# Java

## 类初始化顺序

初始化父类中的静态属性 -> 父类中的静态代码块 -> 初始化子类中的静态属性 -> 子

类中的静态代码块 -> 父类中的普通属性初始化 -> 父类中的普通代码块 -> 父类中的

构造方法 -> 子类中的普通属性初始化-> 子类中的普通代码块 -> 子类中的构造方法。

## StringBuilder和StringBuilder

2. StringBuilder可变字符串，主要用于字符串的拼接，属于线程安全的；（StringBuffer的append操作用了synchronized）

3.StringBuilder可变字符串，主要用于字符串的拼接，属于线程不安全的；

String对象串联的效率最慢，单线程下字符串的串联用StringBuilder，多线程下字符串的串联用StrngBuffer；

## 数据库的数据结构

二叉树

红黑树 带平衡的二叉树

b—tree （理解： 改造的红黑树 ，把红黑树的头部 拉长 使树的高度不会太高，性能也得到巨大的优化）

b+tree mysql 的底层数据结构 改造的b—tree 索引加载到内存中比对 叶子节点存放data

mysql 数据都存储在磁盘

存储引擎

innoDB

## 数据结构

栈 后进后出网络浏览器多会将用户最近访问过的网址组织为一个栈。

场景L:

这样，用户每访问一个新页面，其地址就会被存放至栈顶；而用户每按下一次“后退”按钮，即可沿相反的次序访问此前刚访问过的页面。

类似地，主流的文本编辑器也大都支持编辑操作的历史记录功能（ctrl + z：撤销，ctrl + y：恢复），用户的编辑操作被依次记录在一个栈中。

一旦出现误操作，用户只需按下“撤销”按钮，即可取消最近一次操作并回到此前的编辑状态。

队列 先进先出

在需要公平且经济地对各种自然或社会资源做管理或分配的场合，无论是调度银行和医院的服务窗口，

还是管理轮耕的田地和轮伐的森林，队列都可大显身手。

甚至计算机及其网络自身内部的各种计算资源，无论是多进程共享的 CPU 时间，还是多用户共享的打印机，

数据结构：线性和非线性

数组，栈，队列，哈希表，树 Arraylist底层动态数组扩容

数组的优点

查找速度快

数组的缺点

插入和删除效率低

可能浪费内存

内存空间要求高，必须有足够的连续内存空间。

数组大小固定，不能动态拓展

链表的优点

插入删除速度快

内存利用率高，不会浪费内存

大小没有固定，拓展很灵活。

链表的缺点

不能随机查找，必须从第一个开始遍历，查找效率低

## Mysql索引

https://blog.csdn.net/weixin\_43591980/article/details/113833541?spm=1001.2014.3001.5501

## 乐观锁和悲观锁

悲观锁解决：

利用悲观锁的解决思路是，我们认为数据修改产生冲突的概率比较大，所以在更新之前，我们显示的对要修改的记录进行加锁，直到自己修改完再释放锁。

加锁期间只有自己可以进行读写，其他事务只能读不能写。

乐观锁解决

使用乐观锁的解决思路是，我们认为数据修改产生冲突的概率并不大，多个事务在修改数据的之前先查出版本号，在修改时把当前版本号作为修改条件，

只会有一个事务可以修改成功，其他事务则会失败。

悲观锁

优点：悲观锁利用数据库中的锁机制来实现数据变化的顺序执行，这是最有效的办法

缺点：一个事务用悲观锁对数据加锁之后，其他事务将不能对加锁的数据进行除了查询以外的所有操作，如果该事务执行时间很长，那么其他事务将一直等待，

那势必影响我们系统的吞吐量。

乐观锁

优点：乐观锁不在数据库上加锁，任何事务都可以对数据进行操作，在更新时才进行校验，这样就避免了悲观锁造成的吞吐量下降的劣势。

缺点：乐观锁因为是通过我们人为实现的，它仅仅适用于我们自己业务中，如果有外来事务插入，那么就可能发生错误。

应用场景

悲观锁：因为悲观锁会影响系统吞吐的性能，所以适合应用在写为居多的场景下。

乐观锁：因为乐观锁就是为了避免悲观锁的弊端出现的，所以适合应用在读为居多的场景下。

## 逻辑删除和物理删除

一：逻辑删除

逻辑删除的本质是修改操作，所谓的逻辑删除其实并不是真正的删除，而是在表中将对应的是否删除标识（is\_delete）或者说是状态字段（status）做修改操作。比如0是未删除，1是删除。在逻辑上数据是被删除的，但数据本身依然存在库中。

对应的SQL语句：update 表名 set is\_delete = 1 where id = 1；语句表示，在该表中将id为1的信息进行逻辑删除，那么客户端进行查询id为1的信息，服务器就不会提供信息。倘若想继续为客户端提供该信息，可将 is\_delete 更改为 0 。

二：物理删除

物理删除就是真正的从数据库中做删除操作了。

## 不同编码中文对应的字节

编码：GBK 字节数：2

编码：UTF-8 字节数：3

## BufferedReader的readline方法和read方法

Readline（）方法

1该方法读取一行文本，当遇到换行符\n,回车符\r或者\r\n回车换行，该行结束并返回。没有数据时，将会一直处于等待状态 等待一个换行/回车符而一直阻塞

。因此在进行网络连接时，应该避免使用该方法。

2、readLine()只有在数据流发生异常或者另一端被close()掉时，才会返回null值。

read()

功能：读取单个字符的个数，如果已经读完的话会返回-

　　byte[] bytes = new byte[1024];

int i = -1;

while ((i = bis.read(bytes)) != -1) {//单个读取计数，直到结束返回-1

baos.write(bytes, 0, i);

}

## Socket和http区别

socket本质是编程接口（API），对TCP/IP的封装，TCP/IP也要提供可供程序员做网络开发所用的接口，这就是Socket编程接口；HTTP是轿车，提供了封装或者显示数据的具体形式；Socket是发动机，提供了网络通信的能力。

## 静态代理和动态代理

JDK动态代理 被代理类必须实现接口

CGLIB（Code Generator Library）动态代理---（需要导入第三方包）。被代理类不需要实现接口

## 第三次握手

我觉得一般不用说标志位，面试官主要想听为什么第三次握手，两次握手不行的原因：第三次握手保证了客户端和服务端既能给对方发送消息，又能响应对方

## ArrayList的新API加上lambda表达式

从Java 8开始，使用ArrayList的新API加上lambda表达式，我们可以这样写：

list.forEach(e -> System.out.println(e));

而这里的lambda表达式的内容其实只不过就是把参数传给了println()方法，而没有做任何别的事情，所以可以进一步简写为：

list.forEach(System.out::println);

System.out::println 可以看作 lambda表达式 e -> System.out.println(e) 的缩写形式。

## maven中的三种工程：

1.pom工程：用在父级工程或聚合工程中。用来做jar包的版本控制。在pom文件中加<packaging>pom</packaging>

2.war工程：将会打包成war，发布在服务器上的工程。如网站或服务。

3.jar工程：将会打包成jar用作jar包使用。

## ##架构演进

架构的发展历程是从单体式架构，到分布式架构，到SOA架构，再到微服务架构。

1单体架构：未做任何拆分的Java Web程序

2分布式架构:按照业务垂直划分，每个业务都是单体架构，通过API互相调用

3 SOA架构：SOA是一种面向服务的架构。其应用程序的不同组件通过网络上的通信协议向其它组件提供服务或消费服务，所以也是分布式架构的一种。

4微服务架构在某种程度上是SOA架构的进一步的发展。

微服务目前并没有比较官方的定义

## ##微服务解决方案

目前最流行的两种微服务解决方案是Spring Cloud和Dubbo。

链接: [、详细来源](https://mp.weixin.qq.com/s/gktF-VYvXdpLsr74oi1msg)

## 构造方法

一个类没有定义构造方法（无参），编译器会自动为我们生成一个默认构造方法

**class Person {**

**public** Person() {

}

}

如果我们自定义了一个构造方法，那么，编译器就不再自动创建默认构造方法

## Kill tomcat 进程

输入netstat -ano|findstr 8080    8080为tomcat端口号

输入taskkill /F /PID 13756

不行 需用管理员cmd

## 多线程用while代替if

# 前端

## Es6和js 的关系

Es6 是一种规范 ，js严格执行这种规范

## ES6用法

### 关键字

Let 局部作用域

const 声明常量 一定要初始化 且不能更改

var 全局

### 解构赋值

//1、数组解构

// 传统

let a = 1, b = 2, c = 3

console.log(a, b, c)

// ES6

let [x, y, z] = [1, 2, 3]

console.log(x, y, z)

//2、对象解构

let user = {name: 'Helen', age: 18}

// 传统

let name1 = user.name

let age1 = user.age

console.log(name1, age1)

// ES6

let { name, age } = user//注意：结构的变量必须是user中的属性

console.log(name, age)

### 反引号 `

定义多行字符串，还可以在字符串中加入变量和表达式。

let age = 27

let info = ` I am ${age+1} years old next year.`

### 拓展运算符（...）

// 1、拷贝对象

let person1 = {name: "Amy", age: 15}

let someone = { ...person1 }

console.log(someone) //{name: "Amy", age: 15}

// 2、合并对象

let age = {age: 15}

let name = {name: "Amy"}

let person2 = {...age, ...name}

console.log(person2) //{age: 15, name: "Amy"}

### 箭头函数

## Vue

路由跳转

1 <router-link :to="'orderDetail'">订单详情</router-link>

2 this.$router.push({name: "orderDetail"});

this.$router.push({path: "/orderDetail"});

## 生命周期

重要的两个方法：

created 数据渲染之前

Mounted 数据渲染之后

## Node.js

1 js的运行环境，类似java的jdk环境

2 模拟服务器效果，类似tomcat

## NPM

类似maven

安装了node.js 默认就装了npm

### 项目初始化

#建立一个空文件夹，在命令提示符进入该文件夹 执行命令初始化

npm init

#按照提示输入相关信息，如果是用默认值则直接回车即可。

#name: 项目名称

#version: 项目版本号

#description: 项目描述

#keywords: {Array}关键词，便于用户搜索到我们的项目

#最后会生成package.json文件，这个是包的配置文件，相当于maven的pom.xml

#我们之后也可以根据需要进行修改。

如果想直接生成 package.json 文件，那么可以使用命令

npm init -y

### npm install命令的使用

1#使用 npm install 安装依赖包的最新版，

#安装会自动在项目目录下添加 package-lock.json文件，这个文件帮助锁定安装包的版本

#同时package.json 文件中，依赖包会被添加到dependencies节点下，类似maven中的 <dependencies>

如 npm install jquery

#如果安装时想指定特定的版本

npm install jquery@2.1.x

#devDependencies节点：开发时的依赖包，项目打包到生产环境的时候不包含的依赖

#使用 -D参数将依赖添加到devDependencies节点

npm install --save-dev eslint

#或

npm install -D eslint

#全局安装

#Node.js全局安装的npm包和工具的位置：用户目录\AppData\Roaming\npm\node\_modules

#一些命令行工具常使用全局安装的方式

npm install -g webpack

2 根据配置文件package.json 下载依赖 命令

终端命令 npm install

### 其他命令

#更新包（更新到最新版本）

npm update 包名

#全局更新

npm update -g 包名

#卸载包

npm uninstall 包名

#全局卸载

npm uninstall -g 包名

## Babel

转码器，可以将ES6代码转为ES5代码

1npm install --global babel-cli

#查看是否安装成功

babel –version

2配置

Babel的配置文件是.babelrc，存放在项目的根目录下 文件内容

{

"presets": ["es2015"],

"plugins": []

}

3在项目中安装

npm install --save-dev babel-preset-es2015

4、转码

文件转文件

babel src/example.js -o dist1/compiled.js

文件夹转文件夹

babel 要转换的文件夹路径-d 转换的文件夹路径

如babel src -d dist2

## 模块化

后端 类与类之间的调用为模块化操作

前端 js 与js调用 为模块化操作

CommonJS使用 exports 和require 来导出、导入模块。

**运行程序**

node common-js模块化/引入模块.js

ES6使用 export 和 import 来导出、导入模块。

**S6的模块化无法在Node.js中执行，需要用Babel编辑成ES5后再执行。**

## Webpack

可以将多种静态资源 js、css、less 转换成一个静态文件

## debugger 设置断点调试

## ==比较的是值 ===比较的是值和类型