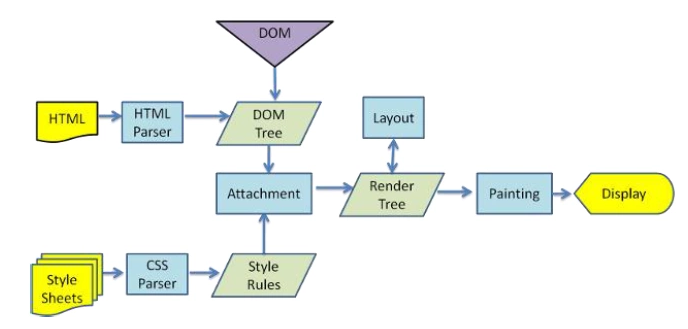


响应报头：常见的响应报头字段有: Server, Connection...。

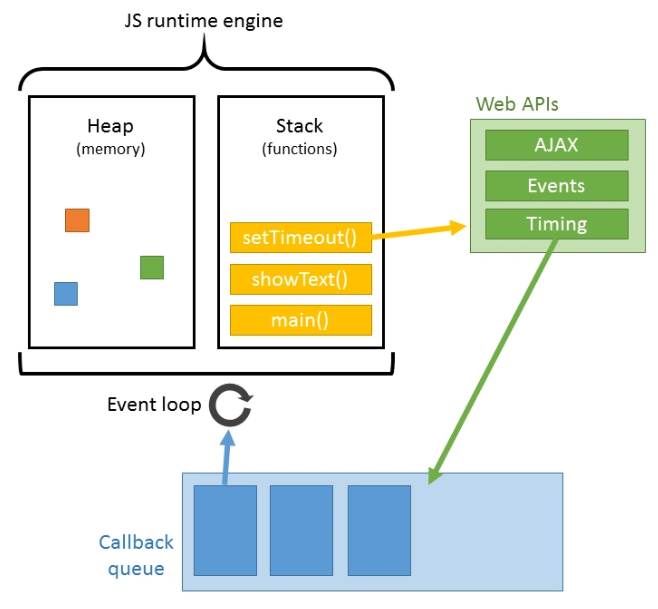
响应报文：服务器返回给浏览器的文本信息，通常HTML, CSS, JS, 图片等文件就放在这一部分。

浏览器解析渲染页面

浏览器在收到HTML,CSS,JS文件后，它是如何把页面呈现到屏幕上的？下图对应的就是WebKit渲染的过程。



浏览器是一个边解析边渲染的过程。首先浏览器解析HTML文件构建DOM树，然后解析CSS文件构建渲染树，等到渲染树构建完成后，浏览器开始布局渲染树并将其绘制到屏幕上。这个过程比较复杂，涉及到两个概念: reflow(回流)和repaint(重绘)。DOM节点中的各个元素都是以盒模型的形式存在，这些都需要浏览器去计算其位置和大小等，这个过程称为reflow;当盒模型的位置,大小以及其他属性，如颜色,字体,等确定下来之后，浏览器便开始绘制内容，这个过程称为repaint。页面在首次加载时必然会经历reflow和repaint。reflow和repaint过程是非常消耗性能的，尤其是在移动设备上，它会破坏用户体验，有时会造成页面卡顿。所以我们应该尽可能少的减少reflow和repaint。



JS的解析是由浏览器中的JS解析引擎完成的。JS是单线程运行，也就是说，在同一个时间内只能做一件事，所有的任务都需要排队，前一个任务结束，后一个任务才能开始。但是又存在某些任务比较耗时，如IO读写等，所以需要一种机制可以先执行排在后面的任务，这就是：同步任务(synchronous)和异步任务(asynchronous)。JS的执行机制就可以看做是一个主线程加上一个任务队列(task queue)。同步任务就是放在主线程上执行的任务，异步任务是放在任务队列中的任务。所有的同步任务在主线程上执行，形成一个执行栈;异步任务有了运行结果就会在任务队列中放置一个事件；脚本运行时先依次运行执行栈，然后会从任务队列里提取事件，运行任务队列中的任务，这个过程是不断重复的，所以又叫做事件循环(Event loop)。

浏览器在解析过程中，如果遇到请求外部资源时，如图像,iconfont,JS等。浏览器将重复1-6过程下载该资源。请求过程是异步的，并不会影响HTML文档进行加载，但是当文档加载过程中遇到JS文件，HTML文档会挂起渲染过程，不仅要等到文档中JS文件加载完毕还要等待解析执行完毕，才会继续HTML的渲染过程。原因是因为JS有可能修改DOM结构，这就意味着JS执行完成前，后续所有资源的下载是没有必要的，这就是JS阻塞后续资源下载的根本原因。CSS文件的加载不影响JS文件的加载，但是却影响JS文件的执行。JS代码执行前浏览器必须保证CSS文件已经下载并加载完毕。

**Web优化**

上面部分主要介绍了一次完整的请求对应的过程，了解该过程的目的无非就是为了Web优化。在谈到Web优化之前，我们回到一个更原始的问题，Web前端的本质是什么。我的理解是: 将信息快速并友好的展示给用户并能够与用户进行交互。快速的意思就是在尽可能短的时间内完成页面的加载，试想一下当你在淘宝购买东西的时候，淘宝页面加载了10几秒才显示出物品，这个时候你还有心情去购买吗？怎么快速的完成页面的加载呢？优雅的学院派雅虎给出了常用的一些手段，也就是我们熟悉的[雅虎34条军规](https://developer.yahoo.com/performance/)。这34军规实际上就是围绕请求过程进行的一些优化方式。

如何尽快的加载资源？答案就是能不从网络中加载的资源就不从网络中加载，当我们合理使用缓存，将资源放在浏览器端，这是最快的方式。如果资源必须从网络中加载，则要考虑缩短连接时间，即DNS优化部分;减少响应内容大小，即对内容进行压缩。另一方面，如果加载的资源数比较少的话，也可以快速的响应用户。当资源到达浏览器之后，浏览器开始进行解析渲染，浏览器中最耗时的部分就是reflow，所以围绕这一部分就是考虑如何减少reflow的次数。

## 项目经验

### 详细问项目中你负责的部分，怎么实现的，碰到的问题，怎么解决的，解决问题的思路

## 数据结构和算法

### 查找算法、排序算法

### 数组、链表对比

### hashmap、hashtable对比

## java

### java内存模型、内存分配、GC

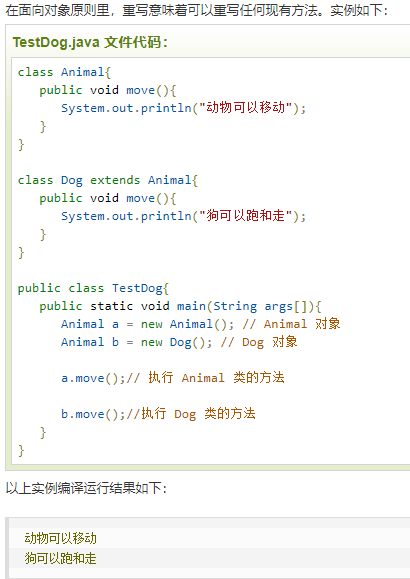
### java重载和重写的区别，使用场景

**重写（override）**

重写是子类对父类的允许访问的方法的实现过程进行重新编写, 返回值和形参都不能改变。**即外壳不变，核心重写！**

重写的好处在于子类可以根据需要，定义特定于自己的行为。 也就是说子类能够根据需要实现父类的方法。

重写方法不能抛出新的检查异常或者比被重写方法申明更加宽泛的异常。例如： 父类的一个方法申明了一个检查异常 IOException，但是在重写这个方法的时候不能抛出 Exception 异常，因为 Exception 是 IOException 的父类，只能抛出 IOException 的子类异常。

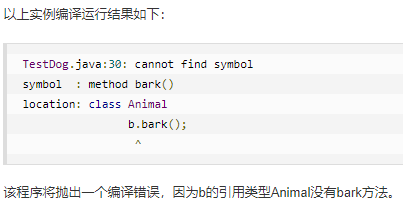


在上面的例子中可以看到，尽管b属于Animal类型，但是它运行的是Dog类的move方法。

这是由于在编译阶段，只是检查参数的引用类型。

然而在运行时，Java虚拟机(JVM)指定对象的类型并且运行该对象的方法。

因此在上面的例子中，之所以能编译成功，是因为Animal类中存在move方法，然而运行时，运行的是特定对象的方法。



**方法的重写规则**

* 参数列表必须完全与被重写方法的相同；
* 返回类型必须完全与被重写方法的返回类型相同；
* 访问权限不能比父类中被重写的方法的访问权限更低。例如：如果父类的一个方法被声明为public，那么在子类中重写该方法就不能声明为protected。
* 父类的成员方法只能被它的子类重写。
* 声明为final的方法不能被重写。
* 声明为static的方法不能被重写，但是能够被再次声明。
* 子类和父类在同一个包中，那么子类可以重写父类所有方法，除了声明为private和final的方法。
* 子类和父类不在同一个包中，那么子类只能够重写父类的声明为public和protected的非final方法。
* 重写的方法能够抛出任何非强制异常，无论被重写的方法是否抛出异常。但是，重写的方法不能抛出新的强制性异常，或者比被重写方法声明的更广泛的强制性异常，反之则可以。
* 构造方法不能被重写。
* 如果不能继承一个方法，则不能重写这个方法。



**重载（overload）**

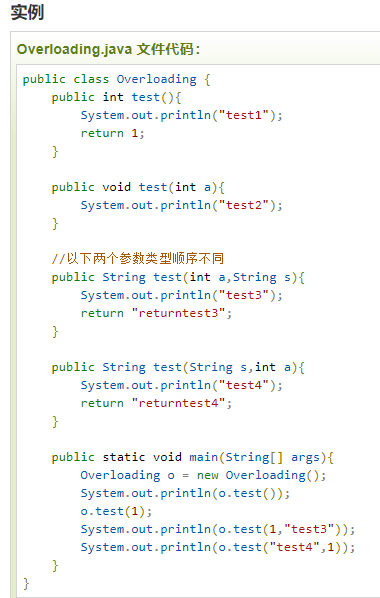
重载(overloading) 是在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。

每个重载的方法（或者构造函数）都必须有一个独一无二的参数类型列表。

最常用的地方就是构造器的重载。

**重载规则:**

* 被重载的方法必须改变参数列表(参数个数或类型不一样)；
* 被重载的方法可以改变返回类型；
* 被重载的方法可以改变访问修饰符；
* 被重载的方法可以声明新的或更广的检查异常；
* 方法能够在同一个类中或者在一个子类中被重载。
* 无法以返回值类型作为重载函数的区分标准。

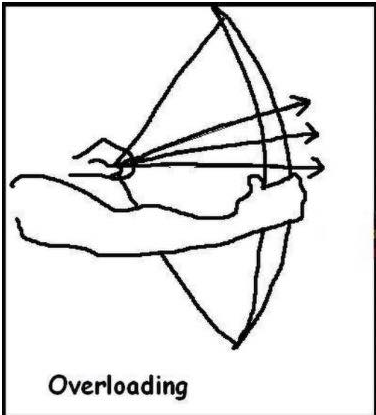
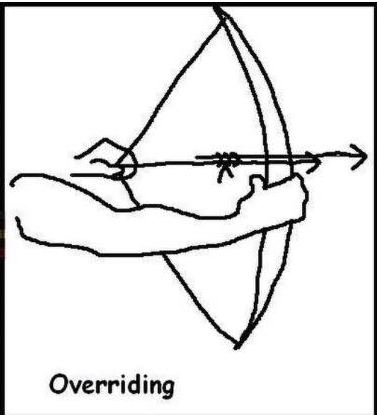




方法的重写(Overriding)和重载(Overloading)是java多态性的不同表现，重写是父类与子类之间多态性的一种表现，重载可以理解成多态的具体表现形式。

* (1)方法重载是一个类中定义了多个方法名相同,而他们的参数的数量不同或数量相同而类型和次序不同,则称为方法的重载(Overloading)。
* (2)方法重写是在子类存在方法与父类的方法的名字相同,而且参数的个数与类型一样,返回值也一样的方法,就称为重写(Overriding)。
* (3)方法重载是一个类的多态性表现,而方法重写是子类与父类的一种多态性表现。





* **重载**反映的是"随机应变". 同样一项功能, 根据数据类型的不同, 采用不同的处理方式. 比如, 同样是吃饭, 去高档餐厅吃西餐, 你会西装革履, 但是去吃火锅, 穿便装就比较合适.
* **重写**反映的是"父子差异". 你"继承"了父亲吃火锅的爱好, 但是吃同一份鸳鸯锅(注意, 数据类型相同) , 你喜欢涮红汤, 你父亲喜欢涮清汤.

### java接口和类、抽象类的区别，使用场景

**接口：**

接口（英文：Interface），在JAVA编程语言中是一个抽象类型，是抽象方法的集合，接口通常以interface来声明。一个类通过继承接口的方式，从而来继承接口的抽象方法。

接口并不是类，编写接口的方式和类很相似，但是它们属于不同的概念。类描述对象的属性和方法。接口则包含类要实现的方法。

除非实现接口的类是抽象类，否则该类要定义接口中的所有方法。

接口无法被实例化，但是可以被实现。一个实现接口的类，必须实现接口内所描述的所有方法，否则就必须声明为抽象类。另外，在 Java 中，接口类型可用来声明一个变量，他们可以成为一个空指针，或是被绑定在一个以此接口实现的对象。

**接口与类相似点：**

一个接口可以有多个方法。

接口文件保存在 .java 结尾的文件中，文件名使用接口名。

接口的字节码文件保存在 .class 结尾的文件中。

接口相应的字节码文件必须在与包名称相匹配的目录结构中。

**接口与类的区别：**

接口不能用于实例化对象。

接口没有构造方法。

接口中所有的方法必须是抽象方法。

接口不能包含成员变量，除了 static 和 final 变量。

接口不是被类继承了，而是要被类实现。

接口支持多继承。

**接口特性**

接口中每一个方法也是隐式抽象的,接口中的方法会被隐式的指定为 public abstract（只能是 public abstract，其他修饰符都会报错）。

接口中可以含有变量，但是接口中的变量会被隐式的指定为 public static final 变量（并且只能是 public，用 private 修饰会报编译错误）。

接口中的方法是不能在接口中实现的，只能由实现接口的类来实现接口中的方法。

**抽象类和接口的区别**

1. 抽象类中的方法可以有方法体，就是能实现方法的具体功能，但是接口中的方法不行。

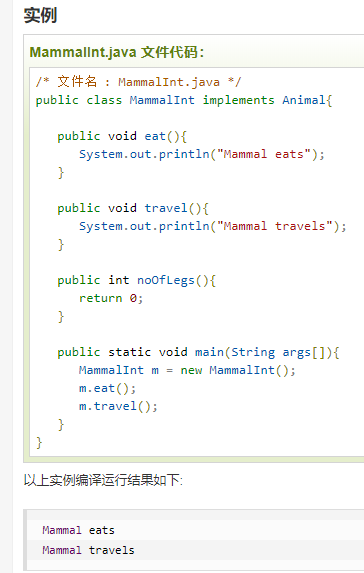
2. 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是 public static final 类型的。

3. 接口中不能含有静态代码块以及静态方法(用 static 修饰的方法)，而抽象类是可以有静态代码块和静态方法。

4. 一个类只能继承一个抽象类，而一个类却可以实现多个接口。







重写接口中声明的方法时，需要注意以下规则：

* 类在实现接口的方法时，不能抛出强制性异常，只能在接口中，或者继承接口的抽象类中抛出该强制性异常。
* 类在重写方法时要保持一致的方法名，并且应该保持相同或者相兼容的返回值类型。
* 如果实现接口的类是抽象类，那么就没必要实现该接口的方法。

在实现接口的时候，也要注意一些规则：

* 一个类可以同时实现多个接口。
* 一个类只能继承一个类，但是能实现多个接口。
* 一个接口能继承另一个接口，这和类之间的继承比较相似。



**接口的多继承**

在Java中，类的多继承是不合法，但接口允许多继承。

在接口的多继承中extends关键字只需要使用一次，在其后跟着继承接口。 如下所示：

public interface Hockey extends Sports, Event

以上的程序片段是合法定义的子接口，与类不同的是，接口允许多继承，而 Sports及 Event 可能定义或是继承相同的方法

**标记接口**

最常用的继承接口是没有包含任何方法的接口。

标记接口是没有任何方法和属性的接口.它仅仅表明它的类属于一个特定的类型,供其他代码来测试允许做一些事情。

标记接口作用：简单形象的说就是给某个对象打个标（盖个戳），使对象拥有某个或某些特权。

例如：java.awt.event 包中的 MouseListener 接口继承的 java.util.EventListener 接口定义如下：

package java.util;

public interface EventListener {}

没有任何方法的接口被称为标记接口。标记接口主要用于以下两种目的：

* **建立一个公共的父接口：**

正如EventListener接口，这是由几十个其他接口扩展的Java API，你可以使用一个标记接口来建立一组接口的父接口。例如：当一个接口继承了EventListener接口，Java虚拟机(JVM)就知道该接口将要被用于一个事件的代理方案。

* **向一个类添加数据类型：**

这种情况是标记接口最初的目的，实现标记接口的类不需要定义任何接口方法(因为标记接口根本就没有方法)，但是该类通过多态性变成一个接口类型。

**Java抽象类**

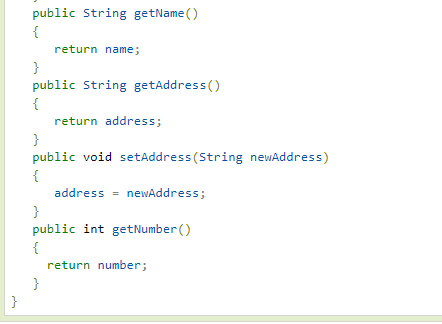
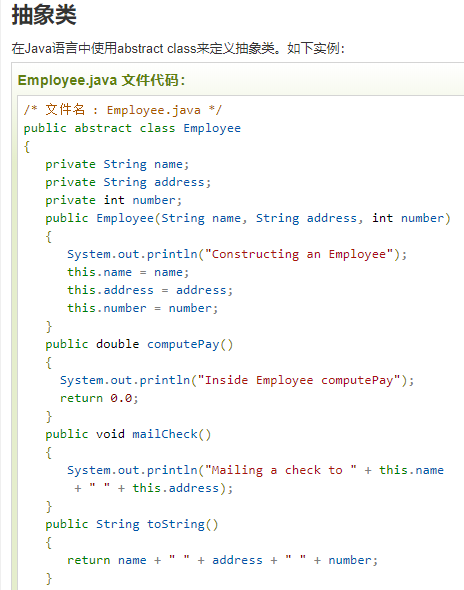
在面向对象的概念中，所有的对象都是通过类来描绘的，但是反过来，并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类。

抽象类除了不能实例化对象之外，类的其它功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。

由于抽象类不能实例化对象，所以抽象类必须被继承，才能被使用。也是因为这个原因，通常在设计阶段决定要不要设计抽象类。

父类包含了子类集合的常见的方法，但是由于父类本身是抽象的，所以不能使用这些方法。

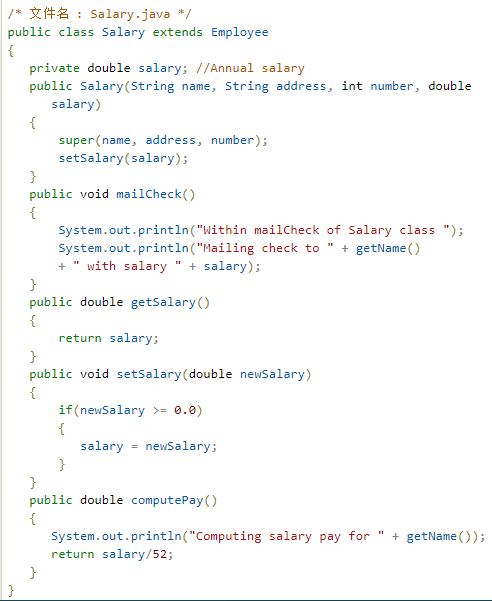
在Java中抽象类表示的是一种继承关系，一个类只能继承一个抽象类，而一个类却可以实现多个接口。





**继承抽象类**

我们能通过一般的方法继承Employee类：



尽管我们不能实例化一个 Employee 类的对象，但是如果我们实例化一个 Salary 类对象，该对象将从 Employee 类继承 7 个成员方法，且通过该方法可以设置或获取三个成员变量。

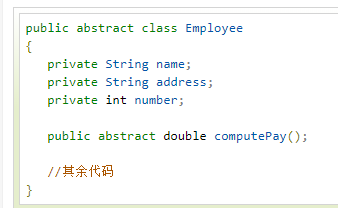


**抽象方法**

如果你想设计这样一个类，该类包含一个特别的成员方法，该方法的具体实现由它的子类确定，那么你可以在父类中声明该方法为抽象方法。

Abstract 关键字同样可以用来声明抽象方法，抽象方法只包含一个方法名，而没有方法体。

抽象方法没有定义，方法名后面直接跟一个分号，而不是花括号。

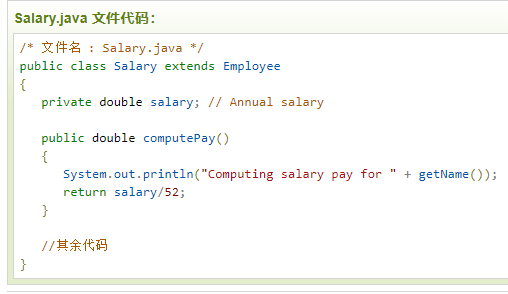


声明抽象方法会造成以下两个结果：

* 如果一个类包含抽象方法，那么该类必须是抽象类。
* 任何子类必须重写父类的抽象方法，或者声明自身为抽象类。

继承抽象方法的子类必须重写该方法。否则，该子类也必须声明为抽象类。最终，必须有子类实现该抽象方法，否则，从最初的父类到最终的子类都不能用来实例化对象。

如果Salary类继承了Employee类，那么它必须实现computePay()方法：



**抽象类总结规定**

* 1. 抽象类不能被实例化(初学者很容易犯的错)，如果被实例化，就会报错，编译无法通过。只有抽象类的非抽象子类可以创建对象。
* 2. 抽象类中不一定包含抽象方法，但是有抽象方法的类必定是抽象类。
* 3. 抽象类中的抽象方法只是声明，不包含方法体，就是不给出方法的具体实现也就是方法的具体功能。
* 4. 构造方法，类方法（用 static 修饰的方法）不能声明为抽象方法。
* 5. 抽象类的子类必须给出抽象类中的抽象方法的具体实现，除非该子类也是抽象类。

### 设计模式

## 数据库

### 索引的优缺点

优点：

通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

 可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

 可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

 在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

 通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能

缺点：

创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

### 索引的实现原理，为什么能加快查询速度

### 什么情况下建立索引

建立索引的几大原则

1.最左前缀匹配原则，非常重要的原则，mysql会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d的顺序可以任意调整。

2.=和in可以乱序，比如a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql的查询优化器会帮你优化成索引可以识别的形式

3.尽量选择区分度高的列作为索引,区分度的公式是count(distinct col)/count(\*)，表示字段不重复的比例，比例越大我们扫描的记录数越少，唯一键的区分度是1，而一些状态、性别字段可能在大数据面前区分度就是0，那可能有人会问，这个比例有什么经验值吗？使用场景不同，这个值也很难确定，一般需要join的字段我们都要求是0.1以上，即平均1条扫描10条记录

4.索引列不能参与计算，保持列“干净”，比如from\_unixtime(create\_time) = ’2014-05-29’就不能使用到索引，原因很简单，b+树中存的都是数据表中的字段值，但进行检索时，需要把所有元素都应用函数才能比较，显然成本太大。所以语句应该写成create\_time = unix\_timestamp(’2014-05-29’);

5.尽量的扩展索引，不要新建索引。比如表中已经有a的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可

6.Where 子句中经常使用的字段应该创建索引，分组字段或者排序字段应该创建索引，两个表的连接字段应该创建索引。

7.like 模糊查询中，右模糊查询（321%）会使用索引，而%321 和%321%会放弃索引而使用全局扫描

应该建立索引的情况

在经常需要搜索的列上，可以加快搜索的速度；

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；

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；

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；

在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；

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。

不应该建立索引的情况

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。

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。

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。

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。

### 事务、锁、sql语句

## 操作系统

### linux常用命令

### 进程、线程对比

进程和线程都是一个时间段的描述，是CPU工作时间段的描述，不过是颗粒大小不同。

进程：

1 进程是资源竞争的基本单位，比如竞争CPU的调度，以及申请内存（物理地址空间）

2 进程之间相互独立安全性高，如果两个进程之间需要进行（事件通知，数据传输，资源共享，进程控制）那么就需要通过进程间通信（管道，消息队列，共享内存，信号量等）的方式来达成。

3 进程有自己的内存，通过分页将虚拟地址空间映射到物理地址空间来存储数据

线程：

1 线程是程序运行的最小单位，线程是进程的一个分流（一个进程至少有一个线程）

2 一个进程内部的多个线程之间共享进程的数据，如果多个线程同时访问临界资源就会存在线程冲突（通过互斥锁来放置线程访问共享资源冲突的问题，有的时候互斥锁会带“死锁”和“饥饿现象”的问题），当然大多数线程内部的数据是单独享有的存储在线程栈上面。

3 线程共享进程的虚拟地址空间（共享段、数据段）、用户ID和组ID、文件描述符表、当前工作目录、但是线程也私有自己的一部分数据例如一组寄存器（用于线程切换时保存独立硬件上下）、用户栈（保存私有数据）、线程优先级等

进程process：进程就是时间总和=执行环境切换时间+程序执行时间-->CPU加载执行环境->CPU执行程序->CPU保存执行环境，实质就是一个程序开始运行到运行结束的时间，针对一个程序只创建了一个进程的情况，一个程序可能创建多个进程，这时候一个进程的结束不代表程序运行结束。

线程thread：线程也是时间总和=执行环境切换时间（共享进程的）+程序模块执行时间-->CPU加载执行环境（共享进程的）->CPU执行程序摸块->CPU保存执行环境（共享进程的）

进程和线程都是描述CPU工作的时间段，线程是更细小的时间段。Java线程实现机制是抢占式的，多个线程轮流执行，同一线程一般不会一直占用CPU执行，所以存在线程上下文的加载与保存。

一个进程实际上可以分为多个称为线程的执行单元组成，每个线程都运行在进程的上下文中，并享有代码和全局数据。操作系统保留进程所需的所有状态信息。这种状态也就是上下文。

线程用于小任务，而进程用于更多的'重量级'的任务- 应用基本执行。 一个线程和进程之间的另一个区别是，在同一进程中的线程共享相同的地址空间，而不同的进程没有。 因此线程可以读写同样的数据结构和变量，便于线程之间的通信。定义方面：进程是程序在某个数据集合上的一次运行活动；线程是进程中的一个执行路径。

### 死锁，原因，解决方法

### 多线程实现方式，创建方法，线程池大小