本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

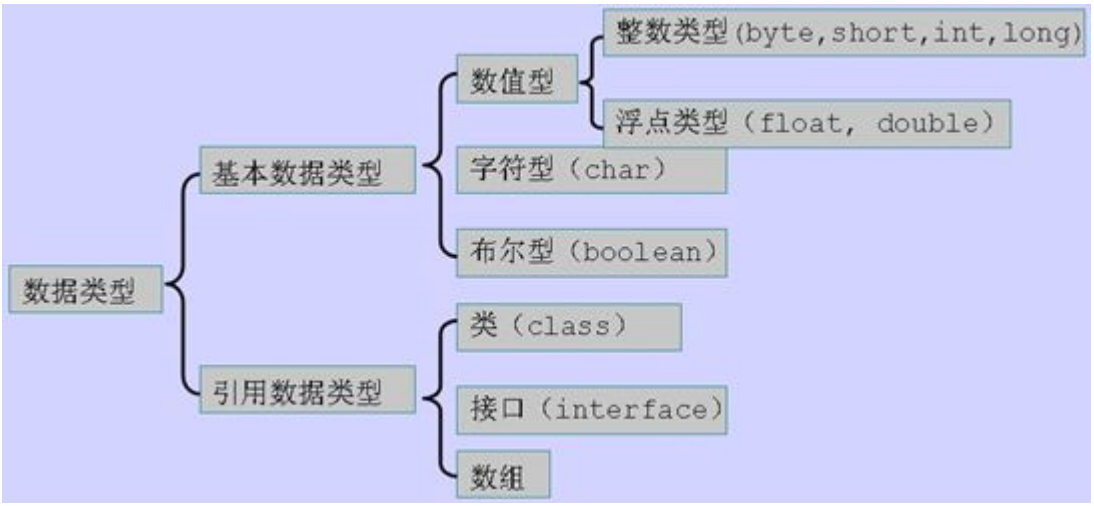
本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

JAVA 基础

JAVA 中的几种基本类型，各占用多少字节？



数据类型	大小	范围	默认值
byte(字节)	8	-128 - 127	0
shot(短整型)	16	-32768 - 32768	0
int(整型)	32	-2147483648-2147483648	0
long(长整型)	64	-9233372036854477808-9233372036854477808	0
float(浮点型)	32	-3.40292347E+38-3.40292347E+38	0.0f
double(双精度)	64	-1.79769313486231570E+308-1.79769313486231570E+308	0.0d
char(字符型)	16	' \u0000 - u\ffff '	'\u0000 '
boolean(布尔型)	1	true/false	false

String 能被继承吗？为什么？

不可以，因为 String 类有 final 修饰符，而 final 修饰的类是不能被继承的，实现细节不允许改变。平常我们定义的 String str=”a”;其实和 String str=new String(“a”)还是有差异的。

前者默认调用的是 String.valueOf 来返回 String 实例对象，至于调用哪个则取决于你的赋值，比如 String num=1,调用的是

```
public static String valueOf(int i) {
```

```
        return Integer.toString(i);
    }
```

后者则是调用如下部分：

```
    public String(String original) {
        this.value = original.value;
        this.hash = original.hash;
    }
```

最后我们的变量都存储在一个 char 数组中

```
 private final char value[];
```

String， Stringbuffer， StringBuilder 的区别。

String 字符串常量(final 修饰，不可被继承)，**String 是常量，当创建之后即不能更改**。(可以通过 StringBuffer 和 StringBuilder 创建 String 对象(常用的两个**字符串操作类**)。)

StringBuffer 字符串变量（线程安全）,其也是 final 类别的，不允许被继承，其中的绝大多数方法都进行了同步处理，包括常用的 Append 方法也做了同步处理(synchronized 修饰)。其自 jdk1.0 起就已经出现。其 toString 方法会进行对象缓存，以减少元素复制开销。

```
public synchronized String toString() {
        if (toStringCache == null) {
            toStringCache = Arrays.copyOfRange(value, 0, count);
        }
        return new String(toStringCache, true);
    }
```

StringBuilder 字符串变量（非线程安全）其自 jdk1.5 起开始出现。与 StringBuffer 一样都继承和实现了同样的接口和类，方法除了没使用 synch 修饰以外基本一致，不同之处在于最后 toString 的时候，会直接返回一个新对象。

```
public String toString() {
        // Create a copy, don't share the array
        return new String(value, 0, count);
    }
```

ArrayList 和 LinkedList 有什么区别。

ArrayList 和 LinkedList 都实现了 List 接口，有以下的不同点：

1、ArrayList 是基于索引的数据接口，它的底层是数组。它可以以 0(1)时间复杂度对元素进行随机访问。与此对应，LinkedList 是以元素列表的形式存储它的数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是 0(n)。

2、相对于 ArrayList，LinkedList 的插入，添加，删除操作速度更快，因为当元素被添加到集合任意位置

的时候，不需要像数组那样重新计算大小或者是更新索引。
3、LinkedList 比 ArrayList 更占内存，因为 LinkedList 为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。

讲讲类的实例化顺序，比如父类静态数据，构造函数，字段，子类静态数据，构造函数，字段，当 new 的时候， 他们的执行顺序。

此题考察的是类加载器实例化时进行的操作步骤（加载-->连接->初始化）。
父类静态代变量、
父类静态代码块、
子类静态变量、
子类静态代码块、
父类非静态变量（父类实例成员变量）、
父类构造函数、
子类非静态变量（子类实例成员变量）、
子类构造函数。
测试 demo：http://blog.csdn.net/u014042066/article/details/77574956
参阅我的博客《深入理解类加载》：http://blog.csdn.net/u014042066/article/details/77394480

用过哪些 Map 类，都有什么区别，HashMap 是线程安全的吗,并发下使用的 Map 是什么，他们内部原理分别是什么，比如存储方式， hashcode，扩容， 默认容量等。

hashMap 是线程不安全的，HashMap 是数组+链表+红黑树（JDK1.8 增加了红黑树部分）实现的，采用哈希表来存储的，
参照该链接：https://zhuanlan.zhihu.com/p/21673805

JAVA8 的 ConcurrentHashMap 为什么放弃了分段锁,有什么问题吗，如果你来设计，你如何设计。

参照：https://yq.aliyun.com/articles/36781

有没有有顺序的 Map 实现类， 如果有， 他们是怎么保证有序的。

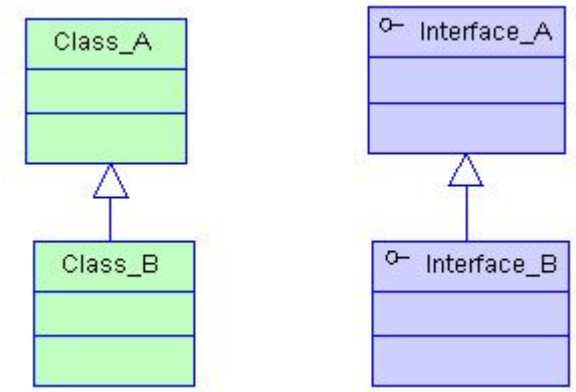
TreeMap 和 LinkedHashMap 是有序的（TreeMap 默认升序，LinkedHashMap 则记录了插入顺序）。
参照：http://uule.iteye.com/blog/1522291

抽象类和接口的区别，类可以继承多个类么，接口可以继承多个接口么,类可以实现多个接口么。

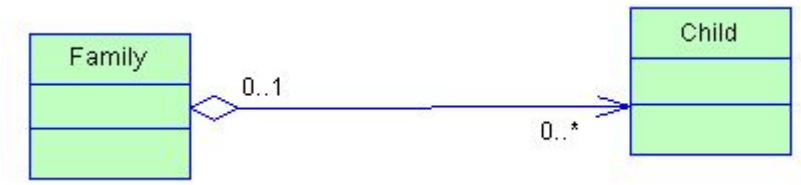
1、抽象类和接口都不能直接实例化，如果要实例化，抽象类变量必须指向实现所有抽象方法的子类对象，接口变量必须指向实现所有接口方法的类对象。
2、抽象类要被子类继承，接口要被类实现。
3、接口只能做方法申明，抽象类中可以做方法申明，也可以做方法实现
4、接口里定义的变量只能是公共的静态的常量，抽象类中的变量是普通变量。
5、抽象类里的抽象方法必须全部被子类所实现，如果子类不能全部实现父类抽象方法，那么该子类只能是抽象类。同样，一个实现接口的时候，如不能全部实现接口方法，那么该类也只能为抽象类。
6、抽象方法只能申明，不能实现。abstract void abc();不能写成 abstract void abc(){}。
7、抽象类里可以没有抽象方法
8、如果一个类里有抽象方法，那么这个类只能是抽象类
9、抽象方法要被实现，所以不能是静态的，也不能是私有的。
10、接口可继承接口，并可多继承接口，但类只能单根继承。

继承和聚合的区别在哪。

继承指的是一个类（称为子类、子接口）继承另外的一个类（称为父类、父接口）的功能，并可以增加它自己的新功能的能力，继承是类与类或者接口与接口之间最常见的关系；在 Java 中此类关系通过关键字 extends 明确标识，在设计时一般没有争议性；



聚合是关联关系的一种特例，他体现的是整体与部分、拥有的关系，即 has-a 的关系，此时整体与部分之间是可分离的，他们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享；比如计算机与 CPU、公司与员工的关系等；表现在代码层面，和关联关系是一致的，只能从语义级别来区分；



参考：http://www.cnblogs.com/jiqing9006/p/5915023.html

讲讲你理解的 nio 和 bio 的区别是啥，谈谈 reactor 模型。

IO 是面向流的，NIO 是面向缓冲区的
参考：https://zhuanlan.zhihu.com/p/23488863
http://developer.51cto.com/art/201103/252367.htm
http://www.jianshu.com/p/3f703d3d804c

反射的原理，反射创建类实例的三种方式是什么

参照：http://www.jianshu.com/p/3ea4a6b57f87?amp

http://blog.csdn.net/yongjian1092/article/details/7364451

反射中，Class.forName 和 ClassLoader 区别。

https://my.oschina.net/gpzhang/blog/486743

描述动态代理的几种实现方式，分别说出相应的优缺点。

Jdk cglib jdk 底层是利用反射机制，需要基于接口方式，这是由于
Proxy.newProxyInstance(target.getClass().getClassLoader(),
target.getClass().getInterfaces(), this);
Cglib 则是基于 asm 框架，实现了无反射机制进行代理，利用空间来换取了时间，代理效率高于 jdk
http://lrd.ele.me/2017/01/09/dynamic_proxy/

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

动态代理与 cglib 实现的区别

同上（基于 invocationHandler 和 methodInterceptor）

为什么 CGlib 方式可以对接口实现代理。

同上 因为 Enhancer

final 的用途

类、变量、方法
http://www.importnew.com/7553.html

写出三种单例模式实现。

懒汉式单例，饿汉式单例，双重检查等
参考：https://my.oschina.net/dyyweb/blog/609021

如何在父类中为子类自动完成所有的 hashcode 和 equals 实现？这么做有何优劣。

同时复写 hashcode 和 equals 方法，优势可以添加自定义逻辑，且不必调用超类的实现。
参照：http://java-min.iteye.com/blog/1416727

请结合 OO 设计理念，谈谈访问修饰符 public、private、protected、default 在应用设计中的作用。

访问修饰符，主要标示修饰块的作用域，方便隔离防护

	同一个类	同一个包	不同包的子类	不同包的非子类
Private	√			
Default	√	√		
Protected	√	√	√	

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

Public	√	√	√	√

public： Java 语言中访问限制最宽的修饰符，一般称之为“公共的”。被其修饰的类、属性以及方法不
仅可以跨类访问，而且允许跨包（package）访问。
private: Java 语言中对访问权限限制的最窄的修饰符，一般称之为“私有的”。被其修饰的类、属性以
及方法只能被该类的对象访问，其子类不能访问，更不能允许跨包访问。
protect: 介于 public 和 private 之间的一种访问修饰符，一般称之为“保护形”。被其修饰的类、
属性以及方法只能被类本身的方法及子类访问，即使子类在不同的包中也可以访问。
default：即不加任何访问修饰符，通常称为“默认访问模式“。该模式下，只允许在同一个包中进行访
问。

深拷贝和浅拷贝区别。

http://www.oschina.net/translate/java-copy-shallow-vs-deep-in-which-you-will-swim

数组和链表数据结构描述，各自的时间复杂度

http://blog.csdn.net/snow_wu/article/details/53172721

error 和 exception 的区别， CheckedException，RuntimeException 的区别

http://blog.csdn.net/woshixuye/article/details/8230407

请列出 5 个运行时异常。

同上

在自己的代码中，如果创建一个 java.lang.String 对象，这个对象是否可以被类加载器加载？为什么

类加载无须等到“首次使用该类”时加载，jvm 允许预加载某些类。。。。
http://www.cnblogs.com/jasonstorm/p/5663864.html

说一说你对 java.lang.Object 对象中 hashCode 和 equals 方法的理解。在什么场景下需要重新实现这两个方法。

参考上边试题

在 jdk1.5 中，引入了泛型，泛型的存在是用来解决什么问题。

泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数，泛型的好处是在编译的时候检查类型安全，并且所有的强制转换都是自动和隐式的，以提高代码的重用率
http://baike.baidu.com/item/java%E6%B3%9B%E5%9E%8B

这样的 a.hashcode() 有什么用，与 a.equals(b)有什么关系。

hashcode
hashcode（）方法提供了对象的 hashCode 值，是一个 native 方法，返回的默认值与 System.identityHashCode(obj)一致。

通常这个值是对象头部的一部分二进制位组成的数字，具有一定的标识对象的意义存在，但绝不定于地址。

作用是：用一个数字来标识对象。比如在 HashMap、HashSet 等类似的集合类中，如果用某个对象本身作为 Key，即要基于这个对象实现 Hash 的写入和查找，那么对象本身如何实现这个呢？就是基于 hashcode 这样一个数字来完成的，只有数字才能完成计算和对比操作。

hashcode 是否唯一
hashcode 只能说是标识对象，在 hash 算法中可以将对象相对离散开，这样就可以在查找数据的时候根据这个 key 快速缩小数据的范围，但 hashcode 不一定是唯一的，所以 hash 算法中定位到具体的链表后，需要循环链表，然后通过 equals 方法来对比 Key 是否是一样的。

equals 与 hashcode 的关系
equals 相等两个对象，则 hashcode 一定要相等。但是 hashcode 相等的两个对象不一定 equals 相等。

https://segmentfault.com/a/1190000004520827

有没有可能 2 个不相等的对象有相同的 hashcode。

有

Java 中的 HashSet 内部是如何工作的。

底层是基于 hashmap 实现的
http://wiki.jikexueyuan.com/project/java-collection/hashset.html

什么是序列化，怎么序列化，为什么序列化，反序列化会遇到什么问题，如何解决。

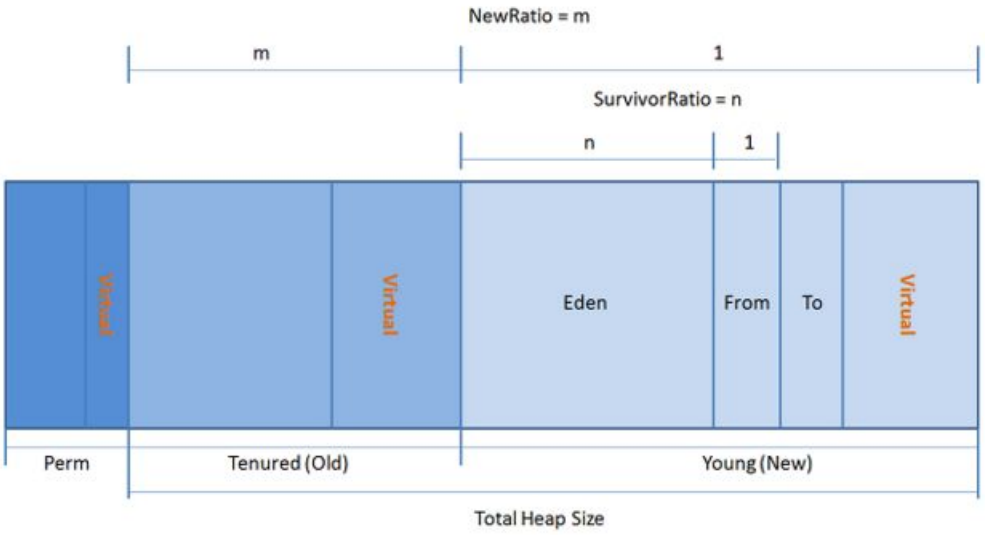
http://www.importnew.com/17964.html

JVM 知识

什么情况下会发生栈内存溢出。

如果线程请求的栈深度大于虚拟机所允许的深度，将抛出 StackOverflowError 异常。 如果虚拟机在动态扩展栈时无法申请到足够的内存空间，则抛出 OutOfMemoryError 异常。
参照：http://wiki.jikexueyuan.com/project/java-vm/storage.html

JVM 的内存结构，Eden 和 Survivor 比例。



eden 和 survior 是按 8 比 1 分配的

http://blog.csdn.net/lojze_ly/article/details/49456255

jvm 中一次完整的 GC 流程是怎样的，对象如何晋升到老年代，说说你知道的几种主要的 jvm 参数。

对象诞生即新生代->eden，在进行 minor gc 过程中，如果依旧存活，移动到 from，变成 Survivor，进行标记代数，如此检查一定次数后，晋升为老年代，

http://www.cnblogs.com/redcreen/archive/2011/05/04/2037056.html

http://ifeve.com/useful-jvm-flags/

https://wangkang007.gitbooks.io/jvm/content/jvmcan_shu_xiang_jie.html

你知道哪几种垃圾收集器，各自的优缺点，重点讲下 cms，包括原理，流程，优缺点

Serial、parNew、ParallelScavenge、SerialOld、ParallelOld、CMS、G1

https://wangkang007.gitbooks.io/jvm/content/chapter1.html

垃圾回收算法的实现原理。

http://www.importnew.com/13493.html

当出现了内存溢出，你怎么排错。

首先分析是什么类型的内存溢出，对应的调整参数或者优化代码。
https://wangkang007.gitbooks.io/jvm/content/4jvmdiao_you.html

JVM 内存模型的相关知识了解多少，比如重排序，内存屏障，happen-before，主内存，工作内存等。

内存屏障：为了保障执行顺序和可见性的一条 cpu 指令
重排序：为了提高性能，编译器和处理器会对执行进行重拍
happen-before：操作间执行的顺序关系。有些操作先发生。
主内存：共享变量存储的区域即是主内存
工作内存：每个线程 copy 的本地内存，存储了该线程以读/写共享变量的副本
http://ifeve.com/java-memory-model-1/
http://www.jianshu.com/p/d3fda02d4cae
http://blog.csdn.net/kenzyq/article/details/50918457

简单说说你了解的类加载器。

类加载器的分类（bootstrap,ext,app,curstom），类加载的流程(load-link-init)
http://blog.csdn.net/gjanyanlig/article/details/6818655/

讲讲 JAVA 的反射机制。

Java 程序在运行状态可以动态的获取类的所有属性和方法，并实例化该类，调用方法的功能
http://baike.baidu.com/link?url=C7p1PeLa3ploAgkfAOK-4XHE8HzQuOAB7K5GPcK_zpbAa_Aw-nO3997K1oir8N--1_wxXZfOThFrEcA0LjVP6wNOwidVTkLBzKlQVK6JvXYvVNhDWV9yF-NIOebtg1hwsnagsjUhOE2wxmiup20RRa#7

你们线上应用的 JVM 参数有哪些。

-server
Xms6000M
-Xmx6000M
-Xmn500M
-XX:PermSize=500M
-XX:MaxPermSize=500M
-XX:SurvivorRatio=65536
-XX:MaxTenuringThreshold=0
-Xnoclassgc
-XX:+DisableExplicitGC
-XX:+UseParNewGC
-XX:+UseConcMarkSweepGC

-XX:+UseCMSCompactAtFullCollection
-XX:CMSFullGCsBeforeCompaction=0
-XX:+CMSClassUnloadingEnabled
-XX:-CMSParallelRemarkEnabled
-XX:CMSInitiatingOccupancyFraction=90
-XX:SoftRefLRUPolicyMSPerMB=0
-XX:+PrintClassHistogram
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
-XX:+PrintHeapAtGC
-Xloggc:log/gc.log

g1 和 cms 区别,吞吐量优先和响应优先的垃圾收集器选择。

Cms 是以获取最短回收停顿时间为目标的收集器。基于标记-清除算法实现。比较占用 cpu 资源，切易造成碎片。
G1 是面向服务端的垃圾收集器，是 jdk9 默认的收集器，基于标记-整理算法实现。可利用多核、多 cpu，保留分代，实现可预测停顿，可控。
http://blog.csdn.net/linhu007/article/details/48897597

请解释如下 jvm 参数的含义：

-server -Xms512m -Xmx512m -Xss1024K
-XX:PermSize=256m -XX:MaxPermSize=512m -XX:MaxTenuringThreshold=20
XX:CMSInitiatingOccupancyFraction=80 -XX:+UseCMSInitiatingOccupancyOnly。

Server 模式启动
最小堆内存 512m
最大 512m
每个线程栈空间 1m
永久代 256
最大永久代 256
最大转为老年代检查次数 20
Cms 回收开启时机：内存占用 80%
命令 JVM 不基于运行时收集的数据来启动 CMS 垃圾收集周期

开源框架知识

简单讲讲 tomcat 结构，以及其类加载器流程。

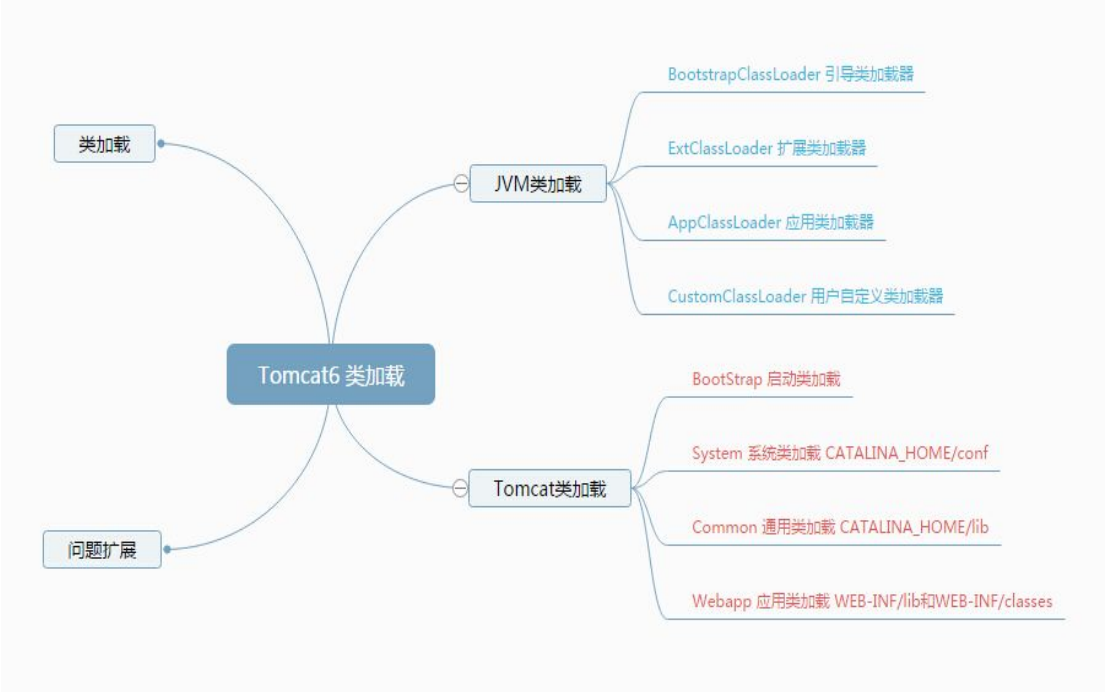
Server- --多个 service

Container 级别的：-->engine--》host-->context

Listenter

Connector

Logging、Naming、Session、JMX 等等



通过 WebappClassLoader 加载 class

http://www.ibm.com/developerworks/cn/java/j-lo-tomcat1/

http://blog.csdn.net/dc_726/article/details/11873343

http://www.cnblogs.com/xing901022/p/4574961.html

http://www.jianshu.com/p/62ec977996df

tomcat 如何调优，涉及哪些参数。

硬件上选择，操作系统选择，版本选择，jdk 选择，配置 jvm 参数，配置 connector 的线程数量，开启 gzip 压缩，trimSpaces，集群等

http://blog.csdn.net/lifetragedy/article/details/7708724

讲讲 Spring 加载流程。

通过 listener 入口，核心是在 AbstractApplicationContext 的 refresh 方法，在此处进行装载 bean 工厂，bean，创建 bean 实例，拦截器，后置处理器等。

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

讲讲 Spring 事务的传播属性。

七种传播属性。

事务传播行为

所谓事务的传播行为是指，如果在开始当前事务之前，一个事务上下文已经存在，此时有若干选项可以指定一个事务性方法的执行行为。在 TransactionDefinition 定义中包括了如下几个表示传播行为的常量：

TransactionDefinition.PROPAGATION_REQUIRED：如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。

TransactionDefinition.PROPAGATION_REQUIRES_NEW：创建一个新的事务，如果当前存在事务，则把当前事务挂起。

TransactionDefinition.PROPAGATION_SUPPORTS：如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。

TransactionDefinition.PROPAGATION_NOT_SUPPORTED：以非事务方式运行，如果当前存在事务，则把当前事务挂起。

TransactionDefinition.PROPAGATION_NEVER：以非事务方式运行，如果当前存在事务，则抛出异常。

TransactionDefinition.PROPAGATION_MANDATORY：如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。

TransactionDefinition.PROPAGATION_NESTED：如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于 TransactionDefinition.PROPAGATION_REQUIRED。

https://www.ibm.com/developerworks/cn/education/opensource/os-cn-spring-trans/

Spring 如何管理事务的。

编程式和声明式

同上

Spring 怎么配置事务（具体说出一些关键的 xml 元素）。

```
<bean id="bankService"
class="footmark.spring.core.tx.programmatic.template.BankServiceImpl">
<property name="bankDao" ref="bankDao"/>
<property name="transactionTemplate" ref="transactionTemplate"/>
</bean><beans.....>
.....
<bean id="bankService"
class="footmark.spring.core.tx.declare.namespace.BankServiceImpl">
<property name="bankDao" ref="bankDao"/>
</bean>
<tx:advice id="bankAdvice" transaction-manager="transactionManager">
<tx:attributes>
<tx:method name="transfer" propagation="REQUIRED"/>
</tx:attributes>
```

```
</tx:advice>

<aop:config>
<aop:pointcut id="bankPointcut" expression="execution(* *.transfer(..))"/>
<aop:advisor advice-ref="bankAdvice" pointcut-ref="bankPointcut"/>
</aop:config>
.....
</beans>
```

同上

说说你对 Spring 的理解，非单例注入的原理？它的生命周期？循环注入的原理， aop 的实现原理，说说 aop 中的几个术语，它们是怎么相互工作的。

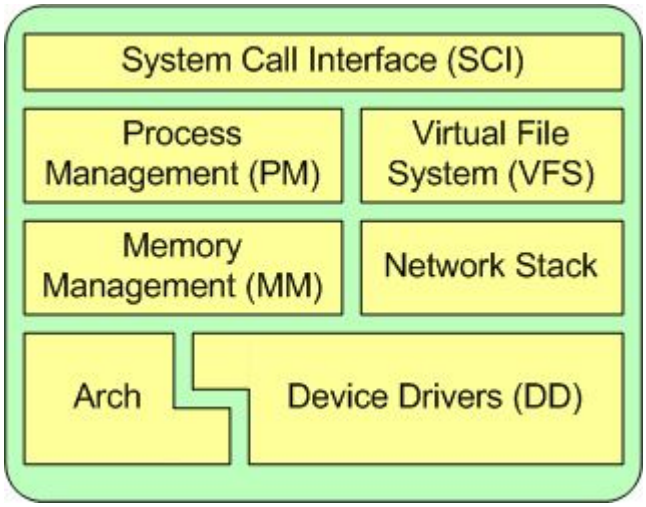
核心组件：bean，context，core，单例注入是通过单例 beanFactory 进行创建，生命周期是在创建的时候通过接口实现开启，循环注入是通过后置处理器，aop 其实就是通过反射进行动态代理，pointcut，advice 等。
Aop 相关：http://blog.csdn.net/csh624366188/article/details/7651702/

Springmvc 中 DispatcherServlet 初始化过程。

入口是 web.xml 中配置的 ds，ds 继承了 HttpServletBean，FrameworkServlet，通过其中的 init 方法进行初始化装载 bean 和实例，initServletBean 是实际完成上下文工作和 bean 初始化的方法。
http://www.mamicode.com/info-detail-512105.html

操作系统

Linux 系统下你关注过哪些内核参数，说说你知道的。




```
Tcp/ip  io  cpu  memory
net.ipv4.tcp_syncookies = 1
#启用 syncookies
net.ipv4.tcp_max_syn_backlog = 8192
#SYN 队列长度
net.ipv4.tcp_synack_retries=2
#SYN ACK 重试次数
net.ipv4.tcp_fin_timeout = 30
#主动关闭方 FIN-WAIT-2 超时时间
net.ipv4.tcp_keepalive_time = 1200
#TCP 发送 keepalive 消息的频度
net.ipv4.tcp_tw_reuse = 1
#开启 TIME-WAIT 重用
net.ipv4.tcp_tw_recycle = 1
#开启 TIME-WAIT 快速回收
net.ipv4.ip_local_port_range = 1024 65000
#向外连接的端口范围
net.ipv4.tcp_max_tw_buckets = 5000
#最大 TIME-WAIT 数量，超过立即清除
net.ipv4.tcp_syn_retries = 2
#SYN 重试次数
echo "fs.file-max=65535" >> /etc/sysctl.conf
sysctl -p
```

http://www.haiyun.me/category/system/

Linux 下 IO 模型有几种，各自的含义是什么。

阻塞式 io，非阻塞 io，io 复用模型，信号驱动 io 模型，异步 io 模型。
https://yq.aliyun.com/articles/46404
https://yq.aliyun.com/articles/46402

epoll 和 poll 有什么区别。

select 的本质是采用 32 个整数的 32 位，即 32*32= 1024 来标识，fd 值为 1-1024。当 fd 的值超过 1024 限制时，就必须修改 FD_SETSIZE 的大小。这个时候就可以标识 32*max 值范围的 fd。
对于单进程多线程，每个线程处理多个 fd 的情况，select 是不适合的。
1.所有的线程均是从 1-32*max 进行扫描，每个线程处理的均是一段 fd 值，这样做有点浪费
2.1024 上限问题，一个处理多个用户的进程，fd 值远远大于 1024
所以这个时候应该采用 poll，
poll 传递的是数组头指针和该数组的长度，只要数组的长度不是很长，性能还是很不错的，因为 poll 一次在内核中申请 4K（一个页的大小来存放 fd），尽量控制在 4K 以内
epoll 还是 poll 的一种优化，返回后不需要对所有的 fd 进行遍历，在内核中维持了 fd 的列表。select 和 poll 是将这个内核列表维持在用户态，然后传递到内核中。但是只有在 2.6 的内核才支持。
epoll 更适合于处理大量的 fd ，且活跃 fd 不是很多的情况，毕竟 fd 较多还是一个串行的操作
https://yq.aliyun.com/articles/10525

平时用到哪些 Linux 命令。

Ls,find,tar,tail,cp,rm,vi，grep,ps,pkill 等等
https://yq.aliyun.com/articles/69417?spm=5176.100240.searchblog.18.Zrbh9R

用一行命令查看文件的最后五行。

Tail -n 5 filename

用一行命令输出正在运行的 java 进程。

ps -ef|grep Java

介绍下你理解的操作系统中线程切换过程。

控制权的转换，根据优先级切换上下文（用户，寄存器，系统）
http://www.cnblogs.com/kkshaq/p/4544426.html

进程和线程的区别。

Linux 实现并没有区分这两个概念（进程和线程）
1. 进程：程序的一次执行
2. 线程：CPU 的基本调度单位
一个进程可以包含多个线程。

http://www.ruanyifeng.com/blog/2013/04/processes_and_threads.html

多线程

多线程的几种实现方式，什么是线程安全。

实现 runable 接口，继承 thread 类。
http://ifeve.com/java-multi-threading-concurrency-interview-questions-with-answers/

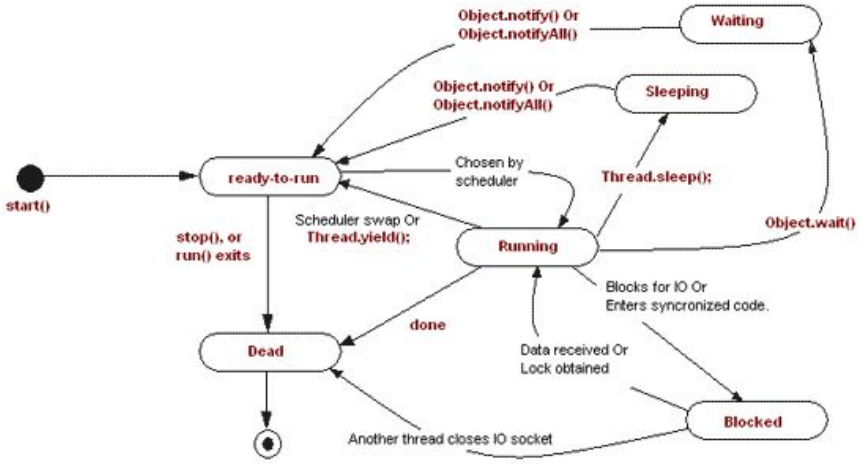
volatile 的原理，作用，能代替锁么。

Volatile 利用内存栅栏机制来保持变量的一致性。不能代替锁，其只具备数据可见性一致性，不具备原子性。
http://blog.csdn.net/gongzi2311/article/details/20715185

画一个线程的生命周期状态图。

新建，可运行，运行中， 睡眠，阻塞，等待，死亡。

http://ifeve.com/thread-status



sleep 和 wait 的区别。

Sleep 是休眠线程，wait 是等待，sleep 是 thread 的静态方法，wait 则是 object 的方法。
Sleep 依旧持有锁，并在指定时间自动唤醒。wait 则释放锁。
http://www.jianshu.com/p/4ec3f4b3903d

Lock 与 Synchronized 的区别。

首先两者都保持了并发场景下的原子性和可见性，区别则是 synchronized 的释放锁机制是交由其自身控制，且互斥性在某些场景下不符合逻辑，无法进行干预，不可人为中断等。
而 lock 常用的则有 ReentrantLock 和 readwritelock 两者，添加了类似锁投票、定时锁等候和可中断锁等候的一些特性。此外，它还提供了在激烈争用情况下更佳的性能。
http://blog.csdn.net/vking_wang/article/details/9952063

synchronized 的原理是什么，解释以下名词：重排序，自旋锁，偏向锁，轻量级锁，可重入锁，公平锁，非公平锁，乐观锁，悲观锁。

Synchronized 底层是通过监视器的 enter 和 exit 实现
https://my.oschina.net/cnarthurs/blog/847801
http://blog.csdn.net/a314773862/article/details/54095819

用过哪些原子类，他们的原理是什么。

AtomicInteger； AtomicLong； AtomicReference； AtomicBoolean；基于 *CAS* 原语实现 ，比较并交换、加载链接/条件存储，最坏的情况下是旋转锁
https://www.ibm.com/developerworks/cn/java/j-jtp11234/index.html
http://www.jmatrix.org/java/848.html

用过线程池吗，newCache 和 newFixed 有什么区别，他们的原理简单概括下，构造函数的各个参数的含义是什么，比如 coreSize，maxsize 等。

newSingleThreadExecutor 返回以个包含单线程的 Executor,将多个任务交给此 Exector 时，这个线程处理完一个任务后接着处理下一个任务，若该线程出现异常，将会有一个新的线程来替代。

newFixedThreadPool 返回一个包含指定数目线程的线程池，如果任务数量多于线程数目，那么没有没有执行的任务必须等待，直到有任务完成为止。

newCachedThreadPool 根据用户的任务数创建相应的线程来处理，该线程池不会对线程数目加以限制，完全依赖于 JVM 能创建线程的数量，可能引起内存不足。

底层是基于 `ThreadPoolExecutor` 实现，借助 `reentrantlock` 保证并发。

`coreSize` 核心线程数，`maxsize` 最大线程数。

http://ifeve.com/java-threadpoolexecutor/

线程池的关闭方式有几种，各自的区别是什么。

Shutdown shutdownNow tryTerminate 清空工作队列，终止线程池中各个线程，销毁线程池
http://blog.csdn.net/xxcupid/article/details/51993235

假如有一个第三方接口，有很多个线程去调用获取数据，现在规定每秒钟最多有 10 个线程同时调用它，如何做到。

`ScheduledThreadPoolExecutor` 设置定时，进行调度。

```
public ScheduledThreadPoolExecutor(int corePoolSize,
                                   ThreadFactory threadFactory) {
```

```
    super(corePoolSize, Integer.MAX_VALUE, 0, TimeUnit.NANOSECONDS,
          new DelayedWorkQueue(), threadFactory);
}
```

http://ifeve.com/java-scheduledthreadpoolexecutor/

spring 的 controller 是单例还是多例，怎么保证并发的安全。

单例
通过单例工厂 DefaultSingletonBeanRegistry 实现单例
通过保 AsyncTaskExecutor 持安全

用三个线程按顺序循环打印 abc 三个字母，比如 abcabcabc。

```
public static void main(String[] args) {
        final    String str="abc";
         ExecutorService executorService= Executors.newFixedThreadPool(3);
         executorService.execute(new Runnable() {
              @Override
              public void run() {
                   System.out.println("1"+str);
              }
         });executorService.execute(new Runnable() {
              @Override
              public void run() {
                   System.out.println("2"+str);
              }
         });executorService.execute(new Runnable() {
              @Override
```

```
                public void run() {
                    System.out.println("2"+str);
                }
            });
    }
```

ThreadLocal 用过么，用途是什么，原理是什么，用的时候要注意什么。

Threadlocal 底层是通过 threadlocalMap 进行存储键值 每个 ThreadLocal 类创建一个 Map，然后用线程的 ID 作为 Map 的 key，实例对象作为 Map 的 value，这样就能达到各个线程的值隔离的效果。

ThreadLocal 的作用是提供线程内的局部变量，这种变量在线程的生命周期内起作用，减少同一个线程内多个函数或者组件之间一些公共变量的传递的复杂度。

谁设置谁负责移除

http://qifuguang.me/2015/09/02/[Java%E5%B9%B6%E5%8F%91%E5%8C%85%E5%AD%A6%E4%B9%A0%E4%B8%83]%E8%A7%A3%E5%AF%86ThreadLocal/

如果让你实现一个并发安全的链表，你会怎么做。

Collections.synchronizedList() ConcurrentLinkedQueue

http://blog.csdn.net/xingjiarong/article/details/48046751

有哪些无锁数据结构，他们实现的原理是什么。

LockFree，CAS
基于 jdk 提供的原子类原语实现，例如 AtomicReference
http://blog.csdn.net/b_h_l/article/details/8704480

讲讲 java 同步机制的 wait 和 notify。

首先这两个方法只能在同步代码块中调用，wait 会释放掉对象锁，等待 notify 唤醒。
http://blog.csdn.net/ithomer/article/details/7685594

多线程如果线程挂住了怎么办。

根据具体情况（sleep,wait,join 等），酌情选择 notifyAll，notify 进行线程唤醒。

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

http://blog.chinaunix.net/uid-122937-id-215913.html

countdowlatch 和 cyclicbarrier 的内部原理和用法，以及相互之间的差别。

CountDownLatch 是一个同步辅助类，在完成一组正在其他线程中执行的操作之前，它运行一个或者多个线程一直处于等待状态。
CyclicBarrier 要做的事情是，让一组线程到达一个屏障（也可以叫同步点）时被阻塞，直到最后一个线程到达屏障时，屏障才会开门，所有被屏障拦截的线程才会继续运行。
CyclicBarrier 初始化的时候，设置一个屏障数。线程调用 await()方法的时候，这个线程就会被阻塞，当调用 await()的线程数量到达屏障数的时候，主线程就会取消所有被阻塞线程的状态。
前者是递减，不可循环，后者是递加，可循环用
countdowlatch 基于 abq cb 基于 ReentrantLock Condition
http://www.jianshu.com/p/a101ae9797e3
http://blog.csdn.net/tolcf/article/details/50925145

使用 synchronized 修饰静态方法和非静态方法有什么区别。

对象锁和类锁
https://yq.aliyun.com/articles/24226

简述 ConcurrentLinkedQueue LinkedBlockingQueue 的用处和不同之处。

LinkedBlockingQueue 是一个基于单向链表的、范围任意的（其实是有界的）、FIFO 阻塞队列。
ConcurrentLinkedQueue 是一个基于链接节点的无界线程安全队列，它采用先进先出的规则对节点进行排序，当我们添加一个元素的时候，它会添加到队列的尾部，当我们获取一个元素时，它会返回队列头部的元素。它采用了“wait－free”算法来实现，该算法在 Michael & Scott 算法上进行了一些修改, Michael & Scott 算法的详细信息可以参见参考资料一。
http://ifeve.com/concurrentlinkedqueue/
http://ifeve.com/juc-linkedblockingqueue/
http://blog.csdn.net/xiaohulunb/article/details/38932923

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

导致线程死锁的原因？怎么解除线程死锁。

死锁问题是多线程特有的问题，它可以被认为是线程间切换消耗系统性能的一种极端情况。在死锁时，线程间相互等待资源，而又不释放自身的资源，导致无穷无尽的等待，其结果是系统任务永远无法执行完成。死锁问题是在多线程开发中应该坚决避免和杜绝的问题。
一般来说，要出现死锁问题需要满足以下条件：
1. 互斥条件：一个资源每次只能被一个线程使用。
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。
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。
4. 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。
只要破坏死锁 4 个必要条件之一中的任何一个，死锁问题就能被解决。
https://www.ibm.com/developerworks/cn/java/j-lo-deadlock/

非常多个线程（可能是不同机器），相互之间需要等待协调，才能完成某种工作，问怎么设计这种协调方案。

此问题的本质是保持顺序执行。可以使用 executors

```
ExecutorService executorService= Executors.newFixedThreadPool(3);
executorService.execute(() -> {
        System.out.println("1"+str);
});executorService.execute(() -> {
        System.out.println("2"+str);
});executorService.execute(() -> { System.out.println("3"+str); });
```

TCP 与 HTTP

http1.0 和 http1.1 有什么区别。

HTTP 1.0 主要有以下几点变化：
请求和相应可以由于多行首部字段构成
响应对象前面添加了一个响应状态行
响应对象不局限于超文本
服务器与客户端之间的连接在每次请求之后都会关闭
实现了 Expires 等传输内容的缓存控制
内容编码 Accept-Encoding、字符集 Accept-Charset 等协商内容的支持
这时候开始有了请求及返回首部的概念，开始传输不限于文本（其他二进制内容）

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

HTTP 1.1 加入了很多重要的性能优化：持久连接、分块编码传输、字节范围请求、增强的缓存机制、传输编码及请求管道。
http://imweb.io/topic/554c5879718ba1240cc1dd8a

TCP 三次握手和四次挥手的流程，为什么断开连接要 4 次,如果握手只有两次，会出现什么。

* 第一次握手(SYN=1, seq=x):

客户端发送一个 TCP 的 SYN 标志位置 1 的包，指明客户端打算连接的服务器的端口，以及初始序号 X,保存在包头的序列号(Sequence Number)字段里。

发送完毕后，客户端进入 `SYN_SEND` 状态。

* 第二次握手(SYN=1, ACK=1, seq=y, ACKnum=x+1):

服务器发回确认包(ACK)应答。即 SYN 标志位和 ACK 标志位均为 1。服务器端选择自己 ISN 序列号，放到 Seq 域里，同时将确认序号(Acknowledgement Number)设置为客户的 ISN 加 1，即 X+1。
发送完毕后，服务器端进入 `SYN_RCVD` 状态。

* 第三次握手(ACK=1，ACKnum=y+1)

客户端再次发送确认包(ACK)，SYN 标志位为 0，ACK 标志位为 1，并且把服务器发来 ACK 的序号字段+1，放在确定字段中发送给对方，并且在数据段放写 ISN 的+1

发送完毕后，客户端进入 `ESTABLISHED` 状态，当服务器端接收到这个包时，也进入 `ESTABLISHED` 状态，TCP 握手结束。

第一次挥手(FIN=1，seq=x)

假设客户端想要关闭连接，客户端发送一个 FIN 标志位置为 1 的包，表示自己已经没有数据可以发送了，但是仍然可以接受数据。

发送完毕后，客户端进入 FIN_WAIT_1 状态。

第二次挥手(ACK=1，ACKnum=x+1)

服务器端确认客户端的 FIN 包，发送一个确认包，表明自己接受到了客户端关闭连接的请求，但还没有准备好关闭连接。

发送完毕后，服务器端进入 CLOSE_WAIT 状态，客户端接收到这个确认包之后，进入

FIN_WAIT_2 状态，等待服务器端关闭连接。

第三次挥手(FIN=1，seq=y)

服务器端准备好关闭连接时，向客户端发送结束连接请求，FIN 置为 1。

发送完毕后，服务器端进入 LAST_ACK 状态，等待来自客户端的最后一个 ACK。

第四次挥手(ACK=1，ACKnum=y+1)

客户端接收到来自服务器端的关闭请求，发送一个确认包，并进入 TIME_WAIT 状态，等待可能出现的要求重传的 ACK 包。

服务器端接收到这个确认包之后，关闭连接，进入 CLOSED 状态。

客户端等待了某个固定时间（两个最大段生命周期，2MSL，2 Maximum Segment Lifetime）之后，没有收到服务器端的 ACK ，认为服务器端已经正常关闭连接，于是自己也关闭连接，进入 CLOSED 状态。
两次后会重传直到超时。如果多了会有大量半链接阻塞队列。
https://segmentfault.com/a/1190000006885287
https://hit-alibaba.github.io/interview/basic/network/TCP.html

TIME_WAIT 和 CLOSE_WAIT 的区别。

TIME_WAIT 状态就是用来重发可能丢失的 ACK 报文。
TIME_WAIT 表示主动关闭，CLOSE_WAIT 表示被动关闭。

说说你知道的几种 HTTP 响应码，比如 200, 302, 404。

1xx：信息，请求收到，继续处理
2xx：成功，行为被成功地接受、理解和采纳
3xx：重定向，为了完成请求，必须进一步执行的动作
4xx：客户端错误，请求包含语法错误或者请求无法实现
5xx：服务器错误，服务器不能实现一种明显无效的请求
200 ok 一切正常
302 Moved Temporatily 文件临时移出
404 not found
https://my.oschina.net/gavinjin/blog/42856

当你用浏览器打开一个链接的时候，计算机做了哪些工作步骤。

Dns 解析-->端口分析-->tcp 请求-->服务器处理请求-->服务器响应-->浏览器解析--->链接关闭

TCP/IP 如何保证可靠性，说说 TCP 头的结构。

使用序号，对收到的 TCP 报文段进行排序以及检测重复的数据；使用校验和来检测报文段的错误；使用确认和计时器来检测和纠正丢包或延时。//TCP 头部，总长度 20 字节

```
typedef struct _tcp_hdr
{
    unsigned short src_port;     //源端口号
    unsigned short dst_port;     //目的端口号
    unsigned int seq_no;          //序列号
    unsigned int ack_no;          //确认号
    #if LITTLE_ENDIAN
    unsigned char reserved_1:4; //保留 6 位中的 4 位首部长度
    unsigned char thl:4;          //tcp 头部长度
    unsigned char flag:6;         //6 位标志
    unsigned char reseverd_2:2; //保留 6 位中的 2 位
    #else
    unsigned char thl:4;          //tcp 头部长度
    unsigned char reserved_1:4; //保留 6 位中的 4 位首部长度
    unsigned char reseverd_2:2; //保留 6 位中的 2 位
    unsigned char flag:6;         //6 位标志
    #endif
    unsigned short wnd_size;     //16 位窗口大小
    unsigned short chk_sum;      //16 位 TCP 检验和
    unsigned short urgt_p;       //16 为紧急指针
}tcp_hdr;
```

https://zh.bywiki.com/zh-hans/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE

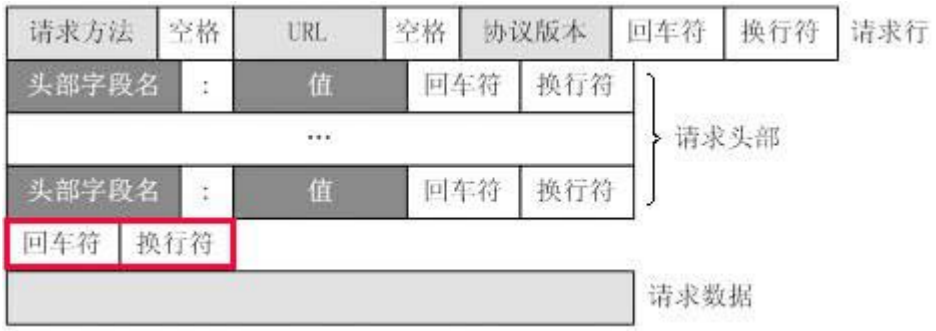
如何避免浏览器缓存。

无法被浏览器缓存的请求：
HTTP 信息头中包含 Cache-Control:no-cache，pragma:no-cache，或 Cache-Control:max-age=0

等告诉浏览器不用缓存的请求
需要根据 Cookie，认证信息等决定输入内容的动态请求是不能被缓存的
经过 HTTPS 安全加密的请求（有人也经过测试发现，ie 其实在头部加入 Cache-Control：max-age 信息，firefox 在头部加入 Cache-Control:Public 之后，能够对 HTTPS 的资源进行缓存，参考《HTTPS 的七个误解》）
POST 请求无法被缓存
HTTP 响应头中不包含 Last-Modified/Etag，也不包含 Cache-Control/Expires 的请求无法被缓存
http://www.alloyteam.com/2012/03/web-cache-2-browser-cache/

简述 Http 请求 get 和 post 的区别以及数据包格式。

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
书签	可收藏为书签	不可收藏为书签
缓存	能被缓存	不能缓存
编码类型	application/x-www-form-urlencoded	application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。
历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。 在发送密码或其他敏感信息时绝不要使用 GET！	POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。



http://www.w3school.com.cn/tags/html_ref_httpmethods.asp
http://www.360doc.com/content/12/0612/14/8093902_217673378.shtml

简述 HTTP 请求的报文格式。

参考上面

HTTPS 的加密方式是什么，讲讲整个加密解密流程。

加密方式是 tls/ssl，底层是通过对称算法，非对称，hash 算法实现
客户端发起 HTTPS 请求 --》2. 服务端的配置 --》
3. 传送证书 ---》4. 客户端解析证书 5. 传送加密信息 6. 服务段解密信息 7. 传输加密后的信息 8. 客户端解密信息
http://www.cnblogs.com/zhuqil/archive/2012/07/23/2604572.html

架构设计与分布式

常见的缓存策略有哪些，你们项目中用到了什么缓存系统，如何设计的。

Cdn 缓存，redis 缓存，ehcache 缓存等
Cdn 图片资源 js 等， redis 一主一从 echcache 缓存数据

用 java 自己实现一个 LRU。

```
final int cacheSize = 100;
Map<String, String> map = new LinkedHashMap<String, String>((int) Math.ceil(cacheSize / 0.75f) + 1, 0.75f, true) {
    @Override
    protected boolean removeEldestEntry(Map.Entry<String, String> eldest) {
    return size() > cacheSize;
    }
};
```

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

分布式集群下如何做到唯一序列号。

Redis 生成，mongodb 的 objectId，zk 生成

http://www.cnblogs.com/haoxinyue/p/5208136.html

设计一个秒杀系统，30 分钟没付款就自动关闭交易。

分流 -- 限流--异步--公平性（只能参加一次）--用户体验（第几位，多少分钟，一抢完）
容错处理
Redis 队列 mysql

30 分钟关闭 可以借助 redis 的发布订阅机制 在失效时进行后续操作，其他 mq 也可以
http://www.infoq.com/cn/articles/yhd-11-11-queuing-system-design

如何使用 redis 和 zookeeper 实现分布式锁？有什么区别优缺点，分别适用什么场景。

首先分布式锁实现常见的有数据库锁(表记录)，缓存锁，基于 zk（临时有序节点可以实现的）的三种

Redis 适用于对性能要求特别高的场景。redis 可以每秒执行 10w 次，内网延迟不超过 1ms
缺点是数据存放于内存，宕机后锁丢失。

锁无法释放？使用 Zookeeper 可以有效的解决锁无法释放的问题，因为在创建锁的时候，客户端会在 ZK 中创建一个临时节点，一旦客户端获取到锁之后突然挂掉（Session 连接断开），那么这个临时节点就会自动删除掉。其他客户端就可以再次获得锁。

非阻塞锁？使用 Zookeeper 可以实现阻塞的锁，客户端可以通过在 ZK 中创建顺序节点，并且在节点上绑定监听器，一旦节点有变化，Zookeeper 会通知客户端，客户端可以检查自己创建的节点是不是当前所有节点中序号最小的，如果是，那么自己就获取到锁，便可以执行业务逻辑了。

不可重入？使用 Zookeeper 也可以有效的解决不可重入的问题，客户端在创建节点的时候，把当前客户端的主机信息和线程信息直接写入到节点中，下次想要获取锁的时候和当前最小的节点中的数据比对一下就可以了。如果和自己的信息一样，那么自己直接获取到锁，如果不一样就再创建一个临时的顺序节点，参与排队。

单点问题？使用 Zookeeper 可以有效的解决单点问题，ZK 是集群部署的，只要集群中有半数以上的机器存活，就可以对外提供服务。

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

如果有人恶意创建非法连接，怎么解决。

可以使用 filter 过滤处理

分布式事务的原理，优缺点，如何使用分布式事务。

Two Phase commit 协议
优点是可以管理多机事务，拥有无线扩展性　确定是易用性难，承担延时风险
JTA，atomiks 等
https://yq.aliyun.com/webinar/join/185?spm=5176.8067841.0.0.RL4GDa

什么是一致性 hash。

一致性 hash 是一种分布式 hash 实现算法。满足平衡性 单调性 分散性 和负载。
http://blog.csdn.net/cywosp/article/details/23397179/

什么是 restful，讲讲你理解的 restful。

REST 指的是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是 RESTful。
http://baike.baidu.com/link?url=fTSAdL-EyYvTp9z7mZsCOdS3kbs4VKKAnpBLg3WS_1Z4cmLMp3S-zrjcy5wakLTO5AIoPTopWVkG-IenloPKxq

如何设计建立和保持 100w 的长连接。

服务器内核调优(tcp，文件数)，客户端调优，框架选择(netty)

如何防止缓存雪崩。

缓存雪崩可能是因为数据未加载到缓存中，或者缓存同一时间大面积的失效，从而导致所有请求都去查数据库，导致数据库 CPU 和内存负载过高，甚至宕机。

解决思路：

1，采用加锁计数，或者使用合理的队列数量来避免缓存失效时对数据库造成太大的压力。这种办法虽然能缓解数据库的压力，但是同时又降低了系统的吞吐量。

2，分析用户行为，尽量让失效时间点均匀分布。避免缓存雪崩的出现。

3，如果是因为某台缓存服务器宕机，可以考虑做主备，比如：redis 主备，但是双缓存涉及到更新事务的问题，update 可能读到脏数据，需要好好解决。

http://www.cnblogs.com/jinjiangongzuoshi/archive/2016/03/03/5240280.html

解释什么是 MESI 协议(缓存一致性)。

MESI 是四种缓存段状态的首字母缩写，任何多核系统中的缓存段都处于这四种状态之一。我将以相反的顺序逐个讲解，因为这个顺序更合理：

失效（Invalid）缓存段，要么已经不在缓存中，要么它的内容已经过时。为了达到缓存的目的，这种状态的段将会被忽略。一旦缓存段被标记为失效，那效果就等同于它从来没被加载到缓存中。
共享（Shared）缓存段，它是和主内存内容保持一致的一份拷贝，在这种状态下的缓存段只能被读取，不能被写入。多组缓存可以同时拥有针对同一内存地址的共享缓存段，这就是名称的由来。
独占（Exclusive）缓存段，和 S 状态一样，也是和主内存内容保持一致的一份拷贝。区别在于，如果一个处理器持有了某个 E 状态的缓存段，那其他处理器就不能同时持有它，所以叫“独占”。这意味着，如果其他处理器原本也持有同一缓存段，那么它会马上变成“失效”状态。
已修改（Modified）缓存段，属于脏段，它们已经被所属的处理器修改了。如果一个段处于已修改状态，那么它在其他处理器缓存中的拷贝马上会变成失效状态，这个规律和 E 状态一样。此外，已修改缓存段如果被丢弃或标记为失效，那么先要把它的内容回写到内存中——这和回写模式下常规的脏段处理方式一样。

说说你知道的几种 HASH 算法，简单的也可以。

哈希(Hash)算法,即散列函数。 它是一种单向密码体制,即它是一个从明文到密文的不可逆的映射,只有加密过程,没有解密过程。 同时,哈希函数可以将任意长度的输入经过变化以后得到固定长度的输出
MD4 MD5 SHA
http://blog.jobbole.com/106733/

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

什么是 paxos 算法。

Paxos 算法是莱斯利·兰伯特（Leslie Lamport，就是 LaTeX 中的"La"，此人现在在微软研究院）于 1990 年提出的一种基于消息传递的一致性算法。

http://baike.baidu.com/item/Paxos%20%E7%AE%97%E6%B3%95

什么是 zab 协议。

ZAB 是 Zookeeper 原子广播协议的简称

整个 ZAB 协议主要包括消息广播和崩溃恢复两个过程，进一步可以分为三个阶段，分别是：

发现 Discovery
同步 Synchronization
广播 Broadcast
组成 ZAB 协议的每一个分布式进程，都会循环执行这三个阶段，将这样一个循环称为一个主进程周期。
https://zzzvvvxxxd.github.io/2016/08/09/ZAB/

一个在线文档系统，文档可以被编辑，如何防止多人同时对同一份文档进行编辑更新。

点击编辑的时候，利用 redis 进行加锁 setNX 完了之后 expire 一下
也可以用版本号进行控制

线上系统突然变得异常缓慢，你如何查找问题。

逐级排查（网络，磁盘，内存，cpu），数据库，日志，中间件等也可通过监控工具排查。

说说你平时用到的设计模式。

单例， 代理，模板，策略，命令
http://www.jianshu.com/p/bdf65e4afbb0

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

Dubbo 的原理，数据怎么流转的，怎么实现集群，负载均衡，服务注册和发现。重试转发，快速失败的策略是怎样的。

Dubbo[]是一个分布式服务框架，致力于提供高性能和透明化的 RPC 远程服务调用方案，以及 SOA 服务治理方案。

Cluster 实现集群
在集群负载均衡时，Dubbo 提供了多种均衡策略，缺省为 random 随机调用。
Random LoadBalance：随机，按权重比率设置随机概率。
RoundRobin LoadBalance：轮循，按公约后的权重比率设置轮循比率。
LeastActive LoadBalance：最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。
ConsistentHash LoadBalance：一致性 Hash，相同参数的请求总是发到同一提供者。当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。
快速失败，只发起一次调用，失败立即报错。

https://my.oschina.net/u/1378920/blog/693374

一次 RPC 请求的流程是什么。

1）服务消费方（client）调用以本地调用方式调用服务；
2）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；
3）client stub 找到服务地址，并将消息发送到服务端；
4）server stub 收到消息后进行解码；
5）server stub 根据解码结果调用本地的服务；
6）本地服务执行并将结果返回给 server stub；
7）server stub 将返回结果打包成消息并发送至消费方；
8）client stub 接收到消息，并进行解码；
9）服务消费方得到最终结果。

异步模式的用途和意义。

异步模式使用与服务器多核，并发严重的场景
可提高服务吞吐量大，不容易受到冲击，可以采用并发策略，提高响应时间

缓存数据过期后的更新如何设计。

失效：应用程序先从 cache 取数据，没有得到，则从数据库中取数据，成功后，放到缓存中。

命中：应用程序从 cache 中取数据，取到后返回。
更新：先把数据存到数据库中，成功后，再让缓存失效。

编程中自己都怎么考虑一些设计原则的，比如开闭原则，以及在工作中的应用。

开闭原则（Open Close Principle）
　　一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。
里氏代换原则（Liskov Substitution Principle）
　　子类型必须能够替换掉它们的父类型。
依赖倒转原则（Dependence Inversion Principle）
　　高层模块不应该依赖低层模块，二者都应该依赖其抽象；抽象不应该依赖细节；细节应该依赖抽象。即针对接口编程，不要针对实现编程
接口隔离原则（Interface Segregation Principle）
　　建立单一接口，不要建立庞大臃肿的接口，尽量细化接口，接口中的方法尽量少
组合/聚合复用原则
　　说要尽量的使用合成和聚合，而不是继承关系达到复用的目的
迪米特法则（Law Of Demeter）
　　迪米特法则其根本思想，是强调了类之间的松耦合，类之间的耦合越弱,越有利于复用，一个处在弱耦合的类被修改，不会对有关系的类造成影响，也就是说，信息的隐藏促进了软件的复用。
单一职责原则（Single Responsibility Principle）
　　一个类只负责一项职责，应该仅有一个引起它变化的原因
http://www.banzg.com/archives/225.html

设计一个社交网站中的“私信”功能，要求高并发、可扩展等等。　画一下架构图。

MVC 模式，即常见的 MVC 框架。

SSM SSH SSI 等

聊了下曾经参与设计的服务器架构。

应用服务器怎么监控性能，各种方式的区别。

如何设计一套高并发支付方案，架构如何设计。

如何实现负载均衡，有哪些算法可以实现。

Zookeeper 的用途，选举的原理是什么。

Mybatis 的底层实现原理。

请思考一个方案，设计一个可以控制缓存总体大小的自动适应的本地缓存。

请思考一个方案，实现分布式环境下的 countDownLatch。

后台系统怎么防止请求重复提交。

可以通过 token 值进行防止重复提交，存放到 redis 中，在表单初始化的时候隐藏在表单中，添加的时候在移除。判断这个状态即可防止重复提交。

如何看待缓存的使用（本地缓存，集中式缓存），简述本地缓存和集中式缓存和优缺点。本地缓存在并发使用时的注意事项。

描述一个服务从发布到被消费的详细过程。

讲讲你理解的服务治理。

如何做到接口的幂等性。

算法

10 亿个数字里里面找最小的 10 个。

有 1 亿个数字，其中有 2 个是重复的，快速找到它，时间和空间要最优。

2 亿个随机生成的无序整数,找出中间大小的值。

给一个不知道长度的（可能很大）输入字符串，设计一种方案，将重复的字符排重。

遍历二叉树。

有 3n+1 个数字，其中 3n 个中是重复的，只有 1 个是不重复的，怎么找出来。

写一个字符串反转函数。

常用的排序算法，快排，归并、冒泡。 快排的最优时间复杂度，最差复杂度。冒泡排

序的优化方案。

二分查找的时间复杂度，优势。

一个已经构建好的 TreeSet，怎么完成倒排序。

什么是 B+树，B-树，列出实际的使用场景。

数据库知识

数据库隔离级别有哪些，各自的含义是什么，MYSQL 默认的隔离级别是是什么。

• 未提交读(Read Uncommitted)：允许脏读，也就是可能读取到其他会话中未提交事务修改的数据

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

• 提交读(Read Committed)：只能读取到已经提交的数据。Oracle 等多数数据库默认都是该级别 (不重复读)

• 可重复读(Repeated Read)：可重复读。在同一个事务内的查询都是事务开始时刻一致的，InnoDB 默认级别。在 SQL 标准中，该隔离级别消除了不可重复读，但是还存在幻象读

• 串行读(Serializable)：完全串行化的读，每次读都需要获得表级共享锁，读写相互都会阻塞

MYSQL 默认是 RepeatedRead 级别

MYSQL 有哪些存储引擎，各自优缺点。

MyISAM： 拥有较高的插入，查询速度，但不支持事务
InnoDB ： 5.5 版本后 Mysql 的默认数据库，事务型数据库的首选引擎，支持 ACID 事务，支持行级锁定
BDB： 源自 Berkeley DB，事务型数据库的另一种选择，支持 COMMIT 和 ROLLBACK 等其他事务特性
Memory ：所有数据置于内存的存储引擎，拥有极高的插入，更新和查询效率。但是会占用和数据量成正比的内存空间。并且其内容会在 Mysql 重新启动时丢失
Merge ：将一定数量的 MyISAM 表联合而成一个整体，在超大规模数据存储时很有用
Archive ：非常适合存储大量的独立的，作为历史记录的数据。因为它们不经常被读取。Archive 拥有高效的插入速度，但其对查询的支持相对较差
Federated： 将不同的 Mysql 服务器联合起来，逻辑上组成一个完整的数据库。非常适合分布式应用
Cluster/NDB ：高冗余的存储引擎，用多台数据机器联合提供服务以提高整体性能和安全性。适合数据量大，安全和性能要求高的应用
CSV： 逻辑上由逗号分割数据的存储引擎。它会在数据库子目录里为每个数据表创建一个.CSV 文件。这是一种普通文本文件，每个数据行占用一个文本行。CSV 存储引擎不支持索引。
BlackHole ：黑洞引擎，写入的任何数据都会消失，一般用于记录 binlog 做复制的中继
另外，Mysql 的存储引擎接口定义良好。有兴趣的开发者通过阅读文档编写自己的存储引擎。
http://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E5%BC%95%E6%93%8E

高并发下，如何做到安全的修改同一行数据。

使用悲观锁 悲观锁本质是当前只有一个线程执行操作，结束了唤醒其他线程进行处理。
也可以缓存队列中锁定主键。

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

乐观锁和悲观锁是什么，INNODB 的行级锁有哪 2 种，解释其含义。

乐观锁是设定每次修改都不会冲突，只在提交的时候去检查，悲观锁设定每次修改都会冲突，持有排他锁。
行级锁分为共享锁和排他锁两种　共享锁又称读锁 排他锁又称写锁
http://www.jianshu.com/p/f40ec03fd0e8

SQL 优化的一般步骤是什么，怎么看执行计划，如何理解其中各个字段的含义。

查看慢日志（show [session|gobal] status ），定位慢查询，查看慢查询执行计划　根据执行计划确认优化方案
Explain sql
select_type:表示 select 类型。常见的取值有 SIMPLE（简单表，即不使用连接或者子查询）、PRIMARY（主查询，即外层的查询）、UNION（union 中的第二个或者后面的查询语句）、SUBQUERY（子查询中的第一个 SELECT）等。
talbe：输出结果集的表。
type:表的连接类型。性能由高到底：system（表中仅有一行）、const（表中最多有一个匹配行）、eq_ref、ref、ref_null、index_merge、unique_subquery、index_subquery、range、idnex 等
possible_keys:查询时，可能使用的索引
key:实际使用的索引
key_len:索引字段的长度
rows：扫描行的数量
Extra：执行情况的说明和描述
http://blog.csdn.net/hsd2012/article/details/51106285

数据库会死锁吗，举一个死锁的例子，mysql 怎么解决死锁。

产生死锁的原因主要是：

（1）系统资源不足。
（2） 进程运行推进的顺序不合适。
（3）资源分配不当等。

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就很低，否则就会因争夺有限的资源而陷入死锁。其次，进程运行推进顺序与速度不同，也可能产生死锁。

产生死锁的四个必要条件：

（1） 互斥条件：一个资源每次只能被一个进程使用。
（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。
（3） 不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。
（4） 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。
这里提供两个解决数据库死锁的方法：

1）重启数据库（谁用谁知道）
2）杀掉抢资源的进程：
先查哪些进程在抢资源：SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX;
杀掉它们：Kill trx_mysql_thread_id；

MYsql 的索引原理，索引的类型有哪些，如何创建合理的索引，索引如何优化。

索引是通过复杂的算法，提高数据查询性能的手段。从磁盘 io 到内存 io 的转变
普通索引，主键，唯一，单列/多列索引建索引的几大原则
1.**最左前缀匹配原则**，非常重要的原则，mysql 会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如 a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d 是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d 的顺序可以任意调整。
2.**=和 in 可以乱序**，比如 a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql 的查询优化器会帮你优化成索引可以识别的形式
3.**尽量选择区分度高的列作为索引**,区分度的公式是 count(distinct col)/count(*)，表示字段不重复的比例，比例越大我们扫描的记录数越少，唯一键的区分度是 1，而一些状态、性别字段可能在大数据面前区分度就是 0，那可能有人会问，这个比例有什么经验值吗？使用场景不同，这个值也很难确定，一般需要 join 的字段我们都要求是 0.1 以上，即平均 1 条扫描 10 条记录
4.**索引列不能参与计算，保持列“干净”**，比如 from_unixtime(create_time) = ’2014-05-29’就不能使用到索引，原因很简单，b+树中存的都是数据表中的字段值，但进行检索时，需要把所有元素都应用函数才能比较，显然成本太大。所以语句应该写成 create_time = unix_timestamp(’2014-05-29’);
5.**尽量的扩展索引，不要新建索引**。比如表中已经有 a 的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

http://tech.meituan.com/mysql-index.html
http://www.cnblogs.com/cq-home/p/3482101.html

聚集索引和非聚集索引的区别。

“聚簇”就是索引和记录紧密在一起。
非聚簇索引 索引文件和数据文件分开存放，索引文件的叶子页只保存了主键值，要定位记录还要去查找相应的数据块。

数据库中 BTREE 和 B+tree 区别。

B+是 btree 的变种，本质都是 btree，btree+与 B-Tree 相比，B+Tree 有以下不同点：
每个节点的指针上限为 2d 而不是 2d+1。
内节点不存储 data，只存储 key；叶子节点不存储指针。

http://lcbk.net/9602.html

Btree 怎么分裂的，什么时候分裂，为什么是平衡的。

Key 超过 1024 才分裂 B 树为甚会分裂？ 因为随着数据的增多，一个结点的 key 满了，为了保持 B 树的特性，就会产生分裂，就向红黑树和 AVL 树为了保持树的性质需要进行旋转一样！

ACID 是什么。

A，atomic，原子性，要么都提交，要么都失败，不能一部分成功，一部分失败。
C，consistent，一致性，事物开始及结束后，数据的一致性约束没有被破坏
I，isolation，隔离性，并发事物间相互不影响，互不干扰。
D，durability,持久性，已经提交的事物对数据库所做的更新必须永久保存。即便发生崩溃，也不能被回滚或数据丢失。

Mysql 怎么优化 table scan 的。

避免在 where 子句中对字段进行 is null 判断
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。
避免在 where 子句中使用 or 来连接条件

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

in 和 not in 也要慎用
Like 查询（非左开头）
使用 NUM=@num 参数这种
where 子句中对字段进行表达式操作 num/2=XX
在 where 子句中对字段进行函数操作

如何写 sql 能够有效的使用到复合索引。

由于复合索引的组合索引，类似多个木板拼接在一起，如果中间断了就无法用了，所以要能用到复合索引，首先开头(第一列)要用上，比如 index(a,b) 这种，我们可以 select table tname where a=XX 用到第一列索引 如果想用第二列 可以 and b=XX 或者 and b like‘TTT%’

mysql 中 in 和 exists 区别。

mysql 中的 in 语句是把外表和内表作 hash 连接，而 exists 语句是对外表作 loop 循环，每次 loop 循环再对内表进行查询。一直大家都认为 exists 比 in 语句的效率要高，这种说法其实是不准确的。这个是要区分环境的。

如果查询的两个表大小相当，那么用 in 和 exists 差别不大。
如果两个表中一个较小，一个是大表，则子查询表大的用 exists，子查询表小的用 in：
not in 和 not exists 如果查询语句使用了 not in 那么内外表都进行全表扫描，没有用到索引；而 not extsts 的子查询依然能用到表上的索引。所以无论那个表大，用 not exists 都比 not in 要快。
1.EXISTS 只返回 TRUE 或 FALSE，不会返回 UNKNOWN。

2.IN 当遇到包含 NULL 的情况，那么就会返回 UNKNOWN。

数据库自增主键可能的问题。

在分库分表时可能会生成重复主键 利用自增比例达到唯一 自增 1 2,3 等
https://yq.aliyun.com/articles/38438

消息队列

用过哪些 MQ，和其他 mq 比较有什么优缺点，MQ 的连接是线程安全的吗，你们公司的 MQ 服务架构怎样的。

根据实际情况说明
我们公司用 activeMQ 因为业务比较简单 只有转码功能，而 amq 比较简单
如果是分布式的建议用 kafka
http://blog.csdn.net/sunxinhere/article/details/7968886

MQ 系统的数据如何保证不丢失。

基本都是对数据进行持久化，多盘存储

rabbitmq 如何实现集群高可用。

集群是保证服务可靠性的一种方式，同时可以通过水平扩展以提升消息吞吐能力。RabbitMQ 是用分布式程序设计语言 erlang 开发的，所以天生就支持集群。接下来，将介绍 RabbitMQ 分布式消息处理方式、集群模式、节点类型，并动手搭建一个高可用集群环境，最后通过 java 程序来验证集群的高可用性。

1. 三种分布式消息处理方式

RabbitMQ 分布式的消息处理方式有以下三种：

1、Clustering：不支持跨网段，各节点需运行同版本的 Erlang 和 RabbitMQ, 应用于同网段局域网。

2、Federation：允许单台服务器上的 Exchange 或 Queue 接收发布到另一台服务器上 Exchange 或 Queue 的消息, 应用于广域网，。

3、Shovel：与 Federation 类似，但工作在更低层次。

RabbitMQ 对网络延迟很敏感，在 LAN 环境建议使用 clustering 方式;在 WAN 环境中，则使用 Federation 或 Shovel。我们平时说的 RabbitMQ 集群，说的就是 clustering 方式，它是 RabbitMQ 内嵌的一种消息处理方式，而 Federation 或 Shovel 则是以 plugin 形式存在。
https://my.oschina.net/jiaoyanli/blog/822011
https://www.ibm.com/developerworks/cn/opensource/os-cn-RabbitMQ/

Redis，Memcached

redis 的 list 结构相关的操作。

LPUSH LPUSHX RPUSH RPUSHX LPOP RPOP BLPOP BRPOP LLEN LRANGE
https://redis.readthedocs.io/en/2.4/list.html

Redis 的数据结构都有哪些。

字符串(strings)：存储整数（比如计数器）和字符串（废话。。），有些公司也用来存储 json/pb 等序列化数据，并不推荐，浪费内存
哈希表(hashes)：存储配置，对象（比如用户、商品），优点是可以存取部分 key，对于经常变化的或者部分 key 要求 atom 操作的适合
列表(lists)：可以用来存最新用户动态，时间轴，优点是有序，确定是元素可重复，不去重
集合(sets)：无序，唯一，对于要求严格唯一性的可以使用
有序集合(sorted sets)：集合的有序版，很好用，对于排名之类的复杂场景可以考虑
https://redis.readthedocs.io/en/2.4/list.html

Redis 的使用要注意什么，讲讲持久化方式，内存设置，集群的应用和优劣势，淘汰策略等。

持久化方式：RDB 时间点快照 AOF 记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。
内存设置 maxmemory used_memory
虚拟内存： vm-enabled yes
3.0 采用 Cluster 方式，
Redis 集群相对单机在功能上存在一些限制， 需要开发人员提前了解，
在使用时做好规避。 限制如下：
1） key 批量操作支持有限。 如 mset、 mget， 目前只支持具有相同 slot 值的
ke
y 执
行批量操作。 对于映射为不同 slot 值的 key 由于执行 mget、 mget 等操作可
能存在于多个节点上因此不被支持。
2） key 事务操作支持有限。 同理只支持多 key 在同一节点上的事务操
作， 当多个 key 分布在不同的节点上时无法使用事务功能。
3） key 作为数据分区的最小粒度， 因此不能将一个大的键值对象如

ha
sh、 list 等映射到不同的节点。
4） 不支持多数据库空间。 单机下的 Redis 可以支持 16 个数据库， 集群模式下只能使用一个数据库空间， 即 db0。
5） 复制结构只支持一层， 从节点只能复制主节点， 不支持嵌套树状复制结构。
Redis Cluster 是 Redis 的分布式解决方案， 在 3.0 版本正式推出， 有效地解决了 Redis 分布式方面的需求。 当遇到单机内存、 并发、 流量等瓶颈时， 可以采用 Cluster 架构方案达到负载均衡的目的。 之前， Redis 分布式方案一般有两种：
• 客户端分区方案， 优点是分区逻辑可控， 缺点是需要自己处理数据路由、 高可用、 故障转移等问题。
• 代理方案， 优点是简化客户端分布式逻辑和升级维护便利， 缺点是加重架构部署复杂度和性能损耗。
现在官方为我们提供了专有的集群方案： Redis Cluster， 它非常优雅地解决了 Redis 集群方面的问题， 因此理解应用好 Redis Cluster 将极大地解放我们使用分布式 Redis 的工作量， 同时它也是学习分布式存储的绝佳案例。

LRU(近期最少使用算法)TTL（超时算法） 去除 ttl 最大的键值
http://wiki.jikexueyuan.com/project/redis/data-elimination-mechanism.html
http://www.infoq.com/cn/articles/tq-redis-memory-usage-optimization-storage
http://www.redis.cn/topics/cluster-tutorial.html

redis2 和 redis3 的区别，redis3 内部通讯机制。

集群方式的区别，3 采用 Cluster，2 采用客户端分区方案和代理方案
通信过程说明：
1） 集群中的每个节点都会单独开辟一个 TCP 通道， 用于节点之间彼此通信， 通信端口号在基础端口上加 10000。
2） 每个节点在固定周期内通过特定规则选择几个节点发送 ping 消息。
3） 接收到 ping 消息的节点用 pong 消息作为响应。

当前 redis 集群有哪些玩法，各自优缺点，场景。

当缓存使用 持久化使用

Memcache 的原理，哪些数据适合放在缓存中。

基于 libevent 的事件处理
内置内存存储方式 SLab Allocation 机制

并不单一的数据删除机制
基于客户端的分布式系统

变化频繁，具有不稳定性的数据,不需要实时入库, (比如用户在线状态、在线人数..)
门户网站的新闻等，觉得页面静态化仍不能满足要求，可以放入到 memcache 中.(配合 jquey 的 ajax 请求)

redis 和 memcached 的内存管理的区别。

Memcached 默认使用 Slab Allocation 机制管理内存，其主要思想是按照预先规定的大小，将分配的内存分割成特定长度的块以存储相应长度的 key-value 数据记录，以完全解决内存碎片问题。
Redis 的内存管理主要通过源码中 zmalloc.h 和 zmalloc.c 两个文件来实现的。
在 Redis 中，并不是所有的数据都一直存储在内存中的。这是和 Memcached 相比一个最大的区别。
http://lib.csdn.net/article/redis/55323

Redis 的并发竞争问题如何解决，了解 Redis 事务的 CAS 操作吗。

Redis 为单进程单线程模式，采用队列模式将并发访问变为串行访问。Redis 本身没有锁的概念，Redis 对于多个客户端连接并不存在竞争，但是在 Jedis 客户端对 Redis 进行并发访问时会发生连接超时、数据转换错误、阻塞、客户端关闭连接等问题，这些问题均是由于客户端连接混乱造成。对此有 2 种解决方法：

1.客户端角度，为保证每个客户端间正常有序与 Redis 进行通信，对连接进行池化，同时对客户端读写 Redis 操作采用内部锁 synchronized。

2.服务器角度，利用 setnx 实现锁。

MULTI，EXEC，DISCARD，WATCH 四个命令是 Redis 事务的四个基础命令。其中：

MULTI，告诉 Redis 服务器开启一个事务。注意，只是开启，而不是执行
EXEC，告诉 Redis 开始执行事务
DISCARD，告诉 Redis 取消事务
WATCH，监视某一个键值对，它的作用是在事务执行之前如果监视的键值被修改，事务会被取消。

可以利用 watch 实现 cas 乐观锁
http://wiki.jikexueyuan.com/project/redis/transaction-mechanism.html
http://www.jianshu.com/p/d777eb9f27df

Redis 的选举算法和流程是怎样的

Raft 采用心跳机制触发 Leader 选举。系统启动后，全部节点初始化为 Follower，term 为 0. 节点如果收到了 RequestVote 或者 AppendEntries，就会保持自己的 Follower 身份。如果一段时间内没收到 AppendEntries 消息直到选举超时，说明在该节点的超时时间内还没发现 Leader，Follower 就会转换成 Candidate，自己开始竞选 Leader。一旦转化为 Candidate，该节点立即开始下面几件事情：

1、增加自己的 term。
2、启动一个新的定时器。
3、给自己投一票。
4、向所有其他节点发送 RequestVote，并等待其他节点的回复。
如果在这过程中收到了其他节点发送的 AppendEntries，就说明已经有 Leader 产生，自己就转换成 Follower，选举结束。

如果在计时器超时前，节点收到多数节点的同意投票，就转换成 Leader。同时向所有其他节点发送 AppendEntries，告知自己成为了 Leader。

每个节点在一个 term 内只能投一票，采取先到先得的策略，Candidate 前面说到已经投给了自己，Follower 会投给第一个收到 RequestVote 的节点。每个 Follower 有一个计时器，在计时器超时时仍然没有接受到来自 Leader 的心跳 RPC, 则自己转换为 Candidate, 开始请求投票，就是上面的的竞选 Leader 步骤。

如果多个 Candidate 发起投票，每个 Candidate 都没拿到多数的投票（Split Vote），那么就会等到计时器超时后重新成为 Candidate，重复前面竞选 Leader 步骤。

Raft 协议的定时器采取随机超时时间，这是选举 Leader 的关键。每个节点定时器的超时时间随机设置，随机选取配置时间的 1 倍到 2 倍之间。由于随机配置，所以各个 Follower 同时转成 Candidate 的时间一般不一样，在同一个 term 内，先转为 Candidate 的节点会先发起投票，从而获得多数票。多个节点同时转换为 Candidate 的可能性很小。即使几个 Candidate 同时发起投票，在该 term 内有几个节点获得一样高的票数，只是这个 term 无法选出 Leader。由于各个节点定时器的超时时间随机生成，那么最先进入下一个 term 的节点，将更有机会成为 Leader。连续多次发生在一个 term 内节点获得一样高票数在理论上几率很小，实际上可以认为完全不可能发生。一般 1-2 个 term 类，Leader 就会被选出来。

Sentinel 的选举流程

Sentinel 集群正常运行的时候每个节点 epoch 相同，当需要故障转移的时候会在集群中选出 Leader 执行故障转移操作。Sentinel 采用了 Raft 协议实现了 Sentinel 间选举 Leader 的算法，

不过也不完全跟论文描述的步骤一致。Sentinel 集群运行过程中故障转移完成，所有 Sentinel 又会恢复平等。Leader 仅仅是故障转移操作出现的角色。

选举流程

1、某个 Sentinel 认定 master 客观下线的节点后，该 Sentinel 会先看看自己有没有投过票，如果自己已经投过票给其他 Sentinel 了，在 2 倍故障转移的超时时间自己就不会成为 Leader。相当于它是一个 Follower。
2、如果该 Sentinel 还没投过票，那么它就成为 Candidate。
3、和 Raft 协议描述的一样，成为 Candidate，Sentinel 需要完成几件事情
1）更新故障转移状态为 start
2）当前 epoch 加 1，相当于进入一个新 term，在 Sentinel 中 epoch 就是 Raft 协议中的 term。
3）更新自己的超时时间为当前时间随机加上一段时间，随机时间为 1s 内的随机毫秒数。
4）向其他节点发送 is-master-down-by-addr 命令请求投票。命令会带上自己的 epoch。
5）给自己投一票，在 Sentinel 中，投票的方式是把自己 master 结构体里的 leader 和 leader_epoch 改成投给的 Sentinel 和它的 epoch。
4、其他 Sentinel 会收到 Candidate 的 is-master-down-by-addr 命令。如果 Sentinel 当前 epoch 和 Candidate 传给他的 epoch 一样，说明他已经把自己 master 结构体里的 leader 和 leader_epoch 改成其他 Candidate，相当于把票投给了其他 Candidate。投过票给别的 Sentinel 后，在当前 epoch 内自己就只能成为 Follower。
5、Candidate 会不断的统计自己的票数，直到他发现认同他成为 Leader 的票数超过一半而且超过它配置的 quorum（quorum 可以参考《redis sentinel 设计与实现》）。Sentinel 比 Raft 协议增加了 quorum，这样一个 Sentinel 能否当选 Leader 还取决于它配置的 quorum。
6、如果在一个选举时间内，Candidate 没有获得超过一半且超过它配置的 quorum 的票数，自己的这次选举就失败了。
7、如果在一个 epoch 内，没有一个 Candidate 获得更多的票数。那么等待超过 2 倍故障转移的超时时间后，Candidate 增加 epoch 重新投票。
8、如果某个 Candidate 获得超过一半且超过它配置的 quorum 的票数，那么它就成为了 Leader。
9、与 Raft 协议不同，Leader 并不会把自己成为 Leader 的消息发给其他 Sentinel。其他 Sentinel 等待 Leader 从 slave 选出 master 后，检测到新的 master 正常工作后，就会去掉客观下线的标识，从而不需要进入故障转移流程。
http://weizijun.cn/2015/04/30/Raft%E5%8D%8F%E8%AE%AE%E5%AE%9E%E6%88%98%E4%B9%8BRedis%20Sentinel%E7%9A%84%E9%80%89%E4%B8%BELeader%E6%BA%90%E7%A0%81%E8%A7%A3%E6%9E%90/

redis 的持久化的机制，aof 和 rdb 的区别。

RDB 定时快照方式(snapshot)： 定时备份，可能会丢失数据
AOF 基于语句追加方式 只追加写操作
AOF 持久化和 RDB 持久化的最主要区别在于，前者记录了数据的变更，而后者是保存了数据本身

本文答案由 ricky 整理，不对之处还望指正，交流群：244930845

redis 的集群怎么同步的数据的。

redis replication redis-migrate-tool 等方式

搜索

elasticsearch 了解多少，说说你们公司 es 的集群架构，索引数据大小，分片有多少，以及一些调优手段。elasticsearch 的倒排索引是什么。

ElasticSearch（简称 ES）是一个分布式、Restful 的搜索及分析服务器，设计用于分布式计算；能够达到实时搜索，稳定，可靠，快速。和 Apache Solr 一样，它也是基于 Lucence 的索引服务器，而 ElasticSearch 对比 Solr 的优点在于：

轻量级：安装启动方便，下载文件之后一条命令就可以启动。

Schema free：可以向服务器提交任意结构的 JSON 对象，Solr 中使用 schema.xml 指定了索引结构。

多索引文件支持：使用不同的 index 参数就能创建另一个索引文件，Solr 中需要另行配置。

分布式：Solr Cloud 的配置比较复杂。

倒排索引是实现“单词-文档矩阵”的一种具体存储形式，通过倒排索引，可以根据单词快速获取包含这个单词的文档列表。倒排索引主要由两个部分组成：“单词词典”和“倒排文件”。

elasticsearch 索引数据多了怎么办，如何调优，部署。

使用 bulk API
初次索引的时候，把 replica 设置为 0
增大 threadpool.index.queue_size
增大 indices.memory.index_buffer_size
增大 index.translog.flush_threshold_ops
增大 index.translog.sync_interval
增大 index.engine.robin.refresh_interval
http://www.jianshu.com/p/5eeeeb4375d4

lucence 内部结构是什么

索引(Index)：
在 Lucene 中一个索引是放在一个文件夹中的。
如上图，同一文件夹中的所有的文件构成一个 Lucene 索引。
段(Segment)：
一个索引可以包含多个段，段与段之间是独立的，添加新文档可以生成新的段，不同的段可以合并。
如上图，具有相同前缀文件的属同一个段，图中共三个段 "_0" 和 "_1"和“_2”。
segments.gen 和 segments_X 是段的元数据文件，也即它们保存了段的属性信息。
文档(Document)：
文档是我们建索引的基本单位，不同的文档是保存在不同的段中的，一个段可以包含多篇文档。
新添加的文档是单独保存在一个新生成的段中，随着段的合并，不同的文档合并到同一个段中。
域(Field)：
一篇文档包含不同类型的信息，可以分开索引，比如标题，时间，正文，作者等，都可以保存在不同的域里。
不同域的索引方式可以不同，在真正解析域的存储的时候，我们会详细解读。
词(Term)：
词是索引的最小单位，是经过词法分析和语言处理后的字符串。