# 一、基础篇

## 面向对象

### 什么是面向对象

起初，面向[对象](https://baike.baidu.com/item/%E5%AF%B9%E8%B1%A1" \t "https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1/_blank)”是专指在[程序设计](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1" \t "https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1/_blank)中采用[封装](https://baike.baidu.com/item/%E5%B0%81%E8%A3%85" \t "https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1/_blank)、[继承](https://baike.baidu.com/item/%E7%BB%A7%E6%89%BF" \t "https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1/_blank)、[多态](https://baike.baidu.com/item/%E5%A4%9A%E6%80%81" \t "https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1/_blank)等设计方法。

面向对象是一种对现实世界理解和抽象的方法，是计算机编程技术 发展到一定阶段后的产物。

### 面向对象、面向过程

面向对象注重结果，面向过程注重过程

面向对象：

优点：易维护、易复用、易扩展，由于面向对象有封装、继承、多态的特效，可以设计出出低耦合的系统，使系统更加灵活、更加易与维护

缺点：性能比面向过程低

面向过程：

优点：性能比面向对象高，因为类调用时需要实例化，开销比较大，比较消耗资源;比如单片机、嵌入式开发、 Linux/Unix等一般采用面向过程开发，性能是最重要的因素。

缺点：没有面向对象易维护、易复用、易扩展

### 面向对象的四大基本特征和五大基本原则

抽象、封装、继承、多态。

单一职责原则、开放封闭原则、里氏替换原则、依赖倒置原则、接口隔离原则

### 平台无关性

Java如何实现的平台无关

为了适应多样性的运行环境，Java编译器生成字节码--一种被设计用来将代码高效地转换到多种硬件和软件平台的架构无关的中间格式。

JVM还支持哪些语言（Kotlin、Groovy、JRuby、Jython、Scala）

### 值传递

值传递、引用传递

值传递：传值（参数）。引用传递：传地址（内存地址）

为什么说Java中只有值传递

### 封装、继承、多态

什么是多态、方法重写与重载

Java的继承与实现

构造函数与默认构造函数

类变量、成员变量和局部变量

成员变量和方法作用域

## Java基础知识

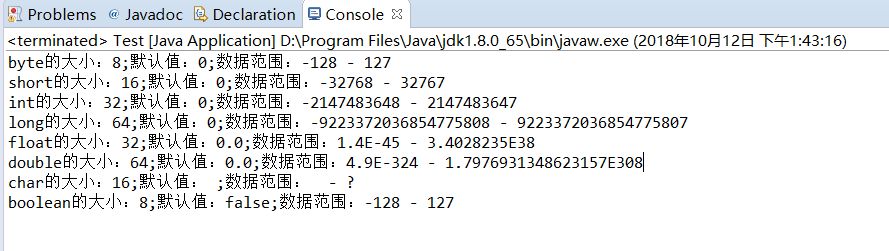
### 基本数据类型

7种基本数据类型：整型、浮点型、布尔型、字符型

八种基本类型。六种数字类型（四个整数型（默认是int 型），两个浮点型（默认是double 型）），一种字符类型，还有一种布尔型。

整型中byte、short、int、long的取值范围

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **数据类型** | **位数** | **默认值** | **取值范围** | **举例说明** |
| byte(位) | 8 | 0 | -2^7 - 2^7-1 | byte b = 10; |
| short(短整数) | 16 | 0 | -2^15 - 2^15-1 | short s = 10; |
| int(整数) | 32 | 0 | -2^31 - 2^31-1 | int i = 10; |
| long(长整数) | 64 | 0 | -2^63 - 2^63-1 | long l = 10l; |
| float(单精度) | 32 | 0.0 | -2^31 - 2^31-1 | float f = 10.0f; |
| double(双精度) | 64 | 0.0 | -2^63 - 2^63-1 | double d = 10.0d; |
| char(字符) | 16 | 空 | 0 - 2^16-1 | char c = 'c'; |
| boolean(布尔值) | 8 | false | true、false | boolean b = true; |



什么是浮点型？什么是单精度和双精度？为什么不能用浮点型表示金额？

### 自动拆装箱

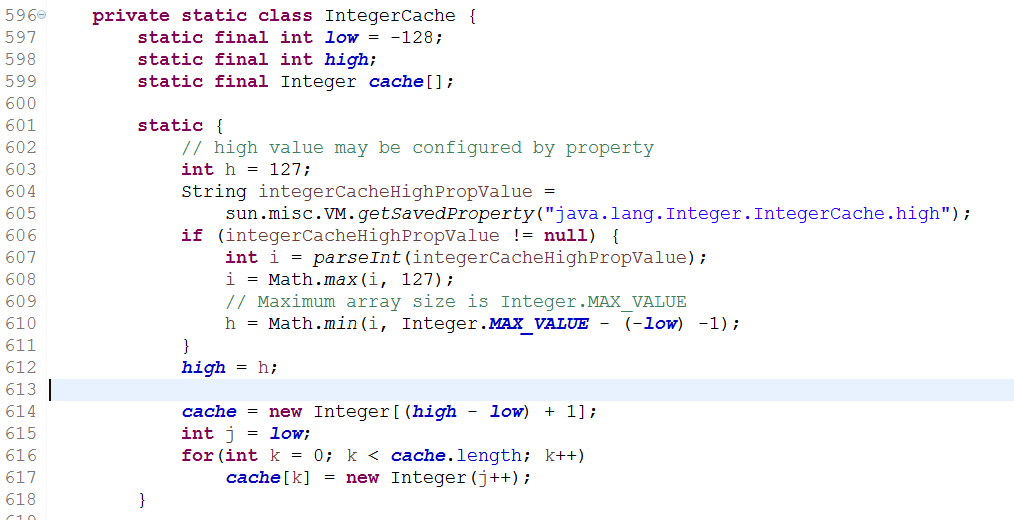
什么是包装类型、什么是基本类型、什么是自动拆装箱

Java 为每个基本数据类型都提供了一个包装类

自动装箱：基本类型自动转为包装类

自动拆箱：包装类转为基本类型

Integer的缓存机制

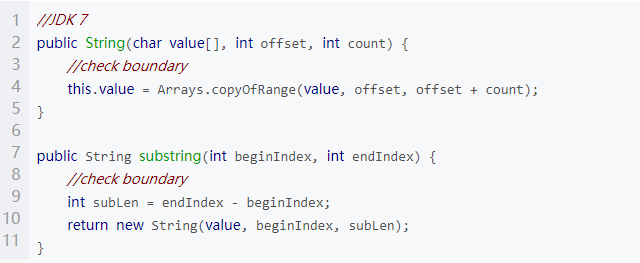
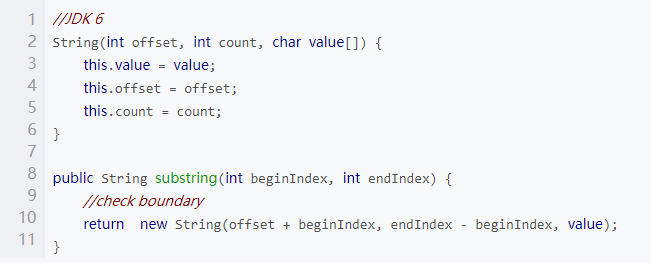


String

字符串的不可变性

一但一个字符串在堆内存创建，它是不能被改变的，我们应该注意所有的关于String类的方法不是改变一个字符串本身，而是重新返回一个新的字符串。如果我们需要一个频繁的改变一个字符串对象，我们可以使用StringBuffer或者StringBuilder，否则将会浪费大量时间进行垃圾回收，因为每次创建一个新的字符串。不过需要注意的是，貌似JDK1.7之前不是所有的方法都是返回新对象，1.7的是返回新的对象。

JDK 6和JDK 7中substring的原理及区别、



replaceFirst、replaceAll、replace区别、

Replace()全部替换

ReplaceAll()全部替换（是根据正则表达式的替换）

ReplaceFirs()只替换第一次出现的

String对“+”的重载、字符串拼接的几种方式和区别

String.valueOf和Integer.toString的区别、

switch对String的支持

字符串池、常量池（运行时常量池、Class常量池）、intern

### 熟悉Java中各种关键字

transient、instanceof、volatile、synchronized、final、static、const 原理及用法。

### 集合类

常用集合类的使用、ArrayList和LinkedList和Vector的区别 、SynchronizedList和Vector的区别、HashMap、HashTable、ConcurrentHashMap区别、

Set和List区别？Set如何保证元素不重复？

Java 8中stream相关用法、apache集合处理工具类的使用、不同版本的JDK中HashMap的实现的区别以及原因

Collection和Collections区别

Arrays.asList获得的List使用时需要注意什么

Enumeration和Iterator区别

fail-fast 和 fail-safe

CopyOnWriteArrayList、ConcurrentSkipListMap

### 枚举

枚举的用法、枚举的实现、枚举与单例、Enum类

Java枚举如何比较

switch对枚举的支持

枚举的序列化如何实现

枚举的线程安全性问题

### IO

字符流、字节流、输入流、输出流、

同步、异步、阻塞、非阻塞、Linux 5种IO模型

BIO、NIO和AIO的区别、三种IO的用法与原理、netty

### Java反射与javassist

反射与工厂模式、 反射有什么作用

反射：应用的是动态编译最大限度发挥了java的灵活性，体现了多  
态的应用，有以降低类之间的藕合性。

静态编译：编译时确定类型，绑定对象。

动态编译：运行时确定类型，绑定对象。

### Class类

java.lang.reflect.\*

### 动态代理

静态代理、动态代理

动态代理和反射的关系

动态代理的几种实现方式

### AOP

### 序列化

什么是序列化与反序列化、为什么序列化、序列化底层原理、序列化与单例模式、protobuf、为什么说序列化并不安全

### 注解

元注解、自定义注解、Java中常用注解使用、注解与反射的结合

Spring常用注解

### JMS

什么是Java消息服务、JMS消息传送模型

### JMX

java.lang.management.\*、 javax.management.\*

### 泛型

泛型与继承、类型擦除、泛型中K T V E ？ object等的含义、泛型各种用法

限定通配符和非限定通配符、上下界限定符extends 和 super

List和原始类型List之间的区别?

List<?>和List之间的区别是什么?

### 单元测试

junit、mock、mockito、内存数据库（h2）

### 正则表达式

java.lang.util.regex.\*

### 常用的Java工具库

commons.lang, commons.\*... guava-libraries netty

### API&SPI

API、API和SPI的关系和区别

如何定义SPI、SPI的实现原理

### 异常

异常类型、正确处理异常、自定义异常

Error和Exception

异常链、try-with-resources

finally和return的执行顺序

### 时间处理

时区、冬令时和夏令时、时间戳、Java中时间API

格林威治时间、CET,UTC,GMT,CST几种常见时间的含义和关系

SimpleDateFormat的线程安全性问题

Java 8中的时间处理

如何在东八区的计算机上获取美国时间

### 编码方式

Unicode、有了Unicode为啥还需要UTF-8

GBK、GB2312、GB18030之间的区别

UTF8、UTF16、UTF32区别

URL编解码、Big Endian和Little Endian

如何解决乱码问题

### 语法糖

Java中语法糖原理、解语法糖

语法糖：switch 支持 String 与枚举、泛型、自动装箱与拆箱、方法变长参数、枚举、内部类、条件编译、 断言、数值字面量、for-each、try-with-resource、Lambda表达式、

### 阅读源代码

String、Integer、Long、Enum、BigDecimal、ThreadLocal、ClassLoader & URLClassLoader、ArrayList & LinkedList、 HashMap & LinkedHashMap & TreeMap & CouncurrentHashMap、HashSet & LinkedHashSet & TreeSet

## Java并发编程

### 并发与并行

什么是并发

什么是并行

并发与并行的区别

### 线程

线程的实现、线程的状态、优先级、线程调度、创建线程的多种方式、守护线程

线程与进程的区别

### 线程池

自己设计线程池、submit() 和 execute()、线程池原理

为什么不允许使用Executors创建线程池

### 线程安全

死锁、死锁如何排查、线程安全和内存模型的关系

锁

CAS、乐观锁与悲观锁、数据库相关锁机制、分布式锁、偏向锁、轻量级锁、重量级锁、monitor、

锁优化、锁消除、锁粗化、自旋锁、可重入锁、阻塞锁、死锁

### 死锁

死锁的原因

死锁的解决办法

### synchronized

synchronized是如何实现的？

synchronized和lock之间关系、不使用synchronized如何实现一个线程安全的单例

synchronized和原子性、可见性和有序性之间的关系

### volatile

happens-before、内存屏障、编译器指令重排和CPU指令重

volatile的实现原理

volatile和原子性、可见性和有序性之间的关系

有了symchronized为什么还需要volatile

sleep 和 wait

wait 和 notify

notify 和 notifyAll

ThreadLocal

写一个死锁的程序

写代码来解决生产者消费者问题

### 并发包

阅读源代码，并学会使用

Thread、Runnable、Callable、ReentrantLock、ReentrantReadWriteLock、Atomic\*、Semaphore、CountDownLatch、、ConcurrentHashMap、Executors

# 二、底层篇

## JVM

### JVM内存结构

方法区（也就是"持久代"），java8里彻底被移除，取而代之的是元数据区

堆

栈（在hotspot JVM中，JVM方法栈--Java虚拟栈，与本地方法栈是同一个）

PC寄存器（程序计数器）

class文件格式、运行时数据区：堆、栈、方法区、直接内存、运行时常量池、

### 堆和栈区别

Java中的对象一定在堆上分配吗？

### Java内存模型

计算机内存模型、缓存一致性、MESI协议

可见性、原子性、顺序性、happens-before、

内存屏障、synchronized、volatile、final、锁

### 垃圾回收

GC算法：标记清除、引用计数、复制、标记压缩、分代回收、增量式回收

GC参数、对象存活的判定、垃圾收集器（CMS、G1、ZGC、Epsilon）

### JVM参数及调优

-Xmx、-Xmn、-Xms、Xss、-XX:SurvivorRatio、

-XX:PermSize、-XX:MaxPermSize、-XX:MaxTenuringThreshold

### Java对象模型

oop-klass、对象头

HotSpot

即时编译器、编译优化

虚拟机性能监控与故障处理工具

jps, jstack, jmap、jstat, jconsole, jinfo, jhat, javap, btrace、TProfiler

## Arthas

类加载机制

classLoader、类加载过程、双亲委派（破坏双亲委派）、模块化（jboss modules、osgi、jigsaw）

### 编译与反编译

什么是编译（前端编译、后端编译）、什么是反编译

JIT、JIT优化（逃逸分析、栈上分配、标量替换、锁优化）

编译工具：javac

反编译工具：javap 、jad 、CRF

# 三、进阶篇

## Java底层知识

### 字节码、class文件格式

CPU缓存，L1，L2，L3和伪共享

### 尾递归

### 位运算

用位运算实现加、减、乘、除、取余

## 设计模式

### 设计模式的六大原则：

开闭原则（Open Close Principle）、里氏代换原则（Liskov Substitution Principle）、依赖倒转原则（Dependence Inversion Principle）

接口隔离原则（Interface Segregation Principle）、迪米特法则（最少知道原则）（Demeter Principle）、合成复用原则（Composite Reuse Principle）

### 了解23种设计模式

创建型模式：单例模式、抽象工厂模式、建造者模式、工厂模式、原型模式。

结构型模式：适配器模式、桥接模式、装饰模式、组合模式、外观模式、享元模式、代理模式。

行为型模式：模版方法模式、命令模式、迭代器模式、观察者模式、中介者模式、备忘录模式、解释器模式（Interpreter模式）、状态模式、策略模式、职责链模式(责任链模式)、访问者模式。

### 会使用常用设计模式

单例的七种写法：懒汉——线程不安全、懒汉——线程安全、饿汉、饿汉——变种、静态内部类、枚举、双重校验锁

工厂模式、适配器模式、策略模式、模板方法模式、观察者模式、外观模式、代理模式等必会

不用synchronized和lock，实现线程安全的单例模式

用CAS是非阻塞算法的常见实现

好处：用CAS的好处在于不需要使用传统的锁机制来保证线程安全,CAS是一种基于忙等待的算法,依赖底层硬件的实现,相对于锁它没有线程切换和阻塞的额外消耗,可以支持较大的并行度。  
 缺点：CAS的一个重要缺点在于如果忙等待一直执行不成功(一直在死循环中),会对CPU造成较大的执行开销。

实现AOP

AOP面向切面编程，通常用来处理事务、日志、缓存

String AOP中的动态代理主要有两种方式，JDK的动态代理和CGLIB动态代理。JDK动态代理通过反射来接受被代理的类，并且要求被代理的类必须实现一个接口。JDK动态代理的核心是InvocationHandler接口和Proxy类。

实现IOC

IOC 依赖注入DI 是一种软件设计模式。是用来解耦合的。

IOC中主要是反射和工程模式实现的。

优点：实现组件间的解耦，提高程序的灵活性和可维护性。

缺点：1、创建对象的步骤变复杂了。

1. 使用反射来创建对象，在效率上会有些损耗，对于程序的灵活性和可维护性来说，这些损耗微不足道。
2. 缺少IDE重构的支持，如果修改了类名，还需要在xml中手动修改。这是所有xml方式的缺憾所在。

nio和reactor设计模式

## 网络编程知识

### tcp、udp、http、https等常用协议

三次握手与四次关闭、流量控制和拥塞控制、OSI七层模型、tcp粘包与拆包

http1.0、 http1.1、 http2.0之间的区别

https://blog.csdn.net/CrankZ/article/details/81239654

http中 get和post区别

最直观的区别GET把参数包含在URL中，POST通过request body传递参数。

GET：浏览器会把http header和data一并发出去，服务器响应200（返回数据）

POST：浏览器先发header，服务器返回100 continue，浏览器再发生data，服务器返回200 （返回数据）

常见的web请求返回的状态码

200、302、301、404、500分别代表什么

200：(成功)服务器请求成功

301：(永久移动)请求的页面转义到新位置，服务器返回此响应时，会自动将请求转到新位置

302：(临时移动)服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

404：(未找到)服务器请求不到页面

500：(服务器内部错误)服务器遇到错误，无法完成请求

http/3

Java RMI，Socket，HttpClient

### cookie 与 session

cookie被禁用，如何实现session

用Java写一个简单的静态文件的HTTP服务器

了解nginx和apache服务器的特性并搭建一个对应的服务器

用Java实现FTP、SMTP协议

进程间通讯的方式

什么是CDN？如果实现？

DNS？

什么是DNS 、记录类型:A记录、CNAME记录、AAAA记录等

域名解析、根域名服务器

DNS污染、DNS劫持、公共DNS：114 DNS、Google DNS、OpenDNS

### 反向代理

正向代理、反向代理

正向代理：正向代理类似一个跳板机，代理访问外部资源。

反向代理：初次接触方向代理的感觉是，客户端是无感知代理的存在的，反向代理对外都是透明的，访问者者并不知道自己访问的是一个代理。因为客户端不需要任何配置就可以访问。

反向代理服务器

## 框架知识

### Servlet

生命周期

1. 加载和实例化
2. 初始化
3. 请求处理
4. 服务终止

线程安全问题

filter和listener

web.xml中常用配置及作用

### Hibernate

什么是OR Mapping

Object-Relation Mapping

Hibernate的缓存机制

一级缓存（session级别）、二级缓存（sessionFactory级别）、查询缓存

Hibernate的懒加载

所谓懒加载（lazy）就是延时加载。

当要访问数据量过大时用懒加载

Hibernate/Ibatis/MyBatis之间的区别

首先Hiberante，iBatis,MyBatis都是对数据访问的封装，最终都是生成sql语句访问数据库。只不过Hibernate封装得比较全面，IBatis只是局部封装，懂Jdbc的程序员可以快速上手，而myBatis是近来流行的一种新的IBatis是对IBatis的升级，可以看着是升级版。，而myBatis是近来流行的一种新的IBatis是对IBatis的升级。

JDBC

JDBC执行的操作

注册驱动（forClass）

编写sql

预编译（通过Connection得到Statement）

设置参数（往Statement中添加参数）

执行sql（通过Statement执行sql）

封装结果（通过ResultSet得到结果）

JDBC的优点：

很清楚明白的看出数据库所执行的操作

JDBC的缺点：

编写工作量巨大

灵活性差

需要及时处理异常并正确关闭数据源

Hibernate

执行过程

读取配置文件完成ORM的映射关系

获取session进行增删改查的操作

Hibernate的优点

维护方便，数据库连接和POJO类的映射都写在xml文档中

操作方便，只需获取session就可以对数据库进行操作，关闭时也只需关闭session

Hibernate的缺点

全映射会造成性能比较差

sql语句由于是hibernate生成而造成了sql的灵活性较差，不方便定制高效sql

自带的HQL语句性能较低

Mybatis

Mybatis优点

半自动的执行方式，除sql外操作由框架完成，方便快捷

手工定制sql，实现sql与java代码分离，灵活性高，方便维护

只需要sql进行操作，不需要学习额外的内容

轻量级的框架，执行速度快

### String AOP和IOC

AOP原理

AOP面向切面编程，通常用来处理事务、日志、缓存

String AOP中的动态代理主要有两种方式，JDK的动态代理和CGLIB动态代理。JDK动态代理通过反射来接受被代理的类，并且要求被代理的类必须实现一个接口。JDK动态代理的核心是InvocationHandler接口和Proxy类。

实现Spring的IOC

IOC 依赖注入DI 是一种软件设计模式。是用来解耦合的。

IOC中主要是反射和工程模式实现的。

优点：实现组件间的解耦，提高程序的灵活性和可维护性。

缺点：1、创建对象的步骤变复杂了。

1. 使用反射来创建对象，在效率上会有些损耗，对于程序的灵活性和可维护性来说，这些损耗微不足道。
2. 缺少IDE重构的支持，如果修改了类名，还需要在xml中手动修改。这是所有xml方式的缺憾所在。

spring四种依赖注入方式

1. set注入
2. 构造器注入
3. 静态工厂注入
4. 动态工厂注入

### Spring MVC

什么是MVC

Model View Controller是一种软件设计典范。它是用一种业务逻辑、数据与界面显示分离的方法来组织代码，将众多的业务逻辑聚集到一个部件里面，在需要改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑，达到减少编码的时间。

### Spring Boot

Spring Boot 2.0、起步依赖、自动配置

起步依赖：

自动配置：应用程序启动时，springboot框架自动检测classpath 里的bean来进行配置的一种机制。

Spring Boot的starter原理

Spring Security

### Spring Cloud

服务发现与注册：Eureka、Zookeeper、Consul

负载均衡：Feign、Spring Cloud Loadbalance

服务配置：Spring Cloud Config

服务限流与熔断：Hystrix

服务链路追踪：Dapper

服务网关、安全、消息

## 应用服务器知识

JBoss

tomcat

jetty

Weblogic

工具

git & svn

maven & gradle

### Intellij IDEA

常用插件：Maven Helper 、FindBugs-IDEA、阿里巴巴代码规约检测、GsonFormat

Lombok plugin、.ignore、Mybatis plugin

# 四、高级篇

## 新技术

### Java 8

Java 8 为我们提供了接口的默认方法和静态方法，接口也可以包含行为，而不仅仅是方法定义。

lambda表达式、Stream API、时间API

lambda表达式：允许我将函数当成参数传递给某个方法，或者把代码本身当做数据处理。

StreamAPI:将生产环境中的函数式编程引入Java库中。极大的简化了集合的操作。

Data/Time API：新的java.time包包含了所有关于时间、日期、时区、Instant、duration、和时钟操作的类。

### Java 9

Java 9增加了List.of()、Set.of()、Map.of()和Map.ofEntries()等工厂方法来创建不可变集合。

Jigsaw、Jshell、Reactive Streams

### Java 10

局部变量类型推断、G1的并行Full GC、ThreadLocal握手机制

### Java 11

ZGC、Epsilon、增强var、

### Spring 5

## 响应式编程

Spring Boot 2.0

http/2

http/3

## 性能优化

使用单例、使用Future模式、使用线程池、选择就绪、减少上下文切换、减少锁粒度、数据压缩、结果缓存

## 线上问题分析

### dump获取

线程Dump、内存Dump、gc情况

### dump分析

分析死锁、分析内存泄露

### dump分析及获取工具

jstack、jstat、jmap、jhat、Arthas

自己编写各种outofmemory，stackoverflow程序

HeapOutOfMemory、 Young OutOfMemory、MethodArea OutOfMemory、ConstantPool OutOfMemory、DirectMemory OutOfMemory、Stack OutOfMemory Stack OverFlow

### Arthas

jvm相关、class/classloader相关、monitor/watch/trace相关、

### options、管道、后台异步任务

文档：https://alibaba.github.io/arthas/advanced-use.html

### 常见问题解决思路

内存溢出、线程死锁、类加载冲突

## 使用工具尝试解决以下问题，并写下总结

当一个Java程序响应很慢时如何查找问题、

当一个Java程序频繁FullGC时如何解决问题、

如何查看垃圾回收日志、

当一个Java应用发生OutOfMemory时该如何解决、

如何判断是否出现死锁、

如何判断是否存在内存泄露

使用Arthas快速排查Spring Boot应用404/401问题

使用Arthas排查线上应用日志打满问题

利用Arthas排查Spring Boot应用NoSuchMethodError

### 编译原理知识

编译与反编译

Java代码的编译与反编译

Java的反编译工具

javap 、jad 、CRF

### 即时编译器

词法分析，语法分析（LL算法，递归下降算法，LR算法），语义分析，运行时环境，中间代码，代码生成，代码优化

操作系统知识

Linux的常用命令

进程间通信

### 进程同步

生产者消费者问题、哲学家就餐问题、读者写者问题

缓冲区溢出

分段和分页

虚拟内存与主存

虚拟内存管理

换页算法

数据库知识

MySql 执行引擎

MySQL 执行计划

如何查看执行计划，如何根据执行计划进行SQL优化

### 索引

Hash索引、B树索引（B+树、和B树、R树）

普通索引、唯一索引

覆盖索引、最左前缀原则、索引下推

### SQL优化

数据库事务和隔离级别

事务的隔离级别、事务能不能实现锁的功能

### 数据库锁

行锁、表锁、使用数据库锁实现乐观锁、

行锁：给某一行加锁select \* from test where id = ‘1’lock in share mode；

表锁：给表加锁。

乐观锁：

悲观锁：

### 连接

内连接，左连接，右连接

### 数据库主备搭建

binlog

redolog

内存数据库

h2

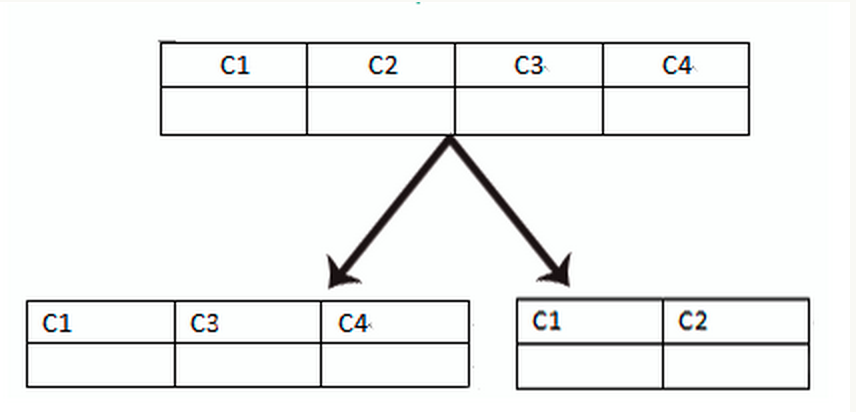
### 分库分表

数据切分：

1、垂直（纵向）切分：

垂直分库：根据业务耦合性，将关联度低的不同表存储在不同的数据库。与“微服务治理”的做法相似，每个微服务都使用单独的数据库。

垂直分表：某个表字段较多，可以新建一张扩展表，将不经常用或字段长度较大的字段拆分出去到扩展表中。



优点：

解决业务系统层面的耦合，业务清晰

与微服务的治理类似，也能对不同业务的数据进行分级管理、维护、监控、扩展等

高并发场景下，垂直切分一定程度的提升IO、数据库连接数、单机硬件资源的瓶颈

缺点：

部分表无法join，只能通过接口聚合方式解决，提升了开发的复杂度

分布式事务处理复杂

依然存在单表数据量过大的问题（需要水平切分）

2、水平（横向）切分：

库内分表、分库分表：根据表内数据内在的逻辑关系，将同一个表按照不同的条件分散到多个数据库或多个表中，每个表中只包含一部分数据，从而使得单个表的数据量变小，达到分布式的效果。

优点：

不存在单库数据量过大、高并发的性能瓶颈，提升系统稳定性和负载能力

应用端改造较小，不需要拆分业务模块

缺点：

跨分片的事务一致性难以保证

跨库的join关联查询性能较差

数据多次扩展难度和维护量极大

水平切分的方法：

"根据数值范围"：

以主键uid为划分依据，按uid的范围将数据水平切分到多个数据库上。例如：user-db1存储uid范围为0~1000w的数据，user-db2存储uid范围为1000w~2000wuid数据。

优点是：扩容简单，如果容量不够，只要增加新db即可。

不足是：请求量不均匀，一般新注册的用户活跃度会比较高，所以新的user-db2会比user-db1负载高，导致服务器利用率不平衡

"根据数值取模"：

也是以主键uid为划分依据，按uid取模的值将数据水平切分到多个数据库上。例如：user-db1存储uid取模得1的数据，user-db2存储uid取模得0的uid数据。

优点是：数据量和请求量分布均均匀

不足是：扩容麻烦，当容量不够时，新增加db，需要rehash。需要考虑对数据进行平滑的迁移。

### 读写分离

1. 将读操作和写操作分离到不同的数据库上，避免主服务器出现性能瓶颈；

　　2. 主服务器进行写操作时，不影响查询应用服务器的查询性能，降低阻塞，提高并发；

3. 数据拥有多个容灾副本，提高数据安全性，同时当主服务器故障时，可立即切换到其他服务器，提高系统可用性；

读写分离的基本原理就是让主数据库处理事务性增、改、删操作（INSERT、UPDATE、DELETE）操作，而从数据库处理SELECT查询操作。数据库复制被用来把事务性操作导致的变更同步到其他从数据库。

### 常用的nosql数据库

Mongodb

Mysql

Oracle

### redis、memcached

Redis

优点：支持多数据结构，支持持久化，支持数据同步。单线程请求，命令串行执行，并发情况下不用考虑数据一致性的问题。支持pub/sub消息订阅机制。

支持简单的业务需求，但业界使用场景很少，并不成熟，即优也缺。

局限性：只能使用单线程，性能受限于CPU的性能。

在String类型上会消耗较多内存，可以使用dict（hash表）压缩存储以降低内存耗用。

不适合在不同数据集之间建立关系，也不适合进行查询搜索。

Mencached

优点：利用多核优势，单实例吞吐量极高，适用于最大程度的抗量。支持直接配置session、handle

局限性：只支持简单的key/value数据结构。

无法持久换，数据不能备份，只能用于缓存，且重启后数据全部丢失。

无法进行数据同步，不能讲MC中的数据同步到其他MC实例中。

Mencached内存分配采用slab allcation机制管理内存，value大小分布差异较大时会造成内存利用率降低，并引发低利用率是依然出现踢出等问题，需要注重value的设计



Redis的好处

(1) 速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)

(2)支持丰富数据类型，支持string，list，set，sorted set，hash

(3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行

(4) 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

****终极策略：**** 使用Redis的String类型做的事，都可以用Memcached替换，以此换取更好的性能提升； 除此以外，优先考虑Redis；

分别使用数据库锁、NoSql实现分布式锁

### 性能调优

数据库连接池

数据结构与算法知识

简单的数据结构

栈、队列、链表、数组、哈希表、

栈和队列的相同和不同之处

栈通常采用的两种存储结构

### 树

二叉树、字典树、平衡树、排序树、B树、B+树、R树、多路树、红黑树

### 堆

大根堆、小根堆

### 图

有向图、无向图、拓扑

## 深度优先和广度优先搜索

全排列、贪心算法、KMP算法、hash算法

海量数据处理

分治，hash映射，堆排序，双层桶划分，Bloom Filter，bitmap，数据库索引，mapreduce等。

两个栈实现队列，和两个队列实现栈

## 大数据知识

### Zookeeper

基本概念、常见用法

Solr，Lucene，ElasticSearch

在linux上部署solr，solrcloud，，新增、删除、查询索引

Storm，流式计算，了解Spark，S4

在linux上部署storm，用zookeeper做协调，运行storm hello world，local和remote模式运行调试storm topology。

Hadoop，离线计算

HDFS、MapReduce

分布式日志收集flume，kafka，logstash

数据挖掘，mahout

## 网络安全知识

### XSS

XSS的防御

### CSRF

注入攻击

SQL注入、XML注入、CRLF注入

文件上传漏洞

加密与解密

对称加密、非对称加密、哈希算法、加盐哈希算法

MD5，SHA1、DES、AES、RSA、DSA

彩虹表

DDOS攻击

DOS攻击、DDOS攻击

memcached为什么可以导致DDos攻击、什么是反射型DDoS

如何通过Hash碰撞进行DOS攻击

SSL、TLS，HTTPS

用openssl签一个证书部署到apache或nginx

# 五、架构篇

## 分布式

数据一致性、服务治理、服务降级

### 分布式事务

2PC、3PC、CAP、BASE、 可靠消息最终一致性、最大努力通知、TCC

### Dubbo

服务注册、服务发现，服务治理

http://dubbo.apache.org/zh-cn/

### 分布式数据库

怎样打造一个分布式数据库、什么时候需要分布式数据库、mycat、otter、HBase

### 分布式文件系统

mfs、fastdfs

### 分布式缓存

缓存一致性、缓存命中率、缓存冗余

### 限流降级

Hystrix、Sentinal

## 算法

共识算法、Raft协议、Paxos 算法与 Raft 算法、拜占庭问题与算法

2PC、3PC

## 微服务

SOA、康威定律

ServiceMesh

sidecar

Docker & Kubernets

Spring Boot

Spring Cloud

### 高并发

### 分库分表

CDN技术

### 消息队列

ActiveMQ

## 监控

### 监控什么

CPU、内存、磁盘I/O、网络I/O等

### 监控手段

进程监控、语义监控、机器资源监控、数据波动

### 监控数据采集

日志、埋点

## Dapper

### 负载均衡

tomcat负载均衡、Nginx负载均衡

四层负载均衡、七层负载均衡

### DNS

DNS原理、DNS的设计

CDN

数据一致性

# 扩展篇

## 云计算

IaaS、SaaS、PaaS、虚拟化技术、openstack、Serverlsess

搜索引擎

Solr、Lucene、Nutch、Elasticsearch

权限管理

Shiro

## 区块链

哈希算法、Merkle树、公钥密码算法、共识算法、Raft协议、Paxos 算法与 Raft 算法、拜占庭问题与算法、消息认证码与数字签名

比特币

挖矿、共识机制、闪电网络、侧链、热点问题、分叉

以太坊

超级账本

人工智能

数学基础、机器学习、人工神经网络、深度学习、应用场景。

常用框架

TensorFlow、DeepLearning4J

IoT

量子计算

AR & VR

其他语言

Groovy、Python、Go、NodeJs、Swift、Rust