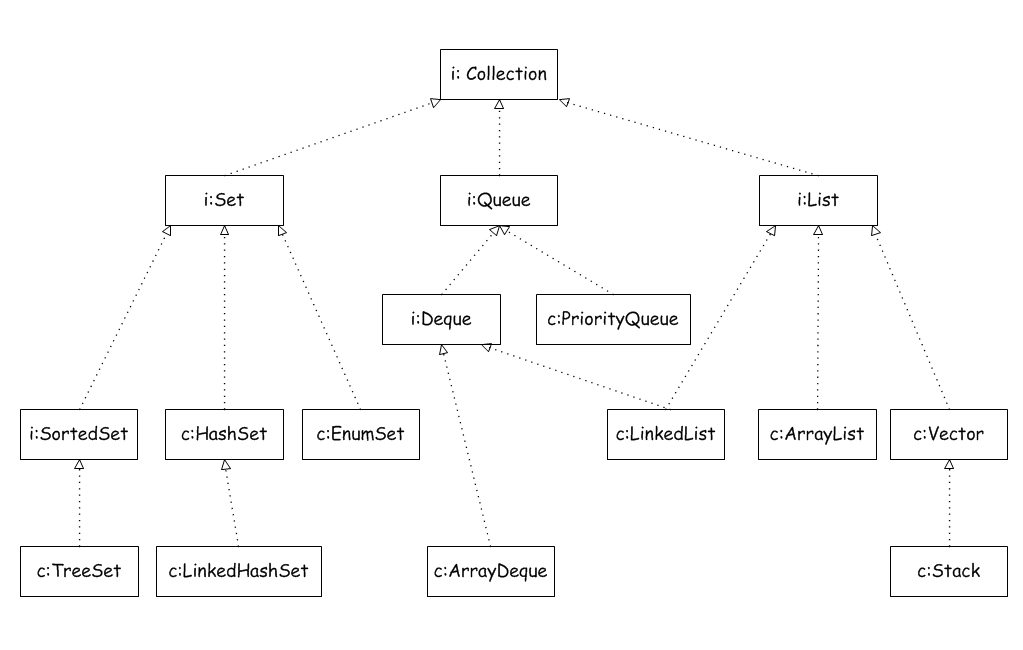
# 1、hashcode相等两个类一定相等吗?equals呢?相反呢?

答：**两个对象equals相等那么hashcode 是一定相等的**；**如果两个对象的hashCode相同，它们equals并不一定相同。所以二者必须同时重写。**

因为例如HasmMap中通过获取key的哈希值，然后通过hash方法计算出一个值，这个值作为比对的依据。也就是，**如果两个对象相等，那么他们的hashCode就一定要相等，不然这里这两个对象就会得到不同hash值从而存在不同的地方。**HashCode说白了是地址值经过一系列的复杂运算得到的结果，而Object中的equals方法底层比较的就是地址值，所以equals()相等，hashCode必定相等，反equals()不等，在java底层进行哈希运算的时候有一定的几率出现相等的hashCode,所以hashCode（）可等可不等。

# 2、介绍一下集合框架?



**1. Set子接口**：**不允许包含相同**的元素，而判断两个对象是否相同则是根据equals方法。

**1.1 HashSet类**

不能保证元素的排列顺序，加入的元素要特别注意hashCode()方法的实现。

HashSet不是同步的，多线程访问同一步HashSet对象时，需要手工同步。

集合元素值可以是null。

**1.2 LinkedHashSet类**

LinkedHashSet类也是根据元素的hashCode值来决定元素的存储位置，但它同时使用链表维护元素的次序。与HashSet相比，特点：

对集合迭代时，按增加顺序返回元素。性能略低于HashSet，因为需要维护元素的插入顺序。但迭代访问元素时会有好性能，因为它采用链表维护内部顺序。

**1.3 SortedSet接口及TreeSet实现类**

TreeSet类是SortedSet接口的实现类。**因为需要排序**，所以性能肯定差于HashSet。

可以定义比较器（Comparator）来实现自定义的排序。**默认自然升序排序。**

**2. List子接口：**：允许**重复**的元素，**有序**集合，所以与Set相比，增加了与索引位置相关的操作。

**2.1 ArrayList**

是数组结构，长度是可变的，原理是（创建新数组+复制数组），查询速度快，增删较慢，不同步。ArrayList不是线程安全的，只能用在单线程环境下，多线程环境下可以考虑使用Collections.synchronizedList（List i）函数返回一个线程安全的ArrayList类，或者使用Concurrent并发包下对应的集合类。

**2.2 Vector实现类**

基于数组实现的List类。Vector是线程安全的。但Vector的性能会比ArrayList低。

Vector提供一个子类Stack，可以挺方便的模拟“栈”这种数据结构（LIFO，后进先出）。

结论：不推荐使用Vector类，即使需要考虑同步，即也可以通过其它方法实现。

**2.3 LinkedList类**

是基于双向循环链表实现的，是链表结构，不同步的，增删速度快，查询较慢。由于实现了Queue接口，因此也可以用于实现堆栈、队列。

**3 Map**

**3.1 HashMap**

数据结构为哈希表，不同步，允许null作为键和值，无序的

**3.1.1 LinkedHashMap :**

类似于HashMap，但是迭代遍历它时，取得“键值对”的顺序是其插入次序，或者是最近最少使用(LRU)的次序。只比HashMap慢一点。而在迭代访问时发而更快，因为它使用链表维护内部次序。

**3.2 TreeMap**

基于红黑树数据结构的实现。查看“键”或“键值对”时，它们会被排序(次序由Comparabel或Comparator决定)。TreeMap的特点在于，你得到的结果是经过排序的。

**3.3 Hashtable**

数据结构为哈希表，同步的，不允许null作为键和值。

**3.3.1 Properties**：属性集，键和值都是字符串，可结合流进行键值的操作

# 3、hashmap hastable 底层实现什么区别?hashtable和concurrenthashtable呢?

Hashtable 也是一个散列表，它存储的内容是键值对(key-value)映射。Hashtable 的方法都是同步的(每个方法前面都加上了synchronized关键字)，这意味着它是线程安全的。**它的key、value都不可以为null**。此外，Hashtable中的映射不是有序的。

在JDK1.6，JDK1.7中，HashMap采用位桶+链表实现，即使用链表处理冲突，同一hash值的链表都存储在一个链表里。但是当位于一个桶中的元素较多，即hash值相等的元素较多时，通过key值依次查找的效率较低。而JDK1.8中，HashMap采用位桶+链表+红黑树实现，当链表长度超过阈值（8）时，将链表转换为红黑树，这样大大减少了查找时间。

# 4、hashmap和treemap什么区别?低层数据结构是什么?

Treemap：TreeMap的实现是红黑树算法的实现。因为TreeMap是有序的，通过comparator接口我们可以对TreeMap的内部排序进行精密的控制。

# 5、线程池用过吗都有什么参数?底层如何实现的?

# 6、sychnized和Lock什么区别?sychnize 什么情况情况是对象锁? 什么时候是全局锁为什么?

synchronized(this)以及非static的synchronized方法，只能防止多个线程同时执行同一个对象的同步代码段。**synchronized锁住的是括号里的对象，而不是代码。对于非static的synchronized方法，锁的就是对象本身也就是this。**

用synchronized(Sync.class)实现了全局锁的效果。

static synchronized方法，static方法可以直接类名加方法名调用，方法中无法使用this，所以它锁的不是this，而是类的Class对象，所以，static synchronized方法也相当于全局锁，相当于锁住了代码段。

# 7、ThreadLocal 是什么底层如何实现?写一个例子呗?

ThreadLocal类用来提供线程内部的局部变量。这些变量在多线程环境下访问(通过get或set方法访问)时能保证各个线程里的变量相对独立于其他线程内的变量，ThreadLocal实例通常来说都是private static类型。

ThreadLocal不是为了解决多线程访问共享变量，而是**为每个线程创建一个单独的变量副本**，每个线程都可以修改自己所拥有的变量副本, 而不会影响其他线程的副本. 其实这也是解决线程安全的问题的一种方法.

ThreadLocal是如何做到为每一个线程维护变量的副本的呢？其实实现的思路很简单：在ThreadLocal类中有一个静态内部类ThreadLocalMap(其类似于Map)，用键值对的形式存储每一个线程的变量副本，ThreadLocalMap中元素的key为当前ThreadLocal对象，而value对应线程的变量副本，每个线程可能存在多个ThreadLocal。

ThreadLocalMap中的每个节点Entry,其键key是ThreadLocal并且还是弱引用，这也导致了后续会产生内存泄漏问题的原因。每个thread中都存在一个map, map的类型是ThreadLocal.ThreadLocalMap. Map中的key为一个threadlocal实例. 这个Map的确使用了弱引用,**不过弱引用只是针对key.** 每个key都弱引用指向threadlocal. **当把threadlocal实例置为null以后,没有任何强引用指向threadlocal实例,所以threadlocal将会被gc回收.** 但是,我们的value却不能回收,因为存在一条从current thread连接过来的强引用. 只有当前thread结束以后, current thread就不会存在栈中,强引用断开, Current Thread, Map, value将全部被GC回收.所以得出一个结论就是只要这个线程对象被gc回收，就不会出现内存泄露，但在threadLocal设为null和线程结束这段时间不会被回收的，就发生了我们认为的内存泄露。比如使用线程池的时候，线程结束是不会销毁的，会再次使用的。就可能出现内存泄露。

public class TreadLocalTest {

static class Student {

private int age;

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

public static void main(String[] args) {

TestThreadLocal t = new TestThreadLocal();

new Thread(t, "Thread A").start();

new Thread(t, "Thread B").start();

}

static class TestThreadLocal implements Runnable {

ThreadLocal<Student> studentLocal = new ThreadLocal<Student>();

@Override

public void run() {

String currentThreadName = Thread.currentThread().getName();

System.out.println(currentThreadName + " is running...");

Random random = new Random();

int age = random.nextInt(100);

System.out.println(currentThreadName + " is set age: " + age);

Student s = getStudent(); //通过这个方法，为每个线程都独立的new一个student对象，

s.setAge(age);

System.out.println("current thread first get age " + currentThreadName + ":" + s.getAge());

}

public Student getStudent() {

Student s = (Student) studentLocal.get();

if (s == null) {

s = new Student();

studentLocal.set(s);

}

return s;

}

}

}

# 8、volitile的工作原理?

# 9、cas知道吗如何实现的?

# 10、请用至少四种写法写一个单例模式?

**JVM**

# 1、请介绍一下JVM内存模型??用过什么垃圾回收器都说说呗

2、线上发送频繁full gc如何处理? CPU 使用率过高怎么办?

如何定位问题?如何解决说一下解决思路和处理方法

3、知道字节码吗?字节码都有哪些?Integer x =5,int y =5，比较x =y 都经过哪些步骤?

4、讲讲类加载机制呗都有哪些类加载器，这些类加载器都加载哪些文件?

手写一下类加载Demo

5、知道osgi吗? 他是如何实现的???

6、请问你做过哪些JVM优化?使用什么方法达到什么效果???

7、classforName("java.lang.String")和String classgetClassLoader() LoadClass("java.lang.String") 什么区别啊??



JVM

**Spring**

1、spring都有哪些机制啊AOP底层如何实现的啊IOC呢??

2、cgLib知道吗?他和jdk动态代理什么区别?手写一个jdk动态代理呗?

**数据库**

1、使用mysq1索引都有哪些原则? ?索引什么数据结构? 3+tree 和B tree 什么区别?

2、mysq1有哪些存储引擎啊?都有啥区别? 要详细!

3、设计高并发系统数据库层面该怎么设计??数据库锁有哪些类型?如何实现呀?

4、数据库事务有哪些?

**分库分表**

1、如何设计可以动态扩容缩容的分库分表方案?

2、用过哪些分库分表中间件，有啥优点和缺点?讲一下你了解的分库分表中间件的底层实现原理?

3、我现在有一个未分库分表的系统，以后系统需分库分表，如何设计，让未分库分表的系统动态切换到分库分表的系统上???TCC? 那若出现网络原因，网络连不通怎么办啊???

4、分布式事务知道吗? 你们怎么解决的?

5、为什么要分库分表啊???

6、分布式寻址方式都有哪些算法知道一致性hash吗?手写一下java实现代码??你若userId取摸分片，那我要查一段连续时间里的数据怎么办???

7、如何解决分库分表主键问题有什么实现方案??

**分布式缓存**

1、redis和memcheched 什么区别为什么单线程的redis比多线程的memched效率要高啊?

2、redis有什么数据类型都在哪些场景下使用啊?

3、reids的主从复制是怎么实现的redis的集群模式是如何实现的呢redis的key是如何寻址的啊?

4、使用redis如何设计分布式锁?使用zk可以吗?如何实现啊这两种哪个效率更高啊??

5、知道redis的持久化吗都有什么缺点优点啊? ?具体底层实现呢?

6、redis过期策略都有哪些LRU 写一下java版本的代码吧??

**分布式服务框架**

1、说一下dubbo的实现过程注册中心挂了可以继续通信吗??

2、zk原理知道吗zk都可以干什么Paxos算法知道吗?说一下原理和实现??

3、dubbo支持哪些序列化协议?hessian 说一下hessian的数据结构PB知道吗为啥PB效率是最高的啊??

4、知道netty吗'netty可以干嘛呀NIO,BIO,AIO 都是什么啊有什么区别啊?

5、dubbo复制均衡策略和高可用策略都有哪些啊动态代理策略呢?

6、为什么要进行系统拆分啊拆分不用dubbo可以吗'dubbo和thrift什么区别啊?

**分布式消息队列**

1、为什么使用消息队列啊消息队列有什么优点和缺点啊?

2、如何保证消息队列的高可用啊如何保证消息不被重复消费啊

3、kafka ，activemq,rabbitmq ，rocketmq都有什么优点，缺点啊???

4、如果让你写一个消息队列，该如何进行架构设计啊?说一下你的思路

**分布式搜索引擎**

1、es的工作过程实现是如何的?如何实现分布式的啊

2、es在数据量很大的情况下( 数十亿级别)如何提高查询效率啊?

3、es的查询是一个怎么的工作过程?底层的lucence介绍一下呗倒排索引知道吗?es和mongdb什么区别啊都在什么场景下使用啊?

**高并发高可用架构设计**

1、如何设计一个高并发高可用系统

2、如何限流?工程中怎么做的，说一下具体实现

3、缓存如何使用的缓存使用不当会造成什么后果?

4、如何熔断啊?熔断框架都有哪些?具体实现原理知道吗?

5、如何降级如何进行系统拆分，如何数据库拆分????



分布式专题架构

**通信协议**

1、说一下TCP 'IP四层?

2、http的工作流程?? ?http1.0 http1.1http2.0 具体哪些区别啊?

3、TCP三次握手，四层分手的工作流程画一下流程图为什么不是四次五次或者二次啊?

4、画一下https的工作流程?具体如何实现啊?如何防止被抓包啊??

**算法**

1、比较简单，我一个文件，有45亿个阿拉伯数字，如何进行去重啊如何找出最大的那个数啊?

**数据结构**

1、二叉树和红黑树等。

**源码中所用到的经典设计思想及常用设计模式**



**福利**

这里给大家提供一个学习交流的平台，Java架构师群：726610841

1. 具有1-5工作经验的，面对目前流行的技术不知从何下手，需要突破技术瓶颈的可以加群。
2. 在公司待久了，过得很安逸，但跳槽时面试碰壁。需要在短时间内进修、跳槽拿高薪的可以加群。
3. 如果没有工作经验，但基础非常扎实，对java工作机制，常用设计思想，常用java开发框架掌握熟练的可以加群。

# Mybatis

## 1 代码自动生成器：generatorConfiguration

① .xml：包含数据库驱动器；数据库连接；生成实体类名，指定包名以及生成的地址；生成Dao文件；对应数据库表；

② 新建文件夹

③java -jar C:\Users\Laura\Desktop\mybatis-generator\mybatis-generator\mybatis-generator-core-1.3.1.jar -configfile C:\Users\Laura\Desktop\mybatis-generator\generatorBp.xml –overwrite

## 2 MyBatis关联查询、嵌套查询

Association

Collection

## 3 缓存使用场景及选择策略

Mybatis的一级缓存是指SqlSession。一级缓存的作用域是一个SqlSession。Mybatis默认开启一级缓存。在同一个SqlSession中，执行相同的查询SQL，第一次会去查询数据库，并写到缓存中；第二次直接从缓存中取。当执行SQL时两次查询中间发生了增删改操作，则SqlSession的缓存清空。

Mybatis的二级缓存是指mapper映射文件。二级缓存的作用域是同一个namespace下的mapper映射文件内容，多个SqlSession共享。Mybatis需要手动设置启动二级缓存。在同一个namespace下的mapper文件中，执行相同的查询SQL，第一次会去查询数据库，并写到缓存中；第二次直接从缓存中取。当执行SQL时两次查询中间发生了增删改操作，则二级缓存清空。

开启：mybatis文件：<setting name="cacheEnabled" value="true" />

在映射文件中，开启二级缓存：<cache/>

**多表操作一定不能使用缓存.**

## 4 Mybatis事务

## 5 MyBatis的动态代理

当定义好一个Mapper接口(UserDao)里，我们并不需要去实现这个类，但sqlSession.getMapper()最终会返回一个实现该接口的对象。这个对象是Mybatis利用jdk的动态代理实现的。这里将介绍这个代理对象的生成过程及其方法的实现过程。

①Mapper代码对象的生成过程：获取每个Mapper接口对应一个MapperProxyFactory对象实例，然后调用MapperRegistry.getMapper()方法，生成一个MapperProxy对象。

②MapperProxyFactory的newInstance方法：创建一个MapperProxy对象，这个方法实现了JDK动态代理中的InvocationHandler接口，说明Mapper接口被代理了，这样子返回的对象就是Mapper接口的子类，方法被调用时会被mapperProxy拦截,也就是执行mapperProxy.invoke()方法 。invoke方法会拦截Mapper接口(UserDao)的所有方法， MapperProxy会根据方法找到对应的MapperMethod对象来实现这次调用。

③MapperMethod对应会读取方法中的注解，从Configuration中找到相对应的MappedStatement对象，再执行。

# Spring

## 1 IOC容器设计原理及高级特性

答：(1)什么是IOC：所谓控制反转，就是把原先我们代码里面需要实现的对象创建、依赖的代码，反转给容器来帮忙实现。通过使用IoC容器，对象依赖关系的管理被反转了，对象之间的相互依赖关系由IoC容器进行管理，并由IoC容器完成对象的注入。Spring就是通过**反射**来实现注入的。具体的注入实现中，主要的注入方式有：接口注入、setter注入、构造器注入三种。Spring的IoC设计中，主要是setter注入和构造器注入方式；相对而言，使用Spring时常见的注入方式是：setter注入。

(2) Spring IOC体系结构：Spring Bean的创建是典型的工厂模式。①BeanFactory作为最顶层的一个接口类，它定义了IOC容器的基本功能规范；②XmlBeanFactory就是针对最基本的IoC容器的实现；③ApplicationContext是Spring提供的一个高级的IoC容器，它除了能够提供IoC容器的基本功能外，还为用户提供了以下的附加服务。

(3) IoC容器的初始化：**IoC容器的初始化由refresh()方法来启动**，其中包括BeanDefinition的Resource定位、载入和注册这三个基本的过程。FileSystemXmlApplicationContext容器时，构造方法做以下两项重要工作：首先，调用父类容器的构造方法为容器设置好Bean资源加载器。然后，再调用【父类AbstractRefreshableConfigApplicationContext】setConfigLocations(configLocations)方法设置Bean定义资源文件的定位路径。

(3.1) BeanDefinition的Resource定位，由**ResourceLoader通过统一的Resource接口完成**。

BeanDefinition载入是从refresh()函数开始的。refresh()是一个模板方法，refresh()方法的作用是：在创建IoC容器前，如果已经有容器存在，则需要把已有的容器销毁和关闭，以保证在refresh之后使用的是新建立起来的IoC容器。构造器中调用【父类】refresh方法，调用obtainFreshBeanFactory()，调用refreshBeanFactory()。

①Bean定义资源文件的载入从【子类AbstractRefreshableApplicationContext】**refreshBeanFactory** ()方法启动。这个方法先判断BeanFactory是否存在，如果存在则先销毁beans并关闭beanFactory；接着调用createBeanFactory()创建DefaultListableBeanFactory类型的IOC容器；然后调用loadBeanDefinitions(beanFactory)装载BeanDefinition【使用了一个委派模式，具体的实现调用子类容器】。

②【子类AbstractXmlApplicationContext】loadBeanDefinitions方法：创建XmlBeanDefinitionReader，即创建Bean读取器，并通过回调设置到容器中去；调用loadBeanDefinitions(XmlBeanDefinitionReader)，真正实现加载的方法；

③loadBeanDefinitions(XmlBeanDefinitionReader)：首先，调用getConfigResources【使用了一个委派模式，具体的实现调用子类容器】获取Bean定义资源的定位；其次，如果getConfigResources不为空，XmlBeanDefinitionReader调用【父类AbstractBeanDefinitionReader】loadBeanDefinitions()读取定位的Bean定义资源；然后，如果子类中获取的Bean定义资源定位为空，则调用getConfigLocations()获取FileSystemXmlApplicationContext构造方法中setConfigLocations方法设置的资源；最后，如果configLocations不为空，XmlBeanDefinitionReader调用【父类AbstractBeanDefinitionReader】loadBeanDefinitions()读取定位的Bean定义资源。FileSystemXmlApplicationContext作为例子分析， getConfigResources的返回值为null，因此程序执行reader.loadBeanDefinitions(configLocations)分支：调用XmlBeanDefinitionReader的loadBeanDefinitions()获取要加载的资源；

④AbstractBeanDefinitionReader#loadBeanDefinitions()：调用其父类DefaultResourceLoader的**getResource**方法获取要加载的资源【FileSystemXmlApplicationContext容器提供了getResourceByPath方法的实现，就是为了处理既不是classpath标识，又不是URL标识的Resource定位这种情况。】

(3.2) XmlBeanDefinitionReader加载BeanDefinition

①回到XmlBeanDefinitionReader的loadBeanDefinitions(Resource …)：调用的doLoadBeanDefinitions方法是从特定XML文件中实际载入Bean定义资源的方法，该方法在载入Bean定义资源之后将其转换为Document对象，解析过程由documentLoader实现。

②接下来调用registerBeanDefinitions启动Spring IoC容器对Bean定义的解析过程：

## 2 AOP设计原理

## 3 FactoryBean和BeanFactory

**答：BeanFactory**作为最顶层的一个接口类，它定义了IoC容器的基本功能规范。BeanFactory里只对IoC容器的基本行为作了定义，根本不关心你的bean是如何定义怎样加载的。在Spring中，所有的Bean都是由BeanFactory来进行管理的。

**FactoryBean**：在使用容器时，可以使用转义符“&”来得到FactoryBean本身，用来区分通过容器来获取FactoryBean产生的对象和获取FactoryBean本身。FactoryBean不是简单的Bean，是一个能产生或者修饰对象生成的工厂Bean，它的实现与设计模式中的工厂模式和修饰器模式类似。

## 4 Spring事务处理机制

## 5 SpringMVC九大组件

## 6 Spring5新特性