并发编程网（http://ifeve.com/）

面视重要内容JAVA篇

1. JAVA反射机制
2. servelet生命周期
3. SpringAOP的原理
4. 动态代理的原理
5. CAS。那些类用到了
6. CAS防止ABA问题
7. sleep和wait区别
8. 用过或者了解什么设计模式
    8. 1. 原则：
        8. 1. 1. 开闭原则
        8. 1. 2. 里氏替换原则
        8. 1. 3. 依赖倒置原则
        8. 1. 4. 单一职责原则
        8. 1. 5. 接口隔离原则
        8. 1. 6. 迪米特法则
        8. 1. 7. 合用复用原则
    8. 2. 创建型
        8. 2. 1. 单例模式
        8. 2. 2. 工厂方法
        8. 2. 3. 抽象工厂
        8. 2. 4. 生成器/建造者
        8. 2. 5. 原型模式
    8. 3. 行为型
    8. 4. 结构型
9. 乐观锁，悲观锁
10. interface与 abstract class的区别
11. java线程池的实现原理，有哪些关键的构造参数
12. String能否被继承
13. hashMap
14. HashTable**
15. sql注入了解么？如何通过sql注入绕开登录密码限制 ，怎么防止**
16. doget和dopost的区别
17. Object类里面有那些基本方法
18. 内存泄漏
19. clone方法，浅拷贝深拷贝
20. notify和notifyAll
21. 线程间通信，进程间的通信
22. 堆栈里分别放的都是什么
23. 多线程编程的几个方式
24. LOCK和Synchronized区别
25. 一个java文件的执行流程以及classLoader的双亲委派机制
26. loadClass和forName的区别

面视重要内容计网篇

面试重要内容操作系统篇

面试重要内容数据库篇

面试重要内容算法篇

面视重要内容JAVA篇

1. JAVA反射机制

动态编译。运行时确定类型，绑定对象。

在实际开发中，我们需要把一个包中的class new出来，但是这个包中的类总是需要变动，那么怎么办，难道总是修改main方法中xxx=new xxx()吗。 这样无疑是麻烦的。而运用反射。我们可以相应的增加一个配置文件，在里面记录包中所有的类名，包中类增加时就加一个类名，删除时就删除一个类名。让main方法去读取这个配置文件中的类名，通过反射获得实例，完全不用我们去修改main方法中的代码。　　反射还有什么用那？他甚至可以修改其他类中的私有属性。 说了这么多，那么我们的开发中，为什么不全部都用反射那？一个原因，开销，它的开销是什么昂贵的，随意尽量在最需要的地方使用反射。

Spring IOC/DI就是运用这个机制

2. servelet生命周期

答： Servlet运行在Servlet容器中，其生命周期由容器来管理。Servlet的生命周期通过javax.servlet.Servlet接口中的init()、service()和destroy()方法来表示 1.加载和实例化 2.初始化 3.请求处理 4.服务终止

3. SpringAOP的原理

面向切面编程，Aspect Oriented Programming

通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术

在指定的地方（一般为某方法开始或结束点，或者异常抛出点）自动去调用某个方法进行处理，一般和IOC结合在一起，直接从IOC容器里面获取实例。

理解AOP： 在所有的连接点（Join point）里面，选出切点（pointcut），进行通知（advice），这一系列动作叫做切面（aspect） 比喻：在所有居民里面，选出有嫌疑的人，进行审问。

4. 动态代理的原理

动态代理利用Java的反射技术(Java Reflection)生成字节码，在运行时创建一个实现某些给定接口的新类（也称"动态代理类"）及其实例。 注意：这里代理的是接口，不是类和抽象类。

注意JDK的反射是基于接口的！也就是你的service一定是有接口的不然是不行的！这时候就有个Cglib可以顶上了！

Cglib采用了底层的字节码技术，为代理类创建了一个子类来代理它！

动态代理其实是一种方便运行时候动态的处理代理方法的调用机制。

通过动态代理可以给我们带来什么呢？通过代理可以让调用者和实现者之间解耦，例如RPC调用，对于我们调用者来说我就是想对用远程的那个方法，对于内部寻址啊，序列化反序列化等等这些交给代理来就行了，这样就能解放我们双手！

或者我们平日开发中需要监控一些方法的执行性能等，这时候就很时候用到代理了。

如我们需要监控每一个方法调用的开始时间和结束时间。

这样的话就把那些非业务逻辑的代码嵌入到我们的业务中，破坏了业务代码的纯粹性，所以我们希望把这个非业务的代码从业务代码中剥离出来，通过动态代理，我们可以将这些代码移除在动态运行的时候再织入它！

我们常见的动态代理有：JDK动态代理、Cglib(基于ASM)等。

5. CAS。那些类用到了

CAS:compare and swap

Lock类用到了

CAS 操作包含三个操作数 —— 内存位置（V）、预期原值（A）和新值(B)。如果内存地址里面的值和A的值是一样的，那么就将内存里面的值更新成B。CAS是通过无限循环来获取数据的，若果在第一轮循环中，a线程获取地址里面的值被b线程修改了，那么a线程需要自旋，到下次循环才有可能机会执行。

6. CAS防止ABA问题

①.CAS容易造成ABA问题。一个线程a将数值改成了b，接着又改成了a，此时CAS认为是没有变化，其实是已经变化过了，而这个问题的解决方案可以使用版本号标识，每操作一次version加1。在java5中，已经提供了AtomicStampedReference来解决问题。

②.CAS造成CPU利用率增加。之前说过了CAS里面是一个循环判断的过程，如果线程一直没有获取到状态，cpu资源会一直被占用。

③.只能保证一个共享变量的原子操作：

当对一个共享变量执行操作时，我们可以使用循环CAS的方式来保证原子操作，但是对多个共享变量操作时，循环CAS就无法保证操作的原子性，这个时候就可以用锁来保证原子性。

7. sleep和wait区别

1.sleep是Thread里面的方法，wait是Object里面的方法。 2.sleep不会去掉对象锁，wait会去掉对象锁，只有调用了notify方法后 才进入对象锁定池准备

8. 用过或者了解什么设计模式

一共23种设计模式，分为3大类

8.1. 原则：

8.1.1. 开闭原则

对拓展开放，对修改关闭

8.1.2. 里氏替换原则

子类可以拓展父类的功能，但不能改变父类原有的功能，也就是说在继承父类时，除了新增功能外，尽量不要重写父类的方法。

8.1.3. 依赖倒置原则

面向接口编程，不要面向实现编程。

常用的，List接口的实现LinkedList，使用就是List l=new LinkedList();

8.1.4. 单一职责原则

单一职责原则规定一个类应该有且仅有一个引起它变化的原因，否则类应该被拆分

8.1.5. 接口隔离原则

序员尽量将臃肿庞大的接口拆分成更小的和更具体的接口，让接口中只包含客户感兴趣的方法。

比如Collection接口拆分为3个子接口List，Set，queue

8.1.6. 迪米特法则

如果两个软件实体无须直接通信，那么就不应当发生直接的相互调用，可以通过第三方转发该调用。其目的是降低类之间的耦合度，提高模块的相对独立性。（需要权衡，避免过度使用该法则， 以防产生大量中介类）

8.1.7. 合用复用原则

要尽量先使用组合或者聚合等关联关系来实现，其次才考虑使用继承关系来实现。

8.2. 创建型

将对象的创建和使用分离，描述怎么样创建

8.2.1. 单例模式

确保类只有一个实例

（ 五种实现方式： 饿汉式 懒汉式 双重检测 静态内部类 枚举 ）

8.2.2. 工厂方法

定义一个创建产品对象的工厂接口，将产品对象的实际创建工作推迟到具体子工厂类当中。

如果要创建的产品不多，只要一个工厂类就可以完成，这种模式叫“简单工厂模式”，它不属于 GoF 的 23 种经典设计模式，它的缺点是增加新产品时会违背“开闭原则”。

8.2.3. 抽象工厂

因为工厂方法只生产一类产品，比如畜牧场只养动物、电视机厂只生产电视机、计算机软件学院只培养计算机软件专业的学生等。

同种类称为同等级，也就是说：工厂方法模式只考虑生产同等级的产品，但是在现实生活中许多工厂是综合型的工厂，能生产多等级（种类）的产品，如农场里既养动物又种植物，电器厂既生产电视机又生产洗衣机或空调，大学既有软件专业又有生物专业等。 本节要介绍的抽象工厂模式将考虑多等级产品的生产，将同一个具体工厂所生产的位于不同等级的一组产品称为一个产品族

抽象工厂（AbstractFactory）模式的定义：是一种为访问类提供一个创建一组相关或相互依赖对象的接口，且访问类无须指定所要产品的具体类就能得到同族的不同等级的产品的模式结构。

抽象工厂模式是工厂方法模式的升级版本，工厂方法模式只生产一个等级的产品，而抽象工厂模式可生产多个等级的产品。

使用抽象工厂模式一般要满足以下条件。

- 系统中有多个产品族，每个具体工厂创建同一族但属于不同等级结构的产品。
- 系统一次只可能消费其中某一族产品，即同族的产品一起使用。

抽象工厂模式除了具有工厂方法模式的优点外，其他主要优点如下。

- 可以在类的内部对产品族中相关联的多等级产品共同管理，而不必专门引入多个新的类来进行管理。
- 当增加一个新的产品族时不需要修改原代码，满足开闭原则。

其缺点是：当产品族中需要增加一个新的产品时，所有的工厂类都需要进行修改。

比如：用抽象工厂模式设计农场类。

分析：农场中除了像畜牧场一样可以养动物，还可以培养植物，如养马、养牛、种菜、种水果等，所以本实例比前面介绍的畜牧场类复杂，必须用抽象工厂模式来实现。

本例用抽象工厂模式来设计两个农场，一个是韶关农场用于养牛和种菜，一个是上饶农场用于养马和种水果，可以在以上两个农场中定义一个生成动物的方法 newAnimal() 和一个培养植物的方法 newPlant()。

抽象工厂定义： newAnimal() ，newPlant()。

具体工厂实现。

抽象产品类：AniMal，Plant

具体产品类：cattle，horse，fruit，vegetable。

所以

抽象工厂模式通常适用于以下场景：

1. 当需要创建的对象是一系列相互关联或相互依赖的产品族时，如电器工厂中的电视机、洗衣机、空调等。
2. 系统中有多个产品族，但每次只使用其中的某一族产品。如有人只喜欢穿某一个品牌的衣服和鞋。
3. 系统中提供了产品的类库，且所有产品的接口相同，客户端不依赖产品实例的创建细节和内部结构。

抽象工厂模式的扩展有一定的“开闭原则”倾斜性：

1. 当增加一个新的产品族时只需增加一个新的具体工厂，不需要修改原代码，满足开闭原则。
2. 当产品族中需要增加一个新种类的产品时，则所有的工厂类都需要进行修改，不满足开闭原则。

另一方面，当系统中只存在一个等级结构的产品时，抽象工厂模式将退化到工厂方法模式。

8.2.4. 生成器/建造者

8.2.5. 原型模式

用一个已经创建的实例作为原型，通过复制该原型对象来创建一个和原型相同或相似的新对象。在这里，原型实例指定了要创建的对象的种类。用这种方式创建对象非常高效，根本无须知道对象创建的细节。

原型模式的实现：

由于 Java 提供了对象的 clone() 方法，所以用 Java 实现原型模式很简单。

原型模式包含以下主要角色。

1. 抽象原型类：规定了具体原型对象必须实现的接口。
2. 具体原型类：实现抽象原型类的 clone() 方法，它是可被复制的对象。
3. 访问类：使用具体原型类中的 clone() 方法来复制新的对象。

原型模式的克隆分为浅克隆和深克隆，Java 中的 Object 类提供了浅克隆的 clone() 方法，具体原型类只要实现 Cloneable 接口就可实现对象的浅克隆，这里的 Cloneable 接口就是抽象原型类。

用原型模式除了可以生成相同的对象，还可以生成相似的对象

原型模式通常适用于以下场景。

- 对象之间相同或相似，即只是个别的几个属性不同的时候。
- 对象的创建过程比较麻烦，但复制比较简单的时候。

原型模式包含以下主要角色。

1. 抽象原型类：规定了具体原型对象必须实现的接口。
2. 具体原型类：实现抽象原型类的 clone() 方法，它是可被复制的对象。
3. 访问类：使用具体原型类中的 clone() 方法来复制新的对象。

8.3. 行为型

描述类/对象之间怎么协作共同完成单个对象无法完成的任务，以及分配职责

8.4. 结构型

描述怎么将类/对象按某种布局组成更大的结构

9. 乐观锁，悲观锁

乐观锁，默认数据不会被修改（实际上是允许修改的），当自己这个线程更新了数据并且准备存入的时候，会判断原数据是否被修改了。典型实现方法CAS

悲观锁：synchronized就是典型的，锁上不允许取和修改。

10. interface与 abstract class的区别

抽象类比接口更快，因为接口需要去类里面寻找对应实现的方法。

抽象类可以有main方法，可以运行。接口不行。

抽象方法可以有public，private，protected和default等修饰符。接口只有public和default（默认实现方法）

11. java线程池的实现原理，有哪些关键的构造参数

https://blog.csdn.net/programmer_at/article/details/79799267

12. String能否被继承

答： 不能

final修饰有3个作用， 1：修饰变量，表示该变量不能被更改，只能赋值一次。 2：修饰方法，表示方法不能被重写。 3：修饰类，表示类不能被继承。

与static不同。 static有4个作用 1：修饰变量，表示整个类里用同一个变量。 2：修饰方法，方法可以直接通过类名调用而不是对象调用。 3：修饰import。import static…，然后在类里面直接直接调用导入的包的方法，而无需通过XXX.XXX()。 4：修饰代码块，代码块只执行一次。

13. hashMap

1.jdk7采用的是链表加数组的形式，jdk8改动后采用的是散列表/红黑树的方式，当某个桶的长度超过8的时候，就把链表转换为红黑树。 **2**.key可以存储null值，会特殊处理放在数组头。 **3**.hashMap是非线程安全的，只是用于单线程环境下。多线程下可以采用concurrent并发包下的concurrentHashMap（使用的分段锁机制）。

14. HashTable**

hashtable其实也是key-value的形式，只不过是线程安全的，但是效率比较低下，因为是用的synchronized关键字对put等操作加锁，而synchronized加锁实际上是对整个对象都加锁，也就是说在进行put等修改table操作的时候，锁住了整个table，从而效率低下。

15. sql注入了解么？如何通过sql注入绕开登录密码限制 ，怎么防止**

就是通过输入特定的用户名或者密码进行注入。比如sql语句为 String sql = "select * from user_table where username= ' "+userName+" ' and password=' "+password+" '"; 那么输入的用户名和密码就可能组成一个sql语句，对数据直接进行操作。

使用PreparedStatement的好处是数据库会对sql语句进行预编译，下次执行相同的sql语句时，数据库端不会再进行预编译了，而直接用数据库的缓冲区，提高数据访问的效率。 SQL注入 攻 击 只 对 Statement有效， 对 PreparedStatement 是无效的；

PreparedStatement可以在传入sql后，执行语句前，给参数赋值，避免了因普通的拼接sql字符串语句所带来的安全问题，而且准备sql和执行sql是在两个语句里面完成的，也提高了语句执行的效率 比如单引号会给你加一个转义，加个斜杠。上面的sql语句在数据库里执行就是这样 select * from t_user where password='ddd' or '1'='1'; 它会把恶意的注入语句预处理为参数

16. doget和dopost的区别

答： Http定义了与服务器交互的不同方法，最基本的方法有4种，分别是GET，POST，PUT，DELETE。GET一般用于获取/查询资源信息，而POST一般用于更新资源信息。 还有OPTIONS,HEAD,TRACE,CONNECT。 但是本质上来说，没有区别。都是HTTP上写的方法，可以在Get里面写Post也可以反过来。 只不过是Get是在报文头里面传，所以参数写在URL里面，所以因为浏览器的长度限制，导致了get方法的长度有限制，而post写在报文体里面，所以没有长度限制，并且对于用户来说，相对安全一些，不过也只是视觉上安全些，如果通过嗅探抓包，那么一样会得到里面的内容。

17. Object类里面有那些基本方法

答：常用的toString，equals，hashcode，wait 还有getClass，返回此 Object 的运行时类。

18. 内存泄漏

内存泄漏（Memory Leak）是指程序中己动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统崩溃等严重后果。

JAVA已经有比较好的内存回收机制了，但任然有内存泄漏问题。

一：当长生命周期的对象持有短生命周期的对象的引用，就很可能发生内存泄漏。尽管短生命周期的对象已经不再需要，但是长生命周期的对象一直持有它的引用导致其无法被回收。例如，缓存系统；加载一个对象放在缓存系统中，一直不去使用这个对象，但是它一直被缓存引用，所以不会被回收导致缓存泄漏。

二：当一个对象被存储进HashSet集合中，就不可修改这个对象中用于计算哈希值的属性了。否则，对象修改后的哈希值与刚添加进HashSet集合时的哈希值不一样，此时如果将当前对象的引用作为参数，用contains方法判断对象是否存在，则会返回找不到对象的结果。这会导致无法从HashSet单独删除当前对象，造成内存泄漏

19. clone方法，浅拷贝深拷贝

浅拷贝：同一个对象 深拷贝：不同对象

使用clone方法（浅拷贝）拷贝对象既复杂又危险，会抛出异常，还需要类型转换，Effective JAVA书中讲过，最好不要用clone，可以使用一个拷贝构造函数或者拷贝工厂来拷贝一个对象。

20. notify和notifyAll

答: notify和notifyAll之间的主要区别在于notify方法只通知一个Thread，notifyAll方法将通知在该监视器上等待的所有线程或锁定。

并且notify会唤醒一个等待的线程，但是不知道唤醒哪一个，这取决于线程调度器，而notifyAll可能造成重置等待，所以使用notifyAll往往只是为了确认所有收件人得到通知（最好少用），容易造成CPU浪费

21. 线程间通信，进程间的通信

22. 堆栈里分别放的都是什么

栈：存放函数的参数值，局部变量的值等，其操作方式类似于数据结构中的栈。

堆：一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS（操作系统）回收，分配方式倒是类似于链表。 栈使用的是一级缓存 堆则是存放在二级缓存中，生命周期由虚拟机的垃圾回收算法来决定（并不是一旦成为垃圾就能被回收）。所以调用这些对象的速度要相对来得低一些。

栈有一个很重要的特殊性，就是存在栈中的数据可以共享。

23. 多线程编程的几个方式

1.继承Thread类。但实际上，Thread类也是实现了Runnable接口。 2.实现Runnable接口 3.实现Callable接口

24. LOCK和Synchronized区别

1.LOCK是一个接口，有实现这个接口的类 Synchronized是实现在JVM的关键字 2.LOCK是乐观锁机制，底层是靠volatile和CAS操作实现的 3.尽量使用Synchronized而不要用Lock，因为JDK1，6之后对Synchronized进行了优化，性能比较好。（例如 1.线程自旋和适应性自旋 2.编译阶段的锁消除 3.锁粗化 4.轻量级锁 5.偏向锁）

25. 一个java文件的执行流程以及classLoader的双亲委派机制

1.编写java程序 2.编译成.class程序（二进制流） 3.通过classLoader加载成为class name对象 4.实例化这个对象

26. loadClass和forName的区别

1.一个类的加载过程，从java变成class文件 2.链接 校验，class的正确性和安全性，准备，为类分配存储空间并设置类的初始值，解析，JVM将常量池里面的符号引用转为直接引用 3.初始化，类变量复制和静态代码块

loadClass在链接之间，不会执行初始化 用于Spring IOC，用来延迟初始化，大量使用提高速度

forName会执行初始化

27. 元空间和永久代

MetaSpace 和 PermGen 字符串常量池在JDK8中被从方法区移到了堆中

方法区也是所有线程共享。主要用于存储类的信息、常量池、方法数据、方法代码等。 方法区是JVM 的规范，永久代（PermGen space）是HotSpot对这种规范的实现。

元空间和永久代类似，都是方法区的实现，不过最大的区别是元空间在本地内存，而永久代在JVM虚拟内存，所以元空间大小只收本地内存大小限制。

28. String里面的intern

比如String str=new String("a"); 创建了2个对象，一个在常量区里面的"a" 一个在java heap里面的str，对a的引用而用str.intern的时候，发现常量区里面已经有了“a”,所以就不会把str放到常量区里面 这个时候再String str1="a"; str1 == str就是false

29. 垃圾回收算法，堆内存的分配，什么时候执行ygc

标记算法算法： 1.引用计数算法 优点：效率高，程序受影响较小。 缺点：无法检测循环引用，导致内存泄漏。 所以非主流 **2.可达性分析算法：** 通过判断对象的引用链是否可达，来决定对象是否回收。（用图论，GC Root根对象，将访问到的所有对象标记为存活，访问不到的就是垃圾对象） 什么对象可以做为GC ROOT 1.虚拟机栈中引用的变量（栈帧中的本地变量表） 2.方法区中常量引用的对象 3.方法区中类静态属性引用对象 4.本地方法栈中Native方法的引用对象 5.活跃线程的引用对象

标记清除算法 标记：从根集合进行扫描，对存活的对象进行标记 回收：对堆内存从头到尾进行线性遍历，回收不可达对象内存。 缺点：造成碎片化

复制算法（年轻代）将可用内存分为两块或者多块，选择一块或者两块作为**对象面**，其他的作为**空闲面**。 对象在**对象面**上创建，当对象面用完的时候，把还存活着的对象复制到**空闲面**，再把对象面所有对象清楚。

1.解决碎片化 2.顺序分配内存 3.使用于对象存活率低的情况（比如：年轻代。老年代一般不能用这个算法）

标记整理算法 标记：从根集合进行扫描，对存活的对象进行标记 清楚：移动所有存活对象，且按照内存地址次序依次排列，然后将末端内存地址以后的内存全部回收。

1.成本高 2.解决碎片化问题 3.避免内存不连续 4.不需要内存互换 5.使用于对象存活率高的情况

分代收集算法（常用）按照对象生命周期的不同，划分不同的区域，采用不同的垃圾收集算法。

分为：新生代，老年代，永久代

新生代，老年代都在java heap

永久代在方法区。

对于新生代，采用复制算法

对于老年代，采用标记清除，或标记整理。

永久代，也采用标记清楚或者标记整理。

30. 线程池

30.1. 常用线程池

顶级抽象类：Exectutor

1.newFixedThreadPool 定长线程池，控制线程最大并发数，超出的线程在队列中等待 2.newCachedThreadPool 可缓存线程池，如果线程池长度超过处理需要，可以灵活回收空闲线程，若无可回收，则新建线程。 3.newSingleThreadExecutor 单线程线程池 4.newSchedulThreadExecutor 定长线程池，支持定时及周期性任务执行

30.2. 线程池中的几种重要的参数

corePoolSize就是线程池中的核心线程数量，这几个核心线程，只是在没有用的时候，也不会被回收

maximumPoolSize就是线程池中可以容纳的最大线程的数量

keepAliveTime，就是线程池中除了核心线程之外的其他的最长可以保留的时间，因为在线程池中，除了核心线程即使在无任务的情况下也不能被清 除，其余的都是有存活时间的，意思就是非核心线程可以保留的最长的空闲时间，

util，就是计算这个时间的一个单位。

workQueue，就是等待队列，任务可以储存在任务队列中等待被执行，执行的是FIFIO原则（先进先出）。

threadFactory，就是创建线程的线程工厂。

handler,是一种拒绝策略，我们可以在任务满了之后，拒绝执行某些任务。

30.3. 说说线程池的拒绝策略

当请求任务不断的过来，而系统此时又处理不过来的时候，我们需要采取的策略是拒绝服务。RejectedExecutionHandler接口提供了拒绝任务处理的自定义方法的机会。在ThreadPoolExecutor中已经包含四种处理策略。

AbortPolicy策略：该策略会直接抛出异常，阻止系统正常工作。

CallerRunsPolicy 策略：只要线程池未关闭，该策略直接在调用者线程中，运行当前的被丢弃的任务。

DiscardOleddestPolicy策略： 该策略将丢弃最老的一个请求，也就是即将被执行的任务，并尝试再次提交当前任务。

DiscardPolicy策略：该策略默默的丢弃无法处理的任务，不予任何处理。

除了JDK默认提供的四种拒绝策略，我们可以根据自己的业务需求去自定义拒绝策略，自定义的方式很简单，直接实现RejectedExecutionHandler接口即可。

30.4. execute和submit的区别？

在前面的讲解中，我们执行任务是用的execute方法，除了execute方法，还有一个submit方法也可以执行我们提交的任务。

这两个方法有什么区别呢？分别适用于在什么场景下呢？我们来做一个简单的分析。

execute适用于不需要关注返回值的场景，只需要将线程丢到线程池中去执行就可以了。

submit方法适用于需要关注返回值的场景

30.5. 五种线程池的使用场景

newSingleThreadExecutor：一个单线程的线程池，可以用于需要保证顺序执行的场景，并且只有一个线程在执行。

newFixedThreadPool：一个固定大小的线程池，可以用于已知并发压力的情况下，对线程数做限制。

newCachedThreadPool：一个可以无限扩大的线程池，比较适合处理执行时间比较小的任务。

newScheduledThreadPool：可以延时启动，定时启动的线程池，适用于需要多个后台线程执行周期任务的场景。

newWorkStealingPool：一个拥有多个任务队列的线程池，可以减少连接数，创建当前可用cpu数量的线程来并行执行。

30.6. 线程池的关闭

关闭线程池可以调用shutdownNow和shutdown两个方法来实现

shutdownNow：对正在执行的任务全部发出interrupt()，停止执行，对还未开始执行的任务全部取消，并且返回还没开始的任务列表。

shutdown：当我们调用shutdown后，线程池将不再接受新的任务，但也不会去强制终止已经提交或者正在执行中的任务。

30.7. 初始化线程池时线程数的选择

如果任务是IO密集型，一般线程数需要设置2倍CPU数以上，以此来尽量利用CPU资源。

如果任务是CPU密集型，一般线程数量只需要设置CPU数加1即可，更多的线程数也只能增加上下文切换，不能增加CPU利用率。

上述只是一个基本思想，如果真的需要精确的控制，还是需要上线以后观察线程池中线程数量跟队列的情况来定。

30.8. 线程池都有哪几种工作队列

1、ArrayBlockingQueue

是一个基于数组结构的有界阻塞队列，此队列按 FIFO（先进先出）原则对元素进行排序。

2、LinkedBlockingQueue

一个基于链表结构的阻塞队列，此队列按FIFO （先进先出） 排序元素，吞吐量通常要高于ArrayBlockingQueue。静态工厂方法Executors.newFixedThreadPool()使用了这个队列

3、SynchronousQueue

一个不存储元素的阻塞队列。每个插入操作必须等到另一个线程调用移除操作，否则插入操作一直处于阻塞状态，吞吐量通常要高于LinkedBlockingQueue，静态工厂方法Executors.newCachedThreadPool使用了这个队列。

4、PriorityBlockingQueue

一个具有优先级的无限阻塞队列。

31. 异常体系

顶层父类：Throwable---->Object 两个子类：Error，Exception 而Exception的两个子类：RuntimeException（比如数组下标越界，空指针）和非RuntimeException（比如IO Exception）

Error错误（程序无法处理的系统错误，比如栈溢出，内存溢出之类的JVM问题）

Exception是程序可以处理的异常，需要尽可能的去处理。

应该设计一个通用的继承自RuntimeException的异常来处理，对用户友好。

try catch块会影响JVM性能优化 每遇到一个异常，就会实例化一个异常并且保存堆栈的快照，开销较大。

所以效率比if else和switch低很多

32. 序列化和反序列化

序列化： 就是将内存中的对象转换为字节序列，方便持久化到磁盘或者网络传输。

反序列化 就是将字节序列转换为内存中的对象

33. 常用数据结构

主要是Collection和Map接口

vector 线性表（arrayList和LinkedList）， 栈， 队列， 哈希表

34. Java集合

List Set Map

其中List和Set继承自Collection接口，Map为独立接口

Set下有：HashSet，LinkedHashSet，TreeSet

List：ArrayList，LinkedList

Map：HashTable，HashMap，LinkedHashMap，TreeMap

Collection接口下还有一个Queue接口，有PriorityQueue类

面视重要内容计网篇

1. session和cookie

首先因为http协议是无状态的，也就是每一次数据交换完毕后连接就关闭了，需要重新建立连接。

怎么样记录去跟踪会话呢，就用session或cookie

答： Cookie以文本文件格式存储在浏览器中，4KB，而session存储在服务端. 我们可以轻松访问cookie值但是我们无法轻松访问seesion，因此它更安全。 设置cookie时间可以使cookie过期。但是使用session-destory（ ），我们将会销毁会话。

2. TCP三次握手，四次挥手

答： 1．为什么建立连接协议是三次握手，而关闭连接却是四次握手呢？ 这是因为服务端的LISTEN状态下的SOCKET当收到SYN报文的连接请求后，它可以把ACK和SYN(ACK起应答作用，而SYN起同步作用)放在一个报文里来发送。但关闭连接时，当收到对方的FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可能未必会马上会关闭SOCKET,也即你可能还需要发送一些数据给对方之后，再发送FIN报文给对方来表示你同意现在可以关闭连接了，所以它这里的ACK报文和FIN报文多数情况下都是分开发送的。

为什么不能用两次握手进行连接？

答：3次握手完成两个重要的功能，既要双方做好发送数据的准备工作(双方都知道彼此已准备好)，也要允许双方就初始序列号进行协商，这个序列号在握手过程中被发送和确认。

现在把三次握手改成仅需要两次握手，死锁是可能发生的。作为例子，考虑计算机S和C之间的通信，假定C给S发送一个连接请求分组，S收到了这个分组，并发 送了确认应答分组。按照两次握手的协定，S认为连接已经成功地建立了，可以开始发送数据分组。可是，C在S的应答分组在传输中被丢失的情况下，将不知道S 是否已准备好，不知道S建立什么样的序列号，C甚至怀疑S是否收到自己的连接请求分组。在这种情况下，C认为连接还未建立成功，将忽略S发来的任何数据分 组，只等待连接确认应答分组。而S在发出的分组超时后，重复发送同样的分组。这样就形成了死锁。

为什么连接的时候是三次握手，关闭的时候却是四次挥手？

答：因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步挥手。

3. ipv4和ipv6

1.ipv4只有32位，ipv6有128位 2.ipv6支持更安全的加密 3.ipv6支持多种协议

4. http和https的区别，以及底层

https就是http加上ssl。 http是明文传输，https是经过ssl加密后的密文传输

端口：

http：80 https:443 **两个是完全不同的连接方式 http:** 一次HTTP操作一个事务 1、Client与Server建立连接，单击某个超链接，HTTP的工作开始。 2、连接建立后，Client发送一个请求给Server，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是MIME信息包括请求修饰符，Client信息和可能的内容。 3、Server接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是MIME信息包括Server信息、实体信息和可能的内容。 4、Client接收Server返回的信息通过浏览器显示在用户的显示屏上，然后Client和Server断开连接。
https:

1、Client使用HTTPS的URL访问Web服务器，要求与Web服务器建立SSL连接。 2、Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。 3、客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。 4、客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。 5、Web服务器利用自己的私钥解密出会话密钥。 6、Web服务器利用会话密钥加密与客户端之间的通信。

面试重要内容操作系统篇

1. 进程间通信**

答： 进程间通信（IPC，InterProcess Communication）是指在不同进程之间传播或交换信息。 IPC的方式通常有管道（包括无名管道和命名管道）、消息队列、信号量、共享存储、Socket、Streams等。其中 Socket和Streams支持不同主机上的两个进程IPC。

2. 进程与线程

1.一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。线程是操作系统可识别的最小执行和调度单位

2.资源分配给进程，同一进程的所有线程共享该进程的所有资源。 同一进程中的多个线程共享代码段(代码和常量)，数据段(全局变量和静态变量)，扩展段(堆存储)。但是每个线程拥有自己的栈段，栈段又叫运行时段，用来存放所有局部变量和临时变量。

3.处理机分给线程，即真正在处理机上运行的是线程。 cpu

4.线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

区别

（1）进程有自己的独立地址空间，线程没有

（2）进程是资源分配的最小单位，线程是CPU调度的最小单位

（3）进程和线程通信方式不同(线程之间的通信比较方便。同一进程下的线程共享数据（比如全局变量，静态变量），通过这些数据来通信不仅快捷而且方便，当然如何处理好这些访问的同步与互斥正是编写多线程程序的难点。而进程之间的通信只能通过[进程通信]的方式进行。

（4）进程上下文切换开销大，线程开销小

（5）一个进程挂掉了不会影响其他进程，而线程挂掉了会影响其他线程

（6）对进程进程操作一般开销都比较大，对线程开销就小了

3. 调度算法

1.FCFS先来先服务 first come first served

2.SJF 最短作业最先调度 Shortest job first

3.SRJF shortest remaining job first SJF的可抢占版本，比SJF更有优势

4.优先权调度 调度优先权最高的

5.Round-robin（RR）轮转调度算法

6.多级队列调度，按一定规则建立多个进程队列，不同队列用不同的算法

7.多级反馈队列，在多级队列的基础上，任务可以在队列间移动，最通用的调度算法

4. 什么是线程安全

如果多线程的程序运行结果是可预期的，而且与单线程的程序运行结果一样，那么说明是“线程安全”的。

5. 线程同步

5.1. 什么是线程同步，互斥

同步就是协同步调，按预定的先后次序进行运行。如：你说完，我再说。这里的同步千万不要理解成那个同时进行，应是指协同、协助、互相配合。线程同步是指多线程通过特定的设置（如互斥量，事件对象，临界区）来控制线程之间的执行顺序（即所谓的同步）也可以说是在线程之间通过同步建立起执行顺序的关系，如果没有同步，那线程之间是各自运行各自的！

线程互斥是指对于共享的进程系统资源，在各单个线程访问时的排它性。当有若干个线程都要使用某一共享资源时，任何时刻最多只允许一个线程去使用，其它要使用该资源的线程必须等待，直到占用资源者释放该资源。线程互斥可以看成是一种特殊的线程同步（下文统称为同步）。

5.2. JAVA实现线程同步

5.2.1. 使用synchronized关键字

5.2.2. wait和notify

5.2.3. 使用特殊域变量volatile实现线程同步

a.volatile关键字为域变量的访问提供了一种免锁机制　b.使用volatile修饰域相当于告诉虚拟机该域可能会被其他线程更新　c.因此每次使用该域就要重新计算，而不是使用寄存器中的值　d.volatile不会提供任何原子操作，它也不能用来修饰final类型的变量

5.2.4. 使用重入锁实现线程同步

在JavaSE5.0中新增了一个java.util.concurrent包来支持同步。

ReentrantLock类是可重入、互斥、实现了Lock接口的锁，它与使用synchronized方法和快具有相同的基本行为和语义，并且扩展了其能力。

注：关于Lock对象和synchronized关键字的选择：　a.最好两个都不用，使用一种java.util.concurrent包提供的机制，能够帮助用户处理所有与锁相关的代码。　b.如果synchronized关键字能满足用户的需求，就用synchronized，因为它能简化代码　c.如果需要更高级的功能，就用ReentrantLock类，此时要注意及时释放锁，否则会出现死锁，通常在finally代码释放锁

5.2.5. 使用局部变量来实现线程同步

如果使用ThreadLocal管理变量，则每一个使用该变量的线程都获得该变量的副本，副本之间相互独立，这样每一个线程都可以随意修改自己的变量副本，而不会对其他线程产生影响。

5.2.6. 使用阻塞队列实现线程同步

前面5种同步方式都是在底层实现的线程同步，但是我们在实际开发当中，应当尽量远离底层结构。 使用javaSE5.0版本中新增的java.util.concurrent包将有助于简化开发。 本小节主要是使用LinkedBlockingQueue来实现线程的同步 LinkedBlockingQueue是一个基于已连接节点的，范围任意的blocking queue。

5.2.7. 使用原子变量实现线程同步

需要使用线程同步的根本原因在于对普通变量的操作不是原子的。

那么什么是原子操作呢？原子操作就是指将读取变量值、修改变量值、保存变量值看成一个整体来操作即-这几种行为要么同时完成，要么都不完成。在java的util.concurrent.atomic包中提供了创建了原子类型变量的工具类，使用该类可以简化线程同步。其中AtomicInteger 表可以用原子方式更新int的值，可用在应用程序中(如以原子方式增加的计数器)，但不能用于替换Integer；可扩展Number，允许那些处理机遇数字类的工具和实用工具进行统一访问。

6. BIO和NIO（IO）

Unix五种IO模型 ：

[1] blocking IO - 阻塞IO [2] nonblocking IO - 非阻塞IO [3] IO multiplexing - IO多路复用 [4] signal driven IO - 信号驱动IO [5] asynchronous IO - 异步IO

前面4种都是同步IO，最后的AIO是异步IO

都是针对网络IO（也就是程序可以预见的阻塞问题），而不是本地IO。本地默认IO不会阻塞。）

BIO：建立了连接就处理，其他的一直在等待，最多开多个线程一起处理。最多只能建立线程个连接。 也就是一个连接对应一个线程。

NIO：程序注册一组Socket文件描述符给操作系统，表示监听这些是否发生IO，（NIO和IO多路复用一般结合在一起用）。 也就是一个有效请求对应一个线程。

AIO:因为NIO在进行IO操作的时候是同步的，而AIO是等IO操作完成后再给线程发出通知，因此AIO是不会阻塞的

6.1. IO多路复用的三种机制

6.1.1. Select

6.1.2. Poll

6.1.3. Epoll

7. 同步和异步

同步是指：当程序1调用程序2时，程序1停下不动，直到程序2完成回到程序1来，程序1才继续执行下去。

异步是指：当程序1调用程序2时，程序1径自继续自己的下一个动作，不受程序2的的影响。

同步是指：发送方发出数据后，等接收方发回响应以后才发下一个数据包的通讯方式。

异步是指：发送方发出数据后，不等接收方发回响应，接着发送下个数据包的通讯方式。

面试重要内容数据库篇

1. 三大范式

1.关系模式R的所有属性都不能在分解为更基本的数据单位时 ，满足第一范式，记为1NF

（数据原子性）

2.如果关系模式R满足第一范式，并且R得所有非主属性都完全依赖于R的每一个候选关键属性，称R满足第二范式，简记为2NF 。

需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。

3.设R是一个满足第一范式条件的关系模式，X是R的任意属性集，如果X非传递依赖于R的任意一个候选关键字，称R满足第三范式，简记为3NF

也就是每个属性都和主键有直接关系而不是间接关系（不然就要拆开为两个表）

简单来说：

第一范式，保证属性为基本属性。

第二范式，让每一列都和主键关联。（而不是只和主键中某一部分关联）

第三范式：每个属性和主键直接关联，如果有间接关联就拆分为多个表。

2. ACID（事务管理）

A： atomicity 原子性——数据库事务是不可分割的工作单位。只有使据库中所有的操作执行成功，才算整个事务成功

C：consistency 一致性——指数据库事务不能破坏关系数据的完整性以及业务逻辑上的一致性。例如对银行转帐事务，不管事务成功还是失败，应该保证事务结束后ACCOUNTS表中Tom和Jack的存款总额为2000元。

I：isolation 隔离性 ——指的是在并发环境中，当不同的事务同时操纵相同的数据时，每个事务都有各自的完整数据空间。

D：durability 持久性 ——指的是只要事务成功结束，它对数据库所做的更新就必须永久保存下来。即使发生系统崩溃，重新启动数据库系统后，数据库还能恢复到事务成功结束时的状态。

3. 并发事务带来的问题

并发事务带来的几个问题：更新丢失，脏读，不可重复读，幻读。

事务隔离级别：未提交读(Read uncommitted)，已提交读(Read committed)，可重复读(Repeatable read)，可序列化(Serializable)

四种隔离级别的比较

读数据一致性及并发副作用 隔离级别	读数据一致性	脏读	不可重复读	幻读
为提交读(read uncommitted)	最低级别，不读物理上顺坏的数据	是	是	是
已提交读(read committed)	语句级	否	是	是
可重复读(Repeatable red)	事务级	否	否	是
可序列化(Serializable)	最高级别，事务级	否	否	否

4. 什么叫视图？游标是什么

> **视图：**

> 是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改会影响基本表。它使得我们获取数据更容易，相比多表查询。
>
> **游标：**
>
> 是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。
>
> Oracle中的游标分为***静态游标和REF游标***。其中，静态游标就像一个数据快照，打开游标后的结果集是对数据库数据的一个备份，数据不随着对表执行DML操作而改变。从这个特性来说，结果集是静态的。
>
> 静态游标包含如下两种类型：
>
> **显式游标**：是指在使用前必须有着明确的游标声明和定义，这样的游标定义会关联数据查询语句，通常会返回一行或多行。打开游标后，用户可以利用游标的位置对结果集进行检索，使之返回单一的行记录，用户可以操作次记录。关闭游标后，就不能再对结果集进行任何操作。显式游标需要用户自己写代码完成，一切由用户控制。
>
> **隐式游标**：和显式游标不同，它被PL/SQL自动管理，也被称为PL/SQL游标。由Oracle自动管理。该游标用户无法控制，但能得到它的属性信息。

5. char和varchar

varchar长度可变，char不可变。

定义一个char[10]和varchar[10],如果存进去的是'abcd',那么char所占的长度依然为10，除了字符'abcd'外，后面跟六个空格，而varchar就立马把长度变为4了，取数据的时候，char类型的要用trim()去掉多余的空格，而varchar是不需要的

1. char的存取数度还是要比varchar要快得多，因为其长度固定，方便程序的存储与查找；但是char也为此付出的是空间的代价，因为其长度固定，所以难免会有多余的空格占位符占据空间，可谓是以空间换取时间效率，而varchar是以空间效率为首位的。
2. char的存储方式是，对英文字符（ASCII）占用1个字节，对一个汉字占用两个字节；而varchar的存储方式是，对每个英文字符占用2个字节，汉字也占用2个字节，两者的存储数据都非unicode的字符数据。

所以，经常变化的字段用varchar，知道固定长度用char，超过255字节的只能用varchar或者text，能哟个varchar就不要用text

6. InnoDB和MyiSam

1、MyISAM：默认表类型，它是基于传统的ISAM类型，ISAM是Indexed Sequential Access Method (有索引的顺序访问方法) 的缩写，它是存储记录和文件的标准方法。不是事务安全的，而且不支持外键，如果执行大量的select，insert MyISAM比较适合。

2、InnoDB：支持事务安全的引擎，支持外键、行锁、事务是他的最大特点。如果有大量的update和insert，建议使用InnoDB，特别是针对多个并发和QPS较高的情况。

6.1. InnoDB

InnoDB是行锁，并且支持事务安全

如果定义了主键，那么InnoDB会选择主键作为聚集索引

如果没有显示的定义主键，那么InnoDB会厕第一个不包含null值的唯一索引作为主键索引。

如果也没有这样的索引，那么InnoDB会选择n内置6字节长的ROWID作为隐含的聚集索引，这个ROWID不像ORACLE的ROWID那样可引用，是隐含的。

6.2. MyiSam

Myisam只支持表锁，消耗巨大。

7. 数据库调优

通过slow 日志来看对应的日志来调优

8. 索引类型

主键索引：数据记录里面不能有 null,数据内容不能重复，在一张表里面不能有 多个主键索引。

普通索引：使用字段关键字建立的索引，主要是提高查询速度

唯一索引：字段数据是唯一的，数据内容里面能否为 null,在一张表里面，是可 以添加多个唯一索引。

全文索引：在比较老的版本中，只有 myisam 引擎支持全文索引，在 innodb5.6 后引擎也支持全文索引，在 mysql 中全文索引不支持中文

又可以划分为：

聚簇索引，非聚簇索引 ————————也称为 二级索引

- 聚簇索引：将数据存储与索引放到了一块，找到索引也就找到了数据
- 非聚簇索引：将数据存储于索引分开结构，索引结构的叶子节点指向了数据的对应行，myisam通过key_buffer把索引先缓存到内存中，当需要访问数据时（通过索引访问数据），在内存中直接搜索索引，然后通过索引找到磁盘相应数据，这也就是为什么索引不在key buffer命中时，速度慢的原因

澄清一个概念：innodb中，在聚簇索引之上创建的索引称之为辅助索引，辅助索引访问数据总是需要二次查找，非聚簇索引都是辅助索引，像复合索引、前缀索引、唯一索引，辅助索引叶子节点存储的不再是行的物理位置，而是主键值

聚簇索引具有唯一性

由于聚簇索引是将数据跟索引结构放到一块，因此一个表仅有一个聚簇索引

一个误区：把主键自动设为聚簇索引

聚簇索引默认是主键，如果表中没有定义主键，InnoDB 会选择一个**唯一的非空索引**代替。如果没有这样的索引，InnoDB 会**隐式定义一个主键**来作为聚簇索引。InnoDB 只聚集在同一个页面中的记录。包含相邻健值的页面可能相距甚远。**如果你已经设置了主键为聚簇索引，必须先删除主键，然后添加我们想要的聚簇索引，最后恢复设置主键即可。**

此时其他索引只能被定义为非聚簇索引。这个是最大的误区。有的主键还是无意义的自动增量字段，那样的话Clustered index对效率的帮助，完全被浪费了。

刚才说到了，聚簇索引性能最好而且具有唯一性，所以非常珍贵，必须慎重设置。**一般要根据这个表最常用的SQL查询方式来进行选择，某个字段作为聚簇索引，或组合聚簇索引**，这个要看实际情况。

9. 数据库索引的优化，什么类型的字段可以建立索引

1.唯一性

2.占用存储空间少的字段更适合选作索引的关键字。例如，与字符串相比，整数字段占用的存储空间较少

3.存储空间固定的字段更适合选作索引的关键字。与 text 类型的字段相比， char 类型的字段较为适合选作索引关键字

4.Where 子句中经常使用的字段应该创建索引，分组字段或者排序字段应该创建索引，两个表的连接字段应该创建索引。

5.更新频繁的字段不适合创建索引，不会出现在 where 子句中的字段不应该创建索引。

6.最左前缀原则。

7.尽量使用前缀索引。

10. 什么情况下索引会失效

使用 or的前后没有都使用索引

用like模糊查询%

11. 存储过程

存储过程是一个预编译的SQL语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次SQL，使用存储过程比单纯SQL语句执行要快。

11.0.1. 存储过程的优缺点？

> **优点：**
>
> 1）存储过程是预编译过的，执行效率高。
>
> 2）存储过程的代码直接存放于数据库中，通过存储过程名直接调用，减少网络通讯。
>
> 3）安全性高，执行存储过程需要有一定权限的用户。
>
> 4）存储过程可以重复使用，可减少数据库开发人员的工作量。

> **缺点：**移植性差

11.1. SQL中存储过程和函数的区别

本质上没区别。只是函数有如：只能返回一个变量的限制。而存储过程可以返回多个。

而函数是可以嵌入在sql中使用的,可以在select中调用，而存储过程不行。

执行的本质都一样。 函数限制比较多，比如不能用临时表，只能用表变量．

还有一些函数都不可用等等．而存储过程的限制相对就比较少

1. 一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。
2. 对于存储过程来说可以返回参数，而函数只能返回值或者表对象。
3. 存储过程一般是作为一个独立的部分来执行（EXEC执行），

   而函数可以作为查询语句的一个部分来调用（SELECT调用），由于函数可以返回一个表对象，

   因此它可以在查询语句中位于FROM关键字的后面。

```
4.  当存储过程和函数被执行的时候，SQL Manager会到procedure cache中去取相应的查询语句，如果在procedure cache里没有相应的查询语句，SQL Manager就会对存储过程和函数进行编译。
```

12. Redis

可基于内存亦可持久化的日志型、Key-Value 数据库， 非关系型数据库。

12.1. Redis支持的数据类型？

String字符串：

格式: set key value

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

string类型是Redis最基本的数据类型，一个键最大能存储512MB。

Hash

格式: hmset name key1 value1 key2 value2

Redis hash 是一个键值(key=>value)对集合。

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

List

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

Set

Redis的Set是string类型的无序集合。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

zset(sorted set：有序集合)

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

zset的成员是唯一的,但分数(score)却可以重复。

12.2. 什么是Redis持久化？Redis有哪几种持久化方式？优缺点是什么？

持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失。

Redis 提供了两种持久化方式:RDB（默认） 和AOF

RDB：

rdb是Redis DataBase缩写

功能核心函数rdbSave(生成RDB文件)和rdbLoad（从文件加载内存）两个函数

AOF:

Aof是Append-only file缩写

每当执行服务器(定时)任务或者函数时flushAppendOnlyFile 函数都会被调用， 这个函数执行以下两个工作

aof写入保存：

WRITE：根据条件，将 aof_buf 中的缓存写入到 AOF 文件

SAVE：根据条件，调用 fsync 或 fdatasync 函数，将 AOF 文件保存到磁盘中。

12.3. 存储结构:

内容是redis通讯协议(RESP)格式的命令文本存储。

比较：

1、aof文件比rdb更新频率高，优先使用aof还原数据。

2、aof比rdb更安全也更大

3、rdb性能比aof好

4、如果两个都配了优先加载AOF

解释下什么是RESP？有什么特点？

Redis Serialization Protocol (Redis序列化协议)

RESP 是redis客户端和服务端之前使用的一种通讯协议；

协议如下：

客户端以规定格式的形式发送命令给服务器；

服务器在执行最后一条命令后，返回结果。

RESP 的特点：实现简单、快速解析、可读性好

12.4. Redis架构模式和特点

1.单机版

问题：

1、内存容量有限 2、处理能力有限 3、无法高可用。

2.主从复制

Redis 的复制（replication）功能允许用户根据一个 Redis 服务器来创建任意多个该服务器的复制品，其中被复制的服务器为主服务器（master），而通过复制创建出来的服务器复制品则为从服务器（slave）。 只要主从服务器之间的网络连接正常，主从服务器两者会具有相同的数据，主服务器就会一直将发生在自己身上的数据更新同步 给从服务器，从而一直保证主从服务器的数据相同。

特点：

1、master/slave 角色

2、master/slave 数据相同

3、降低 master 读压力在转交从库

问题：

无法保证高可用

没有解决 master 写的压力

3.哨兵

Redis sentinel 是一个分布式系统中监控 redis 主从服务器，并在主服务器下线时自动进行故障转移。其中三个特性：

监控（Monitoring）： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。

提醒（Notification）： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。

自动故障迁移（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作。

特点：

1、保证高可用

2、监控各个节点

3、自动故障迁移

缺点：主从模式，切换需要时间丢数据

没有解决 master 写的压力

4.集群（proxy）型

Twemproxy 是一个 Twitter 开源的一个 redis 和 memcache 快速/轻量级代理服务器； Twemproxy 是一个快速的单线程代理程序，支持 Memcached ASCII 协议和 redis 协议。

特点：1、多种 hash 算法：MD5、CRC16、CRC32、CRC32a、hsieh、murmur、Jenkins

2、支持失败节点自动删除

3、后端 Sharding 分片逻辑对业务透明，业务方的读写方式和操作单个 Redis 一致

缺点：增加了新的 proxy，需要维护其高可用。

failover 逻辑需要自己实现，其本身不能支持故障的自动转移可扩展性差，进行扩缩容都需要手动干预

5.集群（直连）型

从redis 3.0之后版本支持redis-cluster集群，Redis-Cluster采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接。

特点：

1、无中心架构（不存在哪个节点影响性能瓶颈），少了 proxy 层。

2、数据按照 slot 存储分布在多个节点，节点间数据共享，可动态调整数据分布。

3、可扩展性，可线性扩展到 1000 个节点，节点可动态添加或删除。

4、高可用性，部分节点不可用时，集群仍可用。通过增加 Slave 做备份数据副本

5、实现故障自动 failover，节点之间通过 gossip 协议交换状态信息，用投票机制完成 Slave到 Master 的角色提升。

缺点：

1、资源隔离性较差，容易出现相互影响的情况。

2、数据通过异步复制,不保证数据的强一致性

12.5. 什么是缓存穿透？如何避免？什么是缓存雪崩？何如避免？

缓存穿透

一般的缓存系统，都是按照key去缓存查询，如果不存在对应的value，就应该去后端系统查找（比如DB）。一些恶意的请求会故意查询不存在的key,请求量很大，就会对后端系统造成很大的压力。这就叫做缓存穿透。

如何避免？

1：对查询结果为空的情况也进行缓存，缓存时间设置短一点，或者该key对应的数据insert了之后清理缓存。

2：对一定不存在的key进行过滤。可以把所有的可能存在的key放到一个大的Bitmap中，查询时通过该bitmap过滤。

3：在第一次请求查询给key对应的数据时加上互斥锁，其他线程走到此步会被阻塞，等到第一个线程查询到数据并写入到缓存再释放锁。后面的线程就直接走缓存拿数据了。

缓存雪崩

当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，会给后端系统带来很大压力。导致系统崩溃。

如何避免？

1：在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个key只允许一个线程查询数据和写缓存，其他线程等待。

2：做二级缓存，A1为原始缓存，A2为拷贝缓存，A1失效时，可以访问A2，A1缓存失效时间设置为短期，A2设置为长期

3：不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀。

面试重要内容算法篇

1. BitMap算法处理大数据。或者分布式计算

通过bit位来标志是否存在 比如一个3，6，1，5，7 就把一个8位的00000000 变成 11101010 从低到高，要考虑大端小端问题。

2. B树，b+树，AVL树

B树分支节点存储数据，B+树在分支节点不存储数据。

优化的B+树在叶子节点有指针指向下一个节点。

3. DP问题的思想（dp和分治，贪心的区别）

DP是解决多阶段决策问题，每个阶段都要做一个决策，全部决策是一个决策序列

分治：区别在于分治出来的子问题必须没有联系，如果有的话，虽然也能解决，但是复杂度太大，不划算

贪心：每次都局部最优，做出的不可撤回的决策，而dp中有考察

4. 错位重排公式

错位重排：Dn=(n-1)* (Dn-1+Dn-2)

n时时针分针重叠公式：n时60* n/11分 平面分割公式： 平面数=n*(n+1)/2 +1. n为线的数量

直线和弦分割圆：n条直径和一条弦把圆分割为3n+1部分

烙饼问题： 饼的面数* 每一面所需要的时间/锅里可以同时烙的面数

银行面视： C/C++ ,JAVA,数据结构与算法，设计模式，数据库（MYSQL，HADOOP），通信原理，操作系统，UI设计（2道题）等知识

2-二叉树， 3-出栈入栈 4-程序设计模式 5-数据库赋予权限操作 6-原型模式 7-hadoop 8-人工智能的开发语言，java/python/c/c#/.net/lisp/prolog/matlab 9-黑盒测试 10-范式 11-排序算法 12-有符号数，左闭右开[-128,128) 13-指针的优缺点 14-栈的优缺点 15-一个二叉排序树从大到小输出