## jvm

### 1.1 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）

stack overflow：栈内存溢出

栈溢出解决方案：  
方案1.调整栈内变量内存分配，限制在1M内  
方案2.调整程序栈大小

### 1.2 Jvm常用内存参数设置：

-Xms ：初始堆内存大小

-Xmx：最大堆内存大小

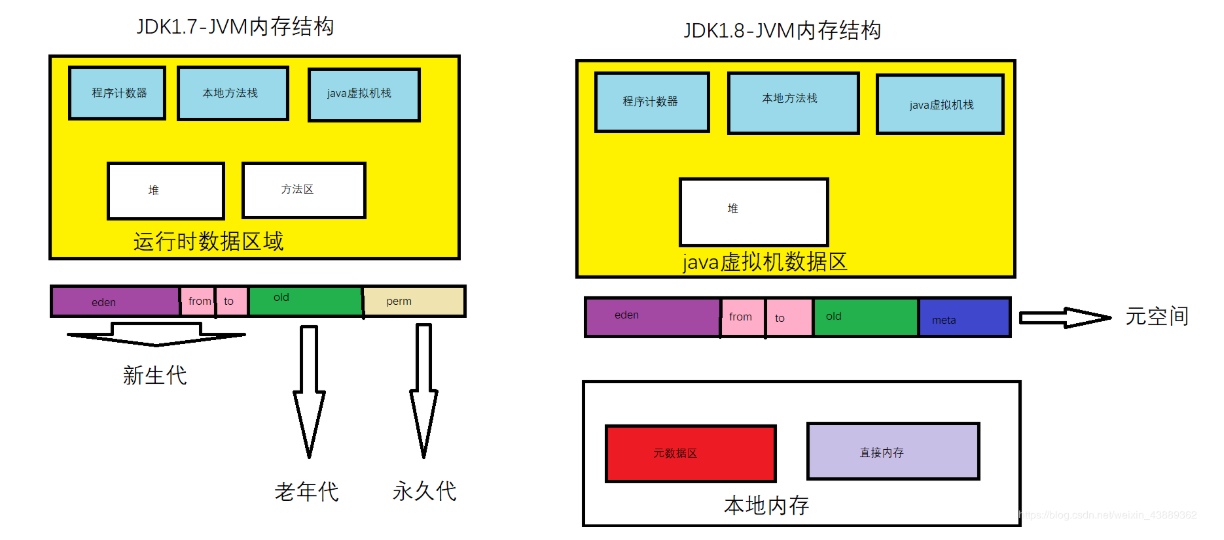
-Xmn：堆中Young区大小

-Xss：java每个线程的Stack(栈)大小

-XX:PermSize：持久代（方法区）的初始内存大小。

-XX:MaxPermSize：持久代（方法区）的最大内存大小。

### 1.3 Jvm在jdk1.7与jdk1.8的区别



**从图中我们可以看到，jdk1.7相较jdk1.8多了一个方法区（一般这么说，其实是永久代），而jdk1.8则多了一个元空间。**

首先，我们应该了解方法区只是JVM规范定义（即jdk1.7和jdk1.8都有方法区这个概念），而永久代和元空间则是方法区的实现

**永久代和元空间的区别：**

永久代：在运行时数据区域开辟空间实现方法区

元空间：在本地内存区域开辟空间实现方法区

**为什么要移除永久代呢？**

永久代中的元数据信息在每次FullGc时可能被收集，为永久代分配多少空间很难确定，超出指定空间容易造成内存泄漏

**元空间的特点：**

1.类及相关的元数据的生命周期与类加载器的一致

2.每个加载器有专门的存储空间

3.只进行线性分配

4.不会单独回收某个类

5.省掉了GC扫描及压缩的时间

6.元空间里的对象的位置是固定的

如果GC发现某个类加载器不再存活了，会把相关的空间整个回收掉

### 1.4 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。

### 1.5 GC分类

针对HotSpot VM的实现，它里面的GC其实准确分类只有两大种：

Partial GC：并不收集整个GC堆的模式

Young GC：只收集young gen的GC。

Old GC：只收集old gen的GC。只有CMS的concurrent collection是这个模式。

Mixed GC：收集整个young gen以及部分old gen的GC。只有G1有这个模式。

Full GC：收集整个堆，包括young gen、old gen、perm gen（如果存在的话）等所有部分的模式。

## 线程

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

了解的线程池：ExecutorService pool = Executors.newFixedThreadPool(10);

线程池中主线程与子线程的执行顺序：

先执行完子线程，再执行主线程。全部的子线程都执行完，主线程再关闭。

线程开启的是start方法，实际上执行的是run方法。

多线程同步：

synchronized 关键字

互斥锁

## 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

## 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

## 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）

## 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。

使用构造器注入的好处：

1. 保证依赖不可变（final关键字）
2. 保证依赖不为空（省去了我们对其检查）
3. 保证返回客户端（调用）的代码的时候是完全初始化的状态
4. 避免了循环依赖

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。

## 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。

## Spring事务

Spring事务其实说的是数据库事务

一般采用声明式事务，也就是注解 **@Transactional ，用在service层**

**@Transactional** 放在类上和方法上的区别：

方法上的**@Transactional 会覆盖类上的 @Transactional**

**@Transactional 的参数解释：**

@Transactional(propagation=Propagation.REQUIRED) : 有事务就处于当前事务中，没事务就创建一个事务

@Transactional(propagation=Propagation.NOT\_SUPPORTED)针对某个方法不开启事务

isolation=Isolation.DEFAULT：事务[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库)的默认隔离级别

readOnly=false：可写 针对 增删改操作

readOnly=true：查询使用只读

## Hibernate

一级缓存与二级缓存：

 一级缓存：基于Session的缓存，缓存内容只在当前session有效，session关闭，缓存内容失效。作用范围较小！ 缓存的事件短。

二级缓存：（SessionFactory缓存）

Hibernate提供了基于应用程序级别的缓存即为二级缓存，可以跨多个session，即不同的session都可以访问缓存数据。

注：session的生命周期(session什么时候会失效)

　 1. 服务器会把长时间没有活动的Session从服务器内存中清除，此时Session便失效。Tomcat中Session的默认失效时间为30分钟。

2. 调用Session的invalidate方法。

## Mybatis

Mybatis的一级、二级缓存：

一级缓存基于sqlSession默认开启,在操作数据库时需要构造SqlSession对象，在对象中有一个HashMap用于存储缓存数据。如果sqlSession执行了DML操作，并且提交到数据库，MyBatis则会清空SqlSession中的一级缓存。

当一个SqlSession结束后该SqlSession中的一级缓存也就不存在了。

二级缓存是mapper级别的缓存。使用二级缓存时，多个SqlSession使用同一个Mapper的sql语句去操作数据库，得到的数据会存在二级缓存区域，它同样是使用HashMap进行数据存储。相比一级缓存SqlSession，二级缓存的范围更大，多个Sqlsession可以共用二级缓存，二级缓存是跨SqlSession的。

## Mysql

Mysql的执行引擎：

innoDB：

优点：InnoDB是一个事务型的存储引擎, 提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别。

缺点：不支持全文索引，而且它没有保存表的行数，当SELECT COUNT(\*) FROM TABLE时需要扫描全表。

Mysql事务的隔离级别：



Mysql的事务特性：

原子性：原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生；

一致性：事务前后数据的完整性必须保持一致；

隔离性：事务的隔离性是指多个用户并发访问数据库时，一个用户的事务不能被其它用户的事务所干扰，多个并发事务之间数据要相互隔离；

持久性：持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响。

Mysql中的脏读、幻读

脏读：

脏读发生在一个事务A读取了被另一个事务B修改，但是还未提交的数据。假如B回退，则事务A读取的是无效的数据

幻读：

幻读发生在当两个完全相同的查询执行时，第二次查询所返回的结果集跟第一个查询不相同。

不可重复读与幻读的区别：

不可重复读：不可重复读的重点是修改: 同样的条件, 你读取过的数据, 再次读取出来发现值不一样了。

幻读：幻读的重点在于新增或者删除 (数据条数变化)。同样的条件, 第1次和第2次读出来的记录数不一样。

Mysql默认的隔离级别是 可重复读；oracle默认的隔离级别是 未提交。

## 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：

## 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。

## Ehcache

Element与java程序是绑在一起的，java程序活着，它就活着。譬如，写一个独立程序放数据，再写一个独立程序拿数据，那么是拿不到数据的。只能在独立程序中才能拿到数据。

CacheManager：是缓存管理器，可以通过单例或者多例的方式创建，也是Ehcache的入口类。  
Cache：每个CacheManager可以管理多个Cache，每个Cache可以采用hash的方式管理多个Element。  
Element：用于存放真正缓存内容的。

Ehcache的缓存数据淘汰策略:

LFU：最少被使用，缓存的元素有一个hit属性，hit值最小的将会被清出缓存。

LRU：最近最少使用，缓存的元素有一个时间戳，当缓存容量满了，而又需要腾出地方来缓存新的元素的时候，那么现有缓存元素中时间戳离当前时间最远的元素将被清出缓存。

Ehcache的缓存数据过期策略:

Ehcache采用的是懒淘汰机制，每次往缓存放入数据的时候，都会存一个时间，在读取的时候要和设置的时间做TTL比较来判断是否过期。

存储类型：序列化的对象

## Redis

Redis属于独立的运行程序，需要单独安装后，使用JAVA中的Jedis来操纵。

Redis的存储类型：String,List,hash,set,sort set

Redis的优势：单线程，执行速度快

Redis的持久化：

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；保证实时数据。(原理是将Reids的操作日志—主要是写操作以追加的方式写入文件)

Redis设置过期时间，过了过期时间，key会被删除吗？

aof是以记录命令（写操作）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。

Redis与ehcache的区别：

**redis是通过socket访问到缓存服务，效率比ecache低，比数据库要快很多，处理集群和分布式缓存方便，有成熟的方案。**

**ehcache直接在jvm虚拟机中缓存，速度快，效率高；但是缓存共享麻烦，集群分布式应用不方便。**

Redis的删除策略：

被动删除：key过期后不会立即删除，再次访问此key的时候，会检查是否过期，过期即删除。

主动删除：key过期后立即删除，这样对cpu压力比较大。

定时删除：定时统一删除过期的key。这个是折中方案。

## 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。

## 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

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中数组长度为什么是16？

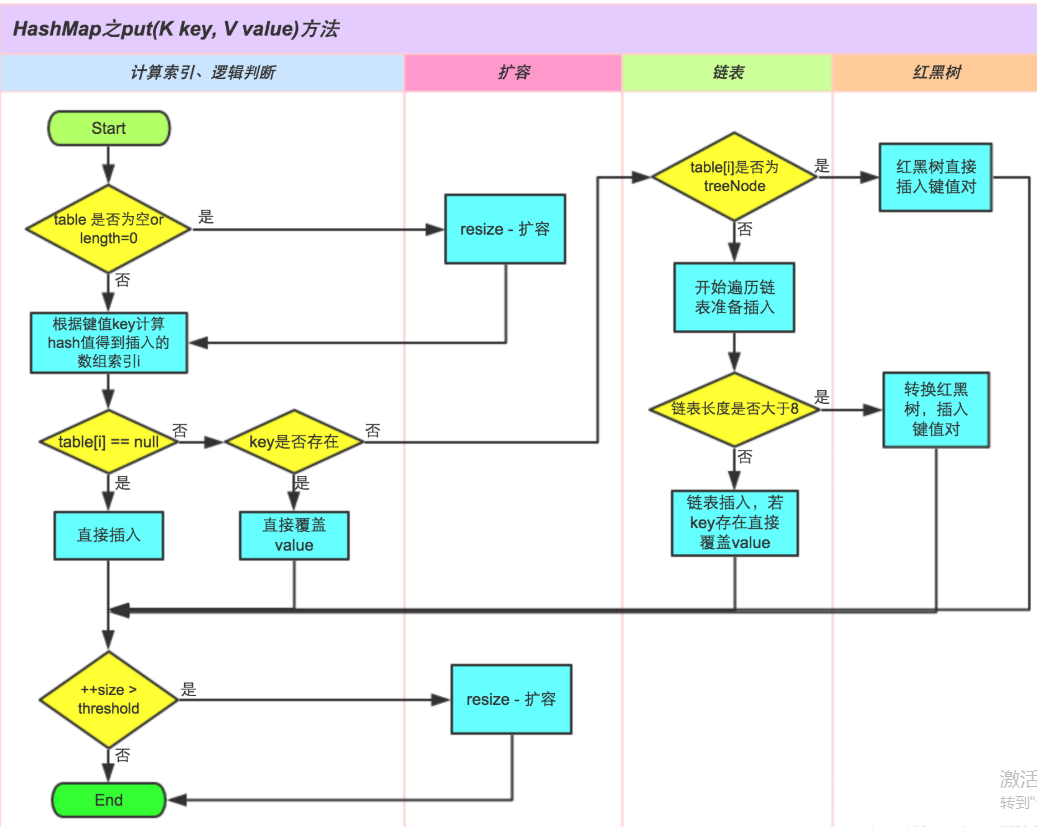
总结： 1 减少hash碰撞 (通过key的hashcode值和数组长度-1的与运算得到数组下标)

         2 提高map查询效率

         3 分配过小防止频繁扩容

         4 分配过大浪费资源

链表长度为8时变成红黑树，长度为6时又变成链表。



HashMap与hashTable、concurrentHashMap的区别：

hashMap线程不安全；初始化容量16；键值对可以为**空**；性能高；位运算符；最大扩容2(30)。

hashTable线程安全；初始化容量**11**；键值对不能为空；每个方法前都有 synchronized 关键字，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。

## Shiro

shiro的权限控制执行流程: application -----> subject -----> securityManager -----> realm -----> 安全数据;。

要控制按钮级别的权限的话，Jsp页面可以使用shiro标签；前后端分离的话可以把本用户对应角色的权限拿到前端进行比较。

Springboot中通过ShiroFilterFactoryBean来创建shiroFilter ,对访问的url 的访问进行控制。

Shiro集成单点：

搭建Cas-server服务器，在cas-client(实际上就是shiro项目)，配置登录路径，直接跳转到cas-server中登录，返回一个带有user信息的ticket，cas-client根据ticket信息进行授权。

## 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

## 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实实现类的方法。

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。

## Linux常用命令

查看linux磁盘占用情况： df -lh

## 时间复杂度

顺序查找：O(n)

二分法查找：O(logn)

## Java中静态方法与非静态方法的调用

静态方法是属于类的，即静态方法是随着类的加载而加载的，在加载类时，程序就会为静态方法分配内存，而非静态方法是属于对象的，对象是在类加载之后创建的，也就是说静态方法先于对象存在，当你创建一个对象时，程序为其在堆中分配内存，一般是通过this指针来指向该对象。静态方法不依赖于对象的调用，它是通过‘类名.静态方法名’这样的方式来调用的。而对于非静态方法，在对象创建的时候程序才会为其分配内存，然后通过类的对象去访问非静态方法。因此在对象未存在时非静态方法也不存在，静态方法自然不能调用一个不存在的方法。

## 过滤器与拦截器的区别

1．拦截器不依赖与servlet容器，过滤器依赖与servlet容器。

2．**拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。**

**3.深度不同：Filter只在Servlet前后起作用，而拦截器能够深入到方法前后。**

4. 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。

## 批处理

jdbc的批处理：

Statement.addBatch(sqlArr[i]);

Statement.executeBatch();

mybatis的批处理：

Bath Update方法

<update id="batchUpdate " parameterType="java.util.List">

    <foreach collection="list" index="index" item="item" separator=";">

  </foreach>

</update>

Springboot+jpa的批处理：

注入

**@PersistenceContext**

EntityManager em;

通过调用 em.persist(*list*.get(i)); //批量插入

em.merge(*list*.get(i)); //批量修改

## Spring AOP中Aspect方法执行顺序

@before around

@before

-- sayHello() -- 执行方法

@after around

@after

@after return

@around 之前之后的顺序是以 JoinPoint.proceed() 为分割的。

## 获取linux用户操作历史

USER\_IP=`who -u am i 2>/dev/null| awk '{print $NF}'|sed -e 's/[()]//g'`

HISTFILESIZE=4000

HISTSIZE=4000

HISTORYFILE=/var/log/history.log

HISTTIMEFORMAT="%F %T ${USER\_IP} `whoami` "

export HISTTIMEFORMAT

export PROMPT\_COMMAND='{ date "+%Y-%m-%d %T $(history 1 | { read x cmd; echo "ip=172.21.5.181,user=$USER,ppid=$PPID,pwd=$PWD,from=$SSH\_CLIENT,command=$cmd"; })"; } >> $HISTORYFILE'

设置 HISTTIMEFORMAT 之后，在控制台输入 history 就会出现类似历史的输出。

设置 PROMPT\_COMMAND 之后，会将linux 的操作历史追加到 history.log中。

## Logstash

启动命令：./bin/logstash -f ../conf/logstash.conf &

logstash.conf内容：

input {

file {

path => "/var/log/history.log"

}

}

filter {

grok {

match => {"message" => "ip=(?<destIp>\S+),user=(?<username>\w+),ppid=(?<ppid>\d+),from=(?<srcIp>\S+)\s+(?<srcPort>\d+)\s+(?<destPort>\d+),pwd=(?<pwd>\S+),command:(?<command>.\*)"}

}

ruby {

code => "event.set('LOAD\_TIME', event.get('@timestamp').time.localtime.strftime('%Y-%m-%d %T'))"

}

}

output {

elasticsearch {

hosts => ["http://172.21.5.181:9200"]

index => "systemlog\_%{+YYYY-MM-dd}"

#user => "elastic"

#password => "changeme"

}

}

主要分为input , filter, output 三部分。

Logstash中 filter 的grok中的匹配用到的是正则匹配。

Ruby 是将默认的timestamp时间字段以固定格式赋值给要新建的字段。