# Java编程

## 基础

### Jdk和Jre和JVM的区别？

JDK（Java Development Kit）、JRE（Java Runtime Environment）和JVM（Java Virtual Machine）是Java开发和运行的关键组件，它们之间的主要区别如下：

1. JVM（Java Virtual Machine）：

- JVM是Java虚拟机的缩写，它是Java程序运行的核心组件。

- JVM负责将Java字节码翻译成机器码，并在各种平台上运行Java程序，实现了Java的"一次编译，到处运行"的跨平台特性。

- JVM还提供了垃圾回收、内存管理和安全性等功能，确保Java程序的稳定性和安全性。

2. JRE（Java Runtime Environment）：

- JRE是Java运行时环境的缩写，它是Java程序的运行环境。

- JRE包含了JVM以及运行Java程序所需的类库和其他支持文件，例如Java标准库、运行时的配置文件等。

- 当用户想要运行Java程序时，只需要安装JRE即可，无需安装JDK。

3. JDK（Java Development Kit）：

- JDK是Java开发工具包的缩写，它是用于开发Java程序的工具集合。

- JDK包含了JRE以及用于开发Java程序的编译器（javac）、调试器（jdb）和其他开发工具（如javap、javadoc等）。

- 开发人员通常在开发Java程序时需要安装JDK，因为它提供了开发过程中所需的所有工具和环境。

重点区别：

JVM是Java程序的运行时引擎，负责将Java字节码转换为机器码并执行程序。

JRE是Java程序的运行环境，包含了JVM以及运行Java程序所需的类库和其他支持文件。

JDK是Java开发工具包，包含了JRE以及用于开发Java程序的编译器和其他开发工具。

### Java语言有哪些特点？

### 什么是字节码？采用字节码的最大好处是什么？

### Java和C++的区别？

### Oracle JDK和OpenJDK的对比？

### Java有哪些数据类型？

### Switch 是否能作用在byte上，是否能作用在long上，是否能作用在String上？

### 用最有效率的方法计算2乘以8

### Math.round(11.5)等于多少？Math.round(-11.5)等于多少？

### float f = 3.4是否正确？

### short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？

### Java语言采用何种编码方案？有什么特点？

### 访问修饰符 public,private,protected,以及不写（默认）时的区别?

### &和&&的区别?

### Java为什么没有goto？

### final有什么用？

### final、finally和finalize的区别？

### this关键字的用法？

### this和 super的区别？

### static 详解/存在的主要意义/独特之处？

### static 应用场景？

### break，continue，return的区别和作用？

## 关键字

## String

### String是最基本的数据类型吗？

### 字符型常量和字符串常量的区别？

### 什么是字符串常量池？

### String有哪些特性？

### String为什么是不可变的？

### String为什么设计为不可变的？

### String真的是不可变的吗？

### 是否可以继承String类？

### String str = “i”和String str = new String(“1”)是一样的吗？

### String s = new String(“1”)创建了几个字符串对象？

### 如何将字符串反转？

### 数组有没有length方法？字符串有没有length方法？

### String类的常用方法都有哪些？

### 在使用HashMap时，用String做key有什么好处？

### String和StringBuffer的区别？

### StringBuilder和StringBuffer的区别？

### HashTable和HashMap、ArrayList和Vector、ArrayList和LinkedList的区别？

## 类和接口

抽象类和接口的对比，区别？

### 自动装箱和拆箱

### int和Integer的区别？

### Integer a = 127 和 Integer b = 127，a == b？

### 抽象类和接口的对比？

### 普通类和抽象类有哪些区别？

### 抽象类能用final修饰吗？

### 创建一个对象用什么关键字？

### 对象实例与对象引用有何不同？

### 成员变量和局部变量的区别有哪些？

### 在Java中定义一个不做事且没有参数的构造方法的作用？

### 一个类的构造方法的作用是什么？若一个类没有声明构造方法，该程序能正常执行吗？

### 构造方法有哪些特性？

### 静态变量和实例变量的区别？

### 静态变量和普通变量的区别？

### 静态方法和实例方法的区别？

### 静态方法内调用非静态成员为什么是非法的？

### 什么是方法的返回值？返回值的作用是什么？

### 什么是内部类？

### 内部类的分类有哪些？

### 内部类的优点？

### 内部类有哪些应用场景？

### 局部内部类和匿名内部类访问局部变量时，为什么变量必须加上final？

### 构造器（constructor）是否可被重写（override）？

### 重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分？

### == 和 equals 的区别是什么？

### HashSet如何检查重复

### 两个对象的 hashCode() 相同，则 equals() 也一定为 true，对吗？

### hashCode和equals方法的关系

### 面试官可能会问你：“你重写过 hashcode 和 equals 么，为什么重写equals时必须重写hashCode方法？”

### hashCode()介绍

### 为什么要有 hashCode

### hashCode()与equals()的相关规定？

### 对象的相等与指向他们的引用相等，两者有什么不同？

### 当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并

### 可返回变化后的结果，那么这里到底是值传递还是引用传递？

### 为什么 Java 中只有值传递？

### 值传递和引用传递有什么区别？

### JDK中常用的包有哪些？

### import java和import javax有什么区别？

## 面向对象

### 面向对象和面向过程的区别？

### 面向搞对象的三大特征？

### 什么是多态？Java语言是如何实现多态的？

### 面向对象的五大基本原则是什么？

## 集合

### 为什么hashmap从链表转换成红黑树的时候需要链表长度大于8数组长度大于64？

选择链表长度为8作为临界值的原因

经过测试后，链表长度大于8后再出现hash碰撞的可能性几乎为0。

由链表转换成红黑树是耗时的，虽然查找效率会提升，但是扩容时需要保证红黑树的平衡，红黑树的旋转也是需要花费时间的。

图形用户界面, 文本

描述已自动生成

## 注解

https://blog.csdn.net/weixin\_67450855/article/details/127689139

Java注解是附加在代码中的一些元信息，用于一些工具在编译、运行时进行解析和使用，起到说明、配置的功能，注解相关类都包含在java.lang.annotation包中。

## IO流

java中IO流分几种？

BIO，NIO，AIO有什么区别？

Files的常用方法有哪些？

## 代理

静态代理就是，编写一个代理类，该类实现与目标对象相同的接口，该代理类中有一个属性是目标对象（被代理的类），在代理类中调用目标对象的同名方法。可以在调用目标对象方法的前后，编写所需要添加的代码，即对目标对象方法进行增强。

静态代理的最大的问题是会带来代理类暴增的问题。需要手写很多类。

动态代理，创建一个代理类可以为很多一样的功能增强需求的类共同使用。

创建代理类，然后再使用代理类根据需要代理的类加载器，接口，对象实例就可以生成具有原来类的接口方法的对象实例了。

<https://blog.csdn.net/biyn9/article/details/131352424>

JDK的动态代理有一个限制，就是使用动态代理的对象必须实现一个或多个接口。如果想代理没有实现接口的类就可以使用CGLIB实现。

Cglib是一个强大的高性能的代码生成包，它可以在运行期扩展Java类与实现Java接口。它广泛的被许多AOP的框架使用，例如Spring AOP和dynaop，为他们提供方法的interception(拦截)。

Cqlib包的底层是通过使用一个小而快的字节码处理框架ASM，来转换字节码并生成新的类。不鼓励直接使用ASM，因为它需要你对JVM内部结构包括class文件的格式和指令集都很熟悉。

直接改的字节码。

## 反射

### 什么是反射机制？

### 反射机制的优缺点？

### 反射机制的应用场景有哪些？

### Java获取反射的3种方法？

什么是方法句柄？

方法句柄什么时候使用？

# Java内存模型

# Java并发编程

## 什么是乐观锁和悲观锁？

### 什么是可重入锁？

### ReentrantLock详解？

# Linux

# JVM

## 垃圾回收

## Java是如何实现的平台无关？

## Java是编译型还是解释型？

## 什么是编译和反编译？

## JIT优化技术是什么？

## 对JDK进程执行kill -9有什么影响？

## Java中的对象一定在堆上分配内存吗？

## Jvm的堆是如何分代的？为什么分代？

## 新生代如果只有一个Eden和一个Survivor可以吗？

## YoungGC和FullGC的触发条件是什么？

## 什么是Stop The World？

## JVM有哪些垃圾回收算法？

## JVM中堆内存中各部分内存区域的大小比值？

## JVM中什么时候需要调整Survivor区占新生代的比例？

## JVM中什么时候需要调整新生代占全部堆内存的比例？

## JVM中什么时候需要调整老年代占全部堆内存的比例？

## JVM如何判断对象是否存活？

## 什么是三色标记算法？

## Java中的4种引用有什么区别？

## 新生代和老年代的GC算法？

## 新生代和老年代的垃圾回收器有何区别？

## G1和CMS有什么区别？

## 为什么G1从JDK9后成为默认的垃圾回收器？

## JDK11中新出的ZGC有什么新特点？

## Java8和Java11中的GC有什么区别？

## 类的生命周期是怎么样的？

## Java中类加载的过程是什么样的？

## 什么是双亲委派？如何破坏？

## 如何判断JVM中类和其它类是不是同一个类？

## JVM如何保证对象分配内存的过程的线程安全？

## 虚拟机中的堆一定是线程共享的吗？

## 常见的JVM工具有哪些？

## 有哪些常用的JVM启动参数？

## 哪些语言有GC机制？

## 一个对象的结构是什么样的？

## JVM是如何创建对象的？

## 字符串常量池是如何实现的？

## 什么是方法区（元空间）？是如何实现的？

## JVM中一次完整的GC流程是如何实现的？

## JVM为什么要把堆和栈区分出来？

## 运行时常量池和字符串常量池的关系是什么？

## 什么是堆外内存？如何使用堆外内存？

## FullGC多久一次算正常？

## 什么是跨代引用？有什么问题？

## 内存泄漏和内存溢出的区别是什么？

## 内存泄漏都有哪些情况？

## 破坏双亲委派之后，能重写String类吗？

## OutOfMemory和 StackOverflow的区别是什么？

## 什么是Class常量池？和运行时常量池的关系是什么？

## Java发生了OOM一定会导致JVM退出吗？

## 什么是safe point,有啥用？

## JDK1.8和1.9中类加载器有哪些不同？

## 什么是逃逸分析？

## 什么是AOT编译？和JIT有什么区别？

## 一个Java进程占用的内存有哪些部分？

## 说一说JVM的并发回收和并行回收？

## 为什么初始标记和重新标记需要STW？而并发标记不需要？

## 介绍下CMS的垃圾回收流程

## Java一定是和平台无关的吗？

## 什么是STW？有什么影响？

## 什么情况下会导致JVM退出？

## Java 类加载的过程？

# Scala编程

## 基础

### scala和java的区别？

|  |  |
| --- | --- |
| Scala | Java |
| 使用val或者var声明变量，类型可以省略不写，编译器通过赋值自动推断类型 | 使用var自动推断类型，或者使用明确的类型声明 |
| 可以省略return关键字和分号 | 不可以省略 |
| For中可以使用守卫，但没有continue和break关键字 | For循环中没有守卫，但是有continue和break关键字 |
| 主构造函数直接在类的声明之后，辅助构造函数命名为this，辅助构造函数必须直接或者间接使用主构造函数 | 构造函数与类名一致 |
| 伴生对象可理解为静态类 |  |
| scala中的赋值语句返回结果是unit,不可以串联,例如x=y=1,x没有被赋值为1 | x=y=1，这样没问题 |

### scala语言有什么特点？

scala同时具有⾯向对象和函数式编程的多种编程范式。

### 什么是函数式编程？有什么优点？

函数式编程是一种编程范式，主要思想就是将程序写成一系列函数嵌套的形式，让程序的条例更清晰，灵活性更强。如函数参数是函数。使用不变变量，函数内不改变函数外的变量的值的纯函数。

### scala中的闭包是什么？

闭包就是能够读取函数外部变量的函数。

### scala中的纯函数是什么？有什么好处？

纯函数所有的输入通过参数传递到函数内部，所有的输出通过返回值传递到函数外部。

纯函数不改变函数外部的变量。闭包或者带有打印语句的函数为非纯函数。

### Scala的函数式编程带来了许多好处，这些好处包括：

### 清晰的代码结构： 函数式编程鼓励使用纯函数，这些函数不依赖于外部状态，也不会修改状态。这使得代码更易于理解和维护，因为函数的行为更加可预测。

### 可组合性： 函数式编程鼓励使用高阶函数和函数组合来构建复杂的功能。这使得代码更具可重用性和可组合性，因为函数可以轻松地与其他函数组合在一起，形成新的功能。

### 并发和并行性： 函数式编程强调不可变数据和纯函数，这使得并发和并行编程更容易。由于函数不会修改状态，因此并发执行不会导致意外的副作用。

### 更少的bug： 由于函数式编程的不可变性和纯函数性质，代码的行为更加可预测，从而减少了由于副作用和状态修改而引起的bug的可能性。

### 更容易的测试： 函数式编程鼓励编写纯函数，这些函数不依赖于外部状态或副作用。这使得测试更容易，因为函数的行为只取决于其输入，而不受外部状态的影响。

### 提高代码的可靠性和健壮性： 函数式编程通过强调不可变性和纯函数性质，可以减少代码中的意外行为，从而提高代码的可靠性和健壮性。

### 总的来说，Scala的函数式编程风格能够带来更清晰、更易于理解和维护的代码，同时也促进了代码的可重用性、可组合性和并发性。

### scala中的模式匹配是什么？

在Scala中，模式匹配是一种强大的功能，对应Java中的switch用于对数据结构进行匹配和解构。它类似于其他编程语言中的switch语句，但更加灵活和强大。模式匹配可以应用于各种数据类型，包括基本类型、集合、元组、样例类等。

Scala的模式匹配由关键字match开启，通常与case子句一起使用。match表达式将一个值与一系列模式进行匹配，然后执行与匹配模式相对应的代码块。每个case子句都包含一个模式和一个代码块，当输入值与该模式匹配时，对应的代码块将被执行。

下面是一个简单的示例，演示了如何在Scala中使用模式匹配：

|  |
| --- |
| def matchExample(x: Any): String = x match {  case 1 => "One"  case "hello" => "A string"  case true => "A boolean"  case \_ => "Something else" // \_ 通配符表示任何匹配  }  println(matchExample(1)) // 输出: One  println(matchExample("hello")) // 输出: A string  println(matchExample(false)) // 输出: Something else |

在这个例子中，输入值被匹配到了相应的case子句，然后执行对应的代码块。最后一个case子句使用了通配符\_，表示除了之前列出的模式之外的任何匹配。

模式匹配不仅限于基本类型，还可以用于复杂的数据结构，比如列表、元组和样例类等。它可以帮助编写简洁而易于理解的代码，同时也增加了代码的健壮性和可维护性。

### case class和class的区别？

case class是一个样例类

样例类默认将主构造器的参数使用val声明,作为类的成员属性，除非显示地声明为var

样例类默认添加伴生对象和全参的apply方法,方便对象构建

样例类默认添加全参的unapply方法,方便模式匹配和属性提取

样例类默认实现了toString, equals, hashcode, copy⽅法（用来构建一个与现有值相同的新对象），适合用于对不可变数据建模。

默认是可序列化的，也就是实现了Serializable

class是一个不具有上述样例类特点的普通类。

### scala中的隐式转换是什么？

隐式函数

在Scala中，隐式转换是一种功能，允许编译器在需要时自动地将一种类型转换为另一种类型。这种转换在编译器的控制下进行，而不需要显式地调用转换函数。隐式转换可以帮助简化代码，提高代码的可读性和可维护性。

假设你有一个表示英寸的类型：

|  |
| --- |
| case class Inch(value: Double)  现在，假设你希望能够将英寸转换为厘米，你可定义一个隐式转换函数来完成这个任务：  implicit def inchToCm(inch: Inch): Double = inch.value \* 2.54  现在，当你尝试将Inch类型的值赋给Double类型的变量时，编译器会自动调用该隐式转换函数：  val inchValue = Inch(10.0)  val cmValue: Double = inchValue // 这里会自动调用隐式转换函数将英寸转换为厘米  这就是一个简单的隐式转换示例，通过隐式转换函数，我们可以实现英寸到厘米的自动转换，而不需要显式调用转换函数。 |

隐式类

隐式变量

### scala中伴生类和伴生对象是什么？

1. 在scala中，object与class名字完全相同，且声明在同一个scala源代码文件中，那么该对象被称为该类的伴⽣对象，该类被称为该对象的伴⽣类;
2. 伴⽣对象和伴⽣类可以互相访问其私有成员。

### scala中Either是什么？

Either是一个类用于返回两种可能的类型，接收结果时可使用模式匹配判定 [Either详解](https://geek-docs.com/scala/scala-questions/276_scala_scala_extract_either_type_parameters.html)

### scala中的柯里化是什么？

柯里化（Currying）是一种函数式编程的技术，其主要思想是将接受多个参数的函数转换为一系列接受单个参数的函数链。这个过程可以让函数更加灵活，并支持部分应用（Partial Application）和函数组合（Function Composition）。

在Scala中，柯里化是一种常见的编程技术，它可以通过在函数参数列表中使用多个参数列表来实现。例如：

def add(x: Int)(y: Int): Int = x + y

这个函数接受两个参数，但是它使用了两个参数列表。这种定义方式使得我们可以像这样使用该函数：

val addOne = add(1) \_ // 返回一个函数，将1加到传入的参数上

val result = addOne(5) // 返回6

### scala的尾递归是什么？

尾递归（Tail Recursion）是一种特殊的递归形式，在函数的最后一步调用自身，并且没有其他操作。这种形式的递归可以被优化为迭代，从而避免了递归调用造成的栈溢出风险。

Scala对尾递归提供了特殊的支持，称为尾递归优化（Tail Call Optimization）。当Scala编译器检测到一个函数调用是尾递归的时候，它会将其转换为迭代形式，从而消除了递归调用所带来的额外开销。

以下是一个简单的尾递归示例，计算阶乘：

object Main extends App {

// 尾递归形式的阶乘函数

def factorial(n: Int): Int = {

@scala.annotation.tailrec

def factorialHelper(n: Int, acc: Int): Int = {

if (n <= 1) acc

else factorialHelper(n - 1, acc \* n) // 尾递归调用

}

factorialHelper(n, 1)

}

val result = factorial(5)

println(result) // 输出: 120

}

在 Scala 中，不加 @scala.annotation.tailrec 注解，编译器也会尝试进行尾递归优化。Scala 编译器会尝试检测函数是否是尾递归的，如果是的话，会自动进行尾递归优化。

然而，这种自动优化并不是总能生效的。编译器有一些限制，只有当递归调用是函数的最后一步操作，并且没有其他操作时，才能进行尾递归优化。如果函数不符合这些条件，编译器将无法进行尾递归优化。

因此，尽管编译器会尝试进行尾递归优化，但是为了确保优化生效，最好使用 @scala.annotation.tailrec 注解明确告知编译器这是一个尾递归函数。这样做可以提高代码的可读性，并在无法自动进行尾递归优化时得到编译器错误提示，避免意外的性能问题。

### scala中高阶函数是什么？

1. 一个函数作为其他函数的参数
2. 一个函数作为其他函数的返回值

### 变长参数跟list作为参数的区别？

变长参数只能作为最后一个形参。变长参数在调用传参时语义更明确。

### 如何实现单例模式？

在 Scala 中，实现单例模式有多种方法，以下是其中两种常见的方式：

1. 使用对象（Object）：

object SingletonObject {

def greet(): Unit = println("Hello from SingletonObject")

}

在这种方式中，Scala 的对象（Object）是一种懒加载的单例，它们在第一次使用时被创建，并且保证在整个应用程序的生命周期中只有一个实例存在。可以直接通过对象名访问其成员，例如 `SingletonObject.greet()`。

2. 使用类和伴生对象：

class SingletonClass private () {

def greet(): Unit = println("Hello from SingletonClass")

}

object SingletonClass {

private val instance = new SingletonClass

def getInstance(): SingletonClass = instance

}

在这种方式中，我们将类的构造函数私有化，然后在伴生对象中创建一个私有的静态实例，并提供一个公共的静态方法 `getInstance()` 来获取该实例。这样做可以保证只有一个实例存在，并且通过 `getInstance()` 方法来获取该实例。

使用哪种方式取决于需求和偏好。通常情况下，使用对象更加简洁和方便，因为对象在定义时就已经是单例了，而使用类和伴生对象需要更多的代码来管理单例实例的创建和访问。

### trait特质是什么

### scala的高阶函数是什么？

### 隐式参数使用的优先顺序是什么？

### Option类型的定义和使用场景？

Option类型的定义和使用场景？

在Java中， null是⼀个关键字，不是⼀个对象，当开发者希望返回⼀个空对象时，却返回了⼀个关键字，为了解决这个问题， Scala建议开发者返回值是空值时，使⽤Option类型，在Scala中null是Null的唯⼀对象，会引起异常， Option则可以避免。 Option有两个⼦类型， Some和None（空值）。

### Unit类型是什么？

Unit代表没有任何意义的值类型，类似于java中的void类型，他是anyval的⼦类型，仅有⼀个实例对象"( )"

### Scala类型系统中Nil, Null, None, Nothing四个类型的区别？

Nil是Null的实例，Null是所有引用类型的子类。

None是Option[Nothing]的实例/对象。

Nothing表示什么都没有，Nothing是所有类型的子类，用于类型系统的完整，比如，在返回值出错不能返回时，表示什么都没有返回。

### 协变和逆变是什么？

首先定义的一个Trait Q[T]{}

• 协变 ： 如果 A 是B的子类，那么Q[A] 是 Q[B]的子类，Q这样定义Trait Q[+T]{}

• 逆变： 如果A是B的子类，那么Q[B]是Q[A]的子类，Q这样定义Trait Q[-T]{}

• 非变： 没有任何从属关系，Q这样定义Trait Q[T]{}

# Scala并发编程

# **Hadoop**

## **基础**

### Hadoop是什么？

### Hadoop HA的整体架构？

### Hadoop的配置文件有哪些？

### Hadoop常用命令？

文本, 信件

描述已自动生成

### MapReduce程序编写实例(word count)

### combiner 和 partition 的作用？

### 如果数据误删，如何抢救?

### 如何理解Hadoop中的数据倾斜现象？

### Hadoop2中block size默认是多大？

### Hadoop3中block size默认是多大？为什么变大了？

### Safe Mode是什么？

### Hadoop启动流程？

### Hadoop 框架中怎么来优化？

## **HDFS**

### HDFS的组织架构？

### HDFS读写流程？

### HDFS 存储的机制?

### NameNode在启动的时候会做哪些操作？

### 如何检查NameNode是否正常运行？重启NameNode的命令是什么？

### 避免NameNode故障导致集群宕机的解决方法是什么？

### HDFS在读取文件的时候，如果其中一个块突然损坏了怎么办？

### HDFS在上传文件的时候，如果其中一个DataNode突然挂掉了怎么办？

### Secondary NameNode的工作机制是怎样的？

### Secondary NameNode不能恢复NameNode的全部数据，那如何保证NameNode数据存储安全？

### 在NameNode HA中，会出现脑裂问题吗？怎么解决脑裂？

### 小文件过多会有什么危害，如何避免？

### NameNode与Secondary NameNode的区别与联系？

NameNode是HDFS的关键组件之一，负责管理文件系统的命名空间和元数据。它维护了文件系统树及其所有文件和目录的元数据，例如文件的名称、目录结构、权限和块列表。

NameNode还负责处理客户端请求，包括读取、写入和删除文件等操作。它通过在内存中维护文件系统元数据来提供快速的访问。

SecondaryNameNode并不是NameNode的备用节点，而是辅助NameNode的工具，用于定期合并和编辑NameNode的编辑日志（Edit Log）和镜像文件（FsImage）。

SecondaryNameNode的主要作用是定期将NameNode的编辑日志与镜像文件合并，然后生成新的镜像文件，这样可以降低NameNode崩溃时恢复数据的时间。

联系：

SecondaryNameNode帮助确保NameNode的稳定性和可靠性。通过定期合并编辑日志和镜像文件，它可以减少在NameNode故障时需要进行的元数据恢复操作的时间。

虽然SecondaryNameNode可以帮助恢复NameNode的镜像文件，但它不能实时替代NameNode。在NameNode发生故障时，需要通过高可用性解决方案（如Hadoop的HA方案）来实现快速的故障转移。

文件系统的命名空间指的是文件系统中文件和目录的层次结构，以及这些文件和目录在该结构中的命名方式。在Hadoop的分布式文件系统（HDFS）中，文件系统的命名空间由文件和目录组成的树形结构构成。

元数据是指描述文件和目录属性的数据，它包括了文件系统中的所有关于文件和目录的信息，但不包括文件的实际内容。在HDFS中，元数据通常包括以下内容：

文件名称：文件的名称，用于标识文件在文件系统中的唯一位置。

文件大小：文件的大小，以字节为单位。

文件权限：文件的访问权限，包括读、写和执行权限。

文件所有者：文件的所有者，即文件的创建者或拥有者。

文件所属组：文件所属的用户组。

文件的块信息：文件在HDFS中的分布式存储位置，包括文件的块大小和每个块所在的DataNode。

文件的时间戳：包括文件的创建时间、最后修改时间和最后访问时间。

目录结构：目录的名称和父目录的信息。

文件副本数量：指定文件的副本数量，即文件在HDFS中的冗余副本数。

这些元数据信息由NameNode维护，并存储在内存中以提供快速访问。元数据的管理对于文件系统的正常运行和性能至关重要。

### Secondary NameNode的目的是什么？

SecondaryNameNode定期触发CheckPoint，代表NameNode合并编辑日志EditLog和镜像文件Fsimage，从而减小EditLog的大小，减少NN启动时间。同时在合并期间，NameNode也可以对外提读写操作。

### DataNode详解

Hadoop中的DataNode是Hadoop分布式文件系统（HDFS）的核心组件之一，负责存储实际的数据块并处理客户端的读写请求。

存储数据块：

DataNode负责存储HDFS中的数据块。数据块是文件在HDFS中的逻辑单位，通常具有默认的大小（例如128MB）。当客户端向HDFS写入文件时，NameNode会将文件划分为多个数据块，并将这些数据块分配给不同的DataNode进行存储。DataNode将数据块存储在本地文件系统上，通常是在本地磁盘上。

处理读写请求：

客户端向HDFS发出读取或写入请求时，请求首先发送到NameNode。NameNode确定数据块的位置信息，并返回给客户端。

客户端通过与对应DataNode通信来读取或写入数据。如果是读取请求，客户端直接从DataNode读取数据块；如果是写入请求，客户端首先将数据块写入到DataNode，然后DataNode会将数据块复制到其他DataNode以提供容错性。

心跳和块报告：

DataNode定期向NameNode发送心跳消息，以确认其正常运行。DataNode还定期向NameNode发送块报告，告知NameNode它所存储的所有数据块的信息，包括数据块的ID、大小和存储位置等。

数据块复制和容错：

HDFS通过数据块的复制来提供容错性和高可用性。当数据块存储在某个DataNode上时，它会被复制到其他DataNode上，以防止数据丢失。

当DataNode发生故障或数据块丢失时，NameNode会使用复制的数据块进行数据恢复，以确保数据的可靠性。

总的来说，Hadoop DataNode的主要工作是存储数据块、处理读写请求、与NameNode保持通信以及实现数据块的复制和容错，从而构建了Hadoop分布式文件系统（HDFS）的基础。

### NameNode和DataNode的心跳机制?

### DataNode 在什么情况下不会备份?

### 一个 DataNode 宕机,怎么一个流程恢复？

### namenode 宕机,怎么解决？

先分析宕机后的损失，宕机后直接导致 client 无法访问，内存中的元数据丢失，但是硬盘中的元数据  
应该还存在，如果只是节点挂了，重启即可，如果是机器挂了，重启机器后看节点是否能重启，不能  
重启就要找到原因修复了。但是最终的解决方案应该是在设计集群的初期就考虑到这个问题，做  
namenode 的 HA。  
3. 一个 datanode 宕机,怎么一个流程恢复  
Datanode 宕机了后，如果是短暂的宕机，可以实现写好脚本监控，将它启动起来。如果是长时间宕机  
了，那么 datanode 上的数据应该已经被备份到其他机器了，那这台 datanode 就是一台新的 datanode  
了，删除他的所有数据文件和状态文件，重新启动。

## **MapReduce**

### MapReduce原理？

### MR中Map Task的工作机制？

### MR中Reduce Task的工作机制？

### 请说下MR中Shuffle阶段？

### Shuffle阶段的数据压缩机制？

### 在写MR时，什么情况下可以使用规约（combiner）？

### 用 MapReduce 来实现下面需求？

现在有 10 个文件夹,每个文件夹都有 1000000 个 URL，现在让你找出 top1000000url。

### Hadoop 中 Combiner 的作用?

### combiner 出现在哪个过程？

### 用 MapReduce 怎么处理数据倾斜问题？

### 我们开发 job 时，是否可以去掉 reduce 阶段？

### MapReduce 的 map 数量 和 reduce 数量怎么确定, 怎么配置?

### 一个Hadoop环境，整合了HBase和Hive，是否有必要给HDFS和HBASE都分别配置压缩策略？请给出对压缩策略的建议？

### 常见的压缩格式？

## Yarn

### YARN的集群架构？

### YARN工作原理?

### YARN的任务提交流程是怎样的?

### YARN的3种资源调度模型?

## Hadoop高可用

# Spark

## 基础

### 为什么Spark比Hadoop的MapReduce快？

1. Shuffle  
   Hadoop map reduce的中间数据必须落盘，spark并不是必须的，shuffle中间数据可以保留在内存中。spark的map reduce中间是宽依赖或者窄依赖，如果是窄依赖，则不需要shuffle。  
   Hadoop map reduce 的中间过程必须排序，spark的shuffle在sort merge join时需要排序，在broadcast hash join、shuffle hash join、broadcast Nested Loop Join和shuffle nested loop join 时并不需要排序。
2. Directed Acyclic Graph (Dag， [有向无环图](https://www.zhihu.com/search?q=%E6%9C%89%E5%90%91%E6%97%A0%E7%8E%AF%E5%9B%BE&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22answer%22%2C%22sourceId%22%3A2977285638%7D))  
   每一次hadoop map reduce的map开始必须从磁盘读取，reduce结果必须写入磁盘。对于spark来说，多个map reduce过程被合并成一个dag，每个stage的数据读取不一定需要从磁盘中读取，除最后的运行结果不一定要保存到磁盘。
3. cache  
   Spark支持将需要反复用到的数据给Cache到内存中，减少数据加载耗时，所以Spark跑机器学习算法比较好（需要对数据进行反复迭代）。
4. 调度方式  
   在不断启停新旧task时候，进程的启动和停止需要做很多初始化工作，因此采用进程的方式会降低执行效率。**当一个机器上多个map task需要读取同一份字典来进行数据过滤或者提取的时候，需要将字典加载到每个map task进程中，会造成资源重复加载，存在浪费内存资源的问题。**Spark通过复用线程池中的线程来减少启动、关闭task所需要的开销。
5. 增加并行度  
   由于将中间结果写到磁盘与从磁盘读取中间结果属于不同的环节，Hadoop将它们简单地通过串行执行衔接起来。Spark把不同的环节抽象为Stage，允许多个Stage既可以串行执行，又可以并行执行。
6. 避免重新计算  
   当Stage中某个分区的Task执行失败后，会重新对此Stage调度，但在重新调度的时候会过滤已经执行成功的分区任务，所以不会造成重复计算和资源浪费。
7. 灵活的内存管理策略  
   Spark将内存分为**堆上的存储内存**、**堆外的存储内存**、**堆上的执行内存**、**堆外的执行内存**4个部分。Spark既提供了**执行内存和存储内存之间固定边界的实现**，**又提供了执行内存和存储内存之间“软”边界的实现**。Spark默认使用“软”边界的实现，执行内存或存储内存中的任意一方在资源不足时都可以借用另一方的内存，最大限度地提高资源的利用率，减少对资源的浪费。Spark由于对内存使用的偏好，内存资源的多寡和使用率就显得尤为重要，为此Spark的内存管理器提供的Tungsten实现了一种与操作系统的内存Page非常相似的数据结构，用于直接操作操作系统内存，节省了创建的Java对象在堆中占用的内存，使得Spark对内存的使用效率更加接近硬件。Spark会给每个Task分配一个配套的任务内存管理器，对Task粒度的内存进行管理。Task的内存可以被多个内部的消费者消费，任务内存管理器对每个消费者进行Task内存的分配与管理，因此Spark对内存有着更细粒度的管理（大概说了一下）。  
   Tungsten Project通过使用更加紧凑和高效的基于列存储的数据处理方式以及支持多种高效的数据结构，对Spark进行了数据结构上的优化，从而提高了Spark的内存处理效率。
8. 检查点支持  
   Spark的RDD之间维护了血缘关系（lineage），一旦某个RDD失败了，则可以由父RDD重建。虽然lineage可用于错误后RDD的恢复，但对于很长的lineage来说，恢复过程非常耗时。如果应用启用了检查点，那么在Stage中的Task都执行成功后，SparkContext将把RDD计算的结果保存到检查点，这样当某个RDD执行失败后，再由父RDD重建时就不需要重新计算，而直接从检查点恢复数据。
9. 更多的内置函数  
   更多来自底层的优化。
10. 列裁剪（大概说一下，了解它的功能）  
    列裁剪（Column Pruning）是 Spark SQL 中一种常用的查询优化技术，它可以从查询中删除不必要的列，只获取必要的数据列，从而减少数据传输和处理的时间和资源，提高查询效率。列裁剪主要是通过过滤掉查询中不需要的列，使得扫描的数据量减少。如果数据存储格式为列式存储（如 Parquet 格式），列裁剪可以进一步映射下推，将列过滤操作下推到数据扫描器中执行，从而进一步减少IO操作。
11. 谓词下推（大概说一下，了解它的功能）  
    谓词下推是 Spark SQL 查询优化的一种技术，它可以将过滤条件尽可能地下推到最底层执行，以减少每一层交互的数据量，从而提升性能。具体来说，谓词下推可以在查询计划中自动将过滤条件下推到关系表的扫描器中，在扫描器中进行数据过滤，减少数据在网络中的传输量和磁盘IO操作，从而实现加速查询的效果。
12. Whole-Stage Code Generation（大概说一下，了解它的功能）  
    Spark中的全阶段代码生成是一种针对SQL和DataFrame等高级API的优化技术，它能够将一整个物理执行计划中的多个操作融合到一起，产生高效的本地机器代码执行（大概说说一下，细节还不懂）。

具体来说，全阶段代码生成通过以下三个步骤实现：

(1) 将物理执行计划中的多个操作合并为一个大的操作。

(2) 使用代码生成技术把这个大操作转换成本地机器代码，然后将代码编译成Java Class对象

(3) 在运行时直接调用生成的Java Class对象对数据进行处理，从而避免了反复地序列化/反序列 化数据, 避免了虚拟函数调 用带来的性能损失。

由于全阶段代码生成技术能够减少序列化和反序列化数据的次数，并且直接生成可执行的本地机器代码，因此它能够极大地提升Spark高级API的计算效率。目前，全阶段代码生成技术已经成为Spark中一个非常重要的优化技术。

13. Hadoop YARN中的ApplicationMaster申请到Container后，具体任务需要利用NodeManager从HDFS的不同节点下载任务所需的资源（如Jar包），这也增加了磁盘I/O。Spark将应用程序上传的资源文件缓冲到Driver本地文件服务的内存中，当Executor执行任务时直接从Driver的内存中读取，也节省了大量的磁盘I/O。

### Spark常用函数/算子?

### 什么是宽依赖和窄依赖？

在Spark中，宽依赖（Wide Dependency）和窄依赖（Narrow Dependency）是指RDD（Resilient Distributed Dataset）之间的依赖关系类型。

1. \*\*窄依赖\*\*：

- 窄依赖指的是每个父RDD分区都只被子RDD的一个分区所使用的依赖关系。换句话说，每个父RDD的分区仅仅会对应到子RDD中的一个分区。

- 窄依赖的好处是，在执行转换操作时，Spark可以更高效地执行任务，因为每个父分区只需要处理一个子分区，这使得Spark可以在同一台机器上进行本地化计算。

- 示例：map、filter等转换操作通常会生成窄依赖。

2. \*\*宽依赖\*\*：

- 宽依赖指的是每个父RDD的分区可能被子RDD的多个分区所使用的依赖关系。这意味着每个父分区可能需要被多个子分区使用，因此需要进行shuffle操作。

- 宽依赖的情况下，Spark需要将数据重新分区并传输到相应的子节点，这导致了更大的性能开销。

- 示例：groupByKey、reduceByKey等需要进行数据重组的操作通常会导致宽依赖。

总的来说，窄依赖通常比宽依赖更加高效，因为它们允许更多的本地化计算，并且不需要进行数据的全局重排。因此，在Spark程序中，尽量避免使用宽依赖，可以提高性能和执行效率。

# Flink

## 基础

### Flink常用函数？

# Kafka

## 基础

### Kafka常用函数？

# Hive

## 基础

### Hive创建表的语句？

### hive 内部表和外部表的区别？

### 生产环境中为什么建议使用外部表？

### Hive常用函数？

### Redis,传统数据库,HBase, Hive 每个之间的区别?

### Hive底层可以使用什么计算引擎？

### hive 有哪些保存元数据的方式，各有什么特点？

### 简述Hive中的虚拟列的作用是什么？使用它的注意事项

# Elasticsearch

# Zookeeper

# CDH-6.3.2

# HBase

## 基础

### HBase 的 row key 怎么创建好？列族怎么创建比较好？

### HBase数据库对行键的设计要求是什么？

### 简述HBase Filter的实现原理是什么？结合实际项目经验，写出几个使用Filter的案例

### 简述HBase性能优化思路

### ROWKEY的后缀匹配怎么实现？例如ROWKEY是yyyyMMDD-UserID形式，如以UserID为条件查询数据，怎样实现？

# Clickhouse

# Redis

## AOF是什么？

## RDB是什么？

## 主从同步？

# MySQL

## 基础

### ACID是什么（数据库四大特性）？

什么是ACID：

ACID，指数据库事务正确执行的四个基本要素的缩写。包含：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)。一个支持事务(Transaction)的数据库，必需要具有这四种特性，否则在事务过程(Transaction processing)当中无法保证数据的正确性，交易过程极可能达不到交易方的要求。

1. 原子性(Atomicity)

说的是一个事务内所有操作共同组成一个原子操作，要么全部成功，要么全部失败。这是最基本的特性，保证了因为一些其它因素或者宕机导致数据库异常

1. 一致性(Consistency)

这个是大家误解最深的，很多博客都喜欢用银行转账的例子来讲一直性，所谓的一致性是基于原子性。

原子性只保证了一个事务内的所有操作同一性，大家同生死，不会出现你死了，我还活着。但是，原子性并没有保证大家同一时刻一起生，一起死。计算机指令是有先后顺序的，这样就决定了一个事务的提交，会经历一个时间过程，那么如果事务提交进行到了一半，我读取了数据库，会不会读到中间结果?

为了防止这样的情况，数据库事物的一致性就规定了事务提交前后，永远只可能存在事务提交前的状态和事务提交后的状态，从一个一致性的状态到另一个一致性状态，而不可能出现中间的过程态。也就是说事务的执行结果是量子化状态，而不是线性状态。

1. 隔离性(Isolation)

事务的隔离性，基于原子性和一致性，因为事务是原子化，量子化的，所以，事务可以有多个原子包的形式并发执行，但是，每个事物互不干扰。

但是，由于多个事务可能操作同一个资源，不同的事务为了保证隔离性，会有很多锁方案，当然这是数据库的实现。

(4) 持久性(Durability)

持久性，当一个事务提交之后，数据库状态永远的发生了改变，这个事务只要提交了，哪怕提交后宕机，它也确确实实的提交了，不会出现因为刚刚宕机了而让提交不生效。

### Mysql存储引擎有哪些？

### MyISAM和Innodb的区别？

### 数据库事务的四个特性及含义？（ACID的含义？）

### 视图的作用，视图可以更改么？

### drop, delete 与 truncate 的区别？

### 索引的工作原理及其种类？

### 连接的种类？

# MongoDB

# Maven

# ELK

# Flume

# Sqoop

# Kerberos