1.java 基础（完）M

1)面向对象的特性

封装：除了需要调用的方法外，一些内容封装在类里不暴露给外面

继承：一个对象可以继承另一个对象

多态：允许将父类实现成多个不同的子类（接口，abstract 类）

抽象：忽略与当前目标无关的方面。

2)hashmap 容量为什么是 2 的幂次？

方便取模和扩容，这样可以直接进行位操作计算

3)了解 Hashmap 吗（说了下），你知道 JDK1.8Hashmap 有什么改动吗（引入红黑树，仔细说了一下）

Hashmap 实现了 map 接口，用来用关键字存取数据。

内部结构是 hash+链表，1.8 以后当超过 8 时转化为红黑树，当小于 6 时重新变为链表。

默认大小 16，默认负载因子 0.75，即超过 12 就开始 rehash

Put 的话先查询 key 的 hashcode，然后高位 16 位和低位 16 为异或运算，最后取模得到 hash 桶的位置

取得位置后进行插入，如果桶是空的则直接插入，不为空则判断知否相等（key），key 不相等则判断是树还是链表，链表还要判断插入后是否超过 8 了，超过 8 则修改为红黑树。插入完成后看是否超过了容量，超了则扩容。

扩容就是对每一个桶中的数据进行遍历，每个 key 计算新的位置并插入。

Hashmap 会有多线程问题，当两个线程同时开始扩容时有可能出现环形链表

http://blog.csdn.net/hhx0626/article/details/54024222

4)hashset 的源码

Hashset 底层是 hashmap 只存储 key，value 为一个通用的 object，要求 key 不重复。

5)深克隆浅克隆

浅克隆是 object 自带的克隆，将会新建一个对象，然后原样拷贝其中的内容。

浅克隆对于基本类型的数据没问题,但是对于对象引用由于复制的是引用所以复制出来的对象跟原对象的引用是相同的。

深克隆一种办法是需要复制的子对象也实现 cloneable 接口。

另一种是使用序列化或者 json 来序列化反序列化出一个新的对象。

6)Java 初始化顺序

父静态变量，父静态代码块，子类静态变量，子类静态代码块，父类非静态变量，父类非静态代码块，父构造器，子类非静态变量，子类非静态代码块，子类构造函数

7)作用域

Public 谁都能，private 自己能，protected 自己、同包、子类都能，默认只有自己和同包能

8)静态内部类实现的单例模式是否有线程安全问题（effective java 推荐的那种方法）

没有，因为 jvm 内部会给类的初始化加锁，只能有一个线程进入初始化逻辑

9)Java IO 模型

IO 主要有 bio，nio，aio，分别为阻塞 io，非阻塞同步 io，非阻塞异步 io

Bio 每次新建立一个连接都要新建一个线程，高并发下消耗大。

Nio 使用 buffer 和 channel 交互读写数据，通过 channel 传输，selector 管理连接

Aio 为真正的异步非阻塞 io

10)Channel 和 buffer

Channel 是用来传输数据的

Buffer 是一块内存区域，可以在这个内存区域中进行数据的读写

11)String,Stringbuffer,StringBuilder 区别

String 拼接会新建 String，Stringbuffer 可以直接拼接，线程安全，StringBuilder 非线程安全，更快。

12)directBuffer 和 buffer 的区别

Directbuffer 内存分配在 jvm 堆之外，buffer 在 jvm 之内，direct 在系统 io 时消耗小。

https://segmentfault.com/a/1190000006824155

13)nio 和 aio 的区别

Nio 是同步非阻塞，aio 是异步非阻塞。

14)同步和异步，阻塞和非阻塞的区别

同步：执行一个操作之后，等待结果，然后才继续执行后续的操作。

异步：执行一个操作后，可以去执行其他的操作，然后等待通知再回来执行刚才没执行完的操作。

阻塞：进程给 CPU 传达一个任务之后，一直等待 CPU 处理完成，然后才执行后面的操作。

非阻塞：进程给 CPU 传达任我后，继续处理后续的操作，隔断时间再来询问之前的操作是否完成。这样的过程其实也叫轮询。

https://www.cnblogs.com/Anker/p/5965654.html

15)Error 和 exception 的区别

Error 是程序本身无法解决的错误，如崩溃等，发生了这些错误程序要终止，exception 是程序可以处理的异常

Exception 还分为运行时异常和受检异常

运行时异常是程序员造成的异常，就算有异常也可以编译通过，受检异常一般是外部异常，如果不 trycatch 或者 throw 则编译会报错。

16)Java9 的特性

http://www.importnew.com/24528.html

17)定时任务怎么实现，并在分布式系统下运行良好

http://blog.csdn.net/xinyuan_java/article/details/51602088

18)优先队列实现关注，具体详细说一下。对耗时的敏感要求比较高吗？

优先队列介绍：

http://www.importnew.com/6932.html

具体实现：堆排序

http://blog.csdn.net/dy5623405/article/details/51487390

19)什么时候发生浮点数精度丢失，存储 0.1 会不会发生精度丢失

0.1 会丢失，浮点数是通过二进制表示的，无法用二进制准确表示的都会丢失，解决方法是用 bigdecimal

https://www.cnblogs.com/kniught-ice/p/4755122.html

20)拆装箱

http://www.importnew.com/18346.html

21)object 的常用方法

Clone：克隆

Getclass：用于反射

Equals：判断相等

Hashcode：hash 值

Tostring

Wait notify notifyall

Finallize

22)Java 的四种引用

http://www.cnblogs.com/huajiezh/p/5835618.html

https://www.zhihu.com/question/37401125

23)手写 java 的 socket 编程，服务端和客户端

https://segmentfault.com/a/1190000006824155

24)八种基本数据类型的大小以及他们的封装类

http://www.runoob.com/java/java-basic-datatypes.html

https://www.cnblogs.com/manhuidhu/p/6639605.html

25)自己定义的类如何实现 hashCode 方法

http://blog.csdn.net/wfg18801733667/article/details/52370996

26)ConncurrentHashMap 和 hashtable 比较

两者都是线程安全的，但是 hashtable 每次操作都得锁住整个类，conncurrenthashmap 不需要。

27)java 是如何实现跨平台的。

Java 程序运行在 jvm 上，而 jvm 能跨平台所以 java 能跨平台。

28)iterator 迭代器用法

调用 itarator

29)重载和重写的区别

重载是相同方法名不同参数可同可不同返回值类型

重写被父类限定了方法名参数和返回值类型，是继承的表现

30)comparable 与 comparator

http://blog.csdn.net/u013256816/article/details/50899416

31)treemap 的实现原理

Map 的实现，key 可排序，底层是红黑树

https://www.cnblogs.com/skywang12345/p/3310928.html

32)JDK1.8 新特性（lambada 表达式啊 JUC 下 Fuction<T,R>，，于是有了 13 题）

Lambda 表达式，使用函数式接口（function）

流处理，可以更加方便的遍历数组

http://www.importnew.com/11908.html#defaultAndStaticMethod

33)数组怎么在内存中存储的

https://www.cnblogs.com/chenpi/p/5489732.html

34)Collections.sort 排序内部原理，快排（回答出来了继续问优化）

排序用 comparable 或者 comparator

http://blog.csdn.net/xx326664162/article/details/52227690

35)知道 java 序列化吗

Java 序列化用在互联网传输数据，本地持久化数据等场景使用

先序列化再反序列化，要保证序列化 id 一致。

https://www.cnblogs.com/wxgblogs/p/5849951.html

36)为什么 hashmap 要用红黑树而不是平衡二叉树

平衡二叉树对于平衡要求高，所以对于时常插入删除的数据结构会很耗时。

https://www.zhihu.com/question/30527705

37)Integer 的常量缓存池的问题(-127~128 范围有个 cache，所以会出现范围内==结果为 true，范围外为 false 的奇怪现象)

38)nio 中 selector 中的 wakeup 什么含义

http://developer.51cto.com/art/201112/306359.htm

39)arraylist.sort 怎么实现的（这个可以看看 TimSort 的思想）

用的 arrays.sort 最终调用的 timsort.sort

40)怎么看待 java 跟 c++（说下区别和自己的感受）

去除指针，没有多重继承，不用自己内存管理

http://developer.51cto.com/art/201106/270422.htm

41)Arraylist 和 Linkedlist 的区别

Arraylist 底层是数组，linkedlist 是链表

42)set 接口实现类有哪些，HashSet 和 TreeSet、LinkedHashSet 区别，TreeSet 如何保证有序

Hashset 底层是个 hashmap，统一存的 value 是个空 object

Linkedhashset 除了像 hashset 以外还维护了个链表保存插入顺序

Treeset 是排序的 set，基于 treemap，treemap 基于红黑树

43)Map 接口实现类，HashMap，TreeMap，WeakHashMap、ConcurrentHashMap

http://blog.csdn.net/yxy000/article/details/70214090

Hashmap hash+链表++红黑树

Treemap 有序 红黑树

Weakhashmap 弱引用，gc 时删除

Concurrenthashmap 线程安全的 hashmap

44)checked unchecked 虚拟机原理怎么做。

Checked 需要编程时抛出和处理，unchecked 不需要

https://www.cnblogs.com/fzqm/p/6803680.html

45)Collections.sort 函数 jdk7 和 jdk8 分别怎么实现的

使用的 timsort（归并）

https://segmentfault.com/a/1190000006253668

46)CopyOnWriteList 底层是什么，适用的情况，vector 的特点，实现的是 List 接口吗。

add 加锁，读不加，适用读少写多需要线程安全的场景，vector 全部方法都加了 sycronized，实现的是 list 接口

47)NIO 除了可以让开发者使用本地内存之外还有什么优势

相比 bio 不用每次新来连接就要新建线程处理。

48)ArrayList 扩充问题.add()方法的底层实现。

Add 直接将数组尾设为值就行，要判断是否够长，不够长扩容先扩为 1.5 倍，不够了就直接设为需要的值。

49)Map,Collection 的关系。

Collection 是数组，map 是 key value

50)final finalize finally

Final 类无法继承，final 方法无法重写，final 变量初始化后无法改变。

Finalize 在垃圾回收的时候会自动调用，用来作资源回收

.Gc（）是建议虚拟机在此时进行垃圾回收，jvm 自己斟酌

Finally 和 trycatch 组合，就算出现了异常也能得到调用

https://www.cnblogs.com/lanxuezaipiao/p/3440471.html

51)用堆实现，那每次 get put 复杂度是多少（lgN）（思想就是并不一定要按优先级排队列的所有对象，复杂度太高，但每次保证能取最大的就行，剩下的顺序不用保证，用堆调整最为合适）

15)JDK1.8 的改进

lambda 表达式、HashMap 升级、ConcurrentHashMap 升级、synchronized 升级），此处没有答出 枚举器的升级（可能是我搞错了），增加了 Iteable.forEach 方法，Iterator.remove()现在有一个默认的，会抛出异常的实现，lambda 表达式的意义

2.多线程（完）M

1）线程的几种实现方式

继承 Thread 类，实现 runnable，实现 callable

2）线程 block 和 wait 状态的区别

Block 无需唤醒，其他线程离开区域就会解锁，wait 需要唤醒

3）死锁条件和如何解决死锁

1.互斥条件：一个资源每次只能被一个线程使用。图上每条路上只能让一个方向的汽车通过，故满足产生死锁的条件之一

2.请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。可以看出，图上每个方向的汽车都在等待其他方向的汽车撤走，故满足产生死锁的条件之二

3.不剥夺条件：进程已获得的资源，在未使用完之前，不能强行剥夺。这里假设没有交警，那么没有人能强行要求其他方向上的汽车撤离，故满足产生死锁的条件之三

4.循环等待条件：若干进程或线程之间形成一种头尾相接的循环等待资源关系。这个在图中很直观地表达出来了

解决就是打破上面条件任意一个

http://blog.csdn.net/dream_18/article/details/52645237

4）那如何安全遍历时修改（CopyOnWriterArrayList）

5）锁的膨胀过程，Synchronized 和 Lock 的区别，底层的 monitor 实现和 unsafe 类的 CAS 函数，参数表示什么，寄存器 cpu 如何做）

区别：synchronized 等待锁会一直等待，lock 等待可以设置一定时间后放弃等待，sychronized 适合竞争少的，lock 适合竞争多的，然而现在 java 已经将 synchronized 优化的跟 lock 差不多了。

锁膨胀：锁对象用对象头的 markworld，中间有两位的标志位用来表示当前锁的状态，一开始是 01，是否为偏向锁：0。当首次有获取操作时开启偏向锁，偏向锁设置 1，当其他线程获取偏向锁时，如果是自己持有锁则直接进，如果不是自己，先检查标志的线程是否为持有锁的状态，不是则 cas 改变为自己持有偏向锁并继续，如果对方持有锁则锁膨胀为轻量级锁，原来持有偏向锁的持有轻量级锁，markworld 存储指向线程的指针，其他线程获取锁先 cas 修改，如果成功了获取锁，失败了自旋，自旋到一定次数升级为重量锁

https://www.cnblogs.com/dsj2016/p/5714921.html

https://www.zhihu.com/question/57774162

5）实现多线程的生产者消费者模型

使用 blockingqueue

http://blog.csdn.net/nickjun123/article/details/47377557

52)并发的问题，线程间通信三种方式

http://ifeve.com/thread-signaling/

6）阻塞队列：

Arrayblockingqueue：需要设置大小的队列

Linkedblockingqueue：不需要设置大小，默认 int 最大值

Priorityblockingqueue：按照值排序处理，其他同上面那个

Synchronizedqueue：没有大小，插入了就阻塞直到有人取

7）了解 AQS 吗，看过源码吗

Aqs 为 reentrantlock，readwritrlock 等的基础类

Reentrantlock：分为公平锁和非公平锁

队列逻辑：先将节点加入队列尾部，然后不停自旋获取锁，查看自己是否为队列头，是就获取锁。

非公平锁：获取的时候先尝试获取一次，失败了先检查锁是否被持有，没有则再试着获取一次，失败了再判断自己是否当前持有锁的对象，是则重入，否则进入队列逻辑，非公平每次自旋都会尝试获取锁

公平锁：如果队列为空则先尝试获取锁，否则如果自己是持有锁的线程则重入，否则进入队列逻辑。

http://www.infoq.com/cn/articles/jdk1.8-abstractqueuedsynchronizer

8）Wait notify 实现阻塞队列

https://www.jianshu.com/p/99b7ef411988

9）有哪些线程安全的 map

Concurrenthashmap concurrentskiplistmap hashtable

10）ConcurrentHashMap jdk7 和 jdk8 的区别

http://www.importnew.com/23610.html

11）ConcurrentHashMap 的 size()怎么做的(并没有完全加锁,而是先乐观的认为不会有写,通过 modCount 判断是否更改)

1.7Size 是连续统计两次各 segment 个数的和，一样则直接返回，不一样则加锁再计算一次。

1.8 有一个 basecount 用来存储总数，各操作通过 cas 来操作这个值，当两个线程同时修改这个值，时发现冲突的线程将该变量存在 countercells 中，最终计数时用 basecount 加每个 countercells 的值。

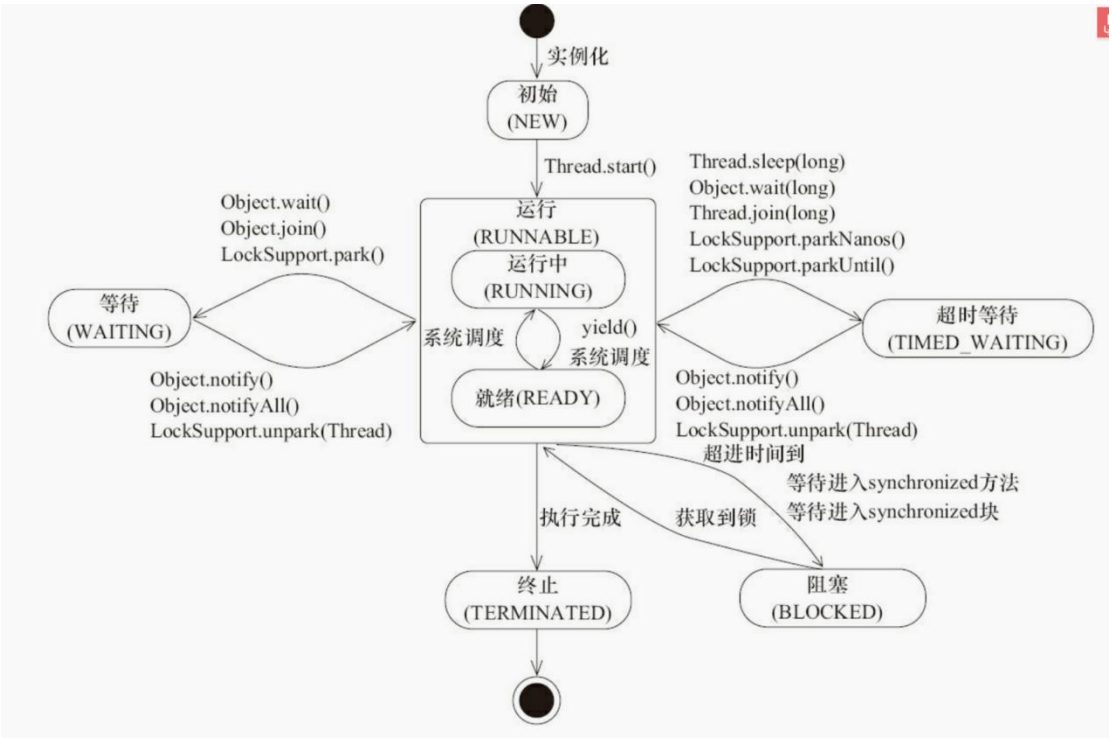
https://www.jianshu.com/p/e694f1e868ec

12）Concurrentlinkedqueue

http://www.baoding-soft.com/news/software/748.html

https://www.jianshu.com/p/7816c1361439

13）线程状态切换（重点说了说 interrut 不会抛出异常，被 wait 会抛出异常并重新设置中断状态为 false，所以如果因为其他操作导致了终端异常需要中心把中断状态置为 true）



14）解释一下 timed_waiting 状态

调用 sleep wait（long）等方法这些带时限的

15）线程中断

http://blog.csdn.net/canot/article/details/51087772

16）线程池的作用

降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。

提高响应速度。当任务到达时，任务可以不需要等到线程创建就能立即执行。

提高线程的可管理性。

17）并发和并行的区别

并行是指两个或者多个事件在同一时刻发生;而并发是指两个或多个事件在同一时间间隔发生。

并行是在不同实体上的多个事件，并发是在同一实体上的多个事件。

在一台处理器上“同时”处理多个任务，在多台处理器上同时处理多个任务。如 hadoop 分布式集群

18）线程异常处理

方法一：子线程中 try... catch...

方法二：为线程设置“未捕获异常处理器”UncaughtExceptionHandler

方法三：通过 Future 的 get 方法捕获异常（推荐）

https://www.cnblogs.com/yangfanexp/p/7594557.html

19）要你设计的话，如何实现一个线程池

核心参数：

Corepoolsize：核心线程数，不会被回收

Maximumpollsize：最大线程数，当任务数超过核心+阻塞队列中等待数时，还能安排最大线程数-核心数的线程，但是超过核心数的线程过段时间会回收。

Keepalivetime：多出来的线程多久回收

Workqueue：用于存储任务的队列

Threadfactory：用来创建线程

Handler：拒绝策略

常用：

FixedThreadPool：需要指定大小的阻塞队列，core 和 max 相同，队列非常大。

SingleThreadExecutor：core 和 max 为 1，其他同上

CachedThreadPool：core 为 0，max 很大，队列容量为 0，说白了来了就建不存

http://www.importnew.com/19011.html

还有个 ScheduledThreadPool

用的 DelayedWorkQueue

http://www.importnew.com/20453.html

20）Synchronized monitor

https://www.cnblogs.com/ygj0930/p/6561667.html

21）Synchronized 用法

Synchronized 方法前：静态方法锁类，非静态锁实例

代码块：this 锁实例，this.class 锁类

http://blog.csdn.net/luoweifu/article/details/46613015

22）ForkJoinPool 是怎么实现的

https://www.cnblogs.com/senlinyang/p/7885964.html

23）为什么匿名内部类的变量必须用 final 修饰，编译器为什么要这么做，否则会出现什么问题

因为会重新拷贝一份，如果不 final，内外会不一致

http://cuipengfei.me/blog/2013/06/22/why-does-it-have-to-be-final/

24）并发编程需要遵守哪几个性质-可见性，原子性，有序性

http://blog.csdn.net/claram/article/details/51683881

25）线程的 sleep 方法和 object 类的 wait 方法有什么区别

Sleep 是阻塞，并没有放弃锁，只是放开 cpu 给其他人运行，而 wait 会放弃锁进入等待队列，等待其他线程 notify 时才会继续争抢锁。

26）线程池有哪些类型，single cached 和 scheduled 的各应用场景

Single 只允许一个线程运行，cached 来多少开多少，scheduled 是定时执行

27）为什么要用线程池

方便管理线程，线程复用

28）Threadlocal

在 threadlocal 类中有一个 map 结构的 threadlocalmap，每个线程自带一个 threadlocalmap 对象，所以每个线程有各自的独享数据。

http://www.importnew.com/20963.html

30）CountDownLatch,CycleBarrier,Semaphore,Exchanger

Countdownlatch:所有线程等待到特殊地点后放开

Cyclebarrier：可重用的 countdown

Semaphore：流量控制，只许一部分线程进入

Exchanger：线程间交换数据

31）IO 密集和 CPU 密集两种情况下，线程池里的线程数应该怎么设

Io 密集由于都在 io，所以线程数设置多些，cpu 密集都在调用 cpu，所以设置跟 cpu 核心数相似。

3.设计模式(完)M

1）抽象工厂和工厂方法模式的区别

工厂模式有一个抽象的单个产品的生产，其他工厂实现这个抽象，抽象工厂是一个工厂抽象生产多个产品，其他工厂实现（生产多个产品）

https://www.zhihu.com/question/20367734

2）哪里用到了工厂模式

Spring ioc 用了

3）双重检验的可见性问题

New 本身存在三个命令 1.为对象分配内存 2.初始化实例对象 3.把引用指向分配的内存空间

指令有可能会重排序，有可能 132，即先指向内存空间而未初始化，所以其他线程有可能检查后发现非空而获取到未初始化完成的对象。

4）各类单例模式

懒加载：判断为空就创建，不然不建（线程不安全）

线程安全懒加载：实例方法直接加锁

饿汉模式：类加载时就初始化

静态内部类：在静态内部类设置实例，第一次被访问时才实例化

双重检验：先检测为空，然后获取锁，然后再确认一遍，最后实例化。

枚举：将该类实现为枚举。

5）观察者模式：

https://www.cnblogs.com/luohanguo/p/7825656.html

6）适配器模式

https://www.cnblogs.com/V1haoge/p/6479118.html

7）代理模式：

https://www.cnblogs.com/cenyu/p/6289209.html

8）装饰者模式：

https://www.cnblogs.com/panhouye/p/6120232.html

4.JVM（完）M

1）了解 Java 内存模型吗？（基于 JSR133 说了一下 JMM，内存重排序，happens before 什么的 刚要说四种内存屏障又被略过去了）

程序在运行时是先将数据读入 CPU 缓存，修改了再写回去，在 cpu 缓存操作的这一段时间其他线程是看不到的，导致数据不一致。这种问题需要 volatile 关键字解决，volatile 通过 happens-before 原则保证可见性

当一个共享变量被 volatile 修饰时，它会保证修改的值会立即被更新到主存，当有其他线程需要读取时，它会去内存中读取新值。

Happens before：

①程序次序规则：一个线程内，按照代码顺序，书写在前面的操作先行发生于书写在后面的操作

②锁定规则：一个 unLock 操作先行发生于后面对同一个锁的 lock 操作

③volatile 变量规则：对一个变量的写操作先行发生于后面对这个变量的读操作

④传递规则：如果操作 A 先行发生于操作 B，而操作 B 又先行发生于操作 C，则可以得出操作 A 先行发生于操作 C

⑤线程启动规则：Thread 对象的 start()方法先行发生于此线程的每个一个动作

⑥线程中断规则：对线程 interrupt()方法的调用先行发生于被中断线程的代码检测到中断事件的发生

⑦线程终结规则：线程中所有的操作都先行发生于线程的终止检测，我们可以通过Thread.join()方法结束、Thread.isAlive()的返回值手段检测到线程已经终止执行

⑧对象终结规则：一个对象的初始化完成先行发生于他的 finalize()方法的开始

加了 volatile 关键字的数据会在线程在自己缓存中修改了数据，则会使其他线程中这个数据的缓存失效，再取只能去内存取。

Read，load，use 有序 assign store write 有序

禁止重排序：

http://blog.csdn.net/jeysine/article/details/68950281

2）双亲委派模型

双亲委派就是加载 java 类的时候优先委托自己的父类加载器，如果父类加载器无法加载再依次往下，这样可以防止一些 java 的基本类与项目中的类名重合导致无法调用基础类的问题。

类加载器从上往下包括：

启动类加载器：将 javahome/lib 下的 jdk 基本类加载

扩展类加载器：用于加载扩展类

应用加载器：用于加载项目 classpath 下的类。

自定义加载器：在应用加载器之下，为自己指定的加载器。

3）Java 运行时数据区域？

包括程序计数器：当前执行程序指令所在地址

Java 栈：java 栈中存放的是栈帧，每个栈帧中存放着局部变量表，操作栈帧，运行时常量池的引用，方法返回地址。

本地方法栈：本地方法的栈

堆：存放对象和数组

方法区：存储类，静态变量，类信息，常量池，编译后的代码。

方法区在 jdk1.8 移除，数据存到了堆和 native memory 中

4） 怎样判断是否需要收集？

引用计数法：对象没有任何引用与之关联(无法解决循环引用)

可达性分析法：通过一组称为 GC Root 的对象为起点,从这些节点向下搜索，如果某对象不能从这些根对象的一个(至少一个)所到达,则判定该对象应当回收。

5） 什么可作为 GCRoot 的对象？

1.虚拟机栈(栈帧中的本地变量表)中引用的对象；

2.方法区中的类静态属性引用的对象

3.方法区中的常量引用的对象

4.原生方法栈（Native Method Stack）中 JNI 中引用的对象。

6）垃圾回收机制

Java 在触发了垃圾回收后会进行可达性分析，只要是发现无法被任何 gcroot 连接到的对象就进入了等待回收的队列,如果在第二次队列扫描前对象又链接到对象上了则能“自救”不然则会被回收，没有被回收的对象被从伊甸区移到存活区，太大了或者代数过多了则移动到老年区，不同代回收算法不同。

gc 时各个代达到什么条件会发生迁徙（达到 xx 比例吧 忘了这个真忘了）

Eden 区放不下了开始 minorgc，survivor 区中装不下的对象进入老年代，当对象大小高于一定数值直接进入老年代。

每次 minorgc 存活下来的对象年龄加 1，到一定年龄进入老年代，如果同年龄对象超过了 survivor 空间的一半，则大于等于该年龄的也会进入老年代。

在 minorgc 之前要先看新生代所有对象大小是否大于老年代的空余空间，小于则直接开始 minorgc，大于且允许担保失败且可用空间大于历代回收到老年代大小的平均值也开始 minorgc，不然直接开始 fullgc。

5）回收机制

新生对应复制，eden 对 survivor8：1：1，老年代对应标记清除（会产生碎片），标记整理，老年代对新生代 2：1。

6）垃圾收集器

Serial/old：单线程收集器，标记整理，回收时线程暂停

Parnew：并发收集器，其他跟 serial 没区别

Parallelscavenge：注重高吞吐量（最短总停顿时间），也可以设置最长停顿时间

Cms：注重停顿时间，使用标记删除算法（并发），为并发收集器。

标记分为三个阶段，初始，并发，重新标记，并发清除

初始记录 root 和可直接关联的对象，并发和程序一块运行，重新再标记一次运行中改变的，第一和三阶段都要暂停程序。

Cms 由于清除是并发的，所以需要留一定余量要不清楚阶段垃圾会塞满。塞满了需要用 serialold 再回收一次。由于标记清除算法会产生碎片，碎片过多会频繁触发 fullgc，可以加配置设定进行了几次开始一次碎片整理。

G1：将原本的分区变为区域块，分别用来存储 eden，survivor，old，humongous（巨型对象），回收在区域间进行合并移动等，这样解决了碎片问题。

http://ifeve.com/%E6%B7%B1%E5%85%A5%E7%90%86%E8%A7%A3g1%E5%9E%83%E5%9C%BE%E6%94%B6%E9%9B%86%E5%99%A8/

http://blog.jobbole.com/109170/

7）Tlab

为了防止多线程分配内存时的激烈竞争，线程初始化时申请一个自己专用的 tlab 区在 eden 里，防止线程间竞争

https://www.jianshu.com/p/cd85098cca39

8）JKD1.8 的 JVM 指令集上有什么更新吗对比 1.7

由于 1.8 彻底去除了 pergemspace 所以永久代相关的命令移除了，新加了一些对元空间的配置（在 jvm 内存空间之外，用的是本地内存）

9）finalize 会不会立即出发 GC，finalize 对象复活

不会，连 gc()都不会立即触发，由于虚拟机在回收垃圾前会先看该类是否有重写 finalize 方法，如果重写的 finalize 中恢复了引用则对象复活，然而只会复活一次

https://www.cnblogs.com/Smina/p/7189427.html

10)类加载机制

类加载过程：

1. 加载：读取类的字节流，转化为方法区的数据机构，生成 class 对象
2. 验证：文件格式，元数据，字节码，符号引用验证
3. 准备：为类变量（static）分配内存和设置初始值（零值）
4. 解析：虚拟机将常量池内的符号引用替换为直接引用的过程。解析动作主要针对类或接口、字段、类方法、接口方法、方法类型、方法句柄和调用点限定符 7 类符号引用进行。
5. 初始化：初始静态变量和执行静态快

类加载触发：

使用 new 关键字实例化对象的时候、读取或设置一个类的静态字段的时候，以及调用

一个类的静态方法的时候。

使用 java.lang.reflect 包的方法对类进行反射调用的时候，如果类没有进行过初始化，则需要先触发其初始化。

当初始化一个类的时候，如果发现其父类还没有进行过初始化，则需要先触发其父类的初始化。

当虚拟机启动时，用户需要指定一个要执行的主类（包含 main()方法的那个类），虚拟机会先初始化这个主类。

11)JVM 如何确立每个类在 JVM 的唯一性

双亲委托机制，这样每个类都只会被一个类加载器加载。

类的全限定名和加载这个类的类加载器的 ID

9）如何防止内存泄露，哪些会容易造成

创建很多个对象而且一直存在不销毁导致 jvm 无法回收，会造成堆溢出，调用栈过多会造成虚拟机栈溢出，加载的类过多或是常量过多会造成方法区溢出。

10）cpu100% 怎么排查（我以为问 Jconsole 的使用，其实是想问 linux 性能监测和调优)

http://blog.csdn.net/wuqiqing_1/article/details/52200000

11）OOM 的分析方法

https://blog.csdn.net/tianya846/article/details/38538411

12）对象的的几种生存状态。

可达：从 gcroot 还是能引用到

可恢复：不可达但是还没 finalize，有自救机会

不可达：finalize 还是无法自救，等待回收

13）Full GC 和 Minor GC 区别，及各自的触发条件

https://www.zhihu.com/question/41922036

14）热加载的原理

https://www.ibm.com/developerworks/cn/java/j-lo-hotdeploy/

5.Mysql（完）M

1）范式

1nf：每列都不可再分割

2nf：不存在对主键的部份依赖(如果有两个列的主键，不能只依赖于其中一个)

3nf：其他列都直接依赖于主键，而不是间接依赖

http://blog.csdn.net/xidianliuy/article/details/51566576

2）了解关系型数据库吗？它的事务？隔离级别之类的

数据库事务为同一次业务请求中的多条数据库操作提供 ACID 的特性。

Atomic(原子性）：原子性确保了同一事务中的操作要不然同时执行要不然都不执行。

Consistent（一致性）：一致性保证了数据与其与所期望实现的业务情况相一致

Insulation(隔离性）：隔离性保证了事物在未提交时无法被其他事务看见，事物执行过程中只能看到在事物开始前其他事务提交的数据。

Durability（持久性）：持久性保证了事务完成后对于数据库事物的影响是永久性的。

隔离级别：

未提交读：事务在执行过程中能够查询其他事务为提交的数据，容易造成“脏读”，当读取到其他事务未提交的数据后该事物回滚会导致当前事务读取到不存在的数据。

提交读：提交读解决了脏读的问题，在事务中只能看见已经提交的数据。

可重复读：可重复读不止保证只能读取已经提交的事务数据外，还保证只能读取到当前事务开始前提交的数据，因此在同一事务中重复查询数据会得到同样的结果。Mysql 的可重

复读使用 Mvcc（版本控制）解决了幻读问题（在事务执行过程中其他事务的添加和删除行数据对查询结果带来影响）。

可串行化：利用加锁导致全部的事务操作都遵行绝对的串行执行。

隔离级别越高，越能保证数据的完整性和一致性，但是对并发性能的影响也越大。

提交读解决脏读

可重复读解决不可重复读

顺序读解决了幻读问题。

https://segmentfault.com/a/1190000012650596?utm_source=tuicool&utm_medium=referral

2)乐观锁与悲观锁？怎么实现的……

悲观锁是在对数据进行修改的过程中锁住需要修改的行，避免并发过程中带来的同时修改的问题。在 mysql 中可以在修改数据库对应行的

乐观锁认为数据一般不会出现并发问题，将并发问题交给用户来决定如何处理，使用版本控制（MVCC）来更新数据，每次开启新事务版本号加 1

Mvcc 通过两个隐藏列（创建时间，删除时间）来实现了

开启新事务时：版本号加 1

Insert 新的行时：新建一行，创建时间为当前事务版本号，删除时间为未指定。

Delete 某行时：更新删除时间为当前事务版本号

Update 某行时：更新被修改的当前最新行的删除时间为当前事务版本号，同时添加一个新的行并将创建时间设定为当前事务版本号.

1)Mysql undo 日志和 redo 日志

http://blog.csdn.net/chast_cn/article/details/50910861

2)数据库连接池如何防止失效

Testwhileidle 设置为 true

http://akhuting.iteye.com/blog/1325169

3)索引（包括分类及优化方式，失效条件，底层结构）

单列索引：一个索引只包含单个列，但一个表中可以有多个单列索引。 这里不要搞混淆了。

普通索引：MySQL 中基本索引类型，没有什么限制，允许在定义索引的列中插入重复值和空值，纯粹为了查询数据更快一点。

唯一索引：索引列中的值必须是唯一的，但是允许为空值，

主键索引：是一种特殊的唯一索引，不允许有空值。

组合索引：在表中的多个字段组合上创建的索引，只有在查询条件中使用了这些字段的左边字段时，索引才会被使用，使用组合索引时遵循最左前缀集合。这个如果还不明白，等后面举例讲解时在细说

全文索引：全文索引，只有在 MyISAM 引擎上才能使用，只能在 CHAR,VARCHAR,TEXT 类型字段上使用全文索引，介绍了要求，说说什么是全文索引，就是在一堆文字中，通过其中的某个关键字等，就能找到该字段所属的记录行，比如有"你是个大煞笔，二货 ..." 通过大煞笔，可能就可以找到该条记录。这里说的是可能，因为全文索引的使用涉及了很多细节，我们只需要知道这个大概意思，如果感兴趣进一步深入使用它，那么看下面测试该索引时，会给出一个博文，供大家参考。

空间索引：空间索引是对空间数据类型的字段建立的索引，MySQL 中的空间数据类型有四种，GEOMETRY、POINT、LINESTRING、POLYGON。在创建空间索引时，使用 SPATIAL 关键字。要求，引擎为 MyISAM，创建空间索引的列，必须将其声明为 NOT NULL。

失效：

1. where 用了！=
2. 使用了函数
3. 模糊搜索时使用了非前缀，如“like %abc”
4. 选择性低
5. Or 条件

Mysql 索引底层实现是 b+树

3）MYSQL 的两种存储引擎区别（事务、锁级别等等），各自的适用场景

Myisam

不支持事务、也不支持外键，优势是访问速度快

Innodb

支持外键和事务，但是访问速度慢磁盘占用大

Memory

内存数据库，支持 hash 索引，hash 索引查找一次到位而不是 b+树要经过多层，但是对于范围查找无效

Merge

一组 myisam 数据库的合集，可以对结构相同的一组列表进行查询，如日志

Innodb 和 myisam 区别

http://blog.csdn.net/lc0817/article/details/52757194

4）mysql 的默认隔离级别（可重复读）

5）Mysql 要加上 nextkey 锁，语句该怎么写

事务内执行 select for update 查询范 围值，会将其自身以及范围内的数据全部加锁。在事务提交前该范围内不会出现新值。

http://blog.csdn.net/tb3039450/article/details/66475638?utm_source=itdadao&utm_medium=referral

6）Mysql 的连接池

连接池在项目创建初期初始化一定数量的连接供系统调用，且是可复用的。

连接池大量减少连接创建时间，需要连接的时候可以直接使用而不需要重新建，使用比普通使用更为直接，获取了连接后可直接执行语句，而不需要自己建立。连接池可以控制资源的使用，可以将连接数控制在一定范围内（最大连接数）防止系统崩溃

https://www.cnblogs.com/aspirant/p/6747238.html

7）事务：

在 MySQL 中使用 START TRANSACTION 或 BEGIN 开启事务，提交事务使用 COMMIT，ROLLBACK 用来放弃事务。MySQL 默认设置了事务的自动提交，即一条 SQL 语句就是一个事务。

总结：

事务的（ACID）特性是由关系数据库管理系统（RDBMS，数据库系统）来实现的。数据库管理系统采用日志来保证事务的原子性、一致性和持久性。

日志记录了事务对数据库所做的更新，如果某个事务在执行过程中发生错误，就可以根据日志，撤销事务对数据库已做的更新，使数据库退回到执行事务前的初始状态。

数据库管理系统采用锁机制来实现事务的隔离性。当多个事务同时更新数据库中相同的数据时，只允许持有锁的事务能更新该数据，其他事务必须等待，

直到前一个事务释放了锁，其他事务才有机会更新该数据。

8）Mysql 连接池对比：

C3p0：单线程，性能较差，适用于小型系统

Druid：高可用、高扩展，多线程，异步，适用于大型系统。

http://blog.csdn.net/fysuccess/article/details/66972554

9）对于建立索引的列，数据是均匀分布好还是不均匀分布好

均匀好，不均匀的数据索引选择性差，如果选择的是出现较多的列，则并不能排除很多行

10）联合

多个列索引，要按顺序查

https://www.cnblogs.com/softidea/p/5977860.html

11）mysql 和 Oracle 的隔离级别分别是什么

Mysql 是重复读，oracle 是读提交

12）分布式事务

https://www.cnblogs.com/zengkefu/p/5677931.html

13）count(1)，count(*)，count(col) 的区别

1 和*几乎无区别，col 没有索引则会慢

14）mysql 语句分类

1：数据定义语言（DDL）

用于创建、修改、和删除数据库内的数据结构，如：1：创建和删除数据库(CREATE DATABASE || DROP DATABASE)；2：创建、修改、重命名、删除表(CREATE TABLE || ALTER TABLE|| RENAME TABLE||DROP TABLE)；3：创建和删除索引(CREATEINDEX || DROP INDEX)

2：数据查询语言（DQL）

从数据库中的一个或多个表中查询数据(SELECT)

3：数据操作语言（DML）

修改数据库中的数据，包括插入(INSERT)、更新(UPDATE)和删除(DELETE)

4：数据控制语言（DCL）

用于对数据库的访问，如：1：给用户授予访问权限（GRANT）;2：取消用户访问权限（REMOKE）

15） int 占多少位

Tinyint：1 字节，smallint：2 字节，mediumint：3 字节，int：4 字节，bigint：8 字节

http://wayne173.iteye.com/blog/1631095

16）like%..%为什么会扫描全表？遵循什么原则？

由于左边也加了%所以不会用索引，而是查询全表，如果对前面的东西没有特殊需求的话可以改为。。。%

https://www.cnblogs.com/chaobest/p/6737901.html

17）mysql 存储过程

https://www.cnblogs.com/chenpi/p/5136483.html

18）mysql 建表需要考虑的东西

http://blog.csdn.net/w384694051/article/details/76998112

19）B+树结构，索引选择原则？

1、 表的某个字段值得离散度越高，该字段越适合选作索引的关键字。主键字段以及唯一性约束字段适合选作索引的关键字，原因就是这些字段的值非常离散。尤其是在主键字段创建索引时，cardinality（基数，集的势）的值就等于该表的行数。MySQL 在处理主键约束以及唯一性约束时，考虑周全。数据库用户创建主键约束的同时，mysql 自动创建主索引（primary index），且索引名称为 Primary；数据库用户创建唯一性索引时，MySQL 自动创

建唯一性索引（unique index），默认情况下，索引名为唯一性索引的字段名。

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，因此，较为适合选作索引关键字。

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（最频繁放最左）。

7、 尽量使用前缀索引。

引入索引的目的就是提高数据的检查效率，因此索引关键字的选择与 select 语句息息相关。这句话有两个含义：一是，select 语句的设计可以决定索引的设计；索引的设计也同样影响着 select 语句的设计。例如原则 1 与原则 2，可以影响 select 语句的设计；而 select 语句中的 where 子句、group by 子句以及，又可以影响索引的设计。两个表的连接字段应该创建索引，外键约束一经创建，MySQL 会自动地创建与外键相对应的索引，这是由于外键字段通常是两个表的连接字段。

复合索引还有一个优点，它通过被称为“最左前缀”（leftmost prefixing）的概念体现出来的。假设向一个表的多个字段（例如 fristname、lastname、address）创建复合索引（索引名为 fname_lname_address）. 当 where 查询条件是以下各种字段的组合是，MySQL 将使用 fname_lname_address 索引。其他情况将无法使用 fname_lname_address 索引。可以理解：一个复合索引（firstname、lastname、address）等效于（firstname，llastname，age）、（firstname，lastname）以及（firstname）三个索引。基于最做前缀原则，应尽量避免创建重复的索引，例如，创建了 fname_lname_address 索引后，就无需再 first_name 子段上单独创建一个索引

20）b 树和 b+树的区别

B+树的优点：

1. 非叶子节点不会带上 ROWID，这样，一个块中可以容纳更多的索引项，一是可以降低树的高度。二是一个内部节点可以定位更多的叶子节点。

2. 叶子节点之间通过指针来连接，范围扫描将十分简单，而对于 B 树来说，则需要在叶子节点和内部节点不停的往返移动。

B 树的优点：

对于在内部节点的数据，可直接得到，不必根据叶子节点来定位。

21）mysql 判断是否为空的函数

Ifnull

22）join 的执行计划？mysql 中采用了哪些算法？

Inner join：只有符合连接条件的数据会显示出来，双方表中不符合条件的数据会被过滤掉。

Left join、left outer join:以左边的表为主导，左边表中不符合连接条件的数据也显示出来

Right join、right outer join:以右边的表为主导，右边表中不符合的数据也会显示出来

Full join：不管是否符合的数据都会显示。

按顺序执行

Left+left=最左以及其他符合的

Left+right=最右加其他符合的

Right+right=最右

Right+left=中

主要看从右向左每句话的指向

算法是 Nested Loop Join

https://segmentfault.com/q/1010000007976070/a-1020000007976892

23）模糊查询

https://www.cnblogs.com/muzixiaodan/p/5583473.html

24）数据库挂了怎么办？除了热备份还有什么方法

主从热备

http://blog.csdn.net/huaweitman/article/details/50853075

25）从底层解释最左匹配原则

因为索引使用的是 b+树，对于数据是按照从左至右的顺序比较数值的，所以指定后面的值时数据是分散的，每层划分的时候并不能知道在哪。

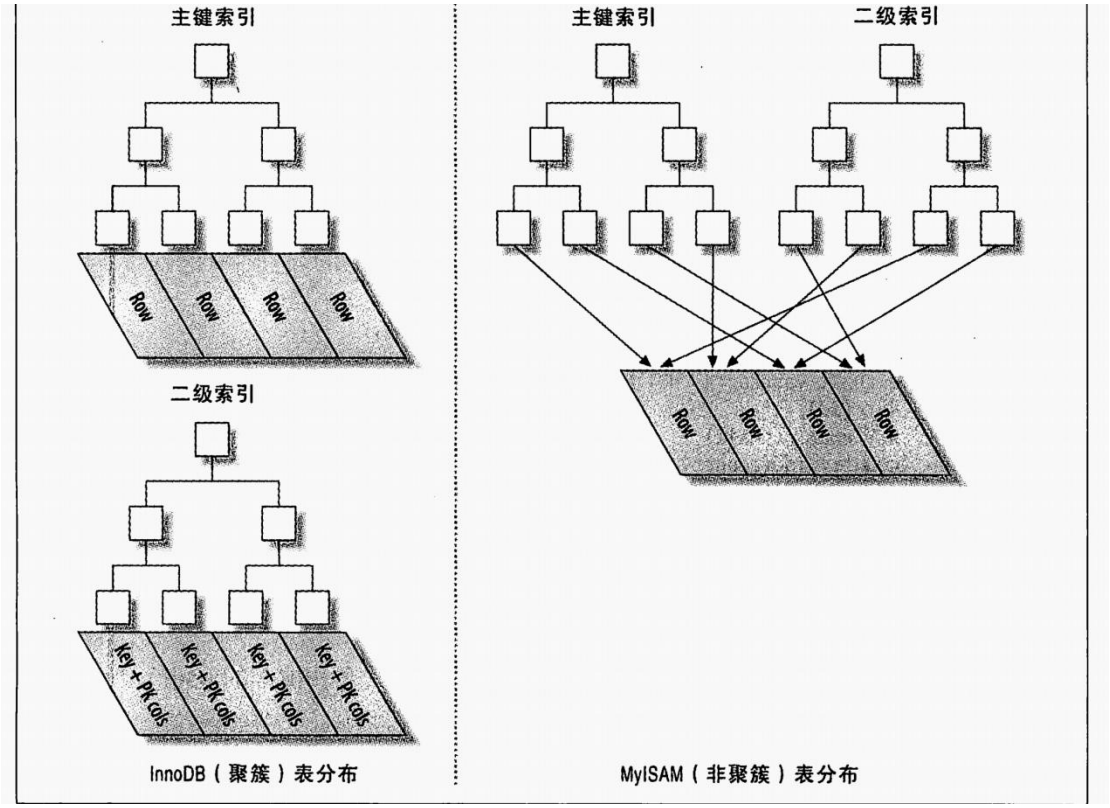
26）数据库连接和 session 会话是一回事儿吗？有什么不同？

Session 对会话是多对一，连接池的情况下 session 关闭会话可能还存在。

http://blog.csdn.net/w13528476101/article/details/78752323

27）非聚簇索引和聚簇索引

使用聚簇索引时主键索引存的是全部值（原来的表），而其他索引叶子结点存储的是主键，非聚簇索引所有叶子节点存的都是列值



http://blog.csdn.net/lisuyibmd/article/details/53004848

28）Mysql、Postgresql 的区别

https://www.cnblogs.com/esther-qing/p/7207369.html?utm_source=itdadao&utm_medium=referral

29）最大的数据量多大，用了索引没有，怎么用的（聊了前缀索引，对于 varchar 类型的值，又聊了聊 char，varchar，text，blob 的关系和区别）。

Char 定长，varchar 不定长，读取上 char 比 varchar 快

Text 和 blob 都是变长

Text 用来存长文本，blob 可以用来存储图片

http://blog.csdn.net/brycegao321/article/details/78038272

30）explain 每个列代表什么含义（关于优化级别 ref 和 all，什么时候应该用到 index 却没用到，关于 extra 列出现了 usetempory 和 filesort 分别的原因和如何着手优化等）

https://www.cnblogs.com/xiaoboluo768/p/5400990.html

31）行锁，哪个引擎支持行锁，行锁的优点

Innodb 有行锁和表锁，myisam 只有表锁

表锁：开销小，加锁快；不会出现死锁；锁定力度大，发生锁冲突概率高，并发度最低

行锁：开销大，加锁慢；会出现死锁；锁定粒度小，发生锁冲突的概率低，并发度高

32）什么是未提交读

未提交读即是事务能读到其他事务尚未提交的内容（脏读）

https://www.cnblogs.com/huanongying/p/7021555.html

33）在 MySQL 存放地理信息位置

http://blog.csdn.net/z69183787/article/details/52857297

34）mysql 如何解决幻读、如何解决不可重复读，了解 MVCC 和 next-key 锁

Mvcc:http://blog.csdn.net/whoamiyang/article/details/51901888

Next-key 解决幻读：

http://blog.csdn.net/tb3039450/article/details/66475638?utm_source=itdadao&utm_medium=referral

35）mysql 主键和唯一索引的区别

Innodb 的主键自带个主键索引，主键索引是一种特殊的唯一索引。

36）mysql 中 b+树的高度

2-3

37）MySQL 和 redis 区别

Redis 结构和功能相对简单，mysql 相对复杂，redis 数据存在内存中并可以持久化，mysql 存在磁盘中。

Redis 存取速度快，mysql 慢

Mysql 适用于逻辑复杂，数据量不太大的数据，redis 适合结构简单但是量大的数据。

38）b 树和 hash 应用场景

Hash 适用于值都不同或是很少重复的等值索引,b 树适合范围索引,多值查询(有重复)，排序等。

https://www.cnblogs.com/heiming/p/5865101.html

39）Mysql 分页关键字

Count，limit

40）数据库备份

https://www.cnblogs.com/SQL888/p/5751631.html

41）Mysql 的优化（高频，索引优化，性能优化）

https://www.cnblogs.com/hepingqingfeng/p/7553428.html

6.算法和数据结构（last）

1)快速排序优化：

三数取中，随机取数，减少递归，规模较小的数组用插入排序。

https://www.cnblogs.com/YuNanlong/p/6115953.html?utm_source=tuicool&utm_medium=referral

2)讲一下稳定的排序算法和不稳定的排序算法

3)各类排序算法以及复杂度分析（快排、归并、堆）

比较排序算法（Comparison Sorts）						
Category	Name	Best	Average	Worst	Memory	Stability
插入排序 (Insertion Sorts)	插入排序 (Insertion Sort)	n	n^2	n^2	1	Stable
	希尔排序 (Shell Sort)	n	$n \log^2 n$	$n \log^2 n$	1	Not Stable
交换排序 (Exchange Sorts)	快速排序 (Quick Sort)	n log n	n log n	n^2	log n	Not Stable
	冒泡排序 (Bubble Sort)	n	n^2	n^2	1	Stable
	鸡尾酒排序 (Cocktail Sort)	n	n^2	n^2	1	Stable
	奇偶排序 (Odd-Even Sort)	n	n^2	n^2	1	Stable
选择排序 (Selection Sorts)	选择排序 (Selection Sort)	n^2	n^2	n^2	1	Not Stable
	堆排序 (Heap Sort)	n log n	n log n	n log n	1	Not Stable
合并排序 (Merge Sorts)	合并排序 (Merge Sort)	n	n log n	n log n	n	Stable
混合排序 (Hybrid Sorts)	内省排序 (Introspective Sort)	n log n	n log n	n log n	log n	Not Stable

https://www.cnblogs.com/gaochundong/p/comparison_sorting_algorithms.html

4)了解链表吗，自己动手写过吗（说了一下链表怎么写，以 CLH 队列为例子说了下双向链表的队列怎么实非递归现

5)二叉树后序遍历

6)如何判断一个树是不是另一颗树的子树

7)股票最长增长区间，优化

8)二叉树相关（层次遍历、求深度、求两个节点距离、翻转二叉树、前中后序遍历）

9)求最大公约数最小公倍数

10)链表相关（插入节点、链表逆置、使用链表进行大数字的加减，双向链表实现队列、寻找链表中的环）

11)堆（大量数据中寻找最大 N 个数字几乎每次都会问，还有堆在插入时进行的调整）

12)图（深度广度优先遍历、单源最短路径、最小生成树）

13)排序（八大排序，各自的时间复杂度、排序算法的稳定性。快排几乎每次都问）
14)识别 2 的 n 次方，写个函数。（最快的是用位操作，大家应该都知道 n&(n-1)可以去掉二进制最右的 1，那 2 的 n 次幂&之后便为 0）
15)二分查找（一般会深入，如寻找数组总和为 K 的两个数字）
16)给定一个数字，一个数组，找出数组中相加等于这两个数的和，不能用数据结构
17)算法，最长递增子序列，一个 dp 数组一个 max 数组，最优情况
18)背包问题
19)数组、链表、二叉树、队列、栈的各种操作（性能，场景）
20)二分查找和各种变种的二分查找
21)动态规划（笔试回回有。。）、贪心。
22)找零钱问题（所有组合）
23)一个只包含小写字母的字符串，去重生成一个只包含单一字母的字符串。例如“abadcab”变成"abdc"，只让用最多一个额外的 int 变量
24)写代码 旋转数组中查找某一个值
25)从 M 个数找到最小的 n 个数
26)用 java 统计一个文本文件中出现的频率最高的 20 个单词
27)红黑树、AVL 树、Hash 树、Tire 树、B 树、B+树。
28)LRU：
http://flychao88.iteye.com/blog/1977653
红黑树:本质上是一颗二叉搜索树，它满足二叉搜索树的基本性质——即树中的任何节点的值大于它的左子节点，且小于它的右子节点。
一颗红黑树必须满足以下几点条件：
规则 1、根节点必须是黑色。
规则 2、任意从根到叶子的路径不包含连续的红色节点。
规则 3、任意从根到叶子的路径的黑色节点总数相同。
作为红黑树节点，其基本属性有：节点的颜色、左子节点指针、右子节点指针、父节点指针、节点的值。
这些约束确保了红黑树的关键特性:从根到叶子的最长的可能路径不多于最短的可能路径的两倍长。结果是这个树大致上是平衡的。
因为操作比如插入、删除和查找某个值的最坏情况时间都要求与树的高度成比例，这个在高度上的理论上限 允许红黑树在最坏情况下都是高效的，而不同于普通的二叉查找树。
在很多树数据结构的表示中，一个节点有可能只有一个子节点，而叶子节点包含数据。用这种范例表示红黑树是可能的，但是这会改变一些性质并使算法复杂。
为此，本文中我们使用"nil 叶子"或"空（null）叶子"，如上图所示，它不包含数据而只充当树在此结束的指示。这些节点在绘图中经常被省略，导致了这些树好像同上述原则相矛盾，而实际上不是这样。
与此有关的结论是所有节点都有两个子节点，尽管其中的一个或两个可能是空叶子。
因为每一个红黑树也是一个特化的二叉查找树，因此红黑树上的只读操作与普通二叉查找树上的只读操作相同。
然而，在红黑树上进行插入操作和删除操作会导致不再符合红黑树的性质。恢复红黑树的性质需要少量（O(log n)）的颜色变更（实际是非常快速的）和不超过三次树旋转（对于插入操作是两次）。
虽然插入和删除很复杂，但操作时间仍可以保持为 O(log n)次

29)图算法（比较少，也就两个最短路径算法理解吧）
30)实现一个优先队列，10 亿个数中找中位数
31)O(1)删除执行链表结点，做分析（其实是要指出剑指 offer 中那个直接 copy 值的方法的缺陷和隐患）
32)二叉树的最长距离（递归的思想）
33)给 200 个 200 个数的数组，找到最大的 200 个
34)海量数据找到出现次数最多的 100 个（内存不足的时候可以先做 hash 分片，最后多路 merge，每次操作可以用 hashMap 计数，也可以自己做 hash 函数计数）
35)100 亿个数找最大 1000 个（说了分片，用堆，再归并）
36)算法题，一个先减后增的数组，查找目标值。（这里并不是查找最值，也不是剑指 offer 上的旋转数组，但是思想上也可以用二分的方式）
37)算法题，两个大数求和，要按高到低位的输入，实时输出结果的对应位，空间 O，时间 O(n)，不借助工具类。（要考虑实时的进位标识，以及多个 9 之后的连续进位标识）
38)算法：int 范围的随机数的阶乘编码实现。（这个题如果直接按最简单的算法题肯定是不行的）
39)二叉树遍历方式，算法实现
40)链表反转
41)手写快速排序与优化
42)二叉树遍历非归并
43)平衡二叉树转化为双向链表？
44)最长公共子序列
45)环状有序数组，中间剪断，寻找其中一个数值
46)一串长字符串，两个一样的数字，寻找其中出现两次的数字。（两种实现方法）
47)二维数组求最大图形面积
48)找到第 k 大的数
49)n 个人中挑选 m 个人
50)左移有序数组的二分查找
51)一道算法题，在一个整形数组中，有正数有负数，找出和最大的子串
52)一道算法题，在一个整形数组中，有正数有负数，找出和最大的子串
53)前缀树
54)敲一个字串匹配问题，写了常规代码。问 kmp 的代码思想，最后问了下正则中用的改进后的 BM 算法。（还有个比较新奇的 Sunday 算法，有兴趣的同学也可以看一下）

7.Oracle(弃）
8.Spring（完）M
1).SpringAop 了解过吗（说了说应用场景和底层实现）
AOP（Aspect-OrientedProgramming，面向方面编程），可以说是 OOP（Object-Oriented Programing，面向对象编程）的补充和完善。
OOP 引入封装、继承和多态性等概念来建立一种对象层次结构，用以模拟公共行为的一个集合。当我们需要为分散的对象引入公共行为的时候，OOP 则显得无能为力。
也就是说，OOP 允许你定义从上到下的关系，但并不适合定义从左到右的关系。例如日志功能。日志代码往往水平地散布在所有对象层次中，而与它所散布到的对象的核心功能毫无关系。

对于其他类型的代码，如安全性、异常处理和透明的持续性也是如此。这种散布在各处的无关的代码被称为横切（cross-cutting）代码，
在 OOP 设计中，它导致了大量代码的重复，而不利于各个模块的重用。
AOP 使用场景：
事务 日志 缓存 权限管理 拦截器
2)Bean 的构造过程，生命周期
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
7.InitializingBean 的 afterPropertiesSet()，如果实现了该接口，则执行其 afterPropertiesSet()方法
8. 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
9. 如果这个 Bean 关联了 BeanPostProcessor 接口，将会调用 postProcessAfterInitialization(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
10. 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
11. 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
3)Spring bean 生命周期前面的过程（总过程）

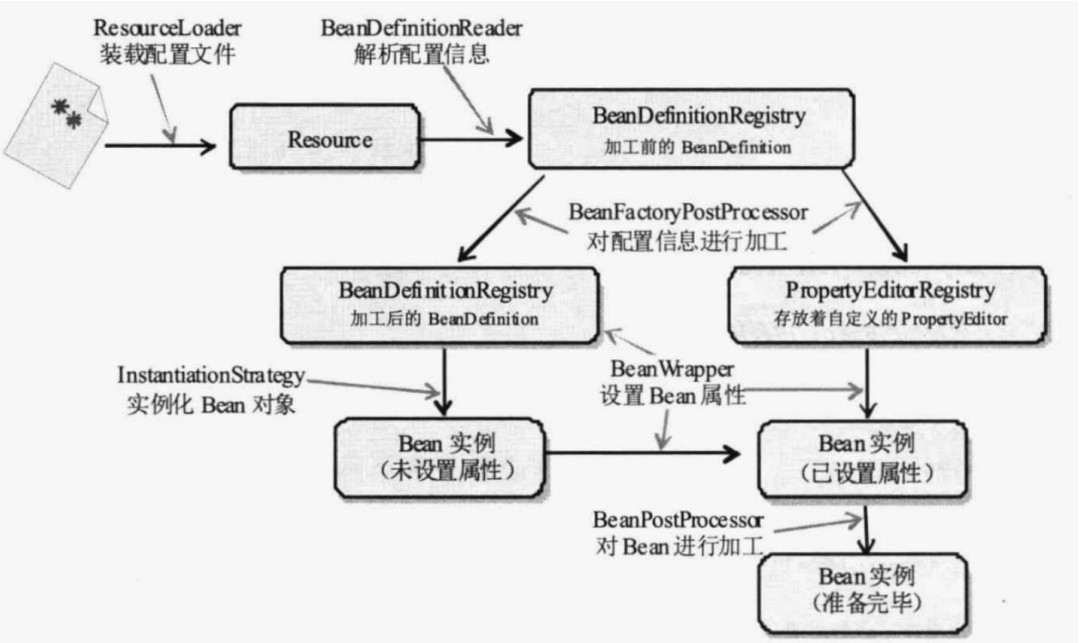


图 6-1　IoC 的流水线

（1）ResourceLoader 从存储介质中加载 Spring 配置信息，并使用 Resource 表示这个配置文件资源。

（2）BeanDefinitionReader 读取 Resource 所指向的配置文件资源，然后解析配置文件。配置文件中的每个<bean>解析成一个 BeanDefinition 对象，并保存到 BeanDefinitionRegistry 中。

（3）容器扫描 BeanDefinitionRegistry 中的 BeanDefinition，使用 Java 反射机制自动识别出 Bean 工厂后处理器（实现 BeanFactoryPostProcessor 接口的 Bean），然后调用这些 Bean 工厂后处理器对 BeanDefinitionRegistry 中的 BeanDefinition 进行加工处理。主要完成以下两项工作：

① 对使用占位符的<bean>元素标签进行解析，得到最终的配置值。这意味着对一些半成品式的 BeanDefinition 对象进行加工处理并得到成品的 BeanDefinition 对象。

② 对 BeanDefinitionRegistry 中的 BeanDefinition 进行扫描，通过 Java 反射机制找出所有属性编辑器的 Bean（实现 java.beans.PropertyEditor 接口的 Bean），并自动将它们注册到 Spring 容器的属性编辑器注册表中（PropertyEditorRegistry）。

（4）Spring 容器从 BeanDefinitionRegistry 中取出加工后的 BeanDefinition，并调用 InstantiationStrategy 着手进行 Bean 实例化的工作。

（5）在实例化 Bean 时，Spring 容器使用 BeanWrapper 对 Bean 进行封装。BeanWrapper 提供了很多以 Java 反射机制操作 Bean 的方法，它将结合该 Bean 的 BeanDefinition 及容器中的属性编辑器，完成 Bean 属性注入工作。

（6）利用容器中注册的 Bean 后处理器（实现 BeanPostProcessor 接口的 Bean）对已经完成属性设置工作的 Bean 进行后续加工，直接装配出一个准备就绪的 Bean。

4) jdk 动态代理和 cglib 代理

Java 动态代理是利用反射机制生成一个实现代理接口的匿名类，在调用具体方法前调

用 InvokeHandler 来处理。

而 cglib 动态代理是利用 asm 开源包，对代理对象类的 class 文件加载进来，通过修改其字节码生成子类来处理。

1、如果目标对象实现了接口，默认情况下会采用 JDK 的动态代理实现 AOP

2、如果目标对象实现了接口，可以强制使用 CGLIB 实现 AOP

JDK 动态代理和 CGLIB 字节码生成的区别？

（1）JDK 动态代理只能对实现了接口的类生成代理，而不能针对类

（2）CGLIB 是针对类实现代理，主要是对指定的类生成一个子类，覆盖其中的方法

因为是继承，所以该类或方法最好不要声明成 final

代理方式	实现	优点	缺点	特点
JDK静态代理	代理类与委托类实现同一接口，并且在代理类中需要硬编码接口	实现简单，容易理解	代理类需要硬编码接口，在实际应用中可能会导致重复编码，浪费存储空间并且效率很低	好像没啥特点
JDK动态代理	代理类与委托类实现同一接口，主要是通过代理类实现InvocationHandler并重写invoke方法来进行动态代理的，在invoke方法中将对方法进行增强处理	不需要硬编码接口，代码复用率高	只能够代理实现了接口的委托类	底层使用反射机制进行方法的调用
CGLIB动态代理	代理类将委托类作为自己的父类并为其中的非final委托方法创建两个方法，一个是与委托方法签名相同的方法，它在方法中会通过super调用委托方法；另一个是代理类独有的方法。在代理方法中，它会判断是否存在实现了MethodInterceptor接口的对象，若存在则将调用intercept方法对委托方法进行代理	可以在运行时对类或者是接口进行增强操作，且委托类无需实现接口	不能对final类以及final方法进行代理	底层将方法全部存入一个数组中，通过数组索引直接进行方法调用

Java 动态代理

http://blog.csdn.net/jiankunking/article/details/52143504

Cglib 代理

https://www.jianshu.com/p/9a61af393e41?from=timeline&isappinstalled=0

5)spring 的上下文和 spring mvc 的上下文之间关系 以及实现

spring.xml 定义的上下文是父上下文，不能调用子容器的上下文，也就是不能调用 springmvc-servlet.xml;

springmvc-servlet.xml 定义的是子类上下问，可以调用父容器 spring.xml 的上下文内容；

action 可以调用 service、dao；

service、dao 不能调用 action；

spring 上下文容器的 aop 不能应用到 action 层，但是可以通过在 springmvc-servlet.xml 中定义来做到类似事情。

6)spring bean 的构造中，有参构造和无参构造有什么区别

无参构造在配置时不需要写每个参数对应的值，而有参构造需要在配置文件中声明对应的值，不然在反射调用构造时会失败。

https://segmentfault.com/q/1010000012511453

7)spring 怎么实现事务的，事务的传播属性

方式一：注解。

方式二：Aspectj AOP 配置事务 。

传播属性常用的有 required，调用方有事务则包裹在该事务内操作，没有则自己建一个事务。

Supports，基本和 required 一致，区别是调用方没事务则自己也不创建事务。

http://blog.csdn.net/bao19901210/article/details/41724355

8)springmvc 的执行流程

1.spring mvc 将所有的请求都提交给 DispatcherServlet,它会委托应用系统的其他模块负责对请求 进行真正的处理工作。

2.DispatcherServlet 查询一个或多个 HandlerMapping,找到处理请求的 Controller.

3.DispatcherServlet 请请求提交到目标 Controller

4.Controller 进行业务逻辑处理后，会返回一个 ModelAndView

5.Dispathcher 查询一个或多个 ViewResolver 视图解析器,找到 ModelAndView 对象指定的视图对象

6.视图对象负责渲染返回给客户端。

http://blog.csdn.net/liangzi_lucky/article/details/52459378

9)springmvc 想要用 HttpServlet 的对象怎么用（在方法参数上加），那是怎么注入进去的，怎么实现的你知道嘛

https://www.cnblogs.com/springsource/p/6728292.html

10)Areturn 前有一个 AOP 拦截，Breturn 前有一个 AOP 拦截，A 调用 B,哪个拦截器先被调用

B

11)spring 特性

Ioc：控制反转，依赖注入

Aop：面向切面编程

依赖注入使得对象的创建并不是通过自己在代码中 new 来创建，这样对于复杂的系统和多层依赖将会非常复杂，而控制反转将这些复杂的工作交给 spring 容器去做处理。

面向切面编程将一些与功能无关具有共性的工作交给 aop 来处理，省去了大量的重复代码。

12)spring 做过文件上传

Multipart 上传

https://www.cnblogs.com/WJ-163/p/6269409.html

13)springIOC 优点

https://www.zhihu.com/question/23277575

不用由去自己管复杂的依赖关系，修改底层实现不用求改上层代码。

14)Bean 注入

属性注入

构造方法注入

工厂方法注入

使用注解的方式注入

https://www.cnblogs.com/wuchanming/p/5426746.html

15)spring 容器里对象默认是单例还是多例

默认为单例

16)Spring 线程安全

通常的 bean 是没有状态的，日常使用只是调用其中的方法，方法内部的数据不存在线程安全问题，而有状态的 bean 可以使用 prototype（取一次实例一个新的）或是 treadlocal 保证线程安全。

17)Spring 容器管理

http://blog.csdn.net/caipeichao2/article/details/39378821

18)Spring 中 autowire 和 resourse 关键字的区别

@Autowired 是默认按照类型装配的 @Resource 默认是按照名称装配的

19)可以给 final 类生成代理

由于 cglib 是生成的子类所以 final 会出错，所以只能使用 java 动态代理。

20)spring 中常用注解？component 和 service 注解的区别？

在功能上是相同的，为了使代码可读性更高，控制层叫 controller，业务层叫 service，普通组件叫 component

21)Spring DAO 层的作用？与直接写 Connection 代码的区别

https://www.cnblogs.com/joenee/p/4903642.html

22)Spring 事务管理和 JDBC 的事务管理的联系？Spring 是否能支持 JDBC 不存在的事物隔离等级，说明理由？(??)

http://blog.csdn.net/zhangyayun0991/article/details/62270196

23)xml 方式和注解方式的区别和使用场景？

https://www.cnblogs.com/iOS-mt/p/6133656.html

注解分散，xml 集中，注解用于 bean 类标识等作用，在 boot 中则是直接用来作为配置文件，xml 在配置一些较为复杂且集中的配置上比较方便，如数据库连接池等。

24)原对象中两个方法，方法 a 与方法 b。使用 spring aop 对该对象进行增强处理，增强处理都为都是输出一条日志。在代理对象的 a 方法中调用 b 方法，会输出几条日志。（答案是一条日志，因为代理对象 a 方法调用的是原对象的 b 方法，而不是代理对象的 b 方法）

25)Spring 和 springMVC 的处理异常问题

项目中 dao 层 service 层 controller 层的异常都往上抛，抛到 DisparcherServlet 更具异常处理类执行，分别有三种：

1. xml 中配置 springmvc 默认的异常处理器
2. 自定义的异常处理器，

https://www.cnblogs.com/shanheyongmu/p/5872442.html

26)springIoC 是干啥的，能解决什么问题？

依赖注入(Dependency Injection)和控制反转(Inversion of Control)是同一个概念。

当某个角色(可能是一个 Java 实例，调用者)需要另一个角色(另一个 Java 实例，被调用者)的协助时，在传统的程序设计过程中，通常由调用者来创建被调用者的实例。

但在 Spring 里，创建被调用者的工作不再由调用者来完成，因此称为控制反转;创建被调用者 实例的工作通常由 Spring 容器来完成，然后注入调用者，因此也称为依赖注入。

不管是依赖注入，还是控制反转，都说明 Spring 采用动态、灵活的方式来管理各种对象。对象与对象之间的具体实现互相透明。

27)有没有了解过开源框架比如 Springboot?

Springboot 约定优于配置，大部分的 maven 配置，配置文件都可以省去，只修改并非默认配置的类即可，节省了开发初期编写配置文件的时间，也是 springcloud 的基础。

28)MVC 框架的实现原理？比如它的 url 怎么映射的……

Url 映射通过扫描所有 requestmaping 注解定义的 bean 来找到对应的 handler 令其运行。

http://blog.csdn.net/scplove/article/details/52294958

29)看过 Springboot 源码吗

没，再见

30)Spring 的 AOP 关于拦截 private 方法一些问题

对于基于接口动态代理的 AOP 事务增强来说，由于接口的方法都必然是 public 的，这就要求实现类的实现方法也必须是 public 的（不能是 protected、private 等），同时不能使用 static 的修饰符。

基于 CGLib 字节码动态代理的方案是通过扩展被增强类，动态创建其子类的方式进行 AOP 增强植入的。由于使用 final、static、private 修饰符的方法都不能被子类覆盖，相应的，这些方法将无法实施 AOP 增强。

31)spring AOP 相关 before after 织入的概念

Before 是前置增强，在方法执行前增强

After 是后置增强，在方法执行后增强。

织人：将切面应用到目标对象并导致代理对象创建的过程

32)讲讲 Spring 中怎么对初始化的 bean 做其他操作。（这里有三种方式，@PostConstruct 注解方式，init-method 的 XML 配置方式，InitializingBean 接口方式）

Postconstruct:

https://www.cnblogs.com/soundcode/p/6367416.html

init-method:

https://www.cnblogs.com/soundcode/p/6367412.html

InitializingBean:

https://www.yiibai.com/spring/spring-initializingbean-and-disposablebean-example.html

33)三种实现上有什么区别（还好看过点源码，其实前两种是一个意思，都是通过反射的方式用 aop 思想实现，可以消除对 spring 的依赖；接口方式是直接调用 afterPropertiesSet 方法，效率更高点。spring 加载 bean 时先判断接口方式，再执行配置注解方式）？

34)如果配置了两个同类但不同 id 的 bean，IOC 容器如何处理

9.Linux（完）M

1）linux 常用命令

https://www.nowcoder.net/discuss/59080

1. ps：查找进程

   -e:显示所有程序

   -f：显示 UID,PPIP,C 与 STIME 栏位。

2. kill：杀死进程

3. killall：相比 kill 是指定 pid 杀死，killall 会杀死所有输入的进程名相似的进程。
4. useradd：添加系统用户
5. usermod：修改用户信息
6. iptables：防火墙
7. nestat：显示网络状态
8. ping：测试网络连通性
9. iptstat:显示 iptables 的运行状态
10. ifconfig：显示网络参数
11. hostname:显示和设置主机名
12. users:显示当前登录系统的所有用户
13. tail:显示文件末尾几行
14. head：显示文件前几行
15. less：从前向后显示全部内容，可前后翻页
16. more：显示内容，只能从后向前
17. cat：显示出所有内容
18. vi：编辑文本
19. mkdir：创建文件
20. rm：删除文件
    1）-f：强制删除
    2）-r：递归删除
21. touch：创建新的空文件
22. rename：修改文件名(批量）
23. mv：移动文件
24. cp：拷贝文件
25. ls：显示目录内容列表
26. chmod：用来变更文件或目录的权限
27. chown：用来变更文件或目录的拥有者或所属群组
28. chgrp：用来变更文件或目录的所属群组
29. stat：显示文件状态
30. grep：文件搜索
31. wc：统计文件的字节数、字数、行数
32. tar：打包和解压
    1）-z:通过 gzip 指令处理备份文件；
    2）-x：从备份文件中还原文件；（解压）
    3）-v：显示指令执行过程；
    4）-f：指定备份文件；
    5）-c：建立新的备份文件；（压缩）
33. gzip：压缩
34. gunzip：解压
35. vmstat：虚拟内存状态
36. ln：为源文件创建连接：
    1）-s：软连接
37. exprot:设置环境变量
38. find：搜索文件

1）-name：搜索文件名

2）-path：搜索路径

2）pid 和 ppid 的区别

1、PID（process ID）：

PID 是程序被操作系统加载到内存成为进程后动态分配的资源。

每次程序执行的时候，操作系统都会重新加载，PID 在每次加载的时候都是不同的。

2、PPID（parent process ID）：PPID 是程序的父进程号。

3）Linux 内核相关（select、poll、epoll）

Bio 是一个线程对一个连接欸，每个线程都是阻塞的，创建大量内存对 cpu 和线程上下文切换都是问题，select 和 poll 是单线程处理全部连接，当没有连接就绪时处于阻塞状态，有就绪则开始轮询全部连接找到就绪的处理。Epoll 则是更进一步，会在有连接就绪时直接知道是哪个，省了轮询的过程。

https://www.cnblogs.com/wanghuaijun/p/6914810.html

4）linux 怎么查看网络状态（vmstat）

netstat -anp 所有监听端口及对应的进程

netstat -tlnp 功能同上

5）硬连接和软连接区别

硬连接： 与普通文件没什么不同，inode 都指向同一个文件在硬盘中的区块，无法针对文件夹添加硬链接

软连接： 保存了其代表的文件的绝对路径，是另外一种文件，在硬盘上有独立的区块，访问时替换自身路径。

Inode 是用来存储文件真正存储位置的，只要还有指向 inode 的个数就不会删除文件，硬链接是两个连接都指向同 inode，软连接会自动转化为绝对路径，所以软连接原文件删除的时候文件就删除了。

https://www.jianshu.com/p/dde6a01c4094

6）查看内存使用

Atop htop ps

https://www.cnblogs.com/zhuiluoyu/p/6154898.html

7）管道的使用 |

管道可以将诸如 ps 等打印出来的显示文本进行过滤输出

8）grep 的使用，一定要掌握，每次都会问在文件中查找

http://man.linuxde.net/grep

9）shell 脚本

鸽了

10）find 命令

http://man.linuxde.net/find

11）awk 使用

https://www.cnblogs.com/ginvip/p/6352157.html

12）linux 用过的命令

cd,mkdir,vim,mv,cp,tar,ps,grep,netstat,ftp,tomcat

13）将当前目录下所有以“.txt”结尾的文件打印出来，再追问，除了“.txt”再加上“.abc”结尾的也打印出来。

Find *.txt

10.Tomcat(完)M

1) tomcat 类加载有什么不同，说加载顺序并不是双亲模型，具体顺序说一下

Common 加载器加载 common 文件夹下的类，该类库可被所有 web 应用访问

Catalina 加载器加载 server 文件夹下的，只能被 tomcat 使用

Shard 加载器加载 shard 文件夹中的类，只能被 web 应用使用，tomcat 用不了

存在 webapp 目录下的类被各个 webapploader 加载。

先在本地缓存中查找是否已经加载过该类(对于一些已经加载了的类，会被缓存在 resourceEntries 这个数据结构中)，如果已经加载即返回，否则 继续下一步。

让系统类加载器(AppClassLoader)尝试加载该类，主要是为了防止一些基础类会被 web 中的类覆盖，如果加载到即返回，返回继续。

前两步均没加载到目标类，那么 web 应用的类加载器将自行加载，如果加载到则返回，否则继续下一步。

最后还是加载不到的话，则委托父类加载器(Common ClassLoader)去加载。

最后两步是违反双亲模型的，会先自己加载类，找不到了再去找父加载器（common）。

https://www.jianshu.com/p/a18aecaecc89

2) Tomcat 结构

https://www.cnblogs.com/zhouyuqin/p/5143121.html

3) Tomcat 加载顺序

第一阶段加载 jvm 需要的类和扩展类，环境变量 classpath 中的 jar，class，第二阶段加载 common，server，shaed 的类，最后加载 class 文件夹内的类和 lib 文件夹下的类。

http://shuhucy.iteye.com/blog/1900231

4) Tomcat 配置

http://blog.csdn.net/qq_29028175/article/details/53363738

5) Tomcat 崩溃

启动定时任务监视 tomcat 运行情况，如果挂掉（进程 id 不存在，测试页面返回状态码不是 200）则重启

https://www.cnblogs.com/ddxueyu/p/6209932.html

11.操作系统(完)M

1）线程进程的区别

从操作系统的角度来说，在引入线程之前，进程作为系统资源分配和调度的最小单位，在引入线程后，进程不再作为调度的最小单位，只作为系统资源的分配的最小单位，线程开始作为系统调度最小单位，线程共享进程的资源。

进程的切换需要加载上下文，执行，保存上下文，而线程由于共享了进程的资源和地址空间，所以省去了很多上下文加载时间

http://blog.csdn.net/yaosiming2011/article/details/44280797

a，地址空间

进程之间地址空间不同，同一个进程中的线程共享该进程的地址空间

b，资源分配

进程间的资源彼此独立，线程间共享

c，创建，销毁，上下文切换开销

线程之间可以互相创建和撤销，进程间是彼此独立的，进程的上下文切换开销比线程要大。

2）缺页中断知道吗？

程序运行时每个进程会分配虚拟的页表以及真实的物理地址，其中虚拟内存的页表比真

实能够分配的物理地址要大，用来使操作系统能够运行比真实物理地址更多的进程，然而由于虚拟地址并不能全部映射到物理地址，所以有一部分虚拟内存页表是分配不到物理地址的，当 cpu 在运行时调用了这类进程时，由于找不到对应的物理地址就造成了缺页中断，需要将一些当前没在使用的进程的物理地址中的数据写入磁盘，然后将虚拟内存页表映射到刚才清出来的物理地址中。

http://blog.csdn.net/u011080472/article/details/51206332

4)进程通信 IPC（几种方式），与线程区别

1. 管道 pipe：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

2. 命名管道 FIFO：有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

4. 消息队列 MessageQueue：消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

5. 共享存储 SharedMemory：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

6. 信号量 Semaphore：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

7. 套接字 Socket：套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

8. 信号（singnal）： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

https://www.cnblogs.com/LUO77/p/5816326.html

5)OS 的几种策略（页面置换，进程调度等，每个里面有几种算法）

页面置换：

最佳置换法，每次发生缺页中断，都将接下来最长时间不用的页面置换出去，是理想的置换法无法实现

FIFO（先进先出算法）：当超出上限时将最加进来的页面置换出去，这样容易导致一些经常使用的也被换出去。

LRU（最近最久未使用）：每次将当前列表中最远没被访问的页换出去。但是需要额外空间存储最近的访问时间。

Clock（时钟置换算法）：将所有页都存在环形链表中，并存一位数字代表是否使用过，开始放入后以及访问时都置为 1，当缺页中断发生时，开始环形扫描链表，找到使用位为 0 的，发现是 1 则置为 0，如果全为 1 则转一圈回来第一个置换掉。

http://blog.csdn.net/a724888/article/details/70038420

调度：

先来先服务：维护队列，按顺序服务。

最短作业优先：耗时的服务放最后，时间越少越前面。

响应比：按照等待时间/耗时排序，耗时进程等待长了会排到前面。

时间轮片：按照队列顺序每一个运行固定的时间。

多级反馈队列：刚进来的排优先度最高的队列进行轮片，当时间片结束后还没完成则往

下一级移动。有优先度高的队列运行完了才等到低的，低优先级运行中又来新任务时间片完了后优先运行新任务。这样新进的先服务，时间长的轮到后面，而且也不会等很长。

http://blog.csdn.net/leex_brave/article/details/51638300

3）抖动知道吗？

因为常访问的进程被换出去了，导致还要换回来

http://blog.csdn.net/qq_32809273/article/details/52830004

4）操作系统死锁

进程间互相抢占资源

http://blog.csdn.net/weiyongle1996/article/details/71511956

5）中断和异常

http://blog.csdn.net/baidu_28312631/article/details/47375209

6）虚拟内存和虚拟地址是什么

所有进程运行时所需要的内存比实际内存要大，为了尽可能多的运行程序，就出现了虚拟内存的用法，程序将需要用的地址映射到虚拟内存中去，虚拟内存和物理地址映射，但不是所有虚拟内存能映射到物理地址，当进程访问到未映射的虚拟内存时，发生缺页中断，需要置换。

7）父子进程、孤儿进程

子进程是父进程的副本，获得了父进程数据空间、堆和栈的副本；父子进程并不共享这些存储空间，共享正文段（即代码段）；因此子进程对变量的所做的改变并不会影响父进程。

如果父进程优先结束了，子进程会被 1 号 init 进程接管。

8）fork 进程时的操作，

Fork 创建出跟原来进程几乎一样的进程，但是会分配新资源。

9）Cpu load 的参数如果为 4，描述一下现在系统处于什么情况

为 4 表示出现问题了，超过 1 表示有任务在等待。

http://blog.csdn.net/scugxl/article/details/77199403

10）锁在操作系统层面如何实现的

https://www.cnblogs.com/lycroseup/p/7486824.html

11）缓存问题，缓存与内存的区别。

http://blog.csdn.net/lansesl2008/article/details/70224079

12）一个进程最多申请多大空间（看机器 cpu 的处理位数看情况）

32 位 4g，64 位 128g

13）怎么保证进程间数据的安全？线程呢？

进程因为各自内存数据不同所以安全，线程要加锁。

14）操作系统磁盘的了解

https://www.cnblogs.com/edisonchou/p/5136649.html

15）32 位操作系统与 64 位操作系统，最大的区别是什么？

http://blog.csdn.net/a141024/article/details/54343950

16）操作系统内存碎片

内存碎片分为内部和外部碎片。

内部碎片是由于单个内存分区比需要分配的内存还大，所以有空余大小。

外存碎片是由于未分配的内存区域太小不足以分配进程需要的内存。如果连续分配容易出现外部碎片，如果将需要的内存分为多个内存块则不容易出现这样的问题。

http://blog.csdn.net/tong5956/article/details/74937178/

12.计算机网络（完）M

1）OSI7 层模型（TCP4 层）每层的协议

TCP、UDP 属于哪个层？　　传输

FTP 在哪个层？　　应用

OSI 分层 （7 层）：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

TCP/IP 分层（4 层）：网络接口层、 网际层、运输层、 应用层。

五层协议（5 层）：物理层、数据链路层、网络层、运输层、 应用层。

每一层的协议如下：

物理层：RJ45、CLOCK、IEEE802.3　　（中继器，集线器，网关）

数据链路：PPP、FR、HDLC、VLAN、MAC　　（网桥，交换机）

网络层：IP、ICMP、ARP、RARP、OSPF、IPX、RIP、IGRP、　（路由器）

传输层：TCP、UDP、SPX

会话层：NFS、SQL、NETBIOS、RPC

表示层：Telnet, Rlogin, SNMP, Gopher

应用层：FTP、Telnet、SMTP、HTTP、WWW、NFS、DNS

每一层的作用：

物理层：通过媒介传输比特,确定机械及电气规范（比特 Bit）

数据链路层：将比特组装成帧和点到点的传递（帧 Frame）

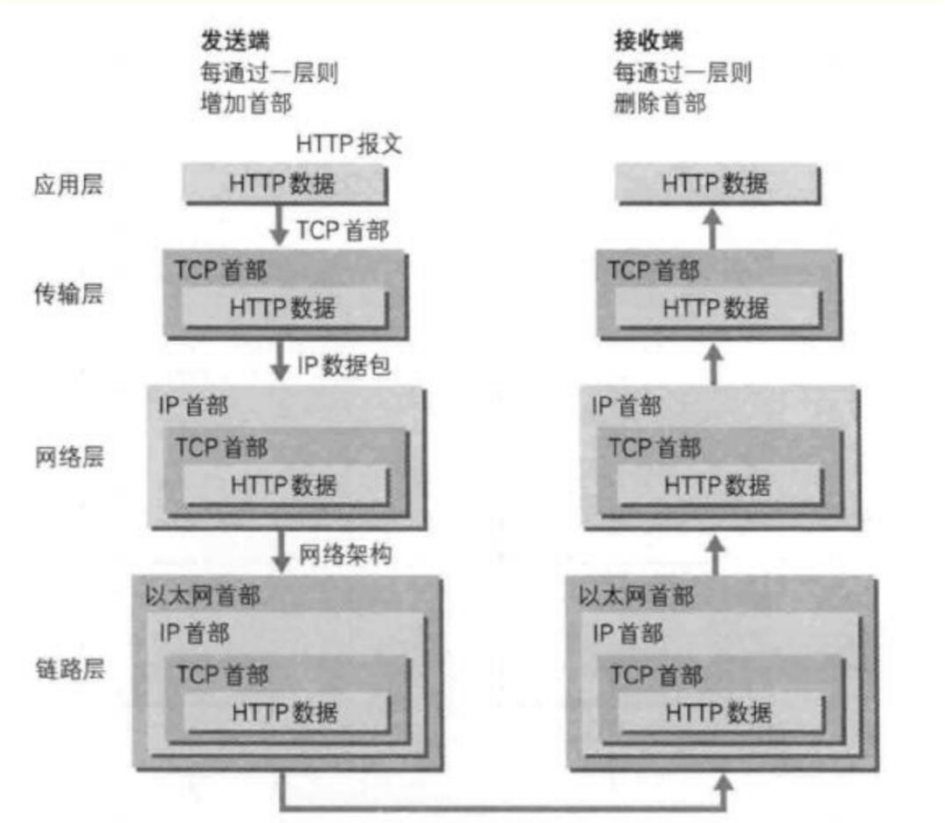
网络层：负责数据包从源到宿的传递和网际互连（包 PackeT）

传输层：提供端到端的可靠报文传递和错误恢复（段 Segment）

会话层：建立、管理和终止会话（会话协议数据单元 SPDU）

表示层：对数据进行翻译、加密和压缩（表示协议数据单元 PPDU）

应用层：允许访问 OSI 环境的手段（应用协议数据单元 APDU）



https://www.cnblogs.com/Robin-YB/p/6668762.html

2）url 到页面的过程

http://blog.csdn.net/xiangriikui/article/details/52207153

3）Forward 和 redirect 的区别

Forward 是服务器自己读取目标 url 的数据并传送给浏览器，浏览器地址栏不变，而 redirect 是通过状态码通知浏览器重新请求新的链接。

http://blog.csdn.net/senmon2004/article/details/654049

4）HTTP：http1.0、1.1、2.0get/post 以及幂等性 http 协议头相关,状态码网络攻击（CSRF、XSS）

http 版本：

1.1 相比 1.0 多加了 keepalive，可以开着长连接，如果有多个请求可以利用单个连接

2.0 相比 1.1 可以同时发送多个请求。1.1 只能顺序发送，如果一个出了问题后面只能阻塞，2.0 可以混在一起发

http://www.blogjava.net/yongboy/archive/2015/03/19/423611.html

Get、post 区别：

Get 会将数据直接显示在 url 上，可以被缓存，可以被缓存为书签，历史日志中有显示。

Post 不会显示。

Get 有长度限制

Post 没有

Get 通常用于可重复提交的操作（对数据库没有做修改）

Post 主要用来处理数据和对数据库做出修改

http 幂等性：

幂等性意指多次操作的结果与一次操作的结果的影响相同。

Get 幂等性：

Get 只为了获取数据，不会对后台的数据产生影响，所以是幂等的。

Delete 幂等性：

Delete 如果删除的是指定某个数据，那么多次提交也只会删除那制定的数据，所以也是幂等的。

Put 幂等性：

Put 表示对同一数据进行修改，所以多次 put 都修改了同样的数据所以是幂等的。

Post 幂等性：

Post 由于涉及到插入和修改，所以无法确保幂等性，需要做出相应的措施解决幂等性的问题。

http://www.cnblogs.com/moonandstar08/p/5334865.html

响应码：

1xx:信息响应类，表示接收到请求并且继续处理

2xx:处理成功响应类，表示动作被成功接收、理解和接受

3xx:重定向响应类，为了完成指定的动作，必须接受进一步处理

4xx:客户端错误，客户请求包含语法错误或者是不能正确执行

5xx:服务端错误，服务器不能正确执行一个正确的请求

详细：

100：接收到并继续处理

200：处理成功

301，302：301 永久指向新网址，302 临时指向新网址

404：找不到页面或处理用的处理类
405：请求的数据和方法中需要的数据不符
500：处理代码发生异常
http://tool.oschina.net/commons?type=5
CSRF、XXS：
http://www.cnblogs.com/wangyuyu/p/3388180.html
5）四次挥手为什么服务端要发两个信息而不是一个
因为有时候服务端还没传完数据，先发送知道要关闭了，发完数据再给客户端发送已经准备关闭的信息
6）TCP/IP 三次握手、四次挥手拥塞控制（过程、阈值）流量控制与滑动窗口 TCP 与 UDP 比较子网划分（一般只有笔试有）DDos 攻击
Tcp:
（1）第一次握手：Client 将标志位 SYN 置为 1，随机产生一个值 seq=J，并将该数据包发送给 Server，Client 进入 SYN_SENT 状态，等待 Server 确认。
（2）第二次握手：Server 收到数据包后由标志位 SYN=1 知道 Client 请求建立连接，Server 将标志位 SYN 和 ACK 都置为 1,ack=J+1,随机产生一个值 seq=K,并将该数据包发送给 Client 以确认连接请求，Server 进入 SYN_RCVD 状态。
（3）第三次握手：Client 收到确认后，检查 ack 是否为 J+1，ACK 是否为 1，如果正确则将标志位 ACK 置为 1，ack=K+1，并将该数据包发送给 Server，Server 检查 ack 是否为 K+1，ACK 是否为 1，如果正确则连接建立成功，Client 和 Server 进入 ESTABLISHED 状态，完成三次握手，随后 Client 与 Server 之间可以开始传输数据了。

（1）第一次挥手：Client 发送一个 FIN，用来关闭 Client 到 Server 的数据传送，Client 进入 FIN_WAIT_1 状态。
（2）第二次挥手：Server 收到 FIN 后，发送一个 ACK 给 Client，确认序号为收到序号+1（与 SYN 相同，一个 FIN 占用一个序号），Server 进入 CLOSE_WAIT 状态。
（3）第三次挥手：Server 发送一个 FIN，用来关闭 Server 到 Client 的数据传送，Server 进入 LAST_ACK 状态。
（4）第四次挥手：Client 收到 FIN 后，Client 进入 TIME_WAIT 状态，接着发送一个 ACK 给 Server，确认序号为收到序号+1，Server 进入 CLOSED 状态，完成四次挥手。
http://www.jianshu.com/p/ef892323e68f
同时打开，同时关闭
http://blog.csdn.net/huoqubing/article/details/6126189
滑动窗口与拥塞控制：
http://www.cnblogs.com/woaiyy/p/3554182.html
区别：
TCP 提供面向连接的、可靠的数据流传输，而 UDP 提供的是非面向连接的、不可靠的数据流传输。（TCP 发出去还会问候核实一下以确保安全; UDP 发出去就不管了 ）
-TCP 传输单位称为 TCP 报文段，UDP 传输单位称为用户数据报。
-TCP 注重数据安全性，UDP 数据传输快，因为不需要连接等待，少了许多操作，但是其安全性却一般。

TCP 对应的协议:
（1） FTP：定义了文件传输协议，使用 21 端口。

（2） Telnet：一种用于远程登陆的端口，使用 23 端口，用户可以以自己的身份远程连接到计算机上，可提供基于 DOS 模式下的通信服务。
（3） SMTP：邮件传送协议，用于发送邮件。服务器开放的是 25 号端口。
（4） POP3：它是和 SMTP 对应，POP3 用于接收邮件。POP3 协议所用的是 110 端口。
（5）HTTP：是从 Web 服务器传输超文本到本地浏览器的传送协议。
UDP 对应的协议：
（1） DNS：用于域名解析服务，将域名地址转换为 IP 地址。DNS 用的是 53 号端口。
（2） SNMP：简单网络管理协议，使用 161 号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。
（3） TFTP(Trival File Transfer Protocal)，简单文件传输协议，该协议在端口 69 上使用 UDP 服务。
7）resp 头中都有什么（主要考察 http 相关知识）
http://www.cnblogs.com/childhooding/p/4349422.html
http://blog.csdn.net/caojunling/article/details/1916899
8）用 int 值表示 ip 如何做（刚好 32 位 bit 一对一映射），写个伪代码做 transfer
http://outofmemory.cn/code-snippet/4837/Java-jiang-ip-charaeter-chuan-turn-huancheng-integer
9）查看 udp 的性能，udp 端口多少，什么时候用 udp?
http://www.cnblogs.com/·smark/p/4496660.html?utm_source=tuicool
10）为什么 tcp 不行？
Udp 适合直播，视频流，由于视频流要传输的数据量大而且并不需要很高的通讯质量。
http://blog.sina.com.cn/s/blog_67bccccb01012ek6.html
11）qq 里哪些用的 tcp 哪些用 udp？分别针对每种情况说一下为什么？
Qq 里聊天信息用 tcp，视频聊天用 udp，qq 聊天数据量小而且要确保送到，视频聊天数据量大，对于数据的完整性要求不高，对性能要求高。
12）说到了 url 有最大长度，就问长度有限制是 get 的原因还是 url 的原因，为什么长度会有限制，是 http 数据包的头的字段原因还是内容字段的原因，详细说明。
Url 最大长度是浏览器的原因，不同的浏览器最大长度不同。
https://www.cnblogs.com/cuihongyu3503319/p/5892257.html
13）说说 http 的缺点。
通信用明文可能被窃取
http://blog.csdn.net/qq_33301113/article/details/78725951
14）然后问我 https ssl tcp 三者关系，其中哪些用到了对称加密，哪些用到了非对称加密，非对称加密密钥是如何实现的。（还好我项目中涉及到了一些加密）
https 为 http 加上了 ssl（加密），将原本明文的 http 数据加密后通过 tcp 协议传输。
客户端先向服务端要公钥，然后生成随机密钥，用公钥加密密钥，附在文件中传给服务端，服务端解密了密钥用对称加密解密。
http://blog.csdn.net/u011068702/article/details/78268552
15）关于加密的私钥和公钥各自如何分配（客户端拿公钥，服务器拿私钥）
16）tcp 的三个状态
Syn（建立） ack（确认） fin（结束）
https://www.cnblogs.com/qingergege/p/6603488.html
17）Udp 可靠传输
http://blog.csdn.net/jxm_96/article/details/53544183

18）http 工作原理

http://blog.csdn.net/hguisu/article/details/8680808

客户机会将请求封装成 http 数据包-->封装成 Tcp 数据包-->封装成 Ip 数据包--->封装成数据帧--->硬件将帧数据转换成 bit 流（二进制数据）-->最后通过物理硬件（网卡芯片）发送到指定地点。

服务器硬件首先收到 bit 流..... 然后转换成 ip 数据包。于是通过 ip 协议解析 Ip 数据包，然后又发现里面是 tcp 数据包，就通过 tcp 协议解析 Tcp 数据包，接着发现是 http 数据包通过 http 协议再解析 http 数据包得到数据。

19）SYN 攻击以及 SYN 攻击的检测方式

http://www.jb51.net/hack/390485.html

只发 syn 不回 ack，让这些半连接占用大量的资源，检测方式就是检测所有连接中半连接的比例

20）Arp 协议

https://www.cnblogs.com/songwenlong/p/6103406.html

21）那客户端是如何认证服务器的真实身份，详细说明一下过程，包括公钥如何申请，哪一层加密哪一层解密。

Ca 证书验证

22）http 状态码，301 和 302 的区别

301 永久跳转，302 临时跳转

http://blog.csdn.net/grandPang/article/details/47448395

23）为什么需要长连接，怎么实现长连接

长连接：

长连接多用于操作频繁，点对点的通讯，而且连接数不能太多的情况。

每个 TCP 连接的建立都需要三次握手，每个 TCP 连接的断开要四次握手。

如果每次操作都要建立连接然后再操作的话处理速度会降低，所以每次操作后，下次操作时直接发送数据就可以了，不用再建立 TCP 连接。例如：数据库的连接用长连接，如果用短连接频繁的通信会造成 socket 错误，频繁的 socket 创建也是对资源的浪费。

短连接：

web 网站的 http 服务一般都用短连接。因为长连接对于服务器来说要耗费一定的资源。像 web 网站这么频繁的成千上万甚至上亿客户端的连接用短连接更省一些资源。试想如果都用长连接，而且同时用成千上万的用户，每个用户都占有一个连接的话，可想而知服务器的压力有多大。所以并发量大，但是每个用户又不需频繁操作的情况下需要短连接。总之：长连接和短连接的选择要根据需求而定。

24）Cookie 的原理

http://blog.csdn.net/fangaoxin/article/details/6952954/

25）servlet 是不是线程安全的

不是

http://www.cnblogs.com/chanshuyi/p/5052426.html

13.项目(完)M

1)实现微信(附近的人)

使用 geohash 处理 xy 值，越相近离得越近

http://blog.jobbole.com/80633/

2)大量 int 去重

使用 bitmap 去重，在原理上是一个 2^32 容量大小的数组分别表示每一种 int 的可能是

否出现，而由于 java 只能用 byte，所以设置一个 2^29 的 byte 数组，每一个 byte 代表 8 个 int 是否存在。

https://www.cnblogs.com/z-sm/p/6238977.html

3)蓄水池

http://blog.csdn.net/huagong_adu/article/details/7619665

4)网站负载变大怎么办……

Html 静态化

图片服务器分离

数据库集群和分表，读写分离

负载均衡

查询缓存

引入 nosql

https://www.cnblogs.com/kevingrace/p/6138150.html

5)Oauth

用于网站登录时使用 qq 微信登陆等手段

https://www.cnblogs.com/flashsun/p/7424071.html

6)分布式 session

Session 广播，Nginx 对应同样的 ip，session 集中存储

http://blog.csdn.net/u014352080/article/details/51764311

7)CDN 加速

为用户的请求根据服务器距用户距离，负载状况，是否存在数据等因素进行负载均衡

https://www.zhihu.com/question/37353035

8)分布式锁

http://www.hollischuang.com/archives/1716

场景：

https://segmentfault.com/q/1010000004605967/a-1020000004608191

9)高并发订单 id 生成策略

1.数据库自增主键，数据库水平切分分担压力

2.预先生成 id

3.根据机器 id，时间戳和序列号生成 id。

https://blog.csdn.net/yeshencat/article/details/72460458

10)分布式缓存的一致性

http://blog.csdn.net/cywosp/article/details/23397179

11)反爬虫的机制，有哪些方式

https://segmentfault.com/a/1190000005840672

12)怎么解决缓存和主存的一致性问题

先删缓存再修改，最多造成 cachemiss，而不会造成不一致

将数据库访问操作单独割出来分离业务与缓存数据库一致性问题

https://mp.weixin.qq.com/s?__biz=MjM5ODYxMDA5OQ==&mid=404087915&idx=1&sn=075664193f334874a3fc87fd4f712ebc&scene=1&srcid=0908q5LhvuehVvGCl53eKx7y&from=groupmessage&isappinstalled=0##

更新数据库时又有写请求时会创建旧数据的缓存，还是会造成一致性问题，所以可以这样：

（1）修改服务 Service 连接池，id 取模选取服务连接，能够保证同一个数据的读写都落在同一个后端服务上

（2）修改数据库 DB 连接池，id 取模选取 DB 连接，能够保证同一个数据的读写在数据库层面是串行的

https://www.cnblogs.com/winner-0715/p/7456151.html

而业务中使用读写分离时，就算能确定先淘汰缓存，再更新数据，再读也不行，因为主从同步有延时，读从库依然容易出现问题，解决方案是在更新完数据库后，用 log 分析再延时进行一次缓存删除，从而达到目的。

https://www.jianshu.com/p/e3f5ff55f0fd

13)分布式事务

http://blog.csdn.net/congyihao/article/details/70195154

14)redis 点赞

https://www.jianshu.com/p/2ab76d5bde71

15)设计一个系统，解决抢购时所需要的大量的短链接的功能，如何保证高并发，如何设计短链接

https://segmentfault.com/a/1190000011171643

16)分布式是怎么提供服务能力的，服务提供者其中一台宕机了，刚好有一台消费者使用到了这台服务器，会出现什么情况

多个机器通过负载均衡提供服务，用户向协调者发送请求后协调者选择一个正在运行的服务器为用户提供服务。

17)那说说 RPC 的要点

服务消费方（client）调用以本地调用方式调用服务；

client stub 接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；

client stub 找到服务地址，并将消息发送到服务端；

server stub 收到消息后进行解码；

server stub 根据解码结果调用本地的服务；

本地服务执行并将结果返回给 server stub；

server stub 将返回结果打包成消息并发送至消费方；

client stub 接收到消息，并进行解码；

服务消费方得到最终结果。

http://www.importnew.com/22003.html

18)序列化的好处与原理

http://www.infoq.com/cn/articles/serialization-and-deserialization/

19)nginx session 共享实现

使用 iphash，相同用户的请求永远会传到固定的服务器，解决 session 问题。

20)大型网站的负载均衡大致怎么弄

使用反向代理，nginx 负载均衡

21)分库分表中间件的大致原理，跨库的 join 怎么做

https://www.cnblogs.com/wangzhongqiu/p/7100332.html

解决：全局表，字段冗余，数据同步，系统层组装(链接表）

http://blog.csdn.net/u011277123/article/details/54374780

22)MySQL 处理高并发，防止库存超卖？

更新时用 sql 对库存进行判断防止超卖，同时对并发量进行限制，如库存 10 个，进来 100 个请求进队列缓存其他人就显示卖光。

http://blog.csdn.net/qq_21267705/article/details/47955539

23)如何获取一个网站每天的访问用户数

1. 如果有 nginx 等反向代理工具，可以直接交给 nginx 进行统计。

http://blog.csdn.net/qq_33324608/article/details/71171574

2.可以使用 servlet 拦截器，对需要的统计请求进行拦截，然后根据需要分类统计或者汇总统计。

3.可以使用 servlet 过滤器，过滤需要统计的请求，同上。

4.Spring aop 也是个不错的选择。

5.如果是 js 脚本，对于不需要访问页面的请求无法统计到。

6.统计容器的 access 日志。

24)设计一个缓存，要求：线程安全，lru，限制容量，可排序。

用 linkedhashmap 实现，继承 linkedhashmap，removeeldestentry 根据存储量返回 true 和 false，初始化时 accessorder 写 true 则按照访问顺序更新队列。

https://www.cnblogs.com/lzrabbit/p/3734850.html

线程安全用 Collections.synchronizedMap 包裹初始化操作

http://blog.csdn.net/u012969732/article/details/48434905

25)实现用户注册时，考虑了哪方面的问题

简单的方面包括：字段有效性验证，密码非对称加密，重复注册检验。

字段有效性：个人信息验证，手机验证码等

非对称加密：md5 等

重复注册：判断对应用户名，手机号等是否有注册过

26)还是在用户注册，高并发情况下，如何优化（提到了 mysql 里唯一索引，有点记不清）

使用唯一索引锁住用户名，防止多个人同时注册同一个用户名。

27)用户注册时，证码验证是如何实现的，如果是在分布式环境里，要怎么验证。

提供单独的验证码生成器，注册验证的时候去验证服务器读取。

28)具体场景，1000 万条数据，分库分表。水平拆分，垂直拆分。如何解决全局主键唯一性问题。

1：单纯将生成主键的事情交给一个数据库管理，会有单点问题，访问量也很大。

2：对生成工作进行分工如一个服务器只生成奇数，一个只生成偶数。

http://blog.csdn.net/bluishglc/article/details/7710738

29)负载均衡算法：随机，轮询，加权随机+轮询，最小连接数算法等

1、轮询法

将请求按顺序轮流地分配到后端服务器上，它均衡地对待后端的每一台服务器，而不关心服务器实际的连接数和当前的系统负载。

2、随机法

通过系统的随机算法，根据后端服务器的列表大小值来随机选取其中的一台服务器进行访问。由概率统计理论可以得知，随着客户端调用服务端的次数增多，

其实际效果越来越接近于平均分配调用量到后端的每一台服务器，也就是轮询的结果。

3、源地址哈希法

源地址哈希的思想是根据获取客户端的 IP 地址，通过哈希函数计算得到的一个数值，用该数值对服务器列表的大小进行取模运算，得到的结果便是客服端要访问服务器的序号。采用源地址哈希法进行负载均衡，同一 IP 地址的客户端，当后端服务器列表不变时，它每次都会映射到同一台后端服务器进行访问。

4、加权轮询法

不同的后端服务器可能机器的配置和当前系统的负载并不相同，因此它们的抗压能力也

不相同。给配置高、负载低的机器配置更高的权重，让其处理更多的请；而配置低、负载高的机器，给其分配较低的权重，降低其系统负载，加权轮询能很好地处理这一问题，并将请求顺序且按照权重分配到后端。

5、加权随机法

与加权轮询法一样，加权随机法也根据后端机器的配置，系统的负载分配不同的权重。不同的是，它是按照权重随机请求后端服务器，而非顺序。

6、最小连接数法

最小连接数算法比较灵活和智能，由于后端服务器的配置不尽相同，对于请求的处理有快有慢，它是根据后端服务器当前的连接情况，动态地选取其中当前

30)SOA

http://www.jdon.com/soa.html

31)CAP 原理和 BASE 理论。

CAP 为分布式服务中较为重要的三要素，

C：一致性

A：可用性

P：分区容忍性(系统中有部分服务或模块挂掉或失效的时候，不影响系统正常服务。)

其中分区容忍性是分布式系统中必备的。

BASE 理论权衡了可用性和一致性，为基本可用和最终一致性。

基本可用行意思是在部分服务器故障时允许一定的延时。

最终一致性强调的是系统中所有的数据副本，在经过一段时间的同步后，最终能够达到一个一致的状态。

https://www.cnblogs.com/duanxz/p/5229352.html

32)加密算法和安全性做了阐述

对称：des

非对称：rsa

不可逆加密：md5

https://www.iplaysoft.com/encrypt-arithmetic.html

33)登陆验证怎么做，为什么用 md5，有没有更好的改进（+salt）

http://blog.csdn.net/blade2001/article/details/6341078

34)影响单台服务器的并发量的因素有哪些？如何优化？

上下文切换：CPU 通过时间片分配算法来循环执行任务，当前任务执行一个时间片后会切换到下一个任务。但是切换前会保存上一个任务的状态，以便下次切换回这个任务时，可以再加载这个任务的状态。任务从保存到再加载的过程就是一次上下文切换。上下文切换会影响多线程的执行速度，因此并发线程数并不是越多越好，需要合理控制并发线程数。减少上下文切换的方法有：

无锁并发编程：多线程竞争锁时，会引起上下文切换，所以多线程处理数据时，可以用一些办法来避免使用锁，比如参考 ConcurrentHashMap 的实现思想，将数据的 ID 按照 Hash 算法取模分段，不同的线程处理不同段的数据。

CAS 算法。Java 的 Atomic 包使用 CAS 算法来更新数据，而不需加锁。

使用最少线程。避免创建不需要的线程。

协程：在单线程里实现多任务的调度，并在单线程里维持多个任务间的切换。

30） 消息队列？应用场景

异步处理，应用解耦，流量削锋和消息通讯

http://blog.csdn.net/seven_____7/article/details/70225830

31）断点续传

https://www.cnblogs.com/findumars/p/5745345.html

14.开源框架（完）M

14.1 nginx（完）M

1)什么是正向代理，什么是反向代理

正向代理是用户通过连上代理服务器转发自己的请求从而访问某些不能访问的网站（如google）

反向代理则是用于装在服务器中来隐藏服务器内部实现以及用于负载均衡，将用户访问的外网地址转化为内网地址。

详细：https://www.cnblogs.com/Anker/p/6056540.html

1）nginx 配置

http://www.cnblogs.com/crazylqy/p/7149774.html

2）Nginx 负载均衡

通过 upstream 定义服务器，用 server 中的 location 绑定对应的负载均衡设置

http://www.cnblogs.com/crazylqy/p/7150250.html

3）Apache 和 Nginx 区别；

1) 高并发响应性能非常好，官方 Nginx 处理静态文件并发 5w/s
2) 反向代理性能非常强。（可用于负载均衡）
3) 内存和 cpu 占用率低。（为 Apache 的 1/5-1/10）
4) 对后端服务有健康检查功能。
5) 支持 PHP cgi 方式和 fastcgi 方式。
6) 配置代码简洁且容易上手。

4） Nginx 七层负载均衡

https://www.cnblogs.com/pycode/p/6588910.html

5）nginx 内部结构了解吗，是怎么实现的？

https://www.cnblogs.com/crazylqy/p/6891929.html

6）nginx 的工作在网络中属于哪一层？

应用层

7） nginx 使用 IP hash 算法如何保证在目标服务器宕机的情况下 保持会话

Session 持久化（如 redis）

8）一台机器上有多个 nginx，如何确定不同 nginx 的启动文件在哪里（查看目录/proc/pid 号码/fd/*，面试官对我 linux 很熟练表示不错）

9）Nginx 请求转发

通过配置文件将指定的请求 pattern 转向目标服务器，同时可以提供负载均衡的功能。

14.2 nosql（完）M

14.2.1 redis（完）M

1)Redis 数据结构

String list set hash zset

http://blog.csdn.net/fan510988896/article/details/71730696

书

2)Redis 单线程多路复用

Redis 采用的是单线程 epoll 模型。

http://blog.csdn.net/liupeng_qwert/article/details/77263187

3)Redis 集群

通过将 hash 槽分片来分给不同的 redis 节点，各自分别负责一块数据，为了防止某个节点挂掉后该数据块的数据复发访问，可以为该节点创建从节点。

https://www.cnblogs.com/liyasong/p/redis_jiqun.html?utm_source=itdadao&utm_medium=referral

4)Redis 集群扩容

http://carlosfu.iteye.com/blog/2243056

5)Redis 设置过期

过期分为定时器删除，惰性删除和定期删除

https://www.cnblogs.com/java-zhao/p/5205771.html

6)Redis 缓存（？？？）

http://www.iigrowing.cn/redis-huan-cun-ji-shu-xue-xi.html

7)redis 相关 两种持久化方式

Rdb 持久化会将当前 redis 的数据存储为快照持久化到硬盘中

Aof 会存储至今的所有操作，恢复时再执行一遍就能恢复。

8)Rdb 为什么不可靠

因为 rdb 需要将所有数据持久化回硬盘所以要耗很多时，持久化有一定的间隔，间隔中间挂掉的话就会损失这段时间的数据

9)Aof 存在什么问题

Aof 存储数据过多的话文件会过大，设置成每次命令都持久化的话会有性能问题

数据存储过多能设置文件压缩，性能问题可设置为每秒持久化或更大间隔，但是有可能损失这段时间的内容。

10)Redis 缓存实现库存

利用 redis 单线程的原子特性设计库存，尝试移除或修改数据，只有成功的被认为是购买成功。

11)Reids 如何淘汰数据

数据在超出设置的内存上限后会淘汰一些数据，有五种淘汰策略

分别为:

淘汰 lru 算法计算出的设置了有效期的数据

淘汰 lru 算法计算出的所有数据

随机淘汰，只淘汰设置有效期的

随机淘汰，都有可能淘汰

淘汰剩余有效期最短的

https://www.cnblogs.com/xiaolang8762400/p/7226269.html

12)Redis 的应用场景

缓存 计数器 队列 位操作 分布式锁 新闻列表 排行榜

https://www.cnblogs.com/NiceCui/p/7794659.html

13)怎么解决缓存穿透

缓存穿透是查询不存在的数据时由于 redis 中查询不到导致要再去数据库查询这个原本不存在的数据，造成关系数据库压力增大。，可以过滤请求或是短时间缓存查询不到的数据。

http://blog.csdn.net/lizhi_java/article/details/68953014?locationNum=14&fps=1

14)同时写入 redis，不能保证线程安全，如何解决
对数据加锁，操作完之前不释放，同时注意要设置过期时间防止死锁
http://www.linuxidc.com/Linux/2014-12/110958.htm
15)Redis hash 结构底层
http://blog.csdn.net/caishenfans/article/details/44784131(延后）
16)Redis 和 memchached
http://blog.csdn.net/tonysz126/article/details/8280696/
17)Topk
https://www.cnblogs.com/mumuxinfei/p/5013357.html
18)Session 共享
　　将原本存在 tomcat 等容器中的 session 数据存储进一个公用的 redis 中，这样当负载均衡到不同的服务器时，都去 redis 中寻找可以找到 session 数据
http://blog.csdn.net/xiao__gui/article/details/52706243
19)Redis 事务
　　将一系列命令先发到 redis 中，发送确认后再执行。
http://www.redis.cn/topics/transactions.html
20)Redis 分布式锁
https://www.cnblogs.com/linjiqin/p/8003838.html
注意要为锁增加过期时间避免死锁，删除锁前要判断持有人是否为自己防止脑裂。
21)集群以及原理
22)主从复制
http://blog.csdn.net/hechurui/article/details/49508813
书

14.2.2 hbase（弃）
14.2.3 mongodb（弃）
14.2.4 memcached（弃）
14.3 zookeeper（完）M
1）Quorum
　　为了保证最终一致性，每个事务要保证被大于一半主机执行，选举时要大于一半同一才行。
　　http://blog.csdn.net/iter_zc/article/details/41212547
2）zookeeper 数据格式
　　树形结构的数据
3）如何利用 zookeeper 进行选举
　　所有的服务器都尝试在 zookeeper 上创建同一个临时节点，而 zookeeper 能保证只有一个服务器能成功，而那个就被当作 master，其他的节点在插入失败后在该节点上注册 wacher，而当 master 节点失效后，由于该临时节点发生了改变从而通知其他 slave 再次开始选举
4）用 zookeeper 做任务分配
　　https://www.jianshu.com/p/c7a927a36802
5）zookeeper 在重新选取 leader 的时候，还可以继续执行事务请求吗
　　不能
6）zookeepe 中 leader 的选举算法，即 fastleaderelection 算法

所有的候选人将自己的 myid 和 zxid 组成投票，遇到比自己大的就修改自己的选票为收到的选票，优先对比 zxid，相等的 zxid 对比 myid。

7）zookeeper 如何保证数据的一致性

同一时间只有 leader 节点能够处理请求，而 leader 节点挂掉后还有崩溃恢复，所以没有问题。

https://www.cnblogs.com/sunddenly/p/4138580.html

8）zookeeper ACL 的权限模式

https://www.cnblogs.com/fesh/p/5798353.html

9）Zab 协议

用 zxid 来进行事务的记录，前 32 位选举轮次，后 32 位事务顺序 id，崩溃恢复时选被提交事务最高的 follower，当再加回集群时 leader 会与 follower 对比事务 id。

书上

10）Paxos 协议

https://www.cnblogs.com/cchust/p/5617989.html

11）Zookeeper 实现分布式锁

书上 6.1.7，和 leader 选举类似，都尝试创建一个节点

12）zookeeper 设计的思想是什么

Cap base

13）zookeeper 如何同步配置

利用 zookeeper 发布与订阅的特性，所有需要该配置文件的服务器注册 watcher 事件到对应文件上，当有更改后通知服务器刷新。

14）Zookeeper 保持一致性用的哪个协议，CA 取舍的哪一个，Raft 呢？

Zookeeper 用的是 zab，cap 取舍为 ap，及保证最终一致性而不是强一致性。

15） 3 个节点挂掉一个能正常工作吗

能，因为过半了，但是再挂就不行了

16）zookeeper 偶数选举问题

偶数节点的集群，则能够容忍挂掉的节点和减去一个服务器时没有区别

14.4 springcloud（完）R

1）eruka:服务治理

主要完成服务注册与服务发现，其他 springcloud 微服务通过将自己注册在 eruka 注册中心来提供服务，同时通过 eruka 发现要调用服务的位置。注册中心互相注册可以保证注册中心的高可用。

2）ribbon：负载均衡

通过注册多个同名的服务后在客户端加上 ribbon 支持并使用 loadbalance 注解修饰的 resttemplate 来调用服务。

3）hystrix：容错保护

当服务超过一定的条件后将服务降级，在调用后会自动执行 falback 的操作而不会傻等服务执行。

4）fegin：同时整合了 hystrix 和 ribbon。

5）zuul：服务路由

6）config：分布式配置中心：通过一些远端服务（如 git）统一存储配置

7）bus：消息总线，如发布配置信息刷新的消息。

14.6 dubbo（暂缓）

1）了解 Dubbo 吗 自己动手打过框架吗
14.7 RocketMq（暂缓）
14.8 ActiveMq（暂缓）
1）activemq 实现原理，是推还是拉
14.9 分布式存储系统（弃）
1)GFS（弃）
2)HDFS（弃）
3)fastDFS(弃）
14.11 kafka（暂缓）
14.12 git（完）M
1）git 常用的操作

```
git init //初始化本地 git 环境
git clone XXX//克隆一份代码到本地仓库
git pull //把远程库的代码更新到工作台
git rebase 合并分支
git pull --rebase origin master //强制把远程库的代码跟新到当前分支上面
git fetch //把远程库的代码更新到本地库
git add . //把本地的修改加到 stage 中
git commit -m 'comments here' //把 stage 中的修改提交到本地库
git push //把本地库的修改提交到远程库中
git branch -r/-a //查看远程分支/全部分支
git checkout master/branch //切换到某个分支
git checkout -b test //新建 test 分支
git checkout -d test //删除 test 分支
git merge master //假设当前在 test 分支上面，把 master 分支上的修改同步到 test 分支上
git blame someFile //查看某个文件的每一行的修改记录（）谁在什么时候修改的）
git status //查看当前分支有哪些修改
git log //查看当前分支上面的日志信息
git diff //查看当前没有 add 的内容
git diff --cache //查看已经 add 但是没有 commit 的内容
git diff HEAD //上面两个内容的合并
git reset --hard HEAD //撤销本地修改
echo $HOME //查看 git config 的 HOME 路径
export $HOME=/c/gitconfig //配置 git config 的 HOME 路径
```

2）git rebase 和 git merge 区别

变基（rebase）是将一系列提交按照原有次序依次应用到另一分支上，而合并（merge）是把最终结果合在一起。

3）要提交到某个分支，你要怎么提交

提交到当前分支上：add，commit
提交到远程分支上：push
提交到别的分支上（如创建一个功能后要合并会主分支）：checkout merge

4）git 和 svn 的区别

Git 自己就有版本库，不需要联网就能工作，svn 需要连上中央库

5）pull 和 pull --rebase 的区别

Pull 为 fetch+merge pull-rebase 为 fetch+rebase

8） git 版本控制器的原理

https://git-scm.com/book/zh/v1/%E8%B5%B7%E6%AD%A5-%E5%85%B3%E4%BA%8E%E7%89%88%E6%9C%AC%E6%8E%A7%E5%88%B6

9） git 工作流

1：集中式工作流：集中式工作流只有 master 一个分支，成员将仓库的项目 clone 后 add 自己的修改，commit 到自己的本地库后 push 到中央仓库。

2：功能分支工作流：功能分支工作流让每个人在各自的分支下开发，并在开发完成时申请合并操作，主管确认可以合并后将对应的分支合并回主分支，这样更容易划分工作。

3. gitflow 工作流：gitflow 工作流分为发布分支，开发分支，hotfix 分支和功能分支，开发分支从发布分支中分离出来用于开发，功能分支直接与开发分支交付，并在测试好较为稳定的时候从开发分支并回发布分支，hotfix 分支用于修复紧急问题，从开发分支中分离出来，当解决问题后将修改合并到开发和发布分支中。

4. Forking 工作流：该工作流各用户都有自己的远程仓库，由主仓库克隆出来，自己进行开发并 push 到自己的远程仓库，并发布 pullrequest 给主仓库维护者，主仓库维护者同意后将改动并回主仓库并通知其他人更新仓库。

https://segmentfault.com/a/1190000002918123

10） git pull 和 fetch 的区别

Pull 为 fetch+merge

11） git 仓库，远程仓库

本地仓库为远程仓库拉下来的备份，可以离线开发，开发后可再推到远程仓库中。

12） git clone

Clone 会把远程仓库里的东西完整的下载到本地。

13） 提交错误的话，怎么撤销

Reset

https://bingohuang.gitbooks.io/progit2/content/02-git-basics/sections/undoing.html

14.13 maven（完）M

https://www.cnblogs.com/tooy/p/7364149.html

循环依赖：http://hck.iteye.com/blog/1728329

2）maven 如何进行依赖管理，如何解决依赖冲突

https://www.cnblogs.com/davenkin/p/advanced-maven-resolve-dependencies-conflicts.html

Maven 是最近匹配功能，距离相同时根据编写顺序。如果需要最新版本可以直接在 pom 文件中声明或是在不要的老版本里加声明不加载

4)maven 的源和插件了解哪些

https://www.cnblogs.com/jqmtony/p/7258511.html

5）maven 的生命周期

验证（validate） - 验证项目是否正确，所有必要的信息可用

编译（compile） - 编译项目的源代码

测试（test） - 使用合适的单元测试框架测试编译的源代码。这些测试不应该要求代码被打包或部署

打包（package） - 采用编译的代码，并以其可分配格式（如 JAR）进行打包。

验证（verify） - 对集成测试的结果执行任何检查，以确保满足质量标准

安装（install） - 将软件包安装到本地存储库中，用作本地其他项目的依赖项

部署（deploy） - 在构建环境中完成，将最终的包复制到远程存储库以与其他开发人员和项目共享。

https://www.cnblogs.com/EasonJim/p/6816340.html

6）Maven 的配置文件

http://blog.csdn.net/u012152619/article/details/51485152

7）maven 的 snapshot 和 released 版本区别

Snapshot 为不稳定版本，这类版本会进行更新，而 maven 会定时下载更新，所以将这类不稳定版本直接引用有可能会导致问题发生（某些方法效果变更和消失等），但是适合开发小组类别的项目组，这样不用频繁迭代版本也能下载到最新代码。而 release 版更新会直接变更版本号，代码本身不会再做改变。

https://www.cnblogs.com/huang0925/p/5169624.html

8）package 类型

常用的有：

Jar：单纯的 java 程序

War：javaweb 程序

Pom：纯配置文件模式（用于同一版本，依赖于同一个 parent 或是 dependencymanagement 后所有的项目对应相同的依赖模块版本会一致）同时可以应用于项目聚合，将多个项目同时打包部署。

9）Maven 怎么把所有包打包

项目的聚合，pom 父模块

https://www.cnblogs.com/scote/p/5786903.html

14.15 mybatis （完）M

1)ibatis 是怎么实现映射的，它的映射原理是什么

使用的 jdk 动态代理，

https://www.jianshu.com/p/adda12b70d59

使用的是 jdbc

2)Mybatis 缓存

https://www.jianshu.com/p/c553169c5921

mybatis 的缓存：分为一级缓存和二级缓存，一级缓存的作用范围为 session,所以当 session commit 或 close 后,缓存就会被清空 ，二级缓存的作用范围为 namespace,映射语句文件中的所有 select 语句都会被缓存,所有 CRUD 的操作都会刷新缓存,缓存会存储 1024 个对象，缓存容易造成脏毒数据，影响真实数据的准确性，实际开发业务中会放弃二级缓存。

redis 的缓存：可控制的后端缓存服务，通常用来缓存后端数据，当程序第二次访问数据库的时候，命中 redis，大大减少数据库的负担，减少访问数据库的链接时间，实际开发过程中都会采用这种缓存方式，达到访问速度和效率的解决方案。

https://www.cnblogs.com/springlight/p/6374372.html

Memchache 缓存：

https://www.cnblogs.com/wangjian1990/p/6731812.html

Redis 和 memchached 的区别：

https://www.cnblogs.com/jeffen/p/6087313.html

3)Mybatis 的功能

数据库操作，缓存，分页（插件）

4)Mybatis 常用标签

Mapper：最外层，用来配置 namespace（实现的接口）

Resultmap：用来映射返回的类

Sql：存储共通的 sql 语句

Include：引用 sql 片段

Select，insert，update，delete：对应数据库增删改查

If：条件语句

Where：拼接查询语句

Trim：拼接语句

Choose：类似于 switch

Collections:列表

Foreach：批量

Association：一对一关系

5)Mybatis 和 hibernate 的区别

Mybatis 易上手，小巧，更容易控制 sql 语句

Hibernate 上手难，功能丰富，不能直接对 sql 进行控制

http://blog.csdn.net/jiuqiyuliang/article/details/45378065

6)Mybatis 批量插入

Foreach

```
<insert id="insertRecords" parameterType="java.util.List">
  insert into t_record ( co_id, c_id,
    o_id, money, time
    )
  values
  <foreach collection="list" item="item" index="index" separator="," >
      (#{item.coId},#{item.cId},#{item.oId},#{item.money},#{item.time})
  </foreach>
</insert>
<insert id="insertSelective" parameterType="cn.edu.bjut.api.order.entity.Record">
```

7)Sessionfactory

http://blog.csdn.net/u013412772/article/details/73648537

8)防止注入攻击原理

#可以防止注入攻击，$用于动态的查询条件等（如 orderby 什么栏）

#防止注入问题的原理是用了 preparedstatement（预编译）

预编译会把不带参数的 sql 语句编译后预先存储，而调用时仅仅把需要的参数进行设置，导致就算注入了些 sql 语句，数据库也单纯将它作为值来看待。

https://www.cnblogs.com/ygj0930/p/5876951.html

9)mybatis 和 hibernate 各自的缓存原理和比较，hibernate 的一级二级和查询缓存，还有针对缓存的 miss 率，置换策略，容量设置和性能的平衡问了自己的理解。

http://blog.csdn.net/tanga842428/article/details/52698657

Mybatis 的二级缓存是 namespace 范围的，而 hibernate 是 sessionfactory 范围的，hibernate 当出现脏读会报错，mybatis 不会。

10)数据库主从备和读写分离原理，ibatis 怎么配置。（这个只讲了数据库层面的原理，比如

监听线程，主机和从机的同步方式等，但是具体代码层面的配置，由于没亲自做过，就说不太知道。）
读写分离：设置两套连接池配置文件，两套事务配置，两套 mapper 文件映射，两套实现接口（分别对应读写）
http://blog.csdn.net/Linpk0315/article/details/51504887
主从：
https://www.cnblogs.com/aegisada/p/5699058.html