## jvm（类加载机制、jvm调优）

类加载机制：加载、链接（验证、准备、解析）、初始化、使用、卸载

加载：通过包名、类名获取定义这个类的二进制字节流

把二进制字节流的静态存储结构转化为方法区可运行的数据结构

在方法区分配类的内存空间，设置初始值

jvm调优：调整内存大小（jvm初始内存和最大内存）、GC策略（串行收集器、并行收集器、并发收集器）

## 多线程 (同步、锁、线程池)

线程状态：新建、就绪、运行、阻塞、死亡；

阻塞状态分为：等待阻塞、同步阻塞、其他阻塞

线程池： newCachedThreadPool、newFixedThreadPool

ExecutorService pool = Executors.newFixedThreadPool(10); 创建固定大小的线程池

## java基本框架（spring、mybatis的工作原理）

spring:

Spring是一个轻量级的IoC和AOP容器框架.

springIOC的注入方式：构造器注入、setter方法注入、根据注解注入

AOP:一般称为面向切面，用于将那些与业务无关，但却对多个对象产生影响的公共行为和逻辑，抽取并封装为一个可重用的模块,减少系统中的重复代码，

降低了模块间的耦合度,可用于权限认证、日志、事务处理。

AOP代理方式：静态代理、动态代理

静态代理：会在编译阶段将AspectJ(切面)织入到Java字节码中，运行的时候就是增强之后的AOP对象。

动态代理：AOP框架不会去修改字节码，而是每次运行时在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，

并回调原对象的方法。

@Autowired和@Resource之间的区别

(1)@Autowired默认是按照类型装配注入的，默认情况下它要求依赖对象必须存在。

(2)@Resource默认是按照名称装配注入的，只有当找不到与名称匹配的bean才会按照类型来装配注入。

BeanFactory与ApplicationContext的区别

BeanFactroy采用的是延迟加载形式来注入Bean的，即只有在使用到某个Bean时(调用getBean())，才对该Bean进行加载实例化。

ApplicationContext，它是在容器启动时，一次性创建了所有的Bean。

声明式事务与编程式事务的区别：

声明式事务：加上@Transactional注解就可以，最细粒度到方法级别

编程式事务：使用TransactionTemplate，最细粒度到方法快级别

mybatis的工作原理：mybatis应用程序通过SqlSessionFactoryBuilder从配置文件中构建出SqlSessionFactory，然后，SqlSessionFactory的实例直接开启一个SqlSession，

再通过SqlSession实例获得Mapper对象并运行Mapper映射的SQL语句，完成对数据库的CRUD和事务提交，之后关闭SqlSession。

## 数据库优化（数据库连接池）

dbcp、c3p0

优化：创建索引、不查\*、exists代替in

锁机制（乐观锁、悲观锁）

数据库的执行计划：

## 中间件

rabitMQ：

a:生产者 b:消费者

5种队列：简单模式（1--1） work模式（1-多）、发布-订阅模式（生产者首先将消息发送到交换机，交换机绑定多个队列，然后被消费）

路由模式：

主题模式：

生产者发送消息到队列 1.获取连接以及mq通道、2.从连接中创建通道、3.创建队列、4.存入信息、5.关闭通道和连接

消费者从消息队列消费 1.获取连接以及mq通道、2.从连接中创建通道、3.定义队列的消费者、4.监听队列、5.获取消息、6.关闭通道和连接

kafka:

生产：1.创建生产者者（声明集群地址，设置key、value的序列化，声明broker地址） 2.根据topic发送消息

消费：1.创建消费者（声明集群地址，设置key、value的序列化，声明broker地址，声明group） 2.通过topic获取信息

## spring是怎么注入bean的?

由Spring容器(比如 ApplicationContext)来实现,Spring启动时会把所有需要的bean扫描并注册到容器里，

在这个过程当中Spring会根据我们定义的bean之间的依赖关系来进行注入，依赖关系的维护方式有两种即XML配置文件或者注解，

Spring启动时会把这些依赖关系转化成Spring能够识别的数据结构BeanDefinition,并根据它来进行bean的初始化，依赖注入等操作。

## 数据库中索引为什么快？

本质是-平衡树（b树或b+树）索引中是包含一个表中列的值（或前缀的字符编码）和它的物理地址的值，通过二分法快速查找对应的值，通过地址查找到数据。

优势：查询快

劣势：创建、维护索引需要耗费时间；占用物理空间；对表的数据进行增删改时，索引也需要维护。

## jvm栈、堆、方法区各存什么?

栈用来存储基本类型变量和对象的引用变量的地址。

堆用来存储new出来的对象和数组。

方法区存储方法和static变量。

## 线程池的作用：减少内存开销；

线程池怎么加入线程：execute()方法；方法里新建线程

## redis的持久化：

将信息存到磁盘，防止关机或宕机信息丢失。RDB和AOF两种方式。

RDB：保存redis某个时间点上的数据集。 缺点：可能会丢失一些实时数据 (将Reids在内存中的数据库记录定时 dump到磁盘上的RDB持久化)

AOF: 设置fsync策略，可以每秒一次；或者每次执行写入命令时fsync; 保证实时数据。 缺点：AOF文件体积要大于RDB文件的体积，根据fsync策略，AOF速度会慢于RDB。

## jdk1.8特性：

lamdba表达式

default关键字：在接口中实现方法

Date API更新 : DateTimeFormatter.ofPattern("yyyy-MM-dd hh:mm:ss"); 配合LocalDateTime使用

调用方式： ：： 例如：A a = A::new; A::aa()[aa方法是A类中的类方法]

## HashMap的本质：是数组与链表的组合。

hashMap怎么put、get

put:首先通过key获取hash值，这个是数组的下标，把entry元素放到数组对应的位置，这个entry元素包括key、value、还有一个指针，指向下一个entry元素。（如果下标一致，在对应的位置向下插入，指针指向原有位置的entry）

get:通过key获取hash值，找到数组对应下标的位置，再根据key找到对应的元素。

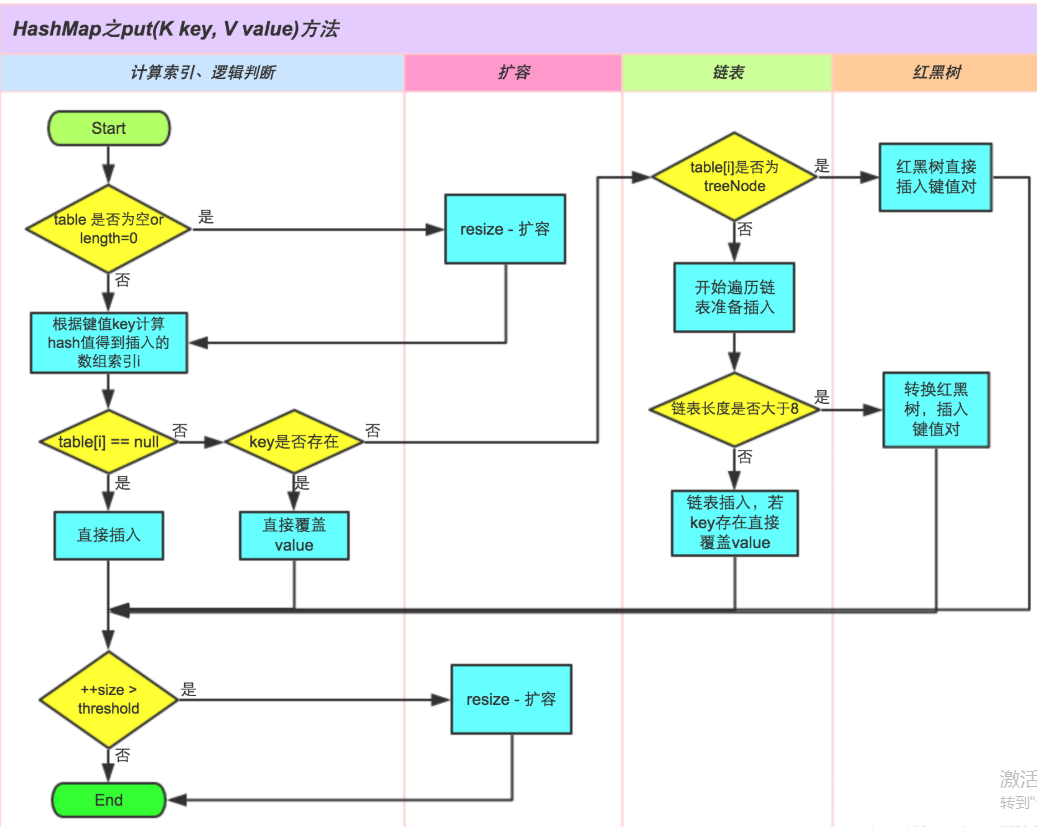
Jdk1.7：数组+链表

Jdk1.8：数组+链表/红黑树

当链表的长度大于8时，将链表转化成红黑树

hashMap源码：一个Node<K,V>，属性：int hash🡪代表数组下标位置，K key，V value，Node<K,V> next 指向的下一个节点

执行顺序：先获取key对应的hashcode值，在新建一个数组（长度为16），然后根据hash & (数组长度-1) 🡪与运算，获得数组的下标位置，将hashMap放在对应的位置，如果数组的该位置上有元素，比较key值是否相等，相等就直接覆盖value，不相等链表就增加一个元素，当链表元素大于8时，转换成红黑树；当数组长度大于12时，就执行resize方法进行扩容，新建长度为32的数组，根据上述方法得到hashmap对应的数组下标，依次对号入座就行。



HashMap与hashTable、concurrentHashMap的区别：

hashMap线程不安全；hashTable每个方法前都有 synchronized 关键字，线程安全，使用一把锁处理并发问题，当有多个线程访问时，需要多个线程竞争一把锁，导致阻塞；concurrentHashMap 中用cas算法实现乐观锁（compareAndSwapInt（目标值，原值，写入的新值）），cas是非阻塞算法的一种常见实现。

## redis设置了过期时间，过了过期时间后，会删除key吗？

aof是以记录命令方式，所以如果还没过期，那么不会有变更的命令，如果过期，会在aof日志插入一条del命令。

rdb的话是内存快照方式，如果持久化时，key已经过期，那么不会持久化，如果在过期之前就已经持久化了，那么在恢复数据时，会判断key是否过期，如果过期不会导入。

## 多线程各个参数的意义：

corePoolSize:最小线程数

maximumPoolSize：最大线程数

long keepAliveTime：线程最大的存活时间

workerQueue：缓存队列

先 达到 corePoolSize，然后 优先放入队列，然后在到MaxPollSize;然后拒绝。

4种拒绝策略：

AbortPolicy （中止） 直接抛出异常

DiscardPolicy （抛弃） 直接抛弃新任务，不执行

DiscardOldestPolicy （抛弃最旧） 抛弃优先级最高的任务，来执行新任务

CallerRunsPolicy （调用者运行）

## Springboot中的各个注解的解释：

@Service: 注解在类上，表示这是一个业务层bean

@Controller：注解在类上，表示这是一个控制层bean

@Repository: 注解在类上，表示这是一个数据访问层bean

@Component： 注解在类上，表示通用bean ，value不写默认就是类名首字母小写

@Autowired：按类型注入.

@Configuration

@Resource默认按照名称方式进行bean匹配。

@RestController与@Controller的区别

@RestController注解相当于@ResponseBody ＋ @Controller合在一起的作用。直接返回json数据，无法返回页面

@Controller可以返回指定的jsp或html页面，要返回json数据的话，要加上@ResponseBody注解。

## Tomcat的配置：

修改内存大小：

Windows下是在bin目录的catalina.bat文件中，添加

JAVA\_OPTS='-Xms256m -Xmx512m'，初始内存是256m，最大内存是256m

Linux下是修改bin目录下的catalina.sh文件。

## Springboot中怎么进行面向切面的使用：

写一个类，上面写上

*@Aspect注解*

*@Component*

对包下的所有类都进行切面控制

*@Pointcut("execution (\* com.topsec.hg..controller..\*.\*(..)) && !execution(\* com.topsec.hg..controller.AuditController.\*(..))")  
public void webLog() {  
}*

在方法中写上环绕

*@Around("webLog()")*

Spring AOP @Around @Before @After的区别

@Before是在所拦截方法执行之前执行一段逻辑。@After 是在所拦截方法执行之后执行一段逻辑。@Around是可以同时在所拦截方法的前后执行一段逻辑。

Aspect类内部的advice按照以下的顺序进行执行

@Around--@before—Method--@Around--@After--@AfterReturning

## 日志级别：

优先级从高到低

Error—Warn—Info--Debug

Logback.xml的配置：

<appender><filter><level></level></filter></>标签和<root><appender-ref ref=’appender里的name’></>标签

## 对springboot-starter的理解：

作用吧：

一键搞定所依赖的jar包，只需写 比如：mybatis-spring-boot-starter

自动加入mybatis的所有依赖，并且解决版本冲突

自动装配：

Springboot项目启动时，starter会自动加载所有的bean(类上标有@Configuration)。

## Es与关系型数据库的区别

## Hashcode与equals的区别

Hashcode相等，equals不一定相等

Equals相等，说明是同一个对象；hashcode不一定相等

为了便于好记，hashcode相当于人名，人相当于对象；equals相等，是同一个人，名字相同；名字相同的人有多个。

## springCloud分布式

eureka：服务注册中心

eureka-server：注册server服务

eureka-client：注册client服务（通过RestTemplate调用server服务的controller）

zuul：配置路由服务，相当于nginx中的反向代理

ribbon：负载均衡的作用，多个服务端可以服务名相同，端口号不同，发布到服务注册中心会显示同一行的两个。

config-server：用来从git/svn上下载文件内容，发布到eureka。

config-client：从config-server中获取在远端的配置文件内容。

## nginx

反向代理配置：

server {

listen 80;

server\_name localhost;

location / {

proxy\_pass http://OrdinaryPolling;

index index.html index.htm index.jsp;

}

}

通过自定义的url（host:port）,映射到服务的url以及开始页面，这样地址栏输入自定义的url，直接跳转到真正服务的页面。

## spring中的各个模式的使用场景

1. 单例模式

单个实例引用，单例模式有且只有一个类实例，并且提供唯一的访问入口。不应该由人来控制，该由代码来限制，强制单例。

使用场景：Spring中的bean都是单例模式 ，数据库连接池一般也采用单例模式。

优势：避免了对共享资源的多重占用。对于一些需要频繁创建和销毁的对象来说可以提高系统的性能。

2.代理模式：

代理模式（Proxy）为另一个对象提供一个替身或占位符以控制对这个对象的访问，其实用最简单的方式来解释代理模式，那就是中介，客户不想直接引用对象，这个“中介”也就是代理对象就起到了一定的作用。在客户端和目标端之间起到一定的作用。

避免直接调用对象，客户端通过代理类调用来调用目标端的方法，并对此方法进行增强。

静态代理与动态代理：

静态代理（实现方式）：目标类与代理类同时实现一个接口，在代理类中实例化目标类，并调用增强后的方法。

jdk动态代理（实现方式）：代理类通过实现InvocationHandler 接口，重写invoke 方法，来实现调用增强后的方法。

jdk动态代理总结：通过类加载器拿到真实实现类和真实实现类接口的字节码文件，通过代理类来调用真是实现类的方法。

3.工厂模式：

使用场景：

1．工厂模式的目的是为了实现解耦，将对象的创建和使用分开，即应用程序将对象的创建和初始化职责交给工厂对象。

2．降低代码重复。如果对象B的创建过程比较复杂，并且很多地方都用到了，那么很可能出现很多重复的代码，通过统一将创建对象B的代码放到工厂里面统一管理，可以减少代码的重复率，同时也方便维护。

在spring 中可以用工厂模式创建 bean ，用Factory-bean 动态工厂创建bean

Spring使用工厂模式配置bean的三种方法

1. 通过静态工厂方法 StaticCarFactory。
2. 通过实例工厂方法 InstanceCarFactory。
3. 通过实现 FactoryBean 接口，通过factorybean 来配置bean。

## Git 与 SVN 的区别

Git是分布式的，svn不是分布式的。

Git 可以不依赖网络做任何事情，分支是存在本地的，只要不合并主干，可以随便玩；

Svn 必须依赖网络才能提交，分支是存在远端的。