**一、基础篇**

**1.1、Java基础**

* 面向对象的特征：继承、封装和多态
* final, finally, finalize 的区别
* Exception、Error、运行时异常与一般异常有何异同
* 请写出5种常见到的runtime exception
* int 和 Integer 有什么区别，Integer的值缓存范围
* 包装类，装箱和拆箱
* String、StringBuilder、StringBuffer
* 重载和重写的区别
* 抽象类和接口有什么区别
* 说说反射的用途及实现
* 说说自定义注解的场景及实现
* HTTP请求的GET与POST方式的区别
* Session与Cookie区别
* 列出自己常用的JDK包
* MVC设计思想
* equals与==的区别
* hashCode和equals方法的区别与联系
* 什么是Java序列化和反序列化，如何实现Java序列化？或者请解释Serializable 接口的作用
* Object类中常见的方法，为什么wait  notify会放在Object里边？
* Java的平台无关性如何体现出来的
* JDK和JRE的区别
* Java 8有哪些新特性

**1.2、Java常见集合**

* List 和 Set 区别
* Set和hashCode以及equals方法的联系
* List 和 Map 区别
* Arraylist 与 LinkedList 区别
* ArrayList 与 Vector 区别
* HashMap 和 Hashtable 的区别
* HashSet 和 HashMap 区别
* HashMap 和 ConcurrentHashMap 的区别
* HashMap 的工作原理及代码实现，什么时候用到红黑树
* 多线程情况下HashMap死循环的问题
* HashMap出现Hash DOS攻击的问题
* ConcurrentHashMap 的工作原理及代码实现，如何统计所有的元素个数
* 手写简单的HashMap
* 看过那些Java集合类的源码

**1.3、进程和线程**

* 线程和进程的概念、并行和并发的概念
* 创建线程的方式及实现
* 进程间通信的方式
* 说说 CountDownLatch、CyclicBarrier 原理和区别
* 说说 Semaphore 原理
* 说说 Exchanger 原理
* ThreadLocal 原理分析，ThreadLocal为什么会出现OOM，出现的深层次原理
* 讲讲线程池的实现原理
* 线程池的几种实现方式
* 线程的生命周期，状态是如何转移的
* 可参考：《[Java多线程编程核心技术](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247484881&idx=2&sn=b0ecf85cd7c9e543c84e7a9859c20a26&chksm=e9c5fc60deb27576a6a9c453dabc585f43d9f29fd8a8f37ed0e7cc2f012c86b23fbd21763a39&scene=21#wechat_redirect)》

**1.4、锁机制**

* 说说线程安全问题，什么是线程安全，如何保证线程安全
* 重入锁的概念，重入锁为什么可以防止死锁
* 产生死锁的四个条件（互斥、请求与保持、不剥夺、循环等待）
* 如何检查死锁（通过jConsole检查死锁）

参考链接：<https://blog.csdn.net/u014039577/article/details/52351626>

<https://www.cnblogs.com/flyingeagle/articles/6853167.html>

###### Jconsole查看死锁：

进入java安装的位置，输入Jconsole，然后弹出界面（或者进入安装目录/java/jdk1.70\_80/bin/，点击Jconsole.exe）

点击线程进入

然后点击检测死锁

然后可以看到造成死锁的两个线程，以及死锁原因

###### Jstack查看死锁：

同样，也是进入jdk安装目录的bin下面，输入jps，先查看我们要检测死锁的进程

然后可以看到进程Test的进程号：8384，然后执行：Jstack -l 8384

查看死锁信息(Java-level deadlock)

* volatile 实现原理（禁止指令重排、刷新内存）

参考链接：

<http://www.importnew.com/23520.html>

<http://ifeve.com/volatile/>

java编程语言允许线程访问共享变量，为了确保共享变量能被准确和一致的更新，线程应该确保通过排他锁单独获得这个变量。Java语言提供了volatile，在某些情况下比锁更加方便。如果一个字段被声明成volatile，java线程内存模型确保所有线程看到这个变量的值是一致的。

Volatile变量修饰符如果使用**恰当**的话，它比synchronized的**使用和执行成本会更低**，因为它不会引起线程上下文的切换和调度

**Lock前缀指令会引起处理器缓存回写到内存**

**一个处理器的缓存回写到内存会导致其他处理器的缓存无效**

* synchronized 实现原理（对象监视器）

**参考链接：**

<https://www.cnblogs.com/paddix/p/5367116.html>

<https://blog.csdn.net/javazejian/article/details/72828483>

**Synchronized是Java中解决并发问题的一种最常用的方法，也是最简单的一种方法。Synchronized的作用主要有三个：**

**（1）确保线程互斥的访问同步代码**

**（2）保证共享变量的修改能够及时可见**

**（3）有效解决重排序问题。**

**从语法上讲，Synchronized总共有三种用法：**

**（1）修饰普通方法**

**（2）修饰静态方法**

**（3）修饰代码块**

当一个线程正在访问一个对象的 synchronized 实例方法，那么其他线程不能访问该对象的其他 synchronized 方法，毕竟一个对象只有一把锁，当一个线程获取了该对象的锁之后，其他线程无法获取该对象的锁，所以无法访问该对象的其他synchronized实例方法，但是其他线程还是可以访问该实例对象的其他非synchronized方法

* synchronized 与 lock 的区别

参考链接：

<https://www.cnblogs.com/baizhanshi/p/6419268.html>

<https://blog.csdn.net/tinyDolphin/article/details/79273103>

如果一个代码块被synchronized修饰了，当一个线程获取了对应的锁，并执行该代码块时，其他线程便只能一直等待，等待获取锁的线程释放锁，而这里获取锁的线程释放锁只会有两种情况：

1）获取锁的线程执行完了该代码块，然后线程释放对锁的占有；

2）线程执行发生异常，此时JVM会让线程自动释放锁。

当有多个线程读写文件时，读操作和写操作会发生冲突现象，写操作和写操作会发生冲突现象，但是读操作和读操作不会发生冲突现象。

但是采用synchronized关键字来实现同步的话，就会导致一个问题：

如果多个线程都只是进行读操作，所以当一个线程在进行读操作时，其他线程只能等待无法进行读操作。

因此就需要一种机制来使得多个线程都只是进行读操作时，线程之间不会发生冲突，通过Lock就可以办到。

另外，通过Lock可以知道线程有没有成功获取到锁。这个是synchronized无法办到的。

总结一下，也就是说Lock提供了比synchronized更多的功能。但是要注意以下几点：

1）Lock不是Java语言内置的，synchronized是Java语言的关键字，因此是内置特性。Lock是一个类，通过这个类可以实现同步访问；

2）Lock和synchronized有一点非常大的不同，采用synchronized不需要用户去手动释放锁，当synchronized方法或者synchronized代码块执行完之后，系统会自动让线程释放对锁的占用；而Lock则必须要用户去手动释放锁，如果没有主动释放锁，就有可能导致出现死锁现象。

* AQS同步队列

<https://www.cnblogs.com/waterystone/p/4920797.html>

<https://blog.csdn.net/zcw4237256/article/details/78552741>

一脸懵逼

* CAS无锁的概念、乐观锁和悲观锁
* 常见的原子操作类

<https://www.cnblogs.com/boothsun/p/8010692.html>

<https://blog.csdn.net/u011116672/article/details/51068828>

原子操作类共有13个类，在java.util.concurrent.atomic包下，可以分为四种类型的原子更新类：原子更新基本类型、原子更新数组类型、原子更新引用和原子更新属性。

* 什么是ABA问题，出现ABA问题JDK是如何解决的
* 乐观锁的业务场景及实现方式
* Java 8并法包下常见的并发类
* 偏向锁、轻量级锁、重量级锁、自旋锁的概念
* 可参考：《[Java多线程编程核心技术](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247484881&idx=2&sn=b0ecf85cd7c9e543c84e7a9859c20a26&chksm=e9c5fc60deb27576a6a9c453dabc585f43d9f29fd8a8f37ed0e7cc2f012c86b23fbd21763a39&scene=21#wechat_redirect)》

**1.5、JVM**

* JVM运行时内存区域划分

参考链接：

<https://www.cnblogs.com/dolphin0520/p/3613043.html>

<https://www.cnblogs.com/wangjzh/p/5258254.html>

根据《Java虚拟机规范》的规定，运行时数据区通常包括这几个部分：程序计数器(Program Counter Register)、Java栈(VM Stack)、本地方法栈(Native Method Stack)、方法区(Method Area)、堆(Heap)。





1）程序计数器（Program Counter Register）：

它保存的是程序当前执行的指令的地址（也可以说保存下一条指令的所在存储单元的地址），当CPU需要执行指令时，需要从程序计数器中得到当前需要执行的指令所在存储单元的地址，然后根据得到的地址获取到指令，在得到指令之后，程序计数器便自动加1或者根据转移指针得到下一条指令的地址，如此循环，直至执行完所有的指令。

2）Java栈也称作虚拟机栈（Java Vitual Machine Stack）：

当线程执行一个方法时，就会随之创建一个对应的栈帧，并将建立的栈帧压栈。当方法执行完毕之后，便会将栈帧出栈。因此可知，线程当前执行的方法所对应的栈帧必定位于Java栈的顶部。

3）本地方法栈：

本地方法栈与Java栈的作用和原理非常相似。区别只不过是Java栈是为执行Java方法服务的，而本地方法栈则是为执行本地方法（Native Method）服务的。

4）堆：

Java中的堆是用来存储对象本身的以及数组（当然，数组引用是存放在Java栈中的）。Java的垃圾回收机制会自动进行处理。因此这部分空间也是Java垃圾收集器管理的主要区域。另外，堆是被所有线程共享的，在JVM中只有一个堆。

5）方法区：

方法区在JVM中也是一个非常重要的区域，它与堆一样，是被线程共享的区域。在方法区中，存储了每个类的信息（包括类的名称、方法信息、字段信息）、静态变量、常量以及编译器编译后的代码等

* 内存溢出OOM和堆栈溢出SOE的示例及原因、如何排查与解决

OOM:

<https://www.cnblogs.com/ThinkVenus/p/6805495.html>（什么是java OOM？如何分析及解决oom问题？）

<https://www.jianshu.com/p/41ffbf31b20c>(什么是OOM？如何解决OOM问题!)

<https://www.cnblogs.com/kongzhongqijing/articles/7283599.html>（常见OOM现象）

SOE：

参考链接：

<https://www.cnblogs.com/csniper/p/5486828.html>

<https://blog.csdn.net/yanghw117/article/details/80889298>

<https://blog.csdn.net/sinat_29912455/article/details/51125748>(异常、堆内存溢出、OOM的几种情况)

  -Xss128k：设置每个线程的堆栈大小。 JDK5.0以后每个线程堆栈大小为1M，以前每个线程堆栈大小为256K。根据应用的线程所需内存大小进行调整。在相同物理内存下，减小这个值能生成更 多的线程。但是操作系统对一个进程内的线程数还是有限制的，不能无限生成，经验值在3000~5000左右。

* 如何判断对象是否可以回收或存活
* 常见的GC回收算法及其含义

标记-清除算法、复制算法、标记-整理算法

Serial：新生代收集器，采用复制算法，单线程。

ParNew：新生代收集器，采用复制算法，多线程。

Parallel Scavenge：新生代收集器，采用复制算法，多线程，注重吞吐量。

Serial Old：老年代收集器，采用标记-整理算法，单线程。

Parallel Old：老年代收集器，采用标记-整理算法，多线程，与Parallel Scavenge结合使用。

CMS：老年代收集器，采用标记-清除算法，相比以上收集器收集过去相对复杂，停止时间短。

G1：年轻代和老年代收集器，基本采用标记-整理算法，局部采用复制算法，收集过程跟CMS相当，但概念差异，是目前最新的收集器之一，使用范围暂时有待检验。

* 常见的JVM性能监控和故障处理工具类：jps、jstat、jmap、jinfo、jconsole等
* JVM如何设置参数
* JVM性能调优
* 类加载器、双亲委派模型、一个类的生命周期、类是如何加载到JVM中的
* 类加载的过程：加载、验证、准备、解析、初始化
* 强引用、软引用、弱引用、虚引用
* Java内存模型JMM

**1.6、设计模式**

* 常见的设计模式
* 设计模式的的六大原则及其含义

<https://www.cnblogs.com/dolphin0520/p/3919839.html>

<https://blog.csdn.net/q291611265/article/details/48465113>

<https://blog.csdn.net/wuxianzhenjia/article/details/80263619>

|  |  |  |
| --- | --- | --- |
| **原则** | **含义** | **具体方法** |
| 开闭原则 | 对扩展开放,对修改关闭 | 多使用抽象类和接口 |
| 里氏代换原则 | 基类可以被子类替换 | 使用抽象类继承,不使用具体类继承 |
| 依赖倒转原则 | 要依赖于抽象,不要依赖于具体 | 针对接口编程,不针对实现编程 |
| 接口隔离原则 | 使用多个隔离的接口,比使用单个接口好 | 建立最小的接口 |
| 迪米特法则 | 一个软件实体应当尽可能少地与其他实体发生相互作用 | 通过中间类建立联系 |
| 合成复用原则 | 尽量使用合成/聚合,而不是使用继承 | 尽量使用合成/聚合,而不是使用继承 |

一、单一职责原则

该原则是指一个类只负责一个功能领域中的相应职责，或者可以定义为：就一个类而言，应该只有一个引起它变化的原因。它是实现高内聚、低耦合的指导方针，它是最简单但又最难运用的原则，需要设计人员发现类的不同职责并将其分离，而发现类的多重职责需要设计人员具有较强的分析设计能力和相关实践经验。

二、开闭原则

该原则是面向对象的可复用设计的第一块基石，它是最重要的面向对象设计原则。一个软件实体应当对扩展开放，对修改关闭。即软件实体应尽量在不修改原有代码的情况下进行扩展，软件实体可以指一个软件模块、一个由多个类组成的局部结构或一个独立的类。

三、里氏替换原则

该原则是指所有引用基类（父类）的地方必须能透明地使用其子类的对象。在软件中将一个基类对象替换成它的子类对象，程序将不会产生任何错误和异常，反过来则不成立，如果一个软件实体使用的是一个子类对象的话，那么它不一定能够使用基类对象。

四、依赖倒置原则

抽象不应该依赖于细节，细节应当依赖于抽象。换言之，要针对接口编程，而不是针对实现编程。  
在实现依赖倒转原则时，我们需要针对抽象层编程，而将具体类的对象通过依赖注入(DI)的方式注入到其他对象中，依赖注入是指当一个对象要与其他对象发生依赖关系时，通过抽象来注入所依赖的对象。常用的注入方式有三种，分别是：构造注入，设值注入（Setter注入）和接口注入。构造注入是指通过构造函数来传入具体类的对象，设值注入是指通过Setter方法来传入具体类的对象，而接口注入是指通过在接口中声明的业务方法来传入具体类的对象。这些方法在定义时使用的是抽象类型，在运行时再传入具体类型的对象，由子类对象来覆盖父类对象。

五、接口隔离原则

使用多个专门的接口，而不使用单一的总接口，即客户端不应该依赖那些它不需要的接口。  
在使用接口隔离原则时，我们需要注意控制接口的粒度，接口不能太小，如果太小会导致系统中接口泛滥，不利于维护；接口也不能太大，太大的接口将违背接口隔离原则，灵活性较差，使用起来很不方便。一般而言，接口中仅包含为某一类用户定制的方法即可，不应该强迫客户依赖于那些它们不用的方法。

六、迪米特法则

一个软件实体应当尽可能少地与其他实体发生相互作用。迪米特法则可降低系统的耦合度，使类与类之间保持松散的耦合关系。迪米特法则要求我们在设计系统时，应该尽量减少对象之间的交互，如果两个对象之间不必彼此直接通信，那么这两个对象就不应当发生任何直接的相互作用，如果其中的一个对象需要调用另一个对象的某一个方法的话，可以通过第三者转发这个调用。简言之，就是通过引入一个合理的第三者来降低现有对象之间的耦合度。

在将迪米特法则运用到系统设计中时，要注意下面的几点：

1）在类的划分上，应当尽量创建松耦合的类，类之间的耦合度越低，就越有利于复用，一个处在松耦合中的类一旦被修改，不会对关联的类造成太大波及；

2）在类的结构设计上，每一个类都应当尽量降低其成员变量和成员函数的访问权限；

3）在类的设计上，只要有可能，一个类型应当设计成不变类；在对其他类的引用上，一个对象对其他对象的引用应当降到最低。

* 常见的单例模式以及各种实现方式的优缺点，哪一种最好，手写常见的单利模式
* 设计模式在实际场景中的应用
* Spring中用到了哪些设计模式
* MyBatis中用到了哪些设计模式
* 你项目中有使用哪些设计模式
* 说说常用开源框架中设计模式使用分析
* 动态代理很重要！！！

**1.7、数据结构**

* 树（二叉查找树、平衡二叉树、红黑树、B树、B+树）

二叉查找树：

<https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%8E%92%E5%BA%8F%E6%A0%91/10905079?fromtitle=%E4%BA%8C%E5%8F%89%E6%9F%A5%E6%89%BE%E6%A0%91&fromid=7077965&fr=aladdin>

平衡二叉树：

<https://baike.baidu.com/item/%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%A0%91/10421057?fr=aladdin>

红黑树：

<https://baike.baidu.com/item/%E7%BA%A2%E9%BB%91%E6%A0%91/2413209?fr=aladdin>

B树:

<https://baike.baidu.com/item/B%E6%A0%91/5411672>

B+树:

<https://www.cnblogs.com/vincently/p/4526560.html>

* 深度有限算法、广度优先算法
* 克鲁斯卡尔算法、普林母算法、迪克拉斯算法
* 什么是一致性Hash及其原理、Hash环问题
* 常见的排序算法和查找算法：快排、折半查找、堆排序等

**1.8、网络/IO基础**

* BIO、NIO、AIO的概念
* 什么是长连接和短连接
* Http1.0和2.0相比有什么区别，可参考《[Http 2.0](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247484611&idx=1&sn=66c875392eedff8150633ddcd5d83e7a&chksm=e9c5fd72deb274648a607b9bc39bac34adadd768577b77354f6dc85422691605e210b69eeb7b&scene=21#wechat_redirect)》
* Https的基本概念
* 三次握手和四次挥手、为什么挥手需要四次
* 从游览器中输入URL到页面加载的发生了什么？可参考《[从输入URL到页面加载发生了什么](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247483724&idx=1&sn=e58dd30d124971c795584e8673d6cc71&chksm=e9c5f8fddeb271ebebbb6c350ed1abc252f1f26b4f35c4ce36e10bde9659a37520feabed2290&scene=21#wechat_redirect)》

**二、数据存储和消息队列**

**2.1、数据库**

* MySQL 索引使用的注意事项

<https://www.cnblogs.com/heyonggang/p/6610526.html>

<https://blog.csdn.net/qq_26963433/article/details/78631487>

<https://www.cnblogs.com/xyhero/p/b0ad525c6a6a5ed2bd7f40918c5dbd98.html>

索引虽然好处很多，但过多的使用索引可能带来相反的问题，索引也是有缺点的：

虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT,UPDATE和DELETE。因为更新表时，mysql不仅要保存数据，还要保存一下索引文件

建立索引会占用磁盘空间的索引文件。一般情况这个问题不太严重，但如果你在要给大表上建了多种组合索引，索引文件会膨胀很宽

索引只是提高效率的一个方式，如果mysql有大数据量的表，就要花时间研究建立最优的索引，或优化查询语句。

使用索引时，有一些技巧：

1.索引不会包含有NULL的列

只要列中包含有NULL值，都将不会被包含在索引中，复合索引中只要有一列含有NULL值，那么这一列对于此符合索引就是无效的。

2.使用短索引

对串列进行索引，如果可以就应该指定一个前缀长度。例如，如果有一个char（255）的列，如果在前10个或20个字符内，多数值是唯一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和I/O操作。

3.索引列排序

mysql查询只使用一个索引，因此如果where子句中已经使用了索引的话，那么order by中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作，尽量不要包含多个列的排序，如果需要最好给这些列建复合索引。

4.like语句操作

一般情况下不鼓励使用like操作，如果非使用不可，注意正确的使用方式。like ‘%aaa%’不会使用索引，而like ‘aaa%’可以使用索引。

5.不要在列上进行运算

6.不使用NOT IN 、<>、！=操作，但<,<=，=，>,>=,BETWEEN,IN是可以用到索引的

7.索引要建立在经常进行select操作的字段上。

这是因为，如果这些列很少用到，那么有无索引并不能明显改变查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

8.索引要建立在值比较唯一的字段上。

9.对于那些定义为text、image和bit数据类型的列不应该增加索引。因为这些列的数据量要么相当大，要么取值很少。

10.在where和join中出现的列需要建立索引。

11.where的查询条件里有不等号(where column != …),mysql将无法使用索引。

12.如果where字句的查询条件里使用了函数(如：where DAY(column)=…),mysql将无法使用索引。

13.在join操作中(需要从多个数据表提取数据时)，mysql只有在主键和外键的数据类型相同时才能使用索引，否则及时建立了索引也不会使用。

* DDL、DML、DCL分别指什么
* explain命令
* left join，right join，inner join
* 数据库事物ACID（原子性、一致性、隔离性、持久性）

<https://www.2cto.com/database/201710/691297.html>

<https://baike.baidu.com/item/acid/10738>

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，必须要具有这四种[特性](https://baike.baidu.com/item/%E7%89%B9%E6%80%A7/3128227)，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。

原子性

整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被[回滚](https://baike.baidu.com/item/%E5%9B%9E%E6%BB%9A)（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性

一个事务可以封装状态改变（除非它是一个只读的）。事务必须始终保持系统处于一致的状态，不管在任何给定的时间[并发](https://baike.baidu.com/item/%E5%B9%B6%E5%8F%91)事务有多少。

也就是说：如果事务是[并发](https://baike.baidu.com/item/%E5%B9%B6%E5%8F%91)多个，系统也必须如同串行事务一样操作。其主要特征是保护性和不变性(Preserving an Invariant)，以转账[案例](https://baike.baidu.com/item/%E6%A1%88%E4%BE%8B)为例，假设有五个账户，每个账户余额是100元，那么五个账户总额是500元，如果在这个5个账户之间同时发生多个转账，无论[并发](https://baike.baidu.com/item/%E5%B9%B6%E5%8F%91)多少个，比如在A与B账户之间转账5元，在C与D账户之间转账10元，在B与E之间转账15元，五个账户总额也应该还是500元，这就是保护性和不变性。

隔离性

隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为[串行](https://baike.baidu.com/item/%E4%B8%B2%E8%A1%8C)化，为了防止事务操作间的混淆，必须串行化或序列化请求，使得在同一时间仅有一个请求用于同一数据。

持久性

在事务完成以后，该事务对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

* 事物的隔离级别（读未提交、读以提交、可重复读、可序列化读）

<https://www.2cto.com/database/201710/691297.html>

<https://blog.csdn.net/Somhu/article/details/78775198>

隔离性的四个级别

READ UNCOMMITTED（未提交读）

在READ UNCOMMITTED级别，事务中的修改，即使没有提交，对其他事务也都是可见的。事务可以读取未提交的数据，这也被称为脏读（Dirty Read）。这个级别会导致很多问题，从性能上来说，READ UNCOMMITTED不会比其他的级别好太多，但却缺乏其他级别的很多好处，除非真的有非常必要的理由，在实际应用中一般很少使用。

READ COMMITTED（提交读）

大多数数据库系统的默认隔离级别都是READ COMMTTED（但MySQL不是）。READ COMMITTED满足前面提到的隔离性的简单定义：一个事务开始时，只能”看见”已经提交的事务所做的修改。换句话说，一个事务从开始直到提交之前，所做的任何修改对其他事务都是不可见的。这个级别有时候叫做不可重复读（nonrepeatble read），因为两次执行同样的查询，可能会得到不一样的结果。

REPEATABLE READ(可重复读)

REPEATABLE READ解决了脏读的问题。该隔离级别保证了在同一个事务中多次读取同样记录结果是一致的。但是理论上，可重复读隔离级别还是无法解决另外一个幻读（Phantom Read）的问题。所谓幻读，指的是当某个事务在读取某个范围内的记录时，另一个事务又在该范围内插入了新的记录，当之前的事务再次读取该范围的记录时，会产生幻行（Phantom Row）。InnoDB和XtraDB存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）解决了幻读的问题。

SERIALIZABLE（串行化）

SERIALIZABLE是最高的隔离级别。它通过强制事务串行执行，避免了前面说的幻读的问题。简单来说，SERIALIZABLE会在读取每一行数据都加锁，所以可能导致大量的超时和锁争用问题。实际应用中也很少用到这个隔离级别，只有在非常需要确保数据的一致性而且可以接受没有并发的情况下，才考虑采用该级别。

* 脏读、幻读、不可重复读

<https://blog.csdn.net/Somhu/article/details/78775198>

也许有很多读者会对上述隔离级别中提及到的 **脏读、不可重复读、幻读** 的理解有点吃力，我在这里尝试使用通俗的方式来解释这三种语义：

**脏读：**所谓的脏读，其实就是读到了别的事务回滚前的脏数据。比如事务B执行过程中修改了数据X，在未提交前，事务A读取了X，而事务B却回滚了，这样事务A就形成了脏读。

也就是说，当前事务读到的数据是别的事务想要修改成为的但是没有修改成功的数据。

**不可重复读：**事务A首先读取了一条数据，然后执行逻辑的时候，事务B将这条数据改变了，然后事务A再次读取的时候，发现数据不匹配了，就是所谓的不可重复读了。

也就是说，当前事务先进行了一次数据读取，然后再次读取到的数据是别的事务修改成功的数据，导致两次读取到的数据不匹配，也就照应了不可重复读的语义。

**幻读：**事务A首先根据条件索引得到N条数据，然后事务B改变了这N条数据之外的M条或者增添了M条符合事务A搜索条件的数据，导致事务A再次搜索发现有N+M条数据了，就产生了幻读。

也就是说，当前事务读第一次取到的数据比后来读取到数据条目少。

不可重复读和幻读比较：

两者有些相似，但是前者针对的是update或delete，后者针对的insert。

* 数据库的几大范式

<https://blog.csdn.net/qq_29232943/article/details/62421776>

<https://www.cnblogs.com/lca1826/p/6601395.html>

 第一范式（1NF）无重复的列

    所谓第一范式（1NF）是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。如果出现重复的属性，就可能需要定义一个新的实体，新的实体由重复的属性构成，新实体与原实体之间为一对多关系。在第一范式（1NF）中表的每一行只包含一个实例的信息。简而言之，第一范式就是无重复的列。

说明：在任何一个关系数据库中，第一范式（1NF）是对关系模式的基本要求，不满足第一范式（1NF）的数据库就不是关系数据库。

**第一范式（1NF）：属性不可分。**

在前面已经介绍了属性值的概念，我们说，它是“不可分的”。而第一范式要求属性也不可分

第二范式（2NF）属性完全依赖于主键 [ 消除部分子函数依赖 ]

如果关系模式R为第一范式，并且R中每一个非主属性完全函数依赖于R的某个候选键， 则称为第二范式模式。

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。这个惟一属性列被称为主关键字或主键、主码。

**第二范式（2NF）：符合1NF，并且，非主属性完全依赖于码。**（注意是完全依赖不能是部分依赖，设有函数依赖W→A，若存在XW，有X→A成立，那么称W→A是局部依赖，否则就称W→A是完全函数依赖）

第三范式（3NF）属性不依赖于其它非主属性 [ 消除传递依赖 ]

如果关系模式R是第二范式，且每个非主属性都不传递依赖于R的候选键，则称R为第三范式模式。

满足第三范式（3NF）必须先满足第二范式（2NF）。第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。

**第三范式（3NF）：符合2NF，并且，消除传递依赖**（也就是每个非主属性都不传递依赖于候选键，判断传递函数依赖，指的是如果存在"A → B → C"的决定关系，则C传递函数依赖于A。）

* 数据库常见的命令
* 说说分库与分表设计
* 分库与分表带来的分布式困境与应对之策（如何解决分布式下的分库分表，全局表？）

##### 说说 SQL 优化之道

<https://www.cnblogs.com/yycc/p/7338894.html>

explain解释

<https://blog.csdn.net/jie_liang/article/details/77340905>

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

select id from t where num is null

可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：

select id from t where num=0

3.应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

4.应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：

select id from t where num=10 or num=20

可以这样查询：

select id from t where num=10

union all

select id from t where num=20

5.in 和 not in 也要慎用，否则会导致全表扫描，如：

select id from t where num in(1,2,3)

对于连续的数值，能用 between 就不要用 in 了：

select id from t where num between 1 and 3

6.下面的查询也将导致全表扫描：

select id from t where name like '%abc%'

7.应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：

select id from t where num/2=100

应改为:

select id from t where num=100\*2

8.应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：

select id from t where substring(name,1,3)='abc'--name以abc开头的id

应改为:

select id from t where name like 'abc%'

9.不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

10.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，

否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

11.不要写一些没有意义的查询，如需要生成一个空表结构：

select col1,col2 into #t from t where 1=0

这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：

create table #t(...)

12.很多时候用 exists 代替 in 是一个好的选择：

select num from a where num in(select num from b)

用下面的语句替换：

select num from a where exists(select 1 from b where num=a.num)

13.并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，

如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。

14.索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，

因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。

一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

15.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。

这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

16.尽可能的使用 varchar 代替 char ，因为首先变长字段存储空间小，可以节省存储空间，

其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

17.任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。

18.避免频繁创建和删除临时表，以减少系统表资源的消耗。

19.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

20.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，

以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

21.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

22.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

23.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

24.与临时表一样，游标并不是不可使用。对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。

在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

25.尽量避免大事务操作，提高系统并发能力。26.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

<https://blog.csdn.net/wuhuagu_wuhuaguo/article/details/72875054>

* MySQL遇到的死锁问题、如何排查与解决

<https://blog.csdn.net/mine_song/article/details/71106410>

MySQL有三种锁的级别：页级、表级、行级。

表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高,并发度最低。

行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。

页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般

Demo:

<https://www.cnblogs.com/sivkun/p/7518540.html>

**死锁的第一种情况**

一个用户A 访问表A(锁住了表A),然后又访问表B；另一个用户B 访问表B(锁住了表B)，然后企图访问表A；这时用户A由于用户B已经锁住表B，它必须等待用户B释放表B才能继续，同样用户B要等用户A释放表A才能继续，这就死锁就产生了。

解决方法：

这种死锁比较常见，是由于程序的BUG产生的，除了调整的程序的逻辑没有其它的办法。仔细分析程序的逻辑，对于数据库的多表操作时，尽量按照相同的顺序进 行处理，尽量避免同时锁定两个资源，如操作A和B两张表时，总是按先A后B的顺序处理， 必须同时锁定两个资源时，要保证在任何时刻都应该按照相同的顺序来锁定资源。

**死锁的第二种情况**

用户A查询一条纪录，然后修改该条纪录；这时用户B修改该条纪录，这时用户A的事务里锁的性质由查询的共享锁企图上升到独占锁，而用户B里的独占锁由于A 有共享锁存在所以必须等A释放掉共享锁，而A由于B的独占锁而无法上升的独占锁也就不可能释放共享锁，于是出现了死锁。这种死锁比较隐蔽，但在稍大点的项 目中经常发生。如在某项目中，页面上的按钮点击后，没有使按钮立刻失效，使得用户会多次快速点击同一按钮，这样同一段代码对数据库同一条记录进行多次操 作，很容易就出现这种死锁的情况。

**死锁的第三种情况**

如果在事务中执行了一条不满足条件的update语句，则执行全表扫描，把行级锁上升为表级锁，多个这样的事务执行后，就很容易产生死锁和阻塞。类似的情 况还有当表中的数据量非常庞大而索引建的过少或不合适的时候，使得经常发生全表扫描，最终应用系统会越来越慢，最终发生阻塞或死锁。

**案例2：**

在开发中，经常会做这类的判断需求：根据字段值查询（有索引），如果不存在，则插入；否则更新。

查看死锁：

show engine innodb status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 读数据一致性 | 脏读 | 不可重复读 | 幻读 |
| 未提交读 | 最低级别 | 是 | 是 | 是 |
| 已提交读 | 语句级 | 否 | 是 | 是 |
| 可重复读 | 事务级 | 否 | 否 | 是 |
| 可序列化 | 最高级别，事务级 | 否 | 否 | 否 |

* 存储引擎的 InnoDB与MyISAM区别，优缺点，使用场景
* 索引类别（B+树索引、全文索引、哈希索引）、索引的原理
* 什么是自适应哈希索引（AHI）
* 为什么要用 B+tree作为MySQL索引的数据结构
* 聚集索引与非聚集索引的区别
* 遇到过索引失效的情况没，什么时候可能会出现，如何解决
* limit 20000 加载很慢怎么解决
* 如何选择合适的分布式主键方案
* 选择合适的数据存储方案
* 常见的几种分布式ID的设计方案
* 常见的数据库优化方案，在你的项目中数据库如何进行优化的

**2.2、Redis**

* Redis 有哪些数据类型，可参考《[Redis常见的5种不同的数据类型详解](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247483987&idx=1&sn=5c5e4cd5bc73a7e6f84e5d6adfab0935&chksm=e9c5fbe2deb272f4b5b75bd2ac92bb27950452623ec83c0e1add7e30c773160421fab1571680&scene=21#wechat_redirect)》
* Redis 内部结构
* Redis 使用场景
* Redis 持久化机制，可参考《[使用快照和AOF将Redis数据持久化到硬盘中](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247483992&idx=1&sn=8f554bc490c4db1a78a30144f873e911&chksm=e9c5fbe9deb272fff47483c241e6d2a7aae99dc8f6fe9fee31f2dd214d0cf81b33d51f7a7dbe&scene=21#wechat_redirect)》
* Redis 集群方案与实现
* Redis 为什么是单线程的？
* 缓存雪崩、缓存穿透、缓存预热、缓存更新、缓存降级
* 使用缓存的合理性问题
* Redis常见的回收策略

**2.3、消息队列**

* 消息队列的使用场景
* 消息的重发补偿解决思路
* 消息的幂等性解决思路
* 消息的堆积解决思路
* 自己如何实现消息队列
* 如何保证消息的有序性

**三、开源框架和容器**

**3.1、SSM/Servlet**

* Servlet的生命周期

参考链接：

<https://blog.csdn.net/paicMis/article/details/52421706>

Servlet 通过调用 init () 方法进行初始化。

Servlet 调用 service() 方法来处理客户端的请求。

Servlet 通过调用 destroy() 方法终止（结束）。

最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

* 转发与重定向的区别

参考链接：<https://blog.csdn.net/lijun102542/article/details/78518733>

转发在服务器端完成的；重定向是在客户端完成的

转发的速度快；重定向速度慢

转发的是同一次请求；重定向是两次不同请求

转发不会执行转发后的代码；重定向会执行重定向之后的代码

转发地址栏没有变化；重定向地址栏有变化

转发必须是在同一台服务器下完成；重定向可以在不同的服务器下完成

* BeanFactory 和 ApplicationContext 有什么区别

参考链接：<https://blog.csdn.net/lxyso/article/details/45446757>

<https://www.cnblogs.com/aspirant/p/9082858.html>

作用：

1. BeanFactory负责读取bean配置文档，管理bean的加载，实例化，维护bean之间的依赖关系，负责bean的声明周期。

2. ApplicationContext除了提供上述BeanFactory所能提供的功能之外，还提供了更完整的框架功能：

a. 国际化支持

b. 资源访问：Resource rs = ctx. getResource(“classpath:config.properties”), “file:c:/config.properties”

c. 事件传递：通过实现ApplicationContextAware接口

3. 常用的获取ApplicationContext的方法

1.利用MessageSource进行国际化

2.强大的事件机制(Event)

3.底层资源的访问

4.对Web应用的支持

5.其它区别   
1).BeanFactroy采用的是延迟加载形式来注入Bean的，即只有在使用到某个Bean时(调用getBean())，才对该Bean进行加载实例化，这样，我们就不能发现一些存在的Spring的配置问题。而ApplicationContext则相反，它是在容器启动时，一次性创建了所有的Bean。这样，在容器启动时，我们就可以发现Spring中存在的配置错误。

2).BeanFactory和ApplicationContext都支持BeanPostProcessor、BeanFactoryPostProcessor的使用，但两者之间的区别是：BeanFactory需要手动注册，而ApplicationContext则是自动注册  
总结  
1.如果使用ApplicationContext，则配置的bean如果是singleton不管你用还是不用，都被实例化。好处是可以预先加载，坏处是浪费内存。  
2.BeanFactory，当使用BeanFactory实例化对象时，配置的bean不会马上被实例化。当你使用该bean时才会被实例化（getBean）。好处是节约内存，缺点是速度比较慢。多用于移动设备的开发上。  
3.一般没有特殊要求，应当使用ApplicationContext完成。

* Spring Bean 的生命周期

参考链接：<https://www.cnblogs.com/kenshinobiy/p/4652008.html>

实现：/study-spring/src/main/java/com/bage/lifecircle/LifeCircleBean.java

1. 实例化一个Bean，也就是我们通常说的new

2. 按照Spring上下文对实例化的Bean进行配置，也就是IOC注入

3. 如果这个Bean实现了BeanNameAware接口，会调用它实现的setBeanName(String beanId)方法，此处传递的是Spring配置文件中Bean的ID

4. 如果这个Bean实现了BeanFactoryAware接口，会调用它实现的setBeanFactory()，传递的是Spring工厂本身（可以用这个方法获取到其他Bean）

5. 如果这个Bean实现了ApplicationContextAware接口，会调用setApplicationContext(ApplicationContext)方法，传入Spring上下文，该方式同样可以实现步骤4，但比4更好，以为ApplicationContext是BeanFactory的子接口，有更多的实现方法

6. 如果这个Bean关联了BeanPostProcessor接口，将会调用postProcessBeforeInitialization(Object obj, String s)方法，BeanPostProcessor经常被用作是Bean内容的更改，并且由于这个是在Bean初始化结束时调用After方法，也可用于内存或缓存技术

7. 如果这个Bean在Spring配置文件中配置了init-method属性会自动调用其配置的初始化方法

8. 如果这个Bean关联了BeanPostProcessor接口，将会调用postAfterInitialization(Object obj, String s)方法

注意：以上工作完成以后就可以用这个Bean了，那这个Bean是一个single的，所以一般情况下我们调用同一个ID的Bean会是在内容地址相同的实例

9. 当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean接口，会调用其实现的destroy方法

10. 最后，如果这个Bean的Spring配置中配置了destroy-method属性，会自动调用其配置的销毁方法

* Spring IOC 如何实现

参考链接：

<https://www.cnblogs.com/ITtangtang/p/3978349.html>

* Spring中Bean的作用域，默认的是哪一个

singleton: 在Spring的IoC容器中只存在一个对象实例，所有该对象的引用都共享这个实例。Spring 容器只会创建该bean定义的唯一实例，这个实例会被保存到缓存中，并且对该bean的所有后续请求和引用都将返回该缓存中的对象实例，一般情况下，无状态的bean使用该scope。

prototype:每次对该bean的请求都会创建一个新的实例，一般情况下，有状态的bean使用该scope。

request：每次http请求将会有各自的bean实例，类似于prototype。

session：在一个http session中，一个bean定义对应一个bean实例。

global session：在一个全局的http session中，一个bean定义对应一个bean实例。典型情况下，仅在使用portlet context的时候有效。

* 说说 Spring AOP、Spring AOP 实现原理
* 动态代理（CGLib 与 JDK）、优缺点、性能对比、如何选择
* Spring 事务实现方式、事务的传播机制、默认的事务类别
* Spring 事务底层原理
* Spring事务失效（事务嵌套），JDK动态代理给Spring事务埋下的坑，可参考《[JDK动态代理给Spring事务埋下的坑！](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247484940&idx=1&sn=0a0a7198e96f57d610d3421b19573002&chksm=e9c5ffbddeb276ab64ff3b3efde003193902c69acda797fdc04124f6c2a786255d58817b5a5c&scene=21#wechat_redirect)》
* 如何自定义注解实现功能
* Spring MVC 运行流程
* Spring MVC 启动流程
* Spring 的单例实现原理
* Spring 框架中用到了哪些设计模式
* Spring 其他产品（Srping Boot、Spring Cloud、Spring Secuirity、Spring Data、Spring AMQP 等）
* 有没有用到Spring Boot，Spring Boot的认识、原理
* MyBatis的原理
* 可参考《[为什么会有Spring](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247484822&idx=1&sn=6fbee2a12b31b6102a18d3725671d41b&chksm=e9c5fc27deb275319641c3f30d168b85c7c196fd276d47efa35046b5dc54f5b77174c5bf8808&scene=21#wechat_redirect)》
* 可参考《[为什么会有Spring AOP](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247484827&idx=1&sn=b9d82f3fced6a875f8dfc22e5849b28e&chksm=e9c5fc2adeb2753c516ef8fc959c0c9dd84ccacaa40473b64bc58b5137c30562a0b45803ba8e&scene=21#wechat_redirect)》

**3.2、Netty**

* 为什么选择 Netty
* 说说业务中，Netty 的使用场景
* 原生的 NIO 在 JDK 1.7 版本存在 epoll bug
* 什么是TCP 粘包/拆包
* TCP粘包/拆包的解决办法
* Netty 线程模型
* 说说 Netty 的零拷贝
* Netty 内部执行流程
* Netty 重连实现

**3.3、Tomcat**

* Tomcat的基础架构（Server、Service、Connector、Container）
* Tomcat如何加载Servlet的
* Pipeline-Valve机制
* 可参考：《[四张图带你了解Tomcat系统架构](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247484905&idx=1&sn=6c8acd89476fadbc4cb9ccfda9c9c2e4&chksm=e9c5fc58deb2754e7519511bb0ed8dcbfa3fe29179663b53f3626643f8b9c82068d9b0464ee6&scene=21#wechat_redirect)！》

**四、分布式**

**4.1、Nginx**

* 请解释什么是C10K问题或者知道什么是C10K问题吗？
* Nginx简介，可参考《[Nginx简介](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247483994&idx=1&sn=b6591f62c7ea6b4adc5a5bf1bf4eac40&chksm=e9c5fbebdeb272fdd865a9c61a380f6b909fc988f99d00ce0aa8c3efca501644db46c40bd4f2&scene=21#wechat_redirect)》
* 正向代理和反向代理.
* Nginx几种常见的负载均衡策略
* Nginx服务器上的Master和Worker进程分别是什么
* 使用“反向代理服务器”的优点是什么?

**4.2、分布式其他**

* 谈谈业务中使用分布式的场景
* Session 分布式方案
* Session 分布式处理
* 分布式锁的应用场景、分布式锁的产生原因、基本概念
* 分布是锁的常见解决方案
* 分布式事务的常见解决方案
* 集群与负载均衡的算法与实现
* 说说分库与分表设计，可参考《[数据库分库分表策略的具体实现方案](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247483931&idx=1&sn=6eda41aa81c1243422a603205d2fad22&chksm=e9c5fbaadeb272bc92537803c14a6f55e1170b1a3b8f60160f66417800c0ace960dfe192717a&scene=21#wechat_redirect)》
* 分库与分表带来的分布式困境与应对之策

**4.3、Dubbo**

* 什么是Dubbo，可参考《[Dubbo入门](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247483791&idx=1&sn=49345f1a022734e81e9257f2b8d38a52&chksm=e9c5f83edeb2712805a77c1e1589e8f1d04bd17e55eeb2a45cabddb46d03615636908f058628&scene=21#wechat_redirect)》
* 什么是RPC、如何实现RPC、RPC 的实现原理，可参考《[基于HTTP的RPC实现](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247483900&idx=1&sn=c5ca198a66a701f81c2ab118fe7a734a&chksm=e9c5f84ddeb2715bc574e467cd6537ef81f223453e0989ffd136976b48dcc2d961a75be596de&scene=21#wechat_redirect)》
* Dubbo中的SPI是什么概念
* Dubbo的基本原理、执行流程

**五、微服务**

**5.1、微服务**

* 前后端分离是如何做的？
* 微服务哪些框架
* Spring Could的常见组件有哪些？可参考《[Spring Cloud概述](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247484125&idx=1&sn=ddba9fba6ae900f5ef71a68f70afebe5&chksm=e9c5fb6cdeb2727a1e8d16a05adb7d9df0170e392ae579d1e40075fb488d680f0c061aa45327&scene=21#wechat_redirect)》
* 领域驱动有了解吗？什么是领域驱动模型？充血模型、贫血模型
* JWT有了解吗，什么是JWT，可参考《[前后端分离利器之JWT](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247485183&idx=1&sn=05dac824dbb534710dd99d6c895fbaf5&chksm=e9c5ff4edeb27658173c8b06ad6d1241d3b7822c734ddf6ac064d40e63cb0cb0a0c90804b9c7&scene=21#wechat_redirect)》
* 你怎么理解 RESTful
* 说说如何设计一个良好的 API
* 如何理解 RESTful API 的幂等性
* 如何保证接口的幂等性
* 说说 CAP 定理、BASE 理论
* 怎么考虑数据一致性问题
* 说说最终一致性的实现方案
* 微服务的优缺点，可参考《[微服务批判](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247485005&idx=1&sn=78a1d286c6a15a81ea5dcf6634a70b54&chksm=e9c5fffcdeb276ea3c766a6e5954685db0e89bea8ff0f47c9a3ff2c0c02991f791a5160287c4&scene=21#wechat_redirect)》
* 微服务与 SOA 的区别
* 如何拆分服务、水平分割、垂直分割
* 如何应对微服务的链式调用异常
* 如何快速追踪与定位问题
* 如何保证微服务的安全、认证

**5.2、安全问题**

* 如何防范常见的Web攻击、如何方式SQL注入
* 服务端通信安全攻防
* HTTPS原理剖析、降级攻击、HTTP与HTTPS的对比

**5.3、性能优化**

* 性能指标有哪些
* 如何发现性能瓶颈
* 性能调优的常见手段
* 说说你在项目中如何进行性能调优

**六、其他**

**6.1、设计能力**

* 说说你在项目中使用过的UML图
* 你如何考虑组件化、服务化、系统拆分
* 秒杀场景如何设计
* 可参考：《[秒杀系统的技术挑战、应对策略以及架构设计总结一二！](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247485294&idx=1&sn=681b3fc8833bc978344f66c8dd33ff32&chksm=e9c5fedfdeb277c96e03e8943bff709ad5354cbbe0c0d894cdb9d1e8cc8da92bf51f1e832e37&scene=21#wechat_redirect)》

**6.2、业务工程**

* 说说你的开发流程、如何进行自动化部署的
* 你和团队是如何沟通的
* 你如何进行代码评审
* 说说你对技术与业务的理解
* 说说你在项目中遇到感觉最难Bug，是如何解决的
* 介绍一下工作中的一个你认为最有价值的项目，以及在这个过程中的角色、解决的问题、你觉得你们项目还有哪些不足的地方

**6.3、软实力**

* 说说你的优缺点、亮点
* 说说你最近在看什么书、什么博客、在研究什么新技术、再看那些开源项目的源代码
* 说说你觉得最有意义的技术书籍
* 工作之余做什么事情、平时是如何学习的，怎样提升自己的能力
* 说说个人发展方向方面的思考
* 说说你认为的服务端开发工程师应该具备哪些能力
* 说说你认为的架构师是什么样的，架构师主要做什么
* 如何看待加班的问题

当然，一个完整的面试肯定不止上述的知识点，其他的诸如：Linux、数据结构、算法、逻辑思维题、系统设计题、职业规划等等都会有所涉及！总之，充分的准备是应对一切面试的不二法宝！