## final能用来修饰抽象类么？

抽象类不能使用final修饰,因为抽象类是用来被继承的,final修饰的表示不可修改,不能被继承

## springMvc应用于控制层，执行流程:

2.1用户发送请求,到前端控制器DispatcherServlet

2.2前端控制器请求处理器映射器HandlerMapping

2.3处理器映射器找到具体的处理器,生成处理器对象及处理器拦截器并返回给前端控制器

2.4前端控制器请求处理器适配器HandlerAdaptor

2.5处理器适配器经过适配调用具体的后端处理器,并返回ModelAndView给前端控制器

2.6前端控制器请求视图解析器ViewResolver

2.7视图解析器返回视图view给前端控制器

2.8前端控制器解析view,渲染视图给用户

## springboot可以精简spring的开发

## Springdata提供基本的增删改查

## orm 对象关系映射

建立实体类和数据库中表的关系，实体类加上@Document(表名)

## Redis数据结构

String，hash,list,set,sortedset

## 高并发解决方案

典型的高并发场景：秒杀和抢购。就是在同一时间点，有很多用户同时访问URL地址，就会产生高并发，高并发可能会导致用户网站打不开，服务器雪崩等情况，

（1）利用缓存。可以对一些数据进行缓存和数据共享，减少数据库的调用

（2）负载均衡。

---硬件四层交换

---软件四层交换

（3）HTML静态化。可以将更新频率很小的内容在进行后台更新的时候进行静态化，这样就避免了大量的数据库访问请求

（4）数据库集群。不同的模块对应不同的数据库或者表，能够低成本的提升系统的性能，并且有很好的扩展性

## 解决RabbitMq消息丢失问题

采用手动消费方式，如果消息没有被消费，消息就会重新加入消息队列，等待被消费

（1）消息持久化

---Exchange持久化 durable：true

---queue持久化

---message持久化发送 deliveryMode=2

（2）确认机制

---消息发送确认

---消息接收确认 默认情况下消息消费者是自动确认消息的，需要设置为手动确认

## JDK，JRE，JVM介绍

JDK:开发环境 JRE：运行环境 JVM:java虚拟机

## ==和equals的区别：

==： 基本数据类型比较值是否相等

引用数据类型比较地址值是否相等

equals：没有基本数据类型

引用数据类型默认情况下比较地址值是否相等，前提是必须重写equals方法

## Math方法中四舍五入实现方法：

给要四舍五入的数字加0.5，向下取整

## springBean的五种作用域：

（1）singleton:单例模式，在SpringIOC中仅存在一个bean实例，是一个默认值

（2）prototype：每次从容器调用bean是都会返回一个新的实例

（3）request：仅作用于web环境，每次http请求后都会创建一个新的bean

（4）session：仅作用于web环境，每个http session共享一个bean，不同的session使用不同的bean

（5）global session：仅作用于web环境，一般应用于Portlet应用环境

## ActiveMq的消息形式：

（1）点对点模式：一个生产者对应一个消费者，生产者发送消息给消费者，消费者没有接受，消息会暂时保存到ActiveMq,消息暂时不会丢失

（2）发布订阅模式：相当于广播模式，默认情况下生产者只向消费者发送一次消息，如果消费者不接受消息就没有了，所以消息容易丢失

## SpringAop实现原理：

SpringAop是oop（面向对象编程）的延续，oop面向对象编程，特征分别是封装，继承，多肽，抽象

SpringAop可以在不修改源码的情况下，对程序中已有的方法进行增强，减少重复代码，降低模块之间的耦合度，提高开发效率，便于维护

spring动态代理技术有两种：

（1）基于接口的动态代理，也就是JDK动态代理，核心是InvocationHandler和Proxy类。

（2）基于子类的动态代理，也就是cglib动态代理。

基于接口的动态代理：要求被代理类最少实现一个接口。

基于子类的动态代理：要求需要导入cglib坐标，被代理类不能被final修饰。

作用：可以在不用修改源码的情况下，增强一些方法。

AOP使用：

前置通知：方法调用之前发出通知。

后置通知：方法完成之后发出通知。

返回后通知：方法正常返回后，调用通知。

环绕通知：通知包裹在被通知的方法周围。

举个例子：假设有一个保存方法，程序即将执行到保存方法时，利用aop面向切面将程序先划分开，这时就可以对保存方法进行增强，以及事务的操作；

项目中用于日志处理和事务处理。

## 为什么要使用spring：

（1）spring是一个轻量级框架，依赖的东西少，占用资源少，部署相对简单

（2）松耦合，即通过xml配置或者注解就可以完成bean的注入

（3）AOP，通过配置或者注解就可以加入面向切面的功能，完成日志，事务的统一处理

（4）方便Mybatis,Hiberate等框架的集成

## SpringCloud

微服务架构是一种架构模式或是一种架构风格，它提倡将单一的应用程序划分成一组小的服务，各服务之间独立运行，互相协调，互相配合，为用户提供最终的价值

## SpringCloud和dubbo有哪些区别？

本质区别：dubbo是基于RPC远程调用，cloud是基于http rest Api调用

dubbo注册中心是zookper,cloud注册中心是Eureka

dubbo服务调用方式是RPC，cloud调用方式是rest API

SpringCloud功能比dubbo更加强大，SpringCloud可以将springBoot,SptingData完美融合，Dubbo构建的微服务就像组装电脑，其中一个零件质量不行就可能会崩盘

SpringCloud在SpringSource（java应用服务器）的整合下，做了大量的兼容测试，但是要想使用非原装的组件，就需要对其基础有足够的了解

## zookper和Eureka的区别

首先说一下CAP是什么,C强一致，A可用性，P分区容错性

一个分布式系统不可能同时满足CAP，由于分区容错性P是分布式系统必须保证的，所以可以在A和P上衡量

zookper遵循CP原则，遵循强一致原则，这就会导致当一个zookper挂了，其他zookper就会进行一次选举，选举期间注册不可用，

也就是这时候如果有用户要使用就用不了了

Eureka遵循AP原则，优先保证可用性，Eureka各个节点之间是平等的，有节点挂掉别的节点依然可以正常工作，用户注册失败时，会自动切换至其他节点，

因此，Eureka可以很好的应对网络故障导致部分节点失去联系的情况，不会像zookper那样整个服务瘫痪

## 谈谈对springBoot和springCloud的理解

SpringBoot是一个快速整合的第三方框架，关注的是微观，快速方便的开发单个个体服务

SpringCloud关注的是宏观，可以将SpringBoot开发出来单个服务管理整合起来

boot就是医院里面一个一个的科室，而Cloud就是把boot组合起来的提供对外的综合医院。

springboot可以单独使用，它不依赖于springcloud，而springcloud必然依赖于springboot，属于依赖关系

## 服务熔断（过载保护）

熔断机制是应对服务雪崩的一种微服务链路保护机制，

（雪崩就是 A为服务提供者, B为A的服务调用者, C和D是B的服务调用者. 当A的不可用,引起B的不可用,并将不可用逐渐放大C和D时, 服务雪崩就形成了）

当某个微服务不可用或者反应时间太长时，会进行服务降级（服务降级就是整体资源快不够用了，就忍痛将某些服务关掉，渡过难关后，再重新开启）

进而熔断该服务的调用，快速返回错误信息，当检测到服务正常后再恢复链路

## java中的设计模式

常用的设计模式：

（1）单例模式 如果一个类始终只能创建一个实例，这个类就是单例类，这种模式就是单例模式，包含懒汉模式和饿汉模式

（2）工厂模式 有一个专门的类来负责创建实例的过程，这些类是由某些抽象类或者接口派生出来的对象树

（3）代理模式 在某些情况下客户不能直接引用另一个类，代理对象可以在目标和客户端之间起到中介作用

（4）观察者模式 spring用于创建IOC容器的监听器就是基于观察者模式的

## spring中用到了那些设计模式

（1）工厂模式。在BeanFactory，ApplicationContext创建中都用到了工厂模式。

（2）模板模式。在BeanFactory，ApplicationContext创建中也用到了模板模式。

（3）代理模式。在Aop实现中用到了JDK动态代理

（4）单例模式。创建bean是用到单例模式

（5）策略模式

## 单例模式中的懒汉模式和饿汉模式的区别

懒汉模式：在类加载时不被初始化，在创建对象是不加synchronized注解，会导致对象访问的线程是不安全的

饿汉模式：在类加载时就被初始化，加载较慢，获取对象较快。线程安全

懒汉是延时加载，在配置文件中可以使用懒汉模式，饿汉在虚拟机启动时就会创建

## 什么是同步请求？什么是异步请求？

同步请求B/S模式：客户端提交请求，等待服务器处理，这个期间客户端浏览器不可以干别的事，请求提交后刷新整个页面

异步请求ajax: 请求通过事件触发，等待服务器处理，这个期间客户端浏览器可以干别的事，提交请求返回对象是不可见层，然后用JavaScript根据返回对象，改变页面数据

## redis缓存实现原理

redis是一种非关系型内存数据库，redis将数据存储在缓存中，查询速度非常快，

redis支持的数据类型：字符串类型，哈希类型，列表类型，集合类型，有序集合类型

应用场景：缓存，聊天室的在线好友，网站访问统计

redis是一个内存数据库，当服务器重启，或者电脑重启，数据可能会丢失，我们可以将数据持久化到硬盘中

redis持久化机制：（1）RDB，默认方式，不需要进行配置。隔一段时间持久化一次

（2）AOF，日志记录的方式，可以记录每一条命令的操作，每次命令操作后，持久化数据。

## 序列化和反序列化

序列化：是将java对象转换为字节流的过程，传递对象时，将对象转换成字节流可以方便传输和保存到硬盘中

反序列化：是将字节流转换为java对象的过程，获取到字节流后，根据字节流所保存的对象状态以及描述信息，通过反序列化重建对象

为什么要进行序列化和反序列化：（1）实现了数据的持久化，通过序列化可以将数据永久的保存到硬盘上

（2）利用序列化实现远程通信

## 控制反转

IOC不是一种技术，而是一种设计思想，在java中，IOC会将设计好的东西交给容器控制，所谓控制，即Ioc容器控制了对象，主要控制外部资源获取，

反转则是由容器来帮忙创建及注入依赖对象

## 依赖注入的几种方式

依赖注入是控制反转的一种实现方式，降低了依赖和被依赖类型之间的耦合

（1）通过setter方法注入

（2）构造器注入

（3）接口注入

Spring支持setter注入和构造器注入，通常使用构造器注入来注入必须的依赖关系，对于可选的依赖关系，则使用setter注入，setter注入需要类提供无参构造器或者无参的静态工厂方法来创建对象

IOC和DI的关系：

IOC就是容器，DI就是注入这一行为，DI就是IOC具体功能的实现。IOC则是DI发挥的平台和空间，IOC和DI相辅相成，他们都是为了解耦而服务的

## 类加载器的作用和分类

什么是类的加载：

当程序要使用某个类时，如果该类还未被加载到内存中，系统会通过类的加载，类的连接，类的初始化这三个步骤对类进行初始化。如果不出现意外，

JVM会连续完成这三个步骤，所以有时也会将这三个步骤统称为类加载或者类初始化。

java源码.java文件在经过java编译器编译后就被转为java字节码文件.class文件，类加载器是用来加载.class文件到java虚拟机中

java中的类加载器大致可以分为两类，一类是系统提供的，另一类是java开发人员编写的

系统提供： （1）引导类加载器（bootStrapClassLoader）用来加载java的核心库

（2）扩展类加载器（extensionsClassLoader）用来加载java的扩展库

（3）系统类加载器（systemClassLoader）根据java应用的类路径加载java类

开发人员编写：继承classLoader实现自己的类加载器

类加载机制：

（1）全盘负责：就是当一个类加载器负责加载一个class文件时，该class所依赖和引用的其他class也将由该类加载器负责加载，除非指定的有另一个类加载器

（2）双亲委派：就是某个特定的类加载器在接到类加载请求时，首先交给父类加载器处理，如果父加载器可以完成，就成功返回，如果完成不了，再自己处理

（3）缓存机制：缓存机制将会保证所有加载过的class文件被缓存，当程序需要使用某个class时，先会从缓存区找，当缓存区不存在该class，

系统会采用二进制，将其转换为class对象，存入缓存区。所以修改了class后，必须重启jvm虚拟机，程序所做的修改才会生效

## java虚拟机的内存模型

java虚拟机的内存空间：

（1）栈内存：方法运行时进入的内存，先进后出

（2）堆内存：new出来的对象进入的内存，数据有初始化值

（3）方法区：字节码文件加载时进入的内存，分为静态方法区和动态方法区

程序运行时（假设程序用的方法时main方法），会将字节码文件加载到方法区，静态方法和静态数据加载到静态方法区，非静态的加载到非静态方法区，方法会先进入到栈内存，

创建的对象会进入堆内存，并且有初始值，赋值后会变成所赋的值，并会记录成员方法的引用地址，调用对象方法时，调用的方法会加载到栈内存，

方法执行完毕，对象的方法会先弹出栈，main方法后弹出

## 什么是锁？什么是死锁？

死锁会让程序一直卡住，无法向下执行，我们只能通过终止并重启程序的方式让程序重新执行。

造成死锁的原因：

（1）当前线程拥有其他线程需要的资源

（2）当前线程等待其他线程拥有的资源

（3）都不放弃拥有的资源

避免死锁的方法：

（1）固定加锁的顺序（解决线程之间交错执行）

（2）开放调用

（3）使用定时锁，如果等待获取时间超时，则抛出异常而不是一直等待

synchronized ReentrantLock

Java中锁的种类：

---可重入锁：又名递归锁，是指一个线程在外层方法获取锁时，在进入内层方法会自动获取锁，可重入方法的一个好处是一定程度上可以避免死锁

----公平锁/非公平锁：公平锁表示线程获取锁的顺序是按照线程加锁的顺序进行分配的，即先来先得，先进先出

非公平锁：一种获取锁的抢占机制，是随机拿到锁的，这个方式可能造成某些线程一直拿不到锁，因此是不公平的

数据库中的锁：

---乐观锁/悲观锁：悲观锁总是假设最坏的情况，每次去拿数据的时候都会怀疑别人会修改，所以每次拿数据的时候都会上锁，这样别人想拿这个数据就会被阻塞，

直到拿到锁（共享资源每次只给一个线程使用，其他线程阻塞，用完后再把资源转让给其他线程），多写的场景下用悲观锁比较合适。

乐观锁总是假设最好的情况，每次拿数据的时候都会认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有更新这个数据，

可以使用版本号机制和CAS算法实现。乐观锁适用于写比较少的场景（多读），这样可以提高吞吐量

更新商品库存时可以使用悲观锁，悲观锁一开始就对商品上锁，这时候就可以做判断和更新

悲观锁可以在更新表数据时，先将表锁起来，这样别人想更改表数据时，就不能修改，只能自己修改

乐观锁可以用于版本更新，乐观锁两种实现方式：

（1）使用数据记录版本机制实现 ，数据版本就是为数据增加一个版本标识，一般是通过为数据库表增加一个数字类型的version字段来实现。

当读取数据时，将version字段的值一同读出，数据每更新一次，对此version值加一，当提交更新时，判断数据库表对应的当前版本信息与

第一次取出来的version值进行比较，如果数据库当前版本与第一次取出来的version值相等，则予以更新，否则认为是过期数据

（2）使用时间戳。在需要乐观锁控制的table中增加一个字段，字段类型使用时间戳（timestamp），在更新提交的时候检查当前数据库中，

数据的时间戳和自己更新前取到的时间戳进行比较，如果一致则OK，否则就是版本冲突

## Lock锁和synchronized锁的区别是什么

（1）Lock是一个接口，而synchronized是java中的关键字，synchronized是在jvm层面上实现的，在代码执行异常时，jvm会自动释放锁，Lock是通过代码实现的，

要保证锁一定被释放，就必须在finally{}中加上unLock()方法

（2）Lock可以让等待锁的线程响应中断，线程中断可以去干别的事务，synchronized不能，使用synchronized时，等待的线程会一直等待，响应不能中断。

（3）通过Lock可以知道有没有成功获取锁，synchronize不能知道。

## 32.1 解决多线程安全问题的几种方式

线程安全：如果你的代码在多线程执行和单线程执行下得到的结果永远都一样，那么你的代码就是线程安全的。

（1）同步代码块。在代码块声明上加synchronized,同步代码块中的锁对象可以是任意对象，但多个线程时，要使用同一个锁对象才能够保证线程安全

（2）同步方法。在方法声明上加上synchronized,同步方法中的锁对象是this；静态同步方法：在方法声明上加上static synchronized，静态同步方法中的锁对象是类名.class

（3）同步锁。Lock接口提供了与synchronized关键字类似的同步功能，但需要手动获取锁和释放锁

卖票案例

## 33. sleep方法和wait方法的区别

（1）sleep方法是Thread的静态方法，wait是object超类的成员方法

（2）sleep方法需要抛异常，wait方法不需要

（3）sleep方法可以在任何地方使用，wait方法只能在同步方法和同步代码块中实现，因为该方法必须由锁对象调用，锁对象必须出现在同步代码块中

（4）wait方法有带参和空参两种：带参数状态下跟sleep效果类似，让程序休眠xx毫秒，休眠过后程序继续执行，

不带参数时,会让程序进入无限休眠状态，必须由notify唤醒，wait方法在等待的过程中，释放锁对象，sleep方法在休眠过程中不释放锁对象

---notify和notifyAll的区别：

notify：随机唤醒单个线程

notifyAll：唤醒所有等待的线程

## 什么是可变参数

public [static] [final] 返回值 方法名称 (参数类型 ... 参数名称){ }

public static int add(int ...data)

参数个数不确定，随意由用户传递

## 进程和线程

进程是正在运行的程序，每一个进程都有他自己的内存空间和系统资源，

线程是进程中的一条执行路径，一个进程如果只有一条执行路径，则被称为单线程，有多条执行路径，则被称为多线程

进程好比一个工厂，线程比如是工厂中的流水线

## 创建线程的方式

（1）继承Thread类创建线程，代码比较简单，但是该类无法继承别的类

（2）实现Runnable接口创建线程，可以继承其他类，统一实现该接口的实例可以共享资源，但是代码比较复杂

（3）实现Callable接口，callable接口中的call()方法有返回值，Runnable接口中的run（）方法没有返回值

（4）使用线程池例如用Executor框架，实现自动化装配，易于管理，循环处理资源

启动线程，调用start方法，run方法是线程的执行体

## ssm整合流程

web三大组件：

servlet,DispatcherServlet前端控制器，拦截前端发过来的请求，指定SpringMVC的文件路径

listener,监听程序启动，启动了就初始化spring容器

fliter，解决post乱码问题

springMvc三大组件：处理器适配器，处理器映射器，视图解析器

mabits:在spring中配置SQLSessionFactory

## 反射原理

java反射就是在运行状态中，对于任意一个类都知道这个类的所有属性和方法；

对于任意一个类都能调用他的属性和方法，并且能改变他的属性。

也就是说反射可以把java中的各种成分映射成一个个的java对象，并且操作这些java对象

实现反射的方式：

（1）通过类名.class获取class对象

（2）通过对象.getClass()方法获取class对象

（3）通过Class.forName(全类名)获取class对象

## Hibernate和mybatis的区别

hibernate的优点：

hibernate是全自动的，对数据库结构提供了较为完整的封装。

O/R映射能力强，可以通过对象关系模型实现对数据库进行操作，自动生成sql，无关数据库性能好坏，开发速度快。

springdata Jpa封装了hibernate，让使用者通过一套注解实现映射配置，无需再为dao层的实现做开发，使用起来更方便

缺点：

学习门槛较高，虽然有hql查询，但是功能不及sql强大，虽然支持原生sql查询，但是开发模式上与orm模式不同，需要转换思维，使用相对不方便

mybatis：

着力点在于pojo和sql之间的映射关系

（1）mybatis是半自动，易于上手和掌握，提供了数据库查询的自动对象绑定功能

（2）可以进行更为细致的sql优化，减少查询字段

缺点：（1）框架比较简陋，功能欠佳，底层数据查询需要自己手写，工作量比较大

（2）二级缓存机制不佳

## mybatis中的动态sql原理分析

mybatis中的动态sql是为了解决手动拼接sql的麻烦，sql语句的主体结构在编译时无法确定，只有在程序执行过程中才能确定，这种sql就是动态sql

提到动态sql就要想到静态sql，静态sql就是在编译阶段就可以知道数据库要做什么事，例如select\*from user where id=1

原理分析需要研究源码，能力有限，研究不了

## 常见的运行时异常

（1）空指针异常

（2）下标越界异常

（3）类转换异常

异常分类：

运行时异常就是在开发中测试功能时，程序终止，控制台出现异常

非运行时异常就是在写代码的时候出现红线，必须处理，不处理编译就不通过

异常体系结构：

Throwable:Error 严重性错误 Exception:RuntimeException:运行时异常 ！RuntimeException:编译时异常

异常产生的原理：

java对异常默认的处理方式，是将异常抛出给上一级。抛出之前，java会根据错误产生的异常类，创建出该类的对象，底层通过throw关键字将异常

抛出给上一级，不断向上抛出，直到抛给虚拟机，虚拟机拿到问题后，会将错误信息和所在位置打印到控制台

异常的处理方式：

（1）问题可以自己处理掉的：使用try...catch处理方式

（2）处理不掉：使用throws抛出

## Map遍历方式

（1）键找值方式遍历map集合（keySet获取所有键，遍历键获取所有值）

（2）获取键值对对象集合，迭代器遍历集合获取键和值

（3）获取键值对对象集合，增强for遍历集合获取键和值

（4）使用map集合中的values方法，拿到所有的值

在保存商品信息时可以使用map集合

## map的实现类

（1）TreeMap:使用二叉树进行存储

（2）HashTable:版本比较低，线程安全，效率低，不能使用null作为key和value，对应的hashMap线程不安全，但是效率高

（3）LinkedHashMap:底层使用链表来维护key-value的次序

（4）properties:以key-value键值对存储信息

## HashMap底层数据结构

Map是一个双列集合，数据结构只针对键有效，跟值无关，键不能重复，

HashMap的数据结构是哈希表（数组+链表），jdk8之后，在哈希表中引入了二叉树，提高了查询效率

HashMap和Hashtable的区别：HashMap是jdk1.2版本出现的，允许存储null键和null值，线程不安全，效率相对高

HashTable是jdk1.0出现的，不允许存储null键和null值，线程安全，效率相对低

可以用来统计商品出现的次数

## 二叉树

TreeMap中的数据是有序的，因为底层数据结构是二叉树

用二叉树存储数据时，存入的第一个数据会被作为一个节点，再存入的数字会跟节点比较，大于该节点的会被存到右边，作为一个节点，

小于该节点的会被存到左边，又会成为一个节点，这样比节点小的数据在左边，比节点大的数据在右边。取数据先取节点左边的，左边取完，

取节点自身，然后取节点右边，这样取出的数据就是排好序的

## 动态代理

特点：字节码随用随创建，随用随加载

分类：（1）基于接口的动态代理，也就是JDK动态代理，核心是InvocationHandler和Proxy类

（2）基于子类的动态代理，也就是cglib动态代理

基于接口的动态代理：要求被代理类最少实现一个接口

基于子类的动态代理：要求需要导入cglib坐标，被代理类不能被final修饰

作用：可以在不用修改源码的情况下，增强一些方法

## jsp乱码问题

中文乱码出现的问题就是因为编码不符，可能出现乱码的地方：

（1）jsp编码乱码（最常见）设置编码的位置位于JSP的第一行，可以设置JSP的编码格式修改默认生成的编码格式

（2）HTML编码乱码：设置content属性中的charset

（3）request获取数据乱码:这种通过URL传参的方式，只能修改Tomcat的传输编码格式，修改Tomcat安装文件中的server.xml，设置URLEncoding编码格式就可以解决传参造成的乱码

（4）response输出信息乱码：设置response中的setContentType就可以解决输出的编码格式

（5）Cookie导致的编码问题：在保存cookie数据之前，首先保证request和response编码正确，使用URLEconder转码并保存，解码时用URLDecoder

## restful风格

一种软件架构风格，主要用于客户端和服务端交互的软件，基于这个风格设计的软件可以更简洁，更有层次，易于实现缓存机制

请求方式：URL+请求方式

## java垃圾回收机制

垃圾：当一个对象的引用没有变量记录时，该对象就会成为垃圾对象，并在垃圾回收器空闲时对其进行清扫

如何校验对象被回收？

重写finalized方法，该方法会在垃圾对象被回收时调用，通过System.gc()通知垃圾回收器工作

如何判断对象是否需要被回收？

引用计数法：每个对象会有一个计数器，对象被引用时，计数器+1,对象不引用后，计数器-1，计数器为0的就是垃圾对象

根搜索方法：判断对象与GC Root对象是否有联系，没有联系就是垃圾对象

java中可以作为GC Root的对象有：

虚拟机栈内存中的对象；

方法区中静态属性引用的对象；

方法区中常量引用的对象；

本地方法区中引用的对象（Native修饰的对象）

## HashCode和equals方法的区别

（1）HashCode和equals都是object类中的方法

如果类中不重写两个方法，那么hashCode返回的是对象的地址值，equals用来比较两个对象的地址值是否相等

如果类中重写两个方法，那么hashCode返回的是根据成员变量计算出的一个整数，equals比较的是两个对象中成员信息是否相等

（2）两个对象通过equals比较是相等的，那么hashCode肯定也相等，也就是equals是绝对可靠的

两个对象通过hashCode比较是相等的，那么equals比一定相等，也就是说hashCode不是绝对可靠的

通过hashCode和equals比较两个对象是否相等时，可以先用hashCode比较，如果两个对象不等，那么两个对象一定不相等，如果hashCode比较的两个

对象相等，再用equals比较，这样既可以判断两个对象是否相等，又可以提高效率

## HashSet是如何保证元素唯一性的

查看add方法的源码可以看出，在添加功能的执行过程中，进行了数据的判断，首先比较对象的哈希值是否相同，这个哈希值是根据对象的hashCode

计算出来的，如果哈希值不同，就直接添加到集合中，如果哈希值相同，继续执行equals进行比较，返回的是true，说明元素重复，不添加，

返回的是false，说明元素不重复，就添加，所以要使用hashSet存储数据，为了保证元素唯一性，就必须重写hashCode和equals方法

## String，stringBuilder，stringBuffer的区别

--String是一个不可改变的字符序列，stringBuilder是一个可以改变的字符序列

--String拼接的效率没有stringBuilder的效率高

String进行字符串拼接时会先创建一个stringBuilder对象，先拼接前两个，把拼接好的转成string，再跟第三个拼接，还是先创建一个stringBuilder

对象，拼接好以后转成string，以此类推，效率很慢

stringBuilder进行拼接时调用append方法，直接将要拼接的字符串放在一起拼接，拼接完成转成字符串，期间值创建一次stringBuilder对象，

拼接效率很高

--stringBuffer是jdk1.0出现的，线程安全（同步）的，效率低

--stringBuilder是jdk1.5出现的，线程不安全（不同步），效率高

## static关键字的特点

（1）静态成员被所在类的对象共享

（2）静态成员可以通过类名来调用

（3）静态成员随着类的加载而加载

（4）静态成员优先于对象存在

---静态方法只能调用静态成员，非静态方法可以调用任意成员

---private修饰的方法不能被重写，private修饰的成员只能在本类中访问。

---静态的方法可以被继承，但是不能重写，静态方法中不能使用this关键字，抽象方法不可以使用static修饰

## 字符串属于什么数据类型

字符串属于引用数据类型，被final修饰，不能被继承，字符串存储在常量池中

String s=new String("xyz);创建了几个对象

（1）如果常量池中已经创建了“xyz“,则不会继续创建，此时只创建了一个对象 new String（"xyz）

(2)如果常量池中没有创建“xyz”,则会创建两个对象，一个对象的值是“xyz”,一个对象是new String（“xyz”）

## IO流

（1）字符流

---字符输入流（Reader）

---字符输出流（Writer）

（2）字节流

---字节输入流（InputStream）

---字节输出流（OutputStream）

字节流读取数据时,读到一个字节就返回一个字节，字符流使用了字节流读到一个或多个字节时，先去查指定的编码表，将查到的字符返回

字节流可以处理所有类型数据，（图片，视频，MP3），字符流只能处理字符数据。处理纯文本数据时，优先选择字符流，除此之外都用字节流

52.数据类型

数据类型分为两大类，基本数据类型，引用数据类型

基本数据类型：byte,short,char,int,long,float,double,boolean

引用数据类型:类，数组，接口

## HashMap和HashTable的区别

相同点：都是双列集合，一个键对应一个值，都是键不能重复，值可以重复

不同点：HashMap是JDK1.2版本出现的，可以存储null键和null值，线程不安全

HashTable是JDK1.0版本出现的，不能存储null键和null值，线程安全

双列集合优先使用HashMap,多线程情况下，可以使用ConcurrentHashMap（线程同步）

HashTable使用不多，但是他的子类Properties集合使用较多，一般配合IO流加载配置文件

## 事务的隔离级别

（1）脏读： 一个事务读到另一个事务还没有提交的数据

（2）不可重复读：一个事务读到了另一个事务已经提交的修改的数据，导致在当前的事务中多次查询的结果不一样

（3）虚读/幻读：一个事务读到另一个事务已经提交的插入的数据，导致在当前的事务中多次查询的结果不一致

## Docker集成了开发环境和生产环境

## Cookie和session的区别

（1）Cookie数据存放在客户端的浏览器上，session数据放在服务器上

（2）很多浏览器都限制一个站点最多保存20个Cookie，单个cookie保存的数据不能超过4k

（3）cookie并不是很安全，别人可以分析存放在本地的cookie并进行cookie欺骗，考虑到安全应该使用session

（4）可以将登陆信息等重要信息保存到session，其他信息如果要保留可以存放在cookie

总而言之，cookie在客户端保存数据，不安全。只能保存字符串，且是少量字符串。

Session在服务端保存数据，安全，可以保存对象数据，数据无限制。

（5）默认情况下，浏览器关闭后，cookie数据被销毁；

Session什么时候被销毁？

1. 服务器关闭
2. Session对象调用invalidate（）
3. Session默认失效时间是30分钟

## http请求中的get方法和post方法的区别

get一般用于查询和获取资源信息，指定的资源经服务器解析后返回响应内容，必要时，可以将查询字符串参数追加到URL末尾，以便将消息发送给服务器。请求参数不安全，提交数据量小。

post一般用于更新资源信息，通常会用来传输实体的本体，提交数据比较安全，提交数据无限制。

## 转发和重定向的区别

（1）重定向是浏览器发送请求并收到响应后再次向一个新地址发请求，转发是服务器收到响应后为了完成响应转到另一个资源

（2）重定向中有两次请求对象，不共享数据；转发只产生一次请求对象且在组件间共享数据

（3）重定向后地址栏地址改变；转发则不会

（4）重定向的新地址可以是任意地址；转发必须是统一应用内的某个资源

转发的方法：

1. 通过HttpServletRequet的getRequestDispatcher()方法获得
2. 通过ServletContext的getRequestDispatcher()方法获得

重定向的方法：

HttpServletResponse的sendRedirect()方法

重定向可以实现图片的异步上传。

一般访问web-inf下的文件只能通过转发来实现

## spring中Bean的作用域有哪些？

（1）singleton（单例模式），使用该属性定义Bean时，IOC容器仅创建一个Bean实例，

IOC容器每次返回的是同一个Bean实例

（2）prototype（原型模式），使用该属性定义Bean时，IOC容器可以创建多个实例，每次返回的都是一个新实例

（3）request（HTTP请求），该属性仅对Http请求产生作用，使用该属性定义Bean时，每次http请求都会创建一个新的bean，适用于WebApplicationContext环境

（4）session（会话），该属性仅适用于http session，同一个session 共享一个bean实例，不同session使用不同的实例

## 63.spring框架实现实例化和依赖注入的方式分别是什么？

（1）使用构造器实例化Bean。SpringIOC容器能使用有参构造器也能使用无参构造器两种方式创建bean

（2）使用静态工厂方式实例化bean。使用这种方式除了指定必须的class属性，还要指定factory-method属性来指定实例化Bean方法，该方法必须是静态方法。使用静态工厂方法也允许指定方法参数，spring IOC容器经调用此属性指定的方法来获取bean

（3）使用实例工厂方法实例化bean。使用这种方式不能指定class属性，此时必须使用factory-bean属性来指定工厂Bean的id，factory-method属性指定实例化bean的方法，使用实例工厂方法也允许指定方法参数，方式和使用构造器方法一样。

**依赖注入的方式：**

1. 基于构造函数的注入

构造函数注入就是通过bean类的构造方法，将bean依赖的对象注入，构造函数的参数一般情况下就是依赖项，spring容器会根据bean中指定的构造函数参数来指定调用哪个构造。

1. 基于set方法注入

将bean所依赖的对象通过设置函数注入，bean需要为注入的依赖对象提供设置方法。

1. 基于自动装配的注入
2. 基于注解的依赖注入

@Autowired注解

默认按类型装配，默认情况下必须要求依赖对象必须存在，如果要允许null值，可以设置他的required属性为false

@Resource注解

---如果同时指定了name和type，则从spring上下文中找到唯一匹配的bean进行装配，找不到则抛出异常。

---如果指定了name，则从上下文中查找名称（id）匹配的bean进行装配，找不到则抛出异常

---如果指定了type，则从上下文中找到类型匹配的唯一bean进行装配，找不到或者找到多个都会抛出异常

---如果既没有指定name，又没有指定type，则自动按照byName方式进行装配，如果没有匹配，则回退为一个原始类型进行匹配，如果匹配则自动装配

## 64.spring和SpringMVC中的常用注解

（1）@component标记在一个类上，用于表示当前类是spring的一个组件，会进入IOC容器，包括三个衍生注解：@Repository@Service@Controller

(2)Repository用于标记在一个类上，代表这是数据访问层（dao层）

（3）@Service用于标记在一个类上，代表这是业务层

（4）@Controller用于标记在一个类上，代表这是控制层

（5）@RequestMapping用于处理请求地址映射，可用于类或方法上，用于方法上表示由当前类的方法来处理请求的URL，写在类上可以窄化请求的URL，也就是访问的URL是由类和方法上的requestMapping组合起来

（6）@requestParam用于将指定的请求参数赋值给方法中的形参

（7）@pathVariable可以获取URL中的动态参数