# 笔记

# 基础理论

## 排序算法

选择排序:

定义 遍历数组，找出数组中最小值和第一个值进行交换，遍历剩下的数组，

找出剩余数组中的最小值与第二个值进行交换，以此类推

希尔排序

定义： 使数组中任意间隔为h的元素是有序的

冒泡排序：

从数组最左边开始，两两比较，若左边的数据小于右边，值交换， 全部比较完， 最 最右边的值是数组最大值。

依次类推

插入排序：

把数组看成有序表和无序表

开始时， 有序数组只有最左边的一个值， 无序表包含剩下的元素，

排序过程 每次从无序表中取出第一个元素，将它与有序数组中的元素进行比较，

并将其插入到指定位置

依次类推

## 名词概念

栈： 后进先出

队列：先进先出

链表：链式存储 物理上非连续 由一系列数据节点构成 --》 数据域和引用域

引用域保存了下个元素存放的地址

树：有且仅有一个根节点，该节点没有前驱节点，

其他节点有且仅有一个前驱节点， 可以m个后继节点

堆 ：

散列表： 散列函数 如果在结构中存在关键字和 T 相等的记录 在F(T)的存储位置可以 找到该记录

线性表：

n个数据元素组成的有限序列

Sdk(software Development Kit ) 软件开发工具包:

API（application Programming Interface）： 应用编程接口

Dll（Dybamic Link Library）: 动态链接库

Maven：项目对象模型

Spring boot ：

是一个框架 简化spring的搭建和开发

Shiro：

是一个安全框架， 认证 授权 会话管理

安全实体： 被保护的对象

权限： 需要被验证的权限对象

分配权限：把安全实体的某些权限分配给某些人员 权限分配

验证权限：某个人是否拥有 某些权限 权限的匹配

Hadoop：

是一个架构 做分布式系统的

什么是（服务器）集群：

很多服务器集中起来一起进行同一种服务， 看似一个服务器

优点： 提高计算速度、做备份

什么是分布式系统（distributed system）：

是一个软性系统、 建立在网络之上

优点： 高内聚、 透明性

什么是软件系统:

由系统软件, 支撑软件, 和应用软件组成的计算机软件系统

是计算机系统中由软件组成的部分

操作系统

是一种系统软件: 管理软硬件资源, 控制程序执行, 组织计算机工作流程, 为用户使用 计算机提供良好的运行环境

什么是内聚

功能角度--------》模块内的联系 模块内的功能联系

一个模块内部各元素彼此结合的紧密程度

什么是耦合

一个软件结构内不同模块之间的连接程度

什么是tcp（Transmission Control Protocol [传输控制协议](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE/9727741" \t "https://baike.baidu.com/item/TCP/_blank)）

是一种通信协议， 面向连接的、基于字节流的传输层通信协议

什么是OSI模型(Open System Interconnect 开放系统互连参考模型）

由低到高：[物理层](https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E5%B1%82" \t "https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E7%B3%BB%E7%BB%9F%E4%BA%92%E8%BF%9E%E5%8F%82%E8%80%83%E6%A8%A1%E5%9E%8B/_blank)、[数据链路层](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E9%93%BE%E8%B7%AF%E5%B1%82" \t "https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E7%B3%BB%E7%BB%9F%E4%BA%92%E8%BF%9E%E5%8F%82%E8%80%83%E6%A8%A1%E5%9E%8B/_blank)、[网络层](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%B1%82" \t "https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E7%B3%BB%E7%BB%9F%E4%BA%92%E8%BF%9E%E5%8F%82%E8%80%83%E6%A8%A1%E5%9E%8B/_blank)、[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82" \t "https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E7%B3%BB%E7%BB%9F%E4%BA%92%E8%BF%9E%E5%8F%82%E8%80%83%E6%A8%A1%E5%9E%8B/_blank)、会话层、[表示层](https://baike.baidu.com/item/%E8%A1%A8%E7%A4%BA%E5%B1%82" \t "https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E7%B3%BB%E7%BB%9F%E4%BA%92%E8%BF%9E%E5%8F%82%E8%80%83%E6%A8%A1%E5%9E%8B/_blank)和[应用层](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E5%B1%82" \t "https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E7%B3%BB%E7%BB%9F%E4%BA%92%E8%BF%9E%E5%8F%82%E8%80%83%E6%A8%A1%E5%9E%8B/_blank)。

什么是UDP（ User Datagram Protocol）用户数据报协议：

是一种无连接的[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82" \t "https://baike.baidu.com/item/UDP/_blank)协议

LDAP:

abbr. 轻量级目录访问协议（Lightweight Directory Access Protocol）

轻型目录访问协议 目录服务 轻量级目录访问协议

数据库三范式

每一列不可再分

其他列必须完全依赖主键列

每个属性都和主键有直接关系

http（超文本传输协议）： 浏览器和服务器的 应用层 通信协议

一种 请求-响应 协议 客户端发请求，服务端响应 采用tcp连接 默认端口 80

http请求包含三部分：

请求方法：

请求头：

请求体：

http响应：

协议-状态码

响应头

响应体

https： 安全的http

SSL(Secure Socket Layer ) 安全套接字: 位于TCP/IP协议和 各种应用层协议之间

TLS（Transport Layer Security，传输层安全）：SSL的升级版

TCP(transmission control protocol )传输层控制协议 :

IP（ internet Protocol 网络之间互联的协议）：

对称加密算法：加密和解密使用相同[密钥](https://baike.baidu.com/item/%E5%AF%86%E9%92%A5" \t "https://baike.baidu.com/item/%E5%AF%B9%E7%A7%B0%E5%8A%A0%E5%AF%86%E7%AE%97%E6%B3%95/_blank)的加密算法

非对称加密算法： 公开密钥，私有秘钥 公开秘钥加密， 私有秘钥解密

同步和非同步：多个任务发生时，一个事件的发生是否会导致整个流程的暂时等待

阻塞和非阻塞：当发出一个请求操作时，如果条件不满足，是会一直等待还是返回一个标志 信息

阻塞IO和非阻塞IO： 查看数据是否就绪的过程中，是一直等待还是返回一个标志信息

同步IO和异步IO：

函数接口：内部只有一个抽象方法的接口

Nginx：是一个高性能的http 和 反向代理服务器

反向代理：以代理服务器获取internet请求，转发给内网服务器，并从内网服务器获得结果 返回客户端

正向代理：

Socket：套接字

使程序可以从网络上读取数据，也可以向网络上写入数据

Socket: 客户端套接字

ServerSocket: 服务端套接字

# Redis

简单动态字符串（simple dynamic string ） SDS

空间预分配：

用于优化SDS的字符串增长操作：

SDS修改操作，对SDS进行扩展，程序不仅会为SDS分配修改所需的必要的空间

还会为SDS分配额外的未使用空间

惰性空间释放：

SDS字符串缩短操作：

对SDS进行缩短，程序并不立即使用内存重分配来回收缩短后多出来的字节，

而是使用free属性将这些字节数量记录下来，等待将来使用

SDS会以处理二进制的方式处理SDS存放在buf数组的数据

链表：

列表键的底层实现之一就是链表

字典:

redis数据库就是使用字典来做为底层实现的

字典还是hash键的底层实现之一

跳跃表：

在每个节点中维持多个指向其他节点的指针，从而达到快速访问节点的目的

redis使用跳跃表作为有序集合键的底层实现之一

redis中用到跳跃表的地方：

有序集合

集群节点中用作内部数据结构

整数集合：

整数集合是集合键的底层实现之一

一个集合只包含整数元素，元素数量不多时，采用整数集合

压缩列表：

是列表键和hash键的底层实现之一

对象：

字符串对象：

编码： int 、raw、embstr

embstr:专门用于保存端字符串的一种编码优化

raw: 字符串、长度大于32字节 值用sds保存，编码应raw

列表对象：

编码： ziplist.linkedlist

ziplist编码的列表使用压缩列表作为底层实现

linkedlist编码的列表使用双端链表作为地层实现

哈希对象：

编码：ziplist 、 hashtable

ziplist编码的哈希对象用压缩列表作为底层实现

hashtable : 字典

有序集合对象：

编码: ziplist 、skiplist

ziplist编码的有序集合对象用压缩列表作为底层实现

skiplist：使用zset结构作为底层实现，

一个zset结构同时包含一个字典 和一个跳跃表

redis过期键的删除策略：

定时删除：（暂时不用）

设置过期键的同时，创建一个定时器，键过期，定时器执行删除操作

优缺点：

对内存是友好的

对cpu时间不友好

惰性删除：

放任过去键不管，每次获取键时，判断是否过期

优缺点：

对cpu时间是有好的

对内存时不友好的

定期删除：

每隔一段时间，对数据库进行检查，删除过期键

上面两个方案的折中

AOF、RDB和复制功能对过期键的处理

RDB持久化

通过保存数据库中的键值对来记录数据库状态状态不同

save: 由服务器进程执行保存工作

bgsave:由子进程执行保存工作

AOF（append only file）持久化

保存redis所执行的写命令来记录数据库状态

3种实现

命令追加：

文件写入：

文件同步：

事件：

文件事件的处理器构成

套接字

IO多路复用程序：

文件事件分派器：

事件处理器：

# 深入理解java虚拟机

Java虚拟机栈：java方法执行的内存模型

本地方法栈： 为虚拟机使用到的native方法服务

Java堆：存放对象实例，

方法区：已被虚拟机加载的类信息、常量。静态变量、即时编译器编译的代码

Java多线程：线程轮流切换并分配处理器执行时间实现的

内存泄漏：Memory Leak

内存溢出：Memory overflow

class对象：用来创建所有类的“常规”对象

对一个方法而言，参数是为该方法提供信息的，而不是让方法改变自己值的

## 自动内存管理机制：

### java内存区域与内存溢出异常

#### 程序计数器（program counter register）：

当前线程所执行的字节码的行号指示器

字节码解释器：

通过改变程序计数器的值来选取下一条需要执行的字节码指令

###### java虚拟机栈（java virtual machine stacks）：

为虚拟机执行java方法（字节码）服务

线程私有，生命周期和线程相同

方法执行时创建一个栈帧

栈帧：存储局部变量表、操作数栈、动态链接。方法出口等

局部变量表：各种基本类型、对象引用

##### 本地方法栈

为虚拟机使用的Native方法服务

##### java堆

线程共享、存放对象实例、垃圾回收器管理的主要活动区域

新生代 ：

老年代：

##### 方法区

线程共享、存放被虚拟机加载的类信息、常量、静态变量、即时编译后的代码等

## 第三章垃圾收集器与分配策略

引用计数法：

标记-清除 算法：

复制算法：

标记-整理 算法：

吞吐量：CPU用于运行用户代码的时间与cpu总耗时时间的比值

jps: 虚拟机进程状态工具

列出正在运行的虚拟机进程

jstat(JVM statistics Monitoring tool )：虚拟机统计信息监视工具

显示本地或者远程虚拟机进程中的类加载、内存、垃圾收集器等运行时数据

Jinfo：java 配置信息工具

实时查看和调试虚拟机的各项参数、

JConsole :java 监视与管理控制台

类文件结构：

class文件:一组以8位字节为基础单位的二进制流

# 深入分析javaWEB

## 深入WEB请求过程

B/S:浏览器服务器

用户输入url 发生的操作：

DNS 把域名解析成对应的IP

根据IP在找服务器

向服务器发送请求

服务器发送响应给用户

发起一个HTTP请求：

本质是建立一个socket连接

HTTP 解析：

http请求头和响应头、状态码

DNS域名解析：

作用：将域名解析成IP

CDN 工作机制（内容分布网络 Content Delivery NetWork）

作用： 缓存访问静态数据 eg： css、js、 图片

负载均衡（Load balance）：平衡工作任务

链路负载均衡

DNS解析成不同的ip，根据ip访问不同的服务器

直接访问目标服务器，速度快

某台服务器挂了，很难及时更新数据

集群负载均衡 （常见）

硬件负载均衡

性能好

硬件价格贵

软件负载均衡

成本低

需要代理服务器，增加网络延时

操作系统负载均衡

利用操作系统级别的中断实现

## JAVA I/O的工作机制

基于字节操作的I/O接口：InputStream 、Outputstream

基于字符操作的I/O接口：Writer 、Reader

基于磁盘操作的I/O接口：File

基于网络操作的I/O接口：Socket

响应时间：将一份数据从一端正确传送至另一端

JAVA Socket 工作机制：

socket: 完成计算机通信

同步和异步：

同步：一个任务的进行需要等待前一个任务的完成

异步：一个任务的进行不需要等待前一个任务的完成

阻塞和非阻塞：

阻塞： cpu停下来等待其他任务的完成

非阻塞：cpu无需停下来，继续执行

javac 编译原理

作用：将java语言规范转换为java虚拟机语言规范

javac编译器处理阶段

词法分析、语法分析、语义分析、代码生成

类加载器的工作机制：

作用：

将class加载到jvm中

判定类由那个类加载器加载

将字节码重新接线成java要求的格式

上级委派机制：

JVM体系详解

类加载器：

jvm启动或类运行时，将class加载到jvm

执行引擎

执行class 文件中包含的字节码指令

内存区：

存储、记录、调度功能模块

本地方法调用：

指令集：

cpu用来计算和控制计算机系统的一套指令的集合

# Spring in Action

Spring的核心策略：

基于pojo的轻量级和最小侵入性编程

通过依赖注入和面向接口实现松耦合

基于切面和惯例进行声明式编程

通过切面和模板减少样板式代码

IOC 控制反转

通过描述来生成或者获取对象

作用：

通过描述或注解 发布和获取bean

通过描述或注解，完成bean之间的依赖关系

DI：

组件之间松耦合

AOP：

功能分离，可重用

### 重新装配bean

装配（wire）：

应用对象之间的协作关系

DI的本质

spring 装配bean的三种方式

xml 显式配置

java 显式配置

注解隐式发现和自动装配

### 高级装配

Profile:

可以灵活的去装配某个bean （是否装配，什么条件去装配）

bean装配有歧义解决方案：

标示首选的bean @primary

限定自动装配的bean ： 使用自定义注解限定

bean的作用域：

@Scope

spel spring 表达式语言

## 面向切面的spring

通知：

切面的工作，称之为通知

切点：

匹配通知所要织入的一个或多个连接点

连接点：

执行过程中可以插入切面的一个点

切面：

通知与切点的结合

引入：

向现有类添加新的方法或属性

织入：

把切面应用到目标对象

通知包含了应用于对象的横切行为，连接点是程序中可以应用通知的方法

切点定义了连接点的具体位置，那些连接点会得到通知

spring 4中类型的aop支持

基于代理的经典 spring aop

纯pojo切面

@Aspectj 注解驱动

注入式Aspectj 切面

spring aop 基于动态代理

## WEB中的spring

### 第七章：Spring mvc 的高级应用

#### spring web flow

就是一个简单的流程管理系统 类似 activiti

#### spring Security

## 后端中的spring

### spring 之JDBC

### 对象-关系映射持久化数据库（hiberbate和jpa）

REST （Representational State Transfer）：

将资源的状态以最合适的形态在客户端和服务端转换

RPC（remote Procedure Call）：远程过程调用

JMS(java message Service): java 消息服务

ActiveMq

AMQP（Advance Message Queue Protocol）: 高级消息队列

RabbitMQ

同步和异步消息 优缺点

Jms的两种模式 点对点， 发布订阅 ActiveMQ

# 高效能程序员的修炼

程序员的八种境界

（不朽、成功（比尔盖茨）、知名、胜任、普通、业余、低调、烂程序员）

培养写作

为什么会有这样的问题

把问题用通俗易懂的话写出来

然后再去说求助的问题

## 第二章： 搞定手头工作

学习、保持热情

以兴趣为导向，踏实努力

学会反思、讨论和学习

尽量一次只做一件事

## 第三章：高效编程的原则

程序出问题，首先找自己的原因

精简代码完成功能（永远都有简化空间）

避免写注释

学会读源代码

加入一个优秀的团队

性能才是王道

## 第四章：招聘程序员需得其法

#### 编码部分

编程

面向对象

脚本编程或正则

数据结构

#### 推销自己

介绍自己

介绍做过的项目

介绍某些知识点及优缺点

如何解决问题

解题思路

## 第五章：团队紧密协作

去掉团队中的混子或者害群之马

避免被团队“枪杀”

团队中良好的交流

结对编码与代码评审

远程办公

## 第六章：工作场所

用不恰当的比较来误导用户

高价格和低价格的比较

利用消费惯性

免费的诱惑

假借社会行为准则

故意允许拖延

利用人的损失厌恶

制造不合理的期望

利用对价格的偏见

## 第七章： 设计时从用户的角度去思考

1、做好细节

程序是所有微笑细节的集合。

2、用户界面代表了软件。—重要性

（如果你的系统有一个糟糕的界面，你基本上一无所有了。哪怕系统内部使用了最高深 的技术，那也于事无补。）

用户界面须优先设计。写代码前，就要想清楚这个程序的用户界面看起来是什么样子的。

纸上画原型设计时一种永远不会过时的方法。

3、分页显示该休矣

一旦你有几千条信息，你要解决的不是分页问题，而是搜索和过滤的问题。

我们应该避免没头没脑地生成一个包含有成千上万个条目的列表，然后用“一刀切”的方法把他们分页显示出来。这样就是把所有负担扔给了用户。这是不合理的。记住！我们发明计算机是为了让人们的生活变得更轻松，而不是更难。

总之，你应该努力不扯上分页这玩意，因为你应该让用户在几个条目中就能找到他们需要的东西。这是高于一切的

## 保护用户数据

所有的网络通信都应该加密码

防范字典式攻击

每个网站采用不同的强类型密码，采用相同的很容易被破解

## 第九章： 代码测试

从客户的角度去思考程序

开发期间出问题是好事（虽然应该尽量避免问题）

代码评审

加大测试力度

可用性测试

# 数据结构 C语言版

1.1数据结构：

定义：研究 对象及其之间关系和操作

结构：

元素之间的关系

数据结构的形式定义：

Date\_Structure = (D,S)

D: 数据元素的有限集

S: D上关系的有限集

数据的逻辑结构

数据之间的逻辑关系

存储结构

数据结构在计算机中的表示

算法的设计取决于逻辑结构，实现依赖于存储结构

抽象数据类型（ADT）

一个数学模型，以及定义在该模型上的一组操作

算法和算法操作：

算法： 对问题求解步骤的描述

度量算法效率：

1. 算法策略
2. 问题规模
3. 书写语言
4. 编译之后的代码质量
5. 机器执行指令的速度

# Effective JAVA

掌握一门语言的三件事情

这门语言的结构（语法）

如何命名你想谈论的事物（词汇）

如何高效表达日常的事物（用法）

## 第二章： 创建和销毁对象

1. 考虑用静态工厂方法代替构造器
2. 遇到多个构造器参数时要考虑构建器
3. 用私有构造器或枚举类型强化Singleton属性
4. 通过私有构造器强化不可实例化的能力
5. 避免创建不必要的对象
   1. 优先使用基本类型而不是装箱基本类型
   2. 当心无意识的自动装箱
6. 消除过期的对象引用
   1. 只要类是自己管理内存，应该警惕内存泄漏
   2. 内存泄漏的另一个常见的来源是缓存
   3. 监听器和回调
7. 避免使用终结方法
   1. 使用终结方法浪费性能
   2. 通常与try-catch-finally 结构结合起来使用
   3. 终结方法：不可预测的，很危险，一般情况下不必要的
   4. 典型例子：inputstream和outputstream的close()方法

## 第三章：对所有对象通用的方法

1. 覆盖equals请遵守约定
   1. 使用==操作检查 参数是否为这个对象的引用
   2. 使用instanceof操作符判断，参数是否为正确的类型
2. 覆盖equals时总要覆盖hashcode
3. 始终要覆盖tostring()
4. 谨慎的覆盖close
5. 考虑实现comparable接口

## 第四章：类和接口

1. 使类和成员的可访问性最小化
   1. 设计良好的模块会隐藏所有的实现细节
   2. 把它的api和实现清晰的隔离开
   3. 使每个类或成员不被外界访问
2. 在公有类中使用访问方法而非公有域（属性）
3. 使可变性最小化
   1. 不可变类是其实例不可被修改的类
   2. 使类成为不可变类的5条规则
      1. 不要提供任何会修改对象状态的方法
      2. 保证类不会被扩展
      3. 使其所有域都为final的
      4. 使所有域都成为私有的
      5. 确保对于任何可变组件的互斥访问
4. 复合优先于继承
   1. 继承： 一个类扩展一个类
      1. 打破了封装性
      2. is-a 的时候使用继承 类B应该扩展类A吗
      3. 子类的某些实现依赖于父类
   2. 复合
      1. 把现有类当做新类的一个组件

装饰器模式：对一个现有的类进行修饰

1. 要么为继承而设计，要么提供文档说明，要么禁止继承
2. 接口优先抽象类
3. 接口值用于定义类型
4. 类层次优先于标签类
5. 用函数对象表示策略
   1. 策略模式：通过传递不同的参数，获取不同的结果
   2. 函数对象：一个类只有一个方法
6. 优先考虑静态成员类
   1. 嵌套类:定义在类内部的类
      1. 目的：为它的外围类提供服务
   2. 嵌套类的四张
      1. 静态成员类
         1. 可以访问外部类的所有成员 包括私有成员
         2. 它的作用和 类中静态成员一样
      2. 非静态成员类
         1. 访问非静态成员类需要外部类的一个实例
      3. 匿名类
         1. 需要在一个地方创建实例，并且已经有了预置类型说明这个类的特征
         2. 用法1、创建函数对象
         3. 创建过程对象
         4. 在静态工厂方法内部
      4. 局部类

## 泛型

1. 不要在新代代码中使用原生态类型
   1. List<E> 新生代 List 原生态
2. 消除非受检警告
   1. Set<A> set = new HashSet<>(); 不建议
   2. Set<A> set = new HashSet<A>(); 建议
   3. @Suppress Warnings 注解
3. 列表优先数组
4. 优先考虑泛型
5. 优先考虑泛型方法
   1. 静态工具方法尤其适合泛型
6. 利用有限制性通配符提升api的灵活性
   1. 参数化类型是不可变的
   2. *？extends E : ？代表的类型是 E的子类型*
   3. *？supperE : ？代表的类型是 E的父类型*
7. 优先考虑类型安全的异构容器
   1. String.class 属于 Class<String> 类型
   2. Integer.class 属于class<Integer> 类型

## 第六章 枚举和注解

1. 用enum代替int常量
   1. 枚举：一组固定的常量组合成合法值的类型
2. 用实例域代替序数
3. 用EnumSet代替位域
   1. 位域：用Or位运算符将几个常量合并到一个集合中
4. 用EnumMap代替序数索引
5. 用接口模拟可伸缩的枚举
6. 注解优先于命名模式
   1. 命名模式：表示某些程序元素需要通过某种工具或框架进行特殊处理
7. 坚持使用override注解
8. 用标记接口定义类型
   1. 标记接口：没有包含方法声明的接口

## 第七章\_方法

1. 检查参数的有效性
2. 必要时进行保护性拷贝（不太懂）
3. 谨慎的设计方法签名
   1. 缩短过长参数列表的三个方法
      1. 把方法分为多个方法
      2. 创建辅助类
      3. 从对象构建到方法调用都采用builder模式
4. 慎用重载
   1. 重载方法是在编译期间就决定的
   2. 重载方法的选择是静态的，覆盖方法的选择是动态的
5. 慎用可变参数
6. 返回0长度的数组或者集合，而不是null
7. 为所有导出api元素编写文档注释

## 第八章\_通用程序设计

1. 将局部变量的作用域最小化
2. for-each 循环优先于传统的for
3. 了解和使用类库
4. 如果需要精确的答案，请避免使用float和double
5. 基本类型优先于装箱基本类型
6. 如果其他类型更适合，请避免使用字符串
7. 当心字符串连接的性能
8. 通过接口引用对象
9. 接口优先于反射机制
10. 谨慎的使用本地方法
11. 谨慎的进行优化
12. 遵守普遍接受的命名惯例

## 第九章——异常

1. 只针对异常的情况才使用异常
2. 对可恢复的情况使用受检异常，对编程错误使用运行时异常
3. 避免不必要的使用受检异常
4. 优先使用标准异常
5. 抛出与抽象相对应的异常
6. 每个方法抛出的异常必须要有文档
7. 在细节消息中包含能捕获失败的信息
8. 努力使失败保持原子性
9. 不要忽略异常

## 第十章——并发

1. 同步访问共享的可变数据
2. 避免过度同步
3. executor和task优先于线程
4. 并发工具优先于wait和notify
5. 线程安全性的文档化
6. 慎用延迟初始化
7. 不要依赖于线程调度器
8. 避免使用线程组

# 第十一章-序列化 （待）

1. 谨慎的实现Serializable接口
   1. 序列化：将对象编码成字节流，并从字节流编码重新构建对象
2. 考虑使用自定义的序列化形式
3. 保护性的编写readObject方法
4. 对于实例控制，枚举类型优先于ReadREsolve
5. 考虑用序列化代理代替序列化实例

# 计算机科学概论

## 基础知识

计算系统（computing system）： 通过交互解决问题的计算机硬件、软件和数据

计算机硬件（computer hardware） : 计算系统的物理原件

计算机软件（computer software） ：提供计算机执行的指令程序

协议：是一套规定，必须严格遵守的规则和过程的代码

抽象：删除了复杂细节的心里模型

算法是解决问题的步骤的逻辑序列

算法思想：能够用按部就班的过程表示问题，从而解决它们

表示法： 用能被有效处理的方式存储数据

程序设计：把算法思想和表示法组织到就是年纪软件中

设计： 使软件满足一种用途

数据：基本值或事实 未处理

信息： 用有效的方式组织或处理过的数据

数据压缩（data compression）：减少存储一段数据所需的空间

多媒体：不同类型的媒体

带宽(band width)：在固定时间内从一个地点传输到另一个地点的最大位数或字节数

压缩率：压缩程度--》原始数据的大小除压缩后的数据大小

有损压缩：会丢失信息的数据压缩技术

无损压缩：不会丢失信息的数据压缩技术

模拟数据和数字数据

模拟数据（analog data）：用连续形式表示的信息

数字数据（digital data）：用离散形式表示的信息

数字化（digitize）： 把信息分割成离散的片段

脉冲编码调制（pluse-code-modulation）：电信号在两个极端之间跳跃的变化

重新计时（reclock）：在信号降级太多之前将它重置为原始状态的行为

n位2进制数字能表示 【2的n次方】 种状态

分配给任何类型的数据的最小存储量通常是2的幂的倍数

负数表示法： 补码的形式

二进制补码: 负数的最左边一位总是1 正数的二进制 源码的反码 +1

实数表示法（小数表示法） --暂略过

溢出（overflow）： 给结果预留的位数存不下计算出的值的状况

字符集(character set):字符和表示他们的代码的清单

## 第一章:简介

### 描述计算机系统的分层

由内而外的介绍计算系统

信息层：计算机上表示信息的方式（1和0）

硬件层：计算机系统的物理硬件

程序设计层:负责处理软件，用于实现计算的指令以及管理的数据

操作系统层： 管理计算机的资源

应用程序层： 解决真实世界的问题

## 第二章：信息层

### 二进制数据和计数系统

位（二进制数字）：二进制系统中的一位数字 0 或1

1字节 = 8位 ：

字： 一个或多个字节， 字中的位数称为计算机的字长

### 数据表示法

文本压缩类型：

关键字编码（keyword encoding）：

用单个字符代替常用的单词

行程长度编码(run length encoding):

把一系列重复字符替换为它们重复出现的次数

赫夫曼编码：

用变长的二进制串表示字符，使常用的字符具有较短的编码

音频信息表示法：

声音：实际上是由与我们的耳膜交互的声波定义的

音频在计算机中的表示：

数字化声波，采集表示声波的电信号，用一系列离散的数值表示它

图形和图形表示法

颜色表示法：表示颜色的数据量

像素（pixels）：用于表示图形的独立点，代表图形的元素

数字化一幅图片： 将图片表示为一套独立的点（像素）

分辨率:一幅图像使用的像素个数

光栅图形格式：逐个像素存储图像信息的格式

GIF（Graphics InterchangeFormat）：图形交换格式

适合颜色较少的图像和图像，是存放线条图像的首选格式

jpeg：

存储照片颜色图像的首选格式

png（Portable network Grapbics）：可一直的网络图像文件格式

矢量图形（vector graphics）：

用线段或集合形表示图形的方法

不适用于:真实世界的图像

适用于：艺术线条和卡通绘画

SVG（scalable Vector graphics）：可缩放矢量图形

视频表示法：

视频编译码器（video codec）：

用于缩减电影大小的方法

视频变译码器的两种压缩方式

时间压缩

根据连续帧之间的差别压缩电影的技术

空间压缩

基于结晶体图形的压缩方法的电影压缩技术

## 第三部分: 硬件层

## 计算机和电学

门（gate）：是对电信号执行基本运算的设备

接收多个输入信号，生成一个输出信号

电路（circuit）：相互关联的门的组合，用于实现特定的逻辑运算

布尔代数（boolean algebra）：

表示二值逻辑函数的数学表示法

逻辑框图（logic diagram）：

电路的图形化表示，每种类型的门都有自己的专业符号

真值表（true table）：

列出了所有可能的输入值和相关的输出值的表

门：

非门（not）

与门（and）

或门：（or）

异或：（XOR）

输入相同则为0， 不同则为1

或非门：（）：对与的运算求逆

与非门：（）：对或的运算求逆

晶体管：作为电线或电阻器的设备，有输入信号的电压点评决定它的作用

半导体：既不是良好的导体也部署绝缘体的材料

组合电路：输出仅由输入值决定的电路

时序电路：输出是输入值和当前状态的函数的电路

电路等价：对应每个输入值的组合，两个电路都生成完全相同的输出

加法器：对二进制值执行的加法运算的电路

半加器：计算两个数位的和并生成正确进位的电路

全加器: 计算两个数位的和，并考虑进位输入的电路

多路复用器：

使用一些输入控制信号决定用哪条输入数据线发送输出信号的电路

## 计算部件

### 冯。诺依曼体系结构

内存部件：存放数据和指令的内存部件

算术逻辑部件: 对数据执行算术和逻辑运算

输入部件：把数据从外部世界转义到计算机中的输入部件

输出部件：

控制器：控制其他部件参与计算机的运行

核心：线性读取，执行周期

内存：存储单元的集合

可编址性：内存中每个可编址位置存储的位数

n位的处理器可以产生2的n次方个不同的地址

算术逻辑部件：

执行算术运算和逻辑运算的计算机部件

寄存器 ：cpu上的一小块存储区域，用于存储中间值或特殊数据

输入输出部件是计算机与外部世界交流的渠道

控制器：控制其他部件的动作，从而执行指令序列的计算机部件

指令寄存器：存放当前执行的指令的寄存器

程序计数器：存放下一条要执行的指令的寄存器

中央处理器（cpu）：

算术逻辑部件和控制器的组合，是计算机用于解释和执行指令的大脑

总线：把机器的主要组成部分连接在一起的一组电线，数据在这组电线中流动

主板： 个人计算机的主电路板

RAM：随机存储器

存储单元可以直接被访问（改写存储单元的内容）

ROM：只读存储器

ROM的内容不能更改，是永久的

二级存储设备

## 程序设计层

### 问题求解和算法设计

问题求解（problem solving）：

找到令人感到困惑的难题的解决方案的行动

如何解决问题

理解问题

找到已知信息和解决方案之间的联系，找不到直接，考虑辅助或间接问题

执行方案

分析得到的解决方案

分治法：

将一个大问题划分为几个可以解决的小的单元

算法（algorithm）：

在有限的时间内用有限的数据解决问题或子问题的明确指令集合

把问题转换为方案的方法

自顶向下设计（功能分解）：

把问题分解成更容易处理的子问题

将这些子问题的解决方案组合起来，构成整体问题的解决方案

模块：一个用于解决问题或子问题的封闭步骤集合

抽象步骤：细节仍未明确的算法步骤

具体步骤：细节完全明确的算法步骤

自顶向上的方法功能分解

分析问题

编写主要模块

编写其余模块

根据需要进行重组和改写

面向对象设计（OOD）：

对象是由数据和处理数据的操作组成

设计重点：

对象以及它们在问题中的交互作用

类：一组具有相似的属性和行为的对象的描述

对象： 类得具体实例

封装： 把数据和行为集中在一起，使数据和行为的逻辑属性与他们的实现细节分离

信息隐蔽：隐蔽模块的细节以控制对这些细节的访问的做法

信息隐藏:隐藏子任务细节的过程

抽象：隐藏细节的结果

数据抽象:把数据的逻辑概观和它的实现分离开

过程抽象：把动作的逻辑概观和它的实现分离开

控制抽象：把控制结构的逻辑概观和它的实现分离开

控制结构：用于改变正常的顺序控制流的语句

## 内存管理：

页式内存管理法

把进程划分为大小固定的页，载入内存中存储在帧中的内存管理方法

帧： 大小固定的一部分主存，用于存放进程页

页：大小固定的一部分进程，存储在内存帧中

页映射表: 操作系统用于记录页和帧直接的关系的表

请求分页：页式内存管理法的扩展，只有当页面被引用（请求）时才会被载入内存

页面交换：把一个页面从二级存储设备载入内存通常

虚拟内存：由于整个程序不必同时处于内存而造成程序大小没有限制的假象

进程控制块（PCB）：操作系统管理进程信息使用的数据结构

上下文切换：当一个进程移出cpu，另一个进程取代它时发生的寄存器的交换

cpu调度：cpu调度算法决定 将cpu给与那个进程

非抢先调度： 一个进程执行完才可以执行下一个进程

抢先调度： 一个进程在执行期间，其他进程可以抢夺cpu

周转周期：进程进入准备就绪状态到它完成之间的时间间隔

时间片: cpu循环调度算法中分配给每个进程的时间量

cpu调度算法：

先到先服务算法：

最短作业优先

循环调度算法：

把处理时间平均分配给所有准备就绪的进程

## 文件系统和目录

### 文件系统

主存：存放活动的程序和正在使用的数据

二级存储设备：永久性

文件：数据的有名集合，用于组织二级存储设备

文件系统：操作系统管理文件的逻辑视图

目录：文件的有名集合

#### 文本文件和二进制文件

文本文件：包含字符的文件

二进制文件：包含特定格式的数据文件，要求给位串一个特定的解释

文件类型：文件中存放关于类型的信息

文件扩展名：文件名中说明文件类型的部分

顺序文件访问法：以线性方式访问文件中的数据的方法

直接文件访问法：通过指定记录编号直接访问文件中的数据的方法

### 磁盘调度

磁盘调度：决定先满足那个磁盘IO的请求操作

磁盘算法

先到先服务磁盘算法

最短寻道时间有限磁盘算法

SCAN磁盘调度算法

## 应用程序层

### 信息系统：

信息系统：组织和分析数据的软件

### 模拟、图形学和其他应用程序

模拟：设计复杂系统的模型，并为观察结果而对该模拟进行试验

### 通信层

带宽：数据从网络中的一个地点传输到另一个地点的速率

协议：网络上格式化和处理数据的一组规则

局域网（LAN）：连接较小地理范围内少量的计算机网络

广域网（WAN）：连接两个或多个局域网的网络

网关：处理它的Lan 和其他网络通信的节点

包：网络上传输的数据单位

包交换：把包单独发送到目的地然后再组装起来的网络通信技术

路由器：指导包在网络上向最终目的地传输的网络设备

中继器：加强和传播信号的网络设备

TCP(传输控制协议)：把消息切割成包，在目的地把包组装成消息，

并负责处理错误的网络协议

网际协议（IP）：处理包通过互相连接的网络传输到最终目的地的路由选择

UDP（用户数据报协议）：tcp的替代品，牺牲了一定的可靠性实现较高传输速率

防火墙：网关机器，过滤网络通信保护网络

访问控制策略：规定接受和拒绝什么类型的网络通信

域名系统：管理主机名解析的分布式系统

域名服务器：把主机名翻译成ip地址的计算机

# GOF23

GOF23:

单一职责原则 （SRP:Simple Responsibility Principle）：

定义： 应该有且仅有一个原因引起类的变更

里式替换：

所有引用基类的地方必须能透明的使用其子类的对象

依赖倒置原则：

高层模块不应该依赖低层模块，两者都应该依赖其抽象；

抽象不应该依赖细节；

细节应该依赖抽象

在Java中的体现

模块间的依赖通过抽象发生，实体类之间不会发生直接的依赖关系

其依赖关系是通过接口或抽象类产生的

接口或抽象类不依赖于实现类

实现类依赖接口或抽象类

接口隔离

接口尽量细化，同时接口中的方法尽量少

单一职责要求的是类和接口职责单一，注重的是职责，这是业务逻辑上的划分，

而接口隔离原则要求接口的方法尽量少。

迪米特法则：

最少知识原则。通俗的讲：一个类应该对自己耦合或者调用的类，知道的越少越好

开闭原则

对修改关闭，对扩展开放

GOF23：

单例设计模式：

确保某一个类只有一个实例，并且自行实例化并向这个系统提供这个实例

模板方法模式：

定义：定义一个操作中的算法框架，将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可定义该算法的某些特定步骤

钩子方法：在抽象类中某个方法的返回值影响了模板方法的执行结果

建造者模式：

将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示

建造者模式的重点：基本方法的调用顺序的安排，即零件的装配

工厂方法的重点：创建零件

代理模式：

为其他对象提供一种代理，以控制对这个对象的访问

动态代理：

在实现阶段不用关心代理谁，在运行阶段才指定代理哪一个对象

原型模式：

用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象

中介者模式：

用一个中介对象封装一系列的对象交互，中介者使各对象不需要显示的相互作用

从而使其耦合松散，而且可以独立的改变他们之间的交互

命令模式：

将一个请求封装成一个对象，从而让你使用不同的请求把客户端参数化

对请求排队或者记录请求日志，可以提供命令道撤销和恢复

责任链模式：

使多个对象都有机会处理请求，从而避免了请求的发送者和接收者之间的耦合关系

将这些对象连城一条链，并沿着这条链传递该请求，直到有对象出来它为止

装饰器模式：

动态的给一个对象添加一些额外的职责，

策略模式：

定义一组算法，将每个算法都封装起来，并且使他们之间可以互换

适配器模式：

将一个类的接口，转换成客户端期望的另一种接口，从而使原本因接口不匹配

而无法在一起工作的两个类能够在一起工作

迭代器模式：

它提供一种方法访问一个容器对象中各个元素，而又不需暴露该对象的内部细节。

# Java编程思想

## 第一章 对象导论

oop(Object oriented Programming) 面向低下设计

UML 统一建模语言

根据问题来描述问题

基本特性

万物皆对象

程序是对象的集合，它们通过发生消息来告知彼此所要做的

每个对象都有自己的有其他对象所构成的存储

每个对象都有其类型

某一特定类型的所有对象都可以接收同样的消息

数据类型 == 类（class）

抽象数据类型 是面向对象的基本设计之一

接口确定了对某一对象发送的请求

理解程序设计：

把对象当成服务的提供者

把问题分解为对象集合

对象的存储位置

寄存器

无法操作，按需分配

堆栈：

对象的引用

堆：

存放所有的java对象

常量存储：

常量值 存在在rom（只读存储）中

非RAM存储

流对象和持久化对象

组合与继承：

继承：” is a”(是一个) 的关系

组合：has a （有一个） 的关系

需要的是继承还是组合

如果必须要向上转型 ----》继承

向上转型： 导出类转型成基类 是安全的

Final 作用在基本类型----》值不可以改变

作用在引用类型---》引用地址不可变

作用在方法上：方法体继承类不可修改

作用在类上： 类不可以被继承

绑定：将一个方法绑定与一个方法主体关联起来

Java 所有方法 通过动态绑定实现多态

构造器内，尽量避免调用其他方法

泛型：

用来指定容器要持有什么类型的对象

泛型的核心概念：

告诉编译器你想使用什么类型，编译器帮你处理一切细节

高可用：无论发生什么情况，系统依旧正常

负载均衡：将请求或者流量，以期望的规则分摊到多个操作单元上进行执行

使用接口的原因：可以向上转型为多个基类、

java编程思想

OOP（Object Oriented Programming）：面向对象程序设计

UML（Unified modelling Language ）：统一建模语言

CGI（common gateway interface）：通用网关接口

向上转型：把子类看成是父类的过程

重载：方法名相同，参数列表不同

多态：

内聚：一个软件构建的的各个方面的组合程度

程序可以通过添加新类型的对象使自身适用于某个特定的问题

面向对象的五个基本特征:

万物皆对象：抽取待求解问题的概念化构架，将其作为对象

程序是对象的集合，他们通过发消息告知彼此需要做的事情

把消息当做对象某个特定对象的方法的调用请求

每个对象都有自己的由其他对象所构成的存储

构建包含现有对象的方式创建一个新的类型的对象

每个对象都有其类型：每个对象都是某个类的实例

对某一特定类型的所有对象都可以接收同样的消息：

调用父类方法可以实现的功能，调用子类方法也一定可以

类： 具有相同特性（数据元素）和行为（功能）的对象集合

如果可以将问题从表象中抽取出来，那么什么样的问题可以马上解决我的问题

将对象看做服务的提供者

访问控制：

public: 任何人可用

private：类型的创建者和类型的内部方法

protected:类型的创建者和类型的内部方法 子类也可使用

# 计算机系统

## 基础知识

位（比特）： 由值0、1组成

字节： 8个比特 --》每个字节表示文本字符

位运算：

与（雨点）：有0为0

或（伙食）：有1为1

异或（疑惑）： 相同出0 ，不同为1

补码:

正数的补码是它本身

负数的补码是它正数源码的取反 在加 1

左移（<<）

左移一位的结果就是原值乘2，左移两位的结果就是原值乘4。

右移（>>）

右移一位的结果就是原值除2，左移两位的结果就是原值除4

系统硬件组成

总线: 贯穿系统个电子管道， 携带信息字节负责在各个部件间传递

IO：系统与外部世界联系的通道

主存：临时存储设备，存放程序和程序处理的数据

处理器（cpu）：解析或执行存储在主存中之类的引擎

上下文：保持跟踪进程运行所需的所有状态信息

--------------------------------------------------------------------------------------------------------------------------------------------------------------

# 操作系统导论：

操作系统是什么：

管理计算机硬件与软件资源的计算机程序（系统级别的软件）

单道批处理系统：（类似单线程）

多道批处理系统： （类似多线程）

多道程序设计：同时在主存中驻留多个程序，由它们竞争cpu的技术

内存管理：理解主存中载有多少个程序，以及他们的位置的动作

进程：程序执行过程中的动态表示法

进程管理：了解活动进程的信息的动作

cpu调度：确定主存中的那个进程可以访问cpu以便执行的动作

分时系统：多个交互用户同时共享cPU时间的系统

虚拟机：分时系统创建的每个用户都有专有机器的假象

主机:一个大型的多用户计算机，通常与早期的分时系统相关

哑终端：早期的分时系统中用户用于访问主机的一套显示器和键盘

实时系统：应用程序的特性决定了响应时间至关重要的系统

响应时间：收到信号和生成响应直接的延迟时间

逻辑地址：对一个存储值的引用，是相对于引用它的程序的

物理地址：主存储设备中的真实地址

地址联编：逻辑地址和物理地址间的映射

固定分区法:把内存分成特定数目的分区以载入程序的内存管理方法

动态分区法:根据容纳程序的需要对内存分区的内存管理方法

页式内存管理法：把进程划分为大小固定的页，载入内存时存储在真的内存管理方法

帧：大小固定的一部分主存，用于存放进程页

页：大小固定的一部分进程，存储在内存帧中

进程控制块：操作系统管理进程信息使用的数据结构

上下文信息：当一个进程移除cpu，另一个进程取代它时，发生的寄存器信息交换

cpu调度：cpu调度算法将决定把cpu给与那个进程

时间片：在cpu循环调度算法中，分配给每个进程的时间量

描述文件，文件系统和目录的用途

文件：数据的有名集合，用于组织二级存储设备

文件系统： 操作系统为它管理的文件提供的逻辑试图

目录: 文件的有名分组

区分文本文件和二进制文件

文本文件: 包含字符的文件

二进制文件：包含特定格式的数据的文件，

信息系统：帮助我们组织和分析数据的软件

# 数据库

悲观锁（pessimistic Lock）

在整个数据处理过程中，将数据处于锁定状态

共享锁：

多个事物对同一数据共享一把锁，只能读不能修改

排他锁：

一个事物获取了一个数据的排他锁，其他事物不能在获取该行的其他锁，可以对事物进 行修改

乐观锁：

乐观锁不会刻意使用数据库本身的锁机制，而是依据数据本身来保证数据的正确 乐观锁的实现

1、版本号

2、时间戳

适用场景

悲观锁

比较适合写入操作比较频繁的场景

乐观锁：

比较适合读取操作比较频繁的场景

# 深入理解java虚拟机

堆：

存储的是对象，每个对象包含与之对应的class的信息

只有一个，被所有线程共享，只存放对象本身

栈：

每个线程包含一个栈，只保存基础数据类型和自定义对象的引用

栈中的数据都是私有的

栈分为三个部分：基本类型变量区，执行环境上下文，操作指令区

方法区：

被所有线程共享，方法区包含 所有的class和static变量

方法区中包含的都是整个程序中永远唯一的元素 如 class、static 变量

java堆的两大区域：

新生代 (young)

eden

survivor(from)

survivor(to)

young中的对象大部分都是朝生夕死

老年代(old)

多次gc之后，仍然活着的对象

GC

Minor GC 新生代GC

Full GC 老年代GC

垃圾收集算法：

mark-sweep 标记-清除

两大阶段： 标记阶段 、 清除阶段

标记阶段的任务是标记出所有需要被回收的对象，清除阶段就是回收被标记的对象所占用的空间

缺点：

容易产生内存碎片，无法为大对象分配足够的空间

Copying 算法 （复制算法）

把可用内存 按容量大小分成大小相等两块

每次只使用其中一块。

GC的时候， 把活着的对象复制到另一块，是其成为被使用的

缺点：

对内存空间的使用 付出了高昂的代价

可以使用的内存缩减到原来的一半

mark-Compact(标记-整理)算法

垃圾收集器

　 新生代GC ：串行GC(SerialGC)、并行回收GC(ParallelScavenge)和并行GC(ParNew)

串行GC：

整个扫描和复制过程采用单线程的方式来进行

适用于单CPU、新生代空间较小及对暂停时间要求不是非常高的应用上

并行GC

整个扫描和复制过程采用多线程的方式来进行，

适用于多CPU、对暂停时间要求较短的应用上

并发GC（CMS）

# ElasticSearch

Es 是基于Lucene的搜索引擎， 支持分布式

index(索引)：一个可以检索的文档对象的集合 类似DB 的datebase

type(类型) ：索引的下一级概念 类似DB 的 table

Document(文档)： 一个可以被检索的基本单位 相当于数据库中的row

field(字段) ： 相当于数据库中的 column

Posting list 就是一个 int 的数组，存储了所有符合某个 term 的文档 id

垂直扩展（向上扩展）：购置性能更强的服务器

水平扩展（向外扩展）：增加更多的服务器

sharding（碎片）: 是把数据库 横向扩展到多个物理节点的一种有效方式

将大数据库分布到多个物理节点的一个分区方案

倒排索引:根据属性值来查找记录

# Spring Cloude

Eureka 服务的注册和发现

Spring Cloude Config 分布式服务配置中心

Feign 声明式服务调用，用于消费服务

spring clode OAuth2 用户安全配置

Ribbon 负载均衡

Hystrix 熔断器

Turbine 聚合多个 Hystrix Dashboard

Spring cloude Sleuth 集成ZipKin，用于服务链路追踪

Spring boot admin 聚合监控微服务的状况

Zuul 服务网关 用于服务智能路由，负载均衡

spriing data jpa 数据库mysql 实体对象持久化采用jpa

Swagger Api 文档组件

RabbitMQ 消息服务器

## spring boot 简介

spring boot是一个为简化配置，致力于快速开发的 spring 框架

简化配置：

自动装配 和 约定(起步依赖)

特点

自动装配:

程序需要什么，spring boot自己去下载装配

起步依赖:

会自动去下载配装需要依赖的包

Actuator:

对程序的状态进行监控

spring boot 的配置文件

swagger2: (没测试)

在线文档的查阅和测试

Eureka : (同款 zookeeper Consul)

服务注册和发现的组件

优缺点：

开源、经过生产环境考验、性能和功能上非常稳定

可以和spring 其他组件无缝对接

Eureka和ribbon、hystrix、Hystrix Dashboard(熔断器监控组件)

熔断器聚合监控（Turbine），以及zuul 很好的配合

负载均衡 Ribbon

负载均衡：将负载分摊到多个执行单元

常见的两种方式

1.独立进程单元：例 nginx

通过负载均衡策略，把请求分摊给多个服务提供者

2.将负载均衡逻辑以代码的封装到服务消费者分客户端上 例 ribbon

LoadBalancerClient

功能：获取负载均衡的服务提供者的实例信息

RestTemplate：

是一个访问 RestFul Api接口的请求框架 （用于消费rest 服务）

声明式调用 Feign

Hystrix 熔断器

功能：阻止 分布式系统中出现的联动故障

Hystrix Dashboard

实时监控熔断器的状态

Turbine 聚合监控

Zuul 路由网关

zuul 与Ribbon 和Eureka 结合，可以实现智能路由和负载均衡

网关将所有服务的api接口统一聚合，并统一对外暴露

对用户身份认证和权限认证

实时监控，实时日志输出

流量监控

方便测试

配置中心 spring cloude Config

# Mysql DBA

## 入门

## mysql 优势：

成本低，效能高，免费，开源，稳定

## mysql引擎

INNODB（5.5版本后的默认引擎）：

灾难恢复性好

支持全部4中级别的事物

使用行级锁

缓冲管理

支持热备份

MyISAM（）：

使用表级锁，并发性差

主机宕机后，表易损坏，灾难恢复性差

无事物支持

Memory 存储引擎

使用内存表可以显著提高数据访问效率

使用表级锁

内存表内存满了，会出现问题

服务器重启数据丢失，复制维护麻烦

索引：

对数据库表中一列或多列的值进行排序的一种结构

加快检索表中的数据

聚簇索引

按照数据存放的物理位置为顺序

提高多行检索的速度

非聚簇索引

单行索引很快

唯一索引:

不允许气筒任何两行具有相同索引值

主键索引

唯一标识表中的一行

聚集索引

表中行的物理顺序与键值的逻辑顺序形态

# 多线程网络编程知识

CAS（compare And Swap）:比较交换技术

volatile: 是一种同步机制，

保证共享变量对所有线程的可见性

这个写操作会导致其他线程中的缓存无效

禁止指令重排序

重排序：是指编译器和处理器为了优化程序性能而对指令进行排序的一种手段

重排序遵守的规则：

重排序不会对存在数据依赖关系的操作进行重排序

重排序是为了优化性能，但是不管怎么重排序，单线程下程序的执行结果不能 被改变

简单总结下，volatile是一种轻量级的同步机制，它主要有两个特性：

一是保证共享变量对所有线程的可见性；

二是禁止指令重排序优化。同时需要注意的是，

volatile对于单个的共享变量的读/写具有原子性，

但是像num++这种复合操作，volatile无法保证其原子性，

解决方案，就是使用并发包中的原子操作类，

通过循环CAS地方式来保证num++操作的原子性

可见性：当一条线程修改了共享变量的值，新值对于其他线程是可以立即得知的

ReentrantLock 是一个 可重入的互斥锁

互斥锁：同一个时刻只有一个线程获得到锁，其他获得锁的线程只能处于同步队列中等 待

只有获得锁的线程释放了锁，后继的线程才能获得锁

可重入： 支持重进入的锁，表示 该锁能够支持一个线程对资源的重复加锁

公平锁和非公平锁：

公平： 不同的线程获得锁的机制是公平的

非公平： 不同的线程获得锁的机制是不公平的

中断响应：

对于synchronized 如果一个线程在等待锁，结果只有两种：

1、获得这把锁继续执行

2、线程保持等待

可重入锁:

提供了第三种可能，线程可以被中断

就是在等待锁的过程中，程序可以根据需要取消对锁的需求

CountDownLatch

使一个或多个线程等待其他线程完成各自的工作后再执行

CylicBarrier 可循环的屏障

让一组线程达到某个屏障点时被阻塞，直到最后一个线程达到屏障的时候，屏障才会开 门，所有被屏障拦截的线程才会执行

Semaphore ：信号量

表示当前公共资源的可用数目

目的：

1、用于多个共享资源的互斥使用

2、用于并发线程数的控制

# 名词解析

SOA（[Service Oriented Architecture](https://baike.baidu.com/item/SOA/9614173" \l "viewPageContent" \o "Service Oriented Architecture)）：面向服务的架构

功能： 将应用程序的不同功能单元（服务）进行拆分

通过接口和契约把这些服务联系起来

RPC(Remote Procedure Call)： 远程过程调用

功能： 通过网络 从远程计算机上请求服务

集群：

多台计算机完成同一份工作，可以被看做是同一台计算机

分布式：

组件之间彼此交互以实现一个共同目标

模块独立，各做各的事，便于扩展，复用性告

解释运行

以解释的方式运行字节码

即：第一句执行一句

编译运行（JIT）

将字节码编译成机器码

直接执行机器码

运行时编译

编译后性能有数量级的提升

类加载器：

读取java字节代码，并转换成java.lang.Class类的一个实例

双亲委托机制：

每次收到类加载请求时，先将请求委派给父类加载器完成（所有加载请求最终都会委派 到顶层Bootstrap ClassLoader 加载器中）

如果父类加载器也无法完成这个加载（该加载器的搜索范围中没有找到对应的类），子 类尝试自己加载

内存溢出（out of memory） :

程序在申请内存时，没有足够的内存空间供其使用

内存泄漏（memory leak） :

程序申请内存后，无法释放已申请的内存空间

不再使用的对象持续占有内存、无用对象的内存得不到及时的释放

## 传统IO

基于字节流和字符流进行操作

## NIO：

基于channel 和buffer 进行操作

非阻塞的模式

三大核心

channel:

双向的，既可以读又可以写

buffer:

selector 运行多个channel

通道： 类似于流，可以异步读写数据（流只能同步）

## netty:

是一个开源框架 jboss提供的

用来开发网络服务器和客户端程

架构： 适用于整个系统范围的设计约束

高层设计: 适用于子系统层次或多个类的层次上的设计约束

inputstream:

read ： 把数据从流中 写到数据中

outputstream:

write: 把数据 写到流中

阻塞和非阻塞：

进程或线程在访问数据的时候，数据是否准备就绪的一种处理方式

阻塞： 缓冲区数据 没有 准备就绪 ，等待数据准备就绪，在继续操作

非阻塞：缓冲区数据 没有准备就绪 ，直接返回， 继续后续操作

同步和异步：

应用程序和操作系统处理IO事件的一种方式

同步：等到IO事件结束，返回结果，然后才可以继续执行

异步：IO操作交给操作系统，继续其他操作，IO结束，给当前线程一个通知

阻塞/非阻塞：进程/线程需要操作的数据如果尚未就绪，是否妨碍了当前进程/线程的后续操作。

同步/异步：数据如果尚未就绪，是否需要等待数据结果。

同步就是多个事物不能同时开工，异步就是多个事物可以同时开工。

所谓同步/异步，关注的是能不能同时开工。

所谓阻塞/非阻塞，关注的是能不能动。

注：一定要去体会“多个事物”，多个线程是多个事物，多个方法是多个事物，

多个语句是多个事物，多个CPU指令是多个事物。等等等等

同步阻塞，相当于一个线程在等待。

同步非阻塞，相当于一个线程在正常运行。

异步阻塞，相当于多个线程都在等待。

异步非阻塞，相当于多个线程都在正常运行。