**堆排序**

**思想：**将待排序序列构造成一个大顶堆，此时，最大值就是根节点。然后将其与末尾的元素交换，此时末尾就是最大值。剩余的n-1个元素重新构造堆，再将根元素与末尾元素交换，反复执行。

初次构建大顶堆的时间复杂度为O(n)。之后需要n-1次交换，每次交换完成后，调整堆的复杂度为log2(n-1)、log2(n-2)、…、1逐步递减。所以，时间复杂度为O(nlogn)。

**写一个线程安全的单例**

利用静态内部类的特性，同时实现懒加载和线程安全

**public class** Singleton {  
 **private** Singleton() {}  
  
 **private static class** SingletonHolder {  
 **private static** Singleton *sInstance* = **new** Singleton();  
 }  
  
 **public** Singleton getInstance() {  
 **return** SingletonHolder.*sInstance*;  
 }  
}

**rabbitMQ交换器和路由的机制**

生产者生产消息，把消息发送给mq，mq通过Exchange将消息转发到订阅的消费者。

三种Exchange

1 Direct Exchange 任何发送到Direct Exchange的消息都会被转发到RouteKey中指定的Queue。

2 Fanout Exchange 任何发送到Fanout Exchange的消息都会被转发到与该Exchange绑定（Binding）的所有Queue上。

3 Topic Exchange 任何发送到Topic Exchange的消息都会被转发到所有关心RouteKey中指定话题的Queue上。

**mybatis配置多数据源**

可以配置多个数据源，然后为他们分别配置不同的mapper和事务，问题是这样就不能根据业务灵活配置。

解决方案是采用基于 **AbstractRoutingDataSource 和 AOP** 的多数据源配置

实现思想

配置多个数据源，并向AbstractRoutingDataSource的实现类中注入。注入对象targetDatasource，这是一个Map，key自己设定，用来标记不同的数据源，value即数据源。

通过Spring AOP指定key（key是一个静态变量，并且用ThreadLocal封装），通过AbstractRoutingDatasource的determineCurrentLookupKey方法实现对不同数据源的路由，这样就配置了在该线程中使用的数据源。

**分表策略**

比如根据用户id分表，假设有n张表，在存储的时候首先对id进行取模操作，根据id%n获取对应的表，再进行相关操作

Mybatis分表

在映射文件中，

<select id=”getOrder” resultMap=”xxx”>

select \* from order\_${tableNum}

where user\_id=#{userId}

</select>

在实际开发中，id更多通过UUID获取，这样，就需要先对其进行hash获取到整数，再进行取模操作。

Volatile

比synchronized关键字更轻量级的同步机制，JVM保证了每次读变量都在内存中读，跳过CPU cache这一步。具备两种特性：

1. 保证此变量对所有线程的可见性；
2. 禁止指令重排序优化，相当于加了一个内存屏障。

**NIO**

BIO（同步阻塞），客户端一个请求对应一个线程，阻塞体现在服务器接收客户端连接请求是阻塞的；同步体现在单线程请求调用read、write方法时，需等到读写操作完成才能返回。

NIO，（同步非阻塞），客户端的连接被注册到selector中，selector轮询注册的连接是否有IO请求，如果有，就在一个线程中处理该次请求。非阻塞体现在客户端的连接请求是非阻塞的；同步体现在单线程请求调用read、write方法时，需等到读写操作完成才能返回。

NIO2，不同于NIO的是读写IO时，方法调用后立即返回，数据先保存到缓冲区ByteBuffer，完成后调用回调方法。

**其他特性**

1. 通知机制，WatchService可以检测文件系统的修改、配置的变化等事件，相比与老式的轮询方式更加简单；
2. 内存映射（memory mapping），MappedByteBuffer，把文件的一段当作内存处理，大大提高I/O性能；
3. 字符编码，Charset，相比getByte更加灵活，在更底层实现，有更好的性能。

**BlockingQueue**

一个线程往里放，一个线程从里面取。当阻塞队列到达临界点，或者消费队列尝试从一个空的队列中提取对象时，产生阻塞。

阻塞队列已满，执行put(o)，产生阻塞，队列中的元素小于阈值时才会释放；

阻塞队列没有元素，执行take()，产生阻塞，队列中有可获取的元素时才会释放。

**ConcurrentMap**

效率优于HashTable，HashTable在操作某个元素时，会锁住所有元素；ConcurrentMap将元素分为若干个桶，操作某个元素时，只锁住对应的桶。

闭锁 CountDownLatch，等待一系列操作完成后再执行。

栅栏 CyclicBarrier，多个线程互相等待。

交换机 Exchanger，两个线程可以互相交换对象，阻塞，直到交换到信息。

**线程池**

ExecutorService三种模式

1. 自定义 new ThreadPoolExecutor(corePoolsize, maximumPoolSize, keepAliveTime, unit, workQueue, handler)
2. 小池子大队列，没有临时线程 Executors.newFixedThreadPool(10)
3. 大池子小队列，没有核心线程，最大线程数为int的最大值，销毁时间60s，Executors.newCachedPool()

Callable，相对于Runnable，能够抛出异常且有返回值，只能通过线程池调用。

**锁**

1. synchronized，同步代码块，公平锁，性能较低，原因：排队机制
2. Lock lock = new ReentrantLock(); 一般通过构造方法传递lock，加锁lock.lock()，释放锁lock.unlock()
3. ReadWriteLock 读写锁，将读写分开，进一步提高性能
4. AutmicInteger 原子性

APR (Apache Portable Runtime)是从操作系统级别来解决异步的IO问题,大幅度的提高性能.

**Spring依赖注入的三种方式**

1. 构造器注入
2. setter注入
3. 接口注入，具备侵入性，要求组件与特定的接口相关联

SSM框架简介

Spring是一个轻量级的控制反转和面向切面的容器框架。IOC，利用工厂模式将对象的创建交给容器（Spring大工厂），只需要配置相应的bean和属性注入；AOP，采用动态代理和静态织入的方式对方法进行装饰，通常时对方法的一个增强或者权限控制。

SpringMVC分离了控制器、视图和分派器，支持Spring的特性。

Mybatis，基于Java的持久层框架，使用简单的XML或者注解配置映射关系，将接口和JavaBean映射程数据库中的记录。

**Spring配置Task的方式**

可以看成一个轻量级的Quartz

1. 定义一个Task类，在xml文件中注册；
2. 将对象和方法注入到MethodInvokingTimerTaskFactoryBean；
3. 利用ScheduledTimerTask类配置task的参数；
4. 在Timer的工厂中，加入刚才配置好的task，TimerFactoryBean。

**单例的应用场景**

1. 网站计数器；
2. 线程池；
3. 数据库连接池；
4. 日志；
5. 配置对象的读取

优点：

资源共享，方便资源之间互相通信，对于需要频繁创建和销毁的对象可以提高性能。

缺点：

不适用于变化频繁的对象；滥用可能会导致因共享对象的程序过多出现溢出；实例化的对象长时间不被利用，会被回收，造成状态丢失。

**Redis应用场景**

Redis内部用一个redisObject来表示所有的key和value，主要信息包括数据类型type、encoding存储方式

1 String

最常用的数据类型，使用场景：粉丝数

get set incr decr mget

2 Hash

使用场景：用户信息

hget hset hgetall

3 List

使用场景：粉丝列表、日志收集器

lpush rpush lpop rpop lrange

4 Set

使用场景：常用来取交集，共同关注、共同喜好

sadd spop smembers sunion

5 Sorted Set

使用场景：有序并且不重复的集合列表，TOP N

zadd zrang zrem zcard

**slf4j的优势**

1. 在你的开源库或者私有库中使用slf4j，可以使它独立于任何日志实现，这就意味着不需要管理多个库或者日志文件；
2. 提供了占位日志记录，通过移除对isDebugEnabled()，isInfoEnabled()等等的检查提高了代码的可读性；
3. 通过使用日志记录方法，知道你使用到的时候，才会去构造日志信息（字符串），这就同时提高了内存和CPU的使用率；
4. 越少的临时字符串，垃圾回收器就意味着越少的工作，这就意味着为你的应用程序提供更好的吞吐量和性能。

**Matcher类**

matches 整个匹配，只有整个字符序列完全匹配成功，才返回True，否则返回False；

lookingAt 部分匹配，总是从第一个字符开始匹配，匹配成功了不再继续匹配，匹配失败了，也不继续匹配；

find 部分匹配，从当前位置开始匹配，找到匹配的字串，将移动下次匹配的位置；

reset 重置位置，如果有参数，则移动到指定的位置。

**Maven中dependency的scope作用域**

1 test测试范围有效，在编译和打包时不会使用这个依赖

2 compile 编译范围有效，编译和打包时会将依赖加进去

3 provided 在编译和测试过程有效，打包时不加入，如servlet-api、tomcat

4 runtime 运行时依赖，编译时不依赖

**Junit如何测试异常、超时**

测试超时

@Test(timeout = 1000)

测试异常

1)try catch 加断言

2)@Test(expected=Exception.class)

3)

@Rule

public ExpectedException expectedEx = ExpectedException.none();

在测试方法中添加

expectedEx.expect(InvalidPasswordException.class);//测试是否抛出异常类

expectedEx.expectMessage(“required”);//参数是Matcher或subString

**Spring和SpringMVC**

在项目中引入Spring和SpringMVC两个框架，Spring是父容器，SpringMVC是子容器。子容器可以看到父容器中注册的Bean，反之就不行。

<context:component-scan base-package=”com.xxx.xxx”/>

功能是扫描配置的base-package包下所有使用了@Component注解的类，它会自动开启以下配置

<context:annotation-config/>

功能是声明了@Required、@Autowired、@PostConstruct、@PersistenceContext、@Resource、@PreDestroy等注解。

SpringMVC需要下面的配置，声明@RequestMapping、@RequestBody、@ResponseBody

<mvc:annotation-driven/>

在Spring中配置扫描，SpringMVC中默认是看不到的，需要配置

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">

<property name="detectHandlerMethodsInAncestorContexts">

<value>true</value>

</property>

</bean>

实际工程中，推荐根据不同的业务模块来划分不同容器中注册不同的Bean，使他们各负其责、明确边界。方式如下

Spring配置

<!-- Spring容器中注册非@controller注解的Bean -->  
<context:component-scan base-package="com.hafiz.www">

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

SpringMVC配置

<!-- SpringMVC容器中只注册带有@controller注解的Bean -->  
<context:component-scan base-package="com.hafiz.www" use-default-filters="false">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Controller" />

</context:component-scan>

use-default-filters设置为false，禁止扫描@Service、@Reposity的Bean，防止SpringMVC把service、dao层的bean重新加载，引起覆盖。

**深拷贝和浅拷贝**

浅拷贝

通过覆盖Object类的clone()方法可以实现浅拷贝。当对象被复制时，只复制它本身和其中包含的值类型的成员变量，而引用类型的成员变量并没有被复制。

深拷贝

对象所包含的所有成员变量将被复制，实现方式两种：

1. 实现Clonable接口，并重写类中的clone()方法，成员变量中引用类型的对应的类也要实现Clonable接口；
2. 实现Serializable接口，通过对象的序列化和反序列化实现克隆，实现真正的深度克隆。

**Shiro配置RememberMe功能**

配置SimpleCookie的bean，设置有效时间等属性；将其注入到CookieRememberMeManager中去；将CookieRememberMeManager注入到Security Manager中去。就实现了记住我的功能。

问题：用户的session信息无法获取，解决办法，设置过滤器，拦截RememberMe功能的请求即可

过滤器需要继承FormAuthenticationFilter，判断subject.isAuthenticated()为false，subject.isRemembered()为true，session.getAttribute(“user”)==null时从subject中取出用户信息存到session中。

**注意，访问一般网页，使用user拦截器即可；访问特殊网页，使用authc拦截器。**

**Git的理解和用法**

远程仓库---git fetch/clone---本地仓库---git reset---缓存区---git checkout files---工作区间

工作区间---git add files---缓存区---git commit---本地仓库---git push---远程仓库

git pull = git fetch + git merge

如何解决冲突：

**Revert**

git checkout head . 放弃本地修改，然后git pull

**Resolve**

git stash 本地修改放入缓存

git pull 从远程仓库获取最新代码

git stash pop 取出本地修改的代码

解决冲突后，提交代码

**分支相关**

git branch dev 创建分支

git checkout dev 切换到分支，Head指向dev

dev开发完成后

git checkout master 切换到master

git merge dev 将dev分支的工作成果合并到master

git branch -d dev 删除分支

**实际开发中的分支策略**

master分支应该是稳定的，仅用于发布新版本

干活都在dev上，发布新版本时，再将dev合并到master上

开发人员都在dev分支上干活，每个人都有自己的分支，时不时往dev分支上合并就可以了。

**Linux命令**

lsof -i:8080 查看端口的使用情况

netstat -tunlp|grep 8080

ps 查看当前进程

id 查看用户id

man adduser adduser –help info adduser 查看指定帮助

rm 删除，-r子目录及子目录所有文件，-f

wc 统计，-c字节数，-l行数，-w字数

进程状态 D不可中断 S休眠 R正在运行 T停止或者追踪 Z僵尸状态 X死掉的进程

&在命令行尾，使进程在后台运行

ps -aux 显示所有进程

job -l 查看后台任务

find [path] -name “\*.txt” -print

df -hl 查看磁盘使用空间

env $HOME 查看所有env

du 显示文件或者目录的大小

**jsonp跨域请求**

在页面上直接发起一个跨域的ajax请求是不可以的，但是，在页面上引入不同域上的js脚本却是可以的。

使用jsonp

$(“#btn”).click(function() {

$.ajax({

[url:”http://xxx.xxx/xxx](url:)”,

type:”GET”,

datatype:”jsonp”,

success:function(data) {

var result = JSON.stringify(data);

$(“#text”).val(result);

}

});

});

可以通过jsonpCallback指定回调函数名称

jsonp只支持GET方式的请求，其原理和<script>脚本请求一样。

**get和post的区别**

get把参数包含在url里，post通过request body传递参数

get请求url有长度限制，post没有

get比post更不安全，因为参数直接暴露在url上，不能用来传递敏感信息

get产生一个TCP数据包，POST产生两个TCP数据包。对于get请求，浏览器会把http header和data一并发送出去，服务器响应200；对于post请求，浏览器先发送header，服务器返回100 continue，浏览器再发送data，服务器响应200 ok。（并不是发送一次就好，在网络差的环境下，两次包的TCP在验证数据的完整性上，有非常大的优点）

**SpringBoot项目在Linux下的部署**

1. jar

mvn clean package -Dmaven.test.skip=true

生成jar包，放在Linux中的任意位置，执行命令

java -jar Demo.jar

1. war

在pom.xml文件中将build节点中的finalName修改为项目名称

去除依赖中的tomcat模块

添加servlet依赖

修改启动类，添加方法SpringApplicationBuilder

mvn clean package -Dmaven.test.skip=true

将war包复制到tomcat下的webapps文件夹下

启动tomcat ./startup.sh

**SpringBoot启动流程**

ServletInitializer指定启动的Application类，Application类上添加注解@SpringBootApplication，并且执行SpringApplication.run()

加载配置文件 application.properties

自动装配 spring-boot-starter-web

加载组件 @Repository @Service @Controller @Component @Entity

应用初始化 启动tomcat、初始化数据源、连接池

**文件上传时，form表单的enctype需要设置为multipart/form-data**

**高并发应对策略**

1 数据库优化，M-S主从复制，读写分离；M-M-Slaves

2 CMS系统生成静态网页

3 图片服务器分离

4 数据库集群和库表散列

5 缓存

6 镜像

7 负载均衡

**过滤器和拦截器**

1 拦截器使用反射机制实现，拦截的是调用方法；过滤器使用Filter实现，拦截的是request对象，基于函数回调实现的；

2 过滤器依赖于servlet容器，拦截范围大；

3 拦截器只能对表现层请求起作用，过滤器对所有请求起作用；

4 拦截器可以访问action上下文、值栈里的对象，过滤器不能；

5 在action的生命周期中拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次；

6 拦截器可以获取IOC容器中的各个bean，而过滤器不行。

过滤器包裹住servlet，servlet包裹住拦截器

**拦截器实现方法：**

1)定义一个类，实现HandlerInterceptor接口，实现方法preHandle（如果返回false，请求将被终止）、postHandle、afterCompletion；（Object参数参考ModelAndView）

2)将拦截器注册到SpringMVC框架中

<mvc:interceptors>

<bean class=”xxx.xxxHandlerInterceptor”></bean>

</mvc:interceptors>

3)配置拦截规则<mvc:mapping path=”/xxx”/>

**SpringBoot实现拦截器的方法：**

与SpringMVC不同的是，需要在实现HandlerInterceptor的类上添加注解@Component；

在入口类的目录下创建一个类继承WebMvcConfigurerAdapter类，并重写addInterceptors方法，类上添加注解@SpringBootConfiguration表明这是一个配置类，将HandlerInterceptor注入进来，在addInterceptors方法中设置拦截规则：

registry.addInterceptor(xxxHandlerInterceptor).addPathPatterns(“/\*\*”);

**同步**

同步代码块可以选择获取哪个对象的对象锁，可以更加精确的控制锁的作用域，不要选择一个可能会在锁的作用域中改变值的实例对象作为锁对象；

同步方法所拥有的锁是该方法所属类的对象锁，即this对象，作用域是整个方法，不同实例对象的同步方法拥有的锁不一样；

静态同步方法拥有的锁是类对象，它于非静态同步方法之间不会有竞争条件。

**AQS**

ReentrantLock synchronized

都具有可重入性

synchronized依赖JVM，ReentranLock依赖于JDK

ReentrantLock性能更好

ReentrantLock需要自己加解锁，但是更加灵活

ReentrantLock可以指定公平锁、非公平锁

Condition类，可以分组唤醒需要唤醒的线程

ReentrantLock能够中断等待锁的线程机制

**InnoDB如何解决幻读的问题**

InnoDB的行锁分为

Record Lock：在索引上对单行记录加锁。

Gap Lock：锁定一个范围的记录，但不包括记录本身。锁加在未使用的空闲空间上，可能是两个索引记录之间，也可能是第一个索引记录之前或最后一个索引之后的空间。

Next-Key Lock：行锁和间隙锁组合起来就叫做Next-Key Lock。锁定一个范围，并且锁定记录本身。对于行的查询，都是采用该方法，主要目的是解决幻读的问题。

尽量使用唯一索引，因为唯一索引会把Next-Key Lock降级为Record Lock。

**RabbitMQ**

面向消息的中间件，主要用于组件之间的解耦，消息的发送者无需知道消息的使用者的存在，反之亦然。RabbitMQ基于Erlang，高并发、高可用。在可用、稳定、可靠性上，强于Kafka，Kafka的性能比RabbitMQ高。

channel.basicQos(1); 限制每次分发消息的数量

消费者可以设置是否自动回复，autoAck。如果为true，每次生产者只要发送消息就会从内存中删除，如果消费者程序异常退出，就无法获取数据；应该设置为false，手动回复，每次消费者收到并通知发送者，最后从队列中删除这条消息，如果消费者异常，mq会把队列中的消息发给其他消费者，如果没有，等消费者启动时再次发送。

**消息确认机制：**

生产者生成一个唯一的correlationId，发送时，添加两个属性：replyTo和correlationId，对于每一个响应消息，检查是否有correlationId与之匹配，匹配上更改状态。

**如何避免消息重复投递或消费？**

消息生产时，MQ内部对每条生产者发送的消息生成一个inner-msg-id，作为去重幂等的依据，避免重复的消息进入队列；消息消费时，通过correlationId作为去重幂等的依据。

**消息持久化**

将交换机\路由的durable属性设置为true。

当发送一条消息时，Rabbit会在消息提交到持久化日志文件后才发送响应。一旦消费了一条信息，把其标志为等待垃圾收集。未消费的消息，崩溃重启会进行重播。

**Java优化方案**

避免重复计算

懒加载

同步代码块

不用的对象及时置为null

不要再循环中使用try catch

容器类设置初始空间

**Mysql优化方案**

**第三方工具pt**

pt-query-digest分析慢查询日志，查什么？1)查询次数多且占用时间长2)IO大的SQL（Row examine扫描行数）3)未命中索引的SQL（Row examin和Row Send对比）

pt-index-usage分析慢查询日志，索引使用情况分析

pt-dumplicate-key-checker 查看重复及冗余索引

**索引优化**

where group by order by on 从句中出现的索引

索引字段越小越好

离散度大的放到联合索引的前面

索引不宜太多，会影响读写速度

**数据库结构优化**

使用最小数据类型，简单数据类型

尽可能not null

尽量不用text，用的话考虑分表

范式化，反范式化

垂直拆分

水平拆分

**系统配置优化**

开启文件数，使用硬件防火墙

配置my.cnf，缓冲池大小、缓冲池个数、缓冲区刷新到磁盘的策略、是否使用独立表空间、读写进程数

**使用Redis如何维护一个排行榜**

今日排行榜

用户积分增加 ZINCRBY rank:20181115 1 user1

取出top10 ZREVRANGE rank:20181115 0 9 withscores

上周排行榜

ZUNIONSTORE rank:last\_week 7 rank:20181105 rank:20181106 … rank:20181111 WEIGHTS 1 1 1 1 1 1 1

ZREVRANGE rank:last\_week 0 9 withscores

**Redis持久化**

RDB方式（默认），在指定的时间里，把数据写到磁盘里

save 900 1

save 300 10

save 60 10000

dbfilename dump.db

AOF方式，以日志的形式，记录每次操作。高可用

三种策略：每秒同步everysec，每修改同步always，不同步no

appendonly yes

appendfsync always/everysec/no

AOF日志文件超出一定限制，会进行重写，优化重复命令和可以合并的命令。好处是：减少AOF日志尺寸，减少内存占用，加快数据库修复时间。

**Redis主从复制**

Master/Slave 读写分离、容灾恢复

配从不配主

从库配置：slaveof 主库IP 主库端口

Slave只能读不能写，只要跟Master断开就需要重新slaveof，除非配置redis.conf

port 6379

pidfile /var/run/redis\_6379.pid

# slaveof <masterip> <masterport>

logfile “/data/logs/redis.master.log”

daemonize yes

常用三种方式：一主二仆 薪火相传 反客为主

手动反客为主 slaveof no one

复制原理：重连时，全量复制；运行过程中，增量复制。

哨兵模式（反客为主的自动版）：新建sentinel.conf文件

sentinel monitor 127.0.0.1 6379 1

复制的缺点：复制延迟

创建集群

redis-trib.rb create --replicas 1 ……

负载均衡，用哈希槽的方式实现，hash slot，将16384个槽分配到集群中的节点，当set一个key时，会用CRC16(key)%16384得到slot，分到对应的节点上。

**http和https的区别**

1 https协议需要CA申请证书

2 http是超文本传输协议，信息是明文传输；https则是具有安全性的ssl加密传输协议；

3 http端口号80，https端口号443

4 http连接很简单，无状态；https是由ssl+http构建的加密传输、身份认证的网络协议，比http协议安全

Session需要依赖Cookie作为识别标志，服务器向客户端发送一个名为JSESSIONID的Cookie，它的值为Session的ID。如果禁用Cookie，则使用URL地址重写的方式将用户Session的重写到url中。

**Session共享**

memcache把服务器中的内存组合起来，形成一个“内存池”。缺点是不能完全利用内存，会产生内存碎片，如果存储不足，还会产生内存溢出。

使用Redis配置流程:

配置分布式session过滤器DelegatingFilterProx，targetBean在Spring容器中定义

配置targetBean SessionRepositoryFilter，它默认使用构造器传参RedisOperationsSessionRepository，它使用构造器传参JedisConnectionFactory

**SSO**

子系统都有相同的父域名，将子系统cookie的domain设置为父域名，就可以让cookie能够被父域名下所有子系统访问。由于子系统可以访问父域下面的cookie，通过Redis实现Session共享即可实现，缺点是不能跨域名。