***1***

**基础篇**

### 01面向对象

**→ 什么是面向对象**

面向对象、面向过程:面向对象是具体化的,流程化的,解决一个问题,需要分步骤分析先后顺序,并逐步实现.面向对象是模型化的,抽象出类,作为封闭对象,在这个环境中有数据也有解决问题的方法,需要使用的时候直接使用就行了,而你不需要知道具体实现的方法.

##### 面向对象的三大基本特征和五大基本原则:

三大基本特征:

封装:封装就是隐藏对象的属性和实现细节,仅对外公开接口,控制在程序中属性的读和修改的访问级别,将抽象得到的数据和行为相结合,形成一个有机的整体.目的是增强安全性和简化编程.

继承:子类继承父类的方法和实例参数.目的:加强构建类的生态,提高代码的**则**:一个类应该仅有一个引起它变化的原因

**开放封闭原则**:既开放又封闭,对扩展开放,对更改封闭.坚持这个原则就必须尽量考虑接口封装.抽象和多态技术

**里氏替换原则**:子类可以替换父类并且出现在父类能够出现的任何地方.父类应该尽可能使用接口或者抽象类来实现.这样在根据新要求扩展父类接口的新子类的时候不影响当前客户端的使用.

**依赖倒置原则**:传统的结构化变成中,最上层的模块通常都要依赖下面的子模块实现,也称为高层依赖底层,该原则就是要逆转这种依赖关系.

接口隔离原则:使用专门的接口要比使用单个接口好得多.

**→ 平台无关性**

**Java 如何实现的平台无关:**一种语言在计算机上的运行不受平台的约束,一次编译到处运行.java经过编译后生成的.class的字节码文件,运行平台上只要有JVM就能运行,不需要进行再次编译.

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:**在函数时将实际参数赋值一份传递到函数中,这样在函数中如果对参数进行修改,将不会影响到实际参数(会创建副本)**传递对象类型,是将实际参数的地址赋值了一份,传递给了形式参数**

**引用传递:**在调用函数时将实际参数赋值一份传递到函数中,这样在函数中如果对参数进行修改,将不会影响到实际参数.(不会创建副本)

**为什么说 Java 中只有值传递:** 8大基本数据类型的传递方式是值传递 引用数据类型是复制参数的地址进行传递,也属于值传递

**→ 封装、继承、多态**

##### 什么是多态、方法重写与重载:

**多态**:同一个方法的调用,由于对象不同可能会有不同的行为

**重载**:函数或者方法又相同的名称,但是参数列表不相同的情形,这样的同名不同参数的函数或者方法之间,互相成为重载

**方法重写**:子类可继承父类的方法,而不需要重新编写相同的方法.但有时子类并不想原封不动地继承父类的方法,而是想作一定的修改.子类函数的访问权限不能小于父类.

需要重写方法的原因:父类的功能无法满足子类的需求.

**Java 的继承与实现**

##### 构造函数与默认构造函数

**构造函数**:java中的一种特殊函数,函数名与类名相同.一般用来初始化成员属性和成员方法的,即new对象产生后,就调用了对象的属性和方法.构造函数是对象一建立就运行,给对象初始化属性和执行方法中的语句.而一般函数是对象调用才执行.

**默认构造函数:**当一个类中没有定义构造函数时,系统会给该类中加一个默认的空参数的构造函数,方便该类初始化.只是该空参构造函数是隐藏不见的.当在该类中自定义了构造函数,默认构造函数就没有了.如果仍要构造函数,则需要自己在类中手动添加.

**类变量、成员变量和局部变量**

**类变量**:归属类的变量,它是通过在定义类的属性时,增加static修饰符,所以又称为静态变量.类变量不仅可以直接通过 类名.变量名 来访问,也可以通过 类对象.变量 来访问.

**成员变量:**归属于类的实例变量,没有经过static修饰,成员变量存在于类之内,方法和代码块之外.实例变量只能通过类对象.变量访问.

**局部变量:**在类中方法体里面所定义的变量,不管是方法的形参,还是方法体内所定义的变量都是局部变量.局部变量的作用域是以其所在方法体的头大括号开始到到尾大括号结束**.**

**成员变量和方法作用域**

### 02 Java 基础知识

#### → 基本数据类型

##### 8 种基本数据类型：整型、浮点型、布尔型、字符型

整型中 byte、short、int、long 的取值范围

byte: −128 ~ 127;2^8

short: −32768 ~ +32767 ;2^15

int: −2147483648 ~ +2147483647;2^31

long:−9223372036854775808~+9223372036854775807;2^63

##### 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？

浮点类型用于表示小数的数据类型,原理是二进制科学计数法.浮点类型有float和double两种

单精度(float):单精度浮点数在机内占4个字节,有效数字8位,表示范围为: -3.40E+38 ~ +3.40E+38

双精度(double):双精度浮点数在机内占8个字节,有效数字16位,表示范围为: -1.79E+308 ~ +1.79E+308

浮点型十进制转换为二进制的时候可能会不准确,这样就导致了误差.

所以表示较大金额的时候会出现精度的损失,这个时候一般使用BigDecimal来进行接受参数

解决精度损失:可以将浮点型放大对应小数点后位数的倍数,然后赋为整型,然后再除以位数,降低精度损失.

##### equals和 == 的区别

==:在基本数据类型中使用,比较的是他们的值;在引用数据类型中使用的时候,比较的是他们在内存中的存放地址(堆内存地址)

equals:不能作用于基本数据类型的变量.如果没有对此方法进行重写,则比较的是引用类型的变量所指向的对象地址.如果重写了,则比较的是所指向的对象的内容.例如:String

#### → 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

包装类型:基本类型不具备对象的性质,为了将其对象化就出现了包装类型,包装类型使其拥有了对象的性质,丰富了基本类型的操作.

基本类型:java中有8中基本类型,基本类型都存储在栈中,因此它们的存取速度要快于存储在堆中的对应包类的实例对象.

##### Integer 的缓存机制

Integer是对小数据(-128-127)是有缓存的,在jvm初始化的时候,数据-128-127之间的数字便被缓存到了本地内存中,这样,如果初始化-128-127之间的数字,便会从内存中直接去出,而不需要再新建一个对象.

例:Integer num1 = 100;

Integer num2 = 100;

Integer num3 = 128;

Integer num4 = 128;

num1 == num2 ; num3 != num4;

#### → String

##### 字符串的不可变性

一旦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.注意:Integer的缓存数据是不会被垃圾回收的.

##### JDK 6 和 JDK 7 中 substring 的原理及区别

private final char value[];

private final int offset;

private final int count;

JDK6中的substring创建了一个新的String对象,但是String的value[]属性域仍然指向堆内存中的原来的那个数组,区别是两个对象的count和offset两个值不同了.问题:当字符串非常长的时候,substring只需要很小的一部分,虽然你只需要一小部分,但是持有了整个长字符串value[]的引用,导致大量内存被占用,JDK6中只能手动指向一个真正的字符串: x = x.substring(begin, end) + "";

JDK7中问题得到了改进,substring()真实的创建了另一个字符数组.

##### replaceFirst、replaceAll、replace 区别

替换范围不同

##### String 对“+”的重载、字符串拼接的几种方式和区别

+ 方式:

concat方式:当两个量都为String类型且值不为null,可以使用

append:当需要拼接至少三个量的时候,可以考虑使用StringBuffer.append()以避免临时字符串的产生,StringBuffer大多数情况线程安全.若不会出现多线程同时对同一实例并发进行append操作,建议使用非线程安全的StringBuilder以获得更好的性能.

StringBuilder：适用于单线程下在字符缓冲区进行大量操作的情况

StringBuffer：适用多线程下在字符缓冲区进行大量操作的情况.StringBuffer中很多方法可以带有synchronized关键字,所以可以保证是线程安全的.进行多线程操作建议使用StringBuffer,如果在单线程情况下,还是建议使用StringBuilder性能更高.

##### String.valueOf 和 Integer.toString 的区别

Integer.toString():该方法返回指定整数的有符号位的String对象,以10进制字符串形式返回.

String.valueOf():重载了很多方法,将不同的类型转换为String.

##### switch 对 String 的支持

switch比较的是字符串常量的哈希值(int类型),但是hash值可能会有冲突,所以还需要再调用equals方法进行二次比较.

##### 字符串池、常量池（运行时常量池、Class 常量池）、intern

平时所说的内存是运行时数据区,字符串的创建有关的是方法区,堆区和栈区.

1. 方法区:存储类信息、常量、静态变量，全局共享。
2. 堆区：存放对象和数组。 全局共享
3. 栈区：基本数据类型、对象的引用都存放在这。线程私有。

字面量创建字符串会现在字符串池中找，看是否有相等的对象，没有的话就在堆中创建，把地址驻留在字符串池；有的话则直接用池中的引用，避免重复创建对象。

new关键字创建时，前面的操作和字面量创建一样，只不过最后在运行时会创建一个新对象，变量所引用的都是这个新对象的地址。

**intern**：如果常量池中存在当前字符串，就会直接返回当前字符串；如果常量池中没有此字符串，会将此字符串放入常量池后，再返回。JDK1.7中调用intern(),如果这个字符串在常量池中是第一次出现,则不会重新创建对象,而是直接返回他在堆中的引用.

**字符串常量池**：字符串常量池在堆中，用来存放字符串常量和字符串对象的引用。字符串常量池中的字符串只存在一份

**class常量池：**class文件中的常量池，用于存放编译器生成的各种字面量和符号引用。在编译阶段存放的是常量的符号引用，也就是说他们存的并不是对象实例。

**方法区中的运行时常量池：**当类加载到内存后，jvm就会将class常量池中的内容存放到运行时常量池中，运行时常量池存在于内存中，也就是class常量池被加载到内存之后的版本。

#### → 熟悉 Java 中各种关键字

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

**transient**：是类型修饰符，只能用来修饰字段，在对象序列化的过程中，标记为transient的变量不会被序列化。把一个对象的表示转化为字节流的过程称为序列化，从字节流中把对象重建出来称为反序列化。transient为不应被串行化的数据提供了一个9语言级的标记数据方法。

**volatile**：变量修饰符。volatile修饰的成员变量在每次被线程访问时，都强迫从共享内存中重读改成员变量的值。而且，当成员变量发生变化时，强迫线程将变化值回写到共享内存。这样在任何时刻，两个不同的线程总是看到某个成员变量是同一个值。

使用建议：在两个或者更多的线程访问的成员变量上使用volatile。当要访问的变量已在synchronized代码块中，或者为常量时，不必使用。

由于使用volatile屏蔽掉了VM中必要的代码优化，所以在效率上比较低，因此一定在必要时才使用此关键字。

**instanceof**：instanceof运算符的前一个操作符是一个引用变量，后一个操作数通常是一个类（可以是接口），用于判断前面的对象是否是后面的类，或者其子类、实现类的实例。类型为boolean。使用instanceof关键字做判断时，左右的操作数必须有继承或者实现关系。注意：在判断某个类的对象是不是其他类的实例时，一定要首先进行向上转型，然后才可用instanceof关键字进行判断，只是基本操作规范

**final：**final关键字可以用来修饰类、方法和变量。

当用来修饰一个类时，表明这个类不能被继承。也就是说，如果一个类你永远不会让他被继承，就可以使用final进行修饰。除非该类真的不会用来继承或者处于安全角度考虑，尽量不要将类设计成final类。

当用来修饰方法时，禁止该方法在子类中被重写。父类中final修饰的方法同时访问控制权限为private，将会导致子类中不能直接继承到此方法，此时可以在子类中定义相同的方法名和参数，此时不在产生重写与final的矛盾，而是在子类中定义了新的方法。类的private方法会隐式地被指定为final方法。

当用来修饰变量时，修饰一个基本数据类型表示该基本数据类型的值一旦在初始化后便不能发生变化，如果final修饰一个引用类型时，则在其初始化猴便不能再让其指向其他对象了，但该引用所指向的对象的内容是可以发生变化的。 修饰成员变量不需要显示初始化，可以在变量声明的时候初始化，也可以在声明变量时不赋初值，在该变量所在类的所有构造函数中对这个变量赋初值。

**static**：用途,方便在没有创建对象的情况下来进行调用(方法/变量).被static关键字修饰的方法或者变量不需要依赖于对象来进行访问,只要类被加载了,就可以通过类名去进行访问。在静态方法中不能访问类的非静态成员变量和非静态成员方法，因此非静态成员方法、变量都是必须依赖具体的对象才能被调用。static关键字还有一个比较关键的作用，用来形成静态代码块以优化程序性能。static块可以置于类中的人和地方，类中可以有多个static块。在初次被加载的时候，会按照static块的顺序来执行每个static块，并且只会执行一次（优化程序性能）。

**静态变量和非静态变量的区别**：静态变量被所有的对象共享，在内存中只有一个副本，它当且仅当在类初次加载时会被初始化。而非静态变量是对象所拥有的，在创建对象的时候被初始化，存在多个副本，各个对象拥有的副本互不影响。

**synchronized**：四种修饰方式:

1. 修饰一个代码块,被修饰的代码块称为同步语句块,其作用的范围是{}中的代码,作用的对象是调用这个代码块的对象
2. 修饰一个方法,被修饰的方法称为同步方法,其作用的范围是整个方法,作用的对象是调用这个方法的对象.
3. 修改一个静态的方法,其作用的范围是整个静态方法,作用的对象是这个类的所有对象
4. 修改一个类,其作用的范围是synchronized后面括号括起来的部分,作用的对象是这个类的所有对象

总结:A.无论synchronized关键字加在方法上还是对象上,如果它作用的对象是非静态的,则它取得的锁是对象;如果synchronized作用的对象是一个静态方法或一个类,则它取得的锁是对类,该类所有的对象是同一把锁

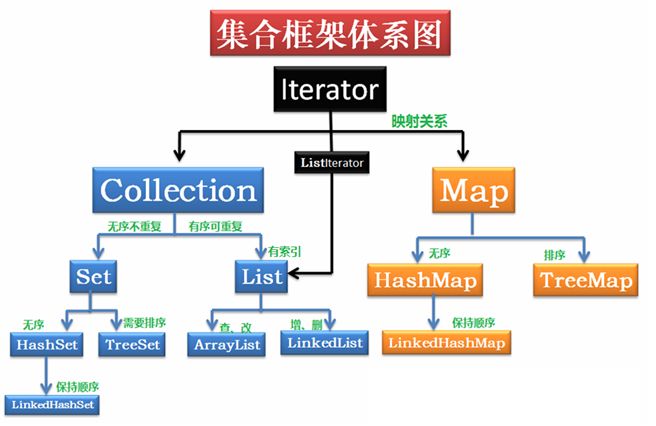
B.每个对象只有一个锁与之相关联,谁拿到这个锁谁就可以运行它所控制的那段代码

C.实现同步是要很大的系统开销作为代价的,甚至可能造成死锁,所以尽量避免无谓的同步控制

const：跟goto相同 保留字以备扩充

#### → 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

#### → 反射

反射与工厂模式、反射有什么用

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 KTVE? 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<Object> 和原始类型 List 之间的区别?

List<?> 和 List<Object> 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

03 阅读源代码

String、Integer、Long、Enum、

BigDecimal、ThreadLocal、ClassLoader & URLClassLoader、

ArrayList & LinkedList、

HashMap & LinkedHashMap & TreeMap & CouncurrentHashMap、HashSet & LinkedHashSet & TreeSet

04 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

#### ****→ 什么是线程，与进程的区别****

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

什么是死锁

死锁如何解决

#### → 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****

#### ****→ 并方包****

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

***2***

**底层篇**

01JVM

#### → JVM 内存结构

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

02 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）

03 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

***3***

**进阶篇**

01 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

02 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****

#### ****→ 实现 AOP****

#### ****→ 实现 IOC****

#### ****→ nio 和 reactor 设计模式****

03 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

#### ****→ http/1.0 http/1.1 http/2 之前的区别****

http 中 get 和 post 区别

常见的 web 请求返回的状态码

404、302、301、500分别代表什么

#### ****→ 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

反向代理服务器

04 框架知识

**→ Servlet**

生命周期

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

#### Hibernate 的懒加载

Hibernate 的缓存机制

Hibernate / Ibatis / MyBatis 之间的区别

#### ****→ Spring****

#### Bean 的初始化

#### AOP 原理

#### 实现 Spring 的IOC

#### Spring 四种依赖注入方式

#### ****→ Spring MVC****

什么是 MVC

Spring mvc 与 Struts mvc 的区别

#### ****→ Spring Boot****

Spring Boot 2.0、起步依赖、自动配置、

Spring Boot 的 starter 原理，自己实现一个 starter

#### ****→ 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

05 应用服务器知识

**→ JBoss**

#### ****→ tomcat****

#### ****→ jetty****

#### ****→ Weblogic****

06 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

***4***

**高级篇**

01 新技术

**→ Java 8**

lambda 表达式、Stream API、时间 API

#### ****→ Java 9****

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****

02 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

03 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

04 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****

#### ****→ 编译过程****

#### 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

05 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****

06 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、

#### ****→ 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****

#### ****→ 读写分离****

#### ****→ 常用的 NoSql 数据库****

redis、memcached

#### ****→ 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****

07 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

#### ****→ 排序算法****

稳定的排序：冒泡排序、插入排序、鸡尾酒排序、桶排序、计数排序、归并排序、原地归并排序、二叉排序树排序、鸽巢排序、基数排序、侏儒排序、图书馆排序、块排序

不稳定的排序：选择排序、希尔排序、Clover 排序算法、梳排序、堆排序、平滑排序、快速排序、内省排序、耐心排序

各种排序算法和时间复杂度

#### ****→ 两个栈实现队列，和两个队列实现栈****

#### ****→ 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。

08 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****

09 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****

***5***

**架构篇**

01 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

02 微服务

SOA、康威定律

#### ****→ ServiceMesh****

sidecar

#### ****→ Docker & Kubernets****

#### ****→ Spring Boot****

#### ****→ Spring Cloud****

03 高并发

**→ 分库分表**

#### ****→ CDN 技术****

#### ****→ 消息队列****

ActiveMQ

04

 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****

05 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

06 DNS

DNS 原理、DNS 的设计

07 CDN

数据一致性