一.get与post请求方法的区别:

1.数据大小限制:get有限制,由于是拼接url,请求长度限制在1024kb以下; post无限制;

2.安全性:get明文提交,post非明文提交;由于post在请求体提交,不会在url被劫持;

3.传输数据位置:get是在请求行传输,而post是在请求体传输数据;

4.对于请求缓存/收藏书签/保留浏览器历史记录,get方式会被缓存,由于采用url请求方式;而post不行;

5.请求参数：querystring 是url的一部分get、post都可以带上。 get的querystring（仅支持urlencode编码），post的参数是放在body（支持多种编码）

6.共同点:二者本质没有区别,底层都是TCP链接;只是get请求方法多用于获取信息,post则用于提交数据;

二.http与https的区别

1.http概念:超文本传输协议，是一个基于请求与响应，无状态的，应用层的协议，常基于TCP/IP协议传输数据;规定了浏览器与服务器传输数据的格式;

2.http各版本差异:

(1)HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法;传输内容格式不限制.

(2)HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法,持久连接(长连接)、节约带宽、HOST域、管道机制、分块传输编码。

(3)HTTP2.0 新的二进制格式（Binary Format），HTTP1.x的解析是基于文本。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认0和1的组合。基于这种考虑HTTP2.0的协议解析决定采用二进制格式，实现方便且健壮。

3.HTTP特点:

(1)无状态：协议对客户端没有状态存储，对事物处理没有"记忆"能力，比如访问一个网站需要反复进行登录操作;
(2)无连接：HTTP/1.1之前，由于无状态特点，每次请求需要通过TCP三次握手四次挥手，和服务器重新建立连接。比如某个客户机在短时间多次请求同一个资源，服务器并不能区别是否已经响应过用户的请求，所以每次需要重新响应请求，需要耗费不必要的时间和流量;
(3)基于请求和响应：基本的特性，由客户端发起请求，服务端响应;
(4)简单快速、灵活;
(5)通信使用明文、请求和响应不会对通信方进行确认、无法保护数据的完整性;

4.https概念:基于HTTP协议，通过SSL或TLS提供加密处理数据、验证对方身份以及数据完整性保护;

5.https特点:

(1)内容加密：采用混合加密技术，中间者无法直接查看明文内容;

(2)验证身份：通过证书认证客户端访问的是自己的服务器;

(3)保护数据完整性：防止传输的内容被中间人冒充或者篡改;

三.TCP/IP协议

1.TCP/IP 意味着 TCP 和 IP 在一起协同工作。

2.TCP 负责应用软件（比如你的浏览器）和网络软件之间的通信。

3.IP 负责计算机之间的通信。

4.TCP 负责将数据分割并装入 IP 包，然后在它们到达的时候重新组合它们。

5.IP 负责将包发送至接受者。

四.TCP三次握手与四次挥手

1.TCP三次握手:通过三次握手和服务器建立连接;

(SYN :同步标志//ack确认标志//FIN结束标志)

2.三次握手:第一次握手:客户端向服务器发送syn=1报文,并置发送序号为X,Client进入SYN_SENT状态,等待Server确认;

第二次握手:Server收到数据包后由标志位SYN=1知道Client请求建立连接,发送syn+ack报文,并将ack=X+1,且随机产生一个序号为Y,并将数据包发送给客户端确认连接请求,server进入SYN_RECV状态;

第三次握手:Client收到确认后,检查ack是否为x+1,如果正确,发送ack=y+1的报文给服务器,服务器检查ack是否为y+1,如果正确则连接建立成功,客户端与服务器端都进入established状态,完成三次握手,随后就可以传输数据了;

3.三次握手的原因:为了防止已失效的连接请求报文段传送到了服务端，因而产生错误;

举例:比如,client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server。本来这是一个早已失效的报文段，但是server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求，于是就向client发出确认报文段，同意建立连接。假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了，由于client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据，但server却以为新的运输连接已经建立，并一直等待client发来数据。所以没有采用“三次握手”，这种情况下server的很多资源就白白浪费掉了。

2.TCP四次挥手:通过四次挥手和服务器断开连接;

第一次挥手:Client发送一个FIN,用来关闭Client到Server的数据传输,Client进入FIN_WAIT_1状态;

第二次挥手:Server收到FIN后,发送一个ack=M+1数据包,Server进入CLOSE_WAIT状态;

第三次挥手:Server发送一个FIN,用来关闭Server到CLIENT的数据传送,Server进入LAST_ACK状态;

第四次挥手:Client收到FIN后,进入TIME_WAIT状态,接着发送一个ack给服务器,确认序号为序号+1,server进入LOSED状态,完成四次挥手;

3.为什么建立连接是三次握手,而关闭连接是四次挥手?

因为服务端在LISTEN状态下,收到建立连接请求的SYN报文后,会将ACK和SYN放在一个报文发送给客户端;而关闭连接时,当收到对方的FIN报文时,仅仅表示对方不再发送数据,但还能接受数据,己方也未必全部数据都发送给对方了,所以己方可以立即close,也可以发送一些数据给对方后,在发送FIN报文给对方来表示同意现在关闭连接,因此,己方ACK和FIN一般会分开发送;

4.为什么会有time_wait状态,经过2MSL才能返回CLose状态?

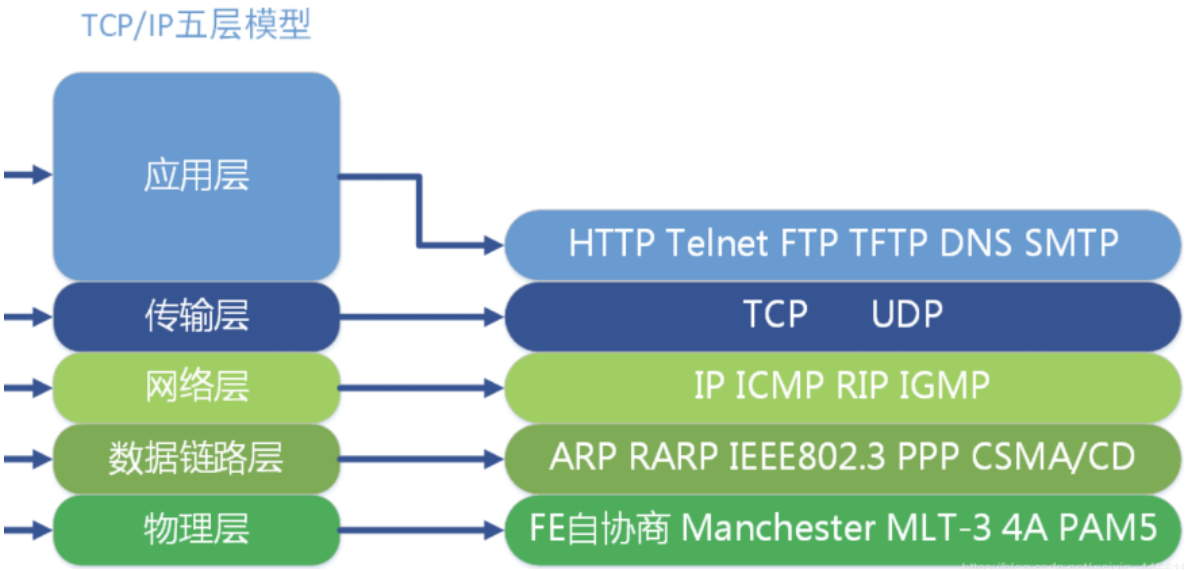
理由一:保证TCP协议的全双工连接能够可靠关闭;

(如果Client直接CLOESD,由于IP协议的不可靠性,或者其他网络原因,导致服务端没有收到客户端最后回复的ACK,那么服务器就会在超时之后继续发送FIN,但由于此时客户端已经closed,那么就会发送连接错误信息,虽不会导致数据丢失,但会导致TCP协议不符合可靠性连接的要求;)

理由二:保证这次连接的重复数据段从网络中消失;

(如果Client直接CLOSED,然后又在像服务器发起一个新连接;假设新连接与已经关闭的老连接端口号是一样的,如果前一次连接的某些数据还滞留在网络中,那么这些数据则会在建立新连接之后到达Server,由于端口号一致,且TCP协议判断不同连接的依据是socket pair,于是就认为延迟的数据是属于新连接,那么就会和真正的数据包发送混淆; TCP连接要在TIME_WAIT状态等待2MSL,这样可以保证本次连接的所有请求数据都从网络中消失;)

五.TCP与UDP的区别



1.面向连接的TCP(Transmission Control Protocol)协议

(1)"面向连接"就是在正式通信前必须要与对方建立起连接，是按照电话系统建模的。比如你给别人打电话，必须等线路接通了、对方拿起话筒才能相互通话。
TCP之所以可靠,是因为在传输数据前需要三次握手确认建立链接;三次"对话"的目的是**使数据包的发送和接收同步**;

(2)TCP总结

优点:能够保证数据传输是完整的;
缺点:由于每次都需要传输确认信息,导致传输效率降低;
场景:多用于必须保证数据完整性的场景,例如文本信息,支付信息等;

2.无连接的UDP(User Datagram Protocol)协议

(1)"无连接"就是在正式通信前不必与对方先建立连接，不管对方状态就直接发送。与手机短信非常相似：你在发短信的时候，只需要输入对方手机号就OK了。

UDP协议采取的方式与TCP完全不同,其根本不关心,对方是否收到数据,甚至不关心,对方的地址是否有效,只要将数据报发送到网络;

(2)UDP总结

优点：传输效率高
缺点：可靠性不如TCP协议
场景：UDP适用于一次只传送少量数据、对可靠性要求不高的应用环境。比如QQ消息,视频聊天,语音聊天

六.websocket 和http区别

1.websocket和http介绍;

(1)二者都是应用层协议,基于TCP/IP的可靠性传输协议;其中websocket是HTML5中的协议,支持持久连接,且是双向通信协议,一开始的握手需要借助HTTP请求完成;而http协议(长连接[keep_alive])不支持持久连接;

(2)http缺点: 在HTTP中一个request只能有一个response,而且这个response也是**被动**的，不能主动发起,**实时性不强**,每次都要采用三次握手建立TCP连接,造成资源浪费,**效率低。**

(3)针对http缺点提出:**1.ajax轮询**:其原理是让浏览器隔个几秒就发送一次请求，询问服务器是否有新信息。**2.长轮询**:采用轮询的方式，不过采取的是阻塞模型（一直打电话，没收到就不挂电话），也就是说，客户端发起连接后，如果没消息，就一直不返回Response给客户端（对于PHP有最大执行时间，建议没消息，执行到一定时间也返回）。直到有消息才返回，返回完之后，客户端再次建立连接，周而复始。这两种方式不断发起连接,且每次都需要重新传输校验信息,很消耗资源;**ajax轮询:需要服务器有很快的处理速度和资源.long poll:需要有很高的并发，也就是说同时接待客户的能力。（场地大小）**

(3)WebSocket优点:

解决被动性问题:只需要完成一次握手，完成协议升级后(http-->websocket)两者之间就直接可以创建持久性的连接，并进行双向数据传输。不用再次发起网络请求,服务端有信息就会主动推送给客户端.比如在app上websocket获取并展示客户的实时消费情况;

解决服务器上消耗资源的问题:

我们所用的程序是要经过两层代理的，即HTTP协议在Nginx等服务器的解析下，然后再传送给相应的Handler（php等）来处理。简单地说，我们有一个非常快速的 接线员Nginx，他负责把问题转交给相应的客服（Handler）。Websocket就解决了这样一个难题，建立后，可以直接跟接线员建立持久连接，有信息的时候客服想办法通知接线员，然后接线员在统一转交给客户。由于Websocket只需要一次HTTP握手，所以说整个通讯过程是建立在一次连接/状态中，也就避免了HTTP的非状态性，服务端会一直知道你的信息，直到你关闭请求，这样就解决了接线员要反复解析HTTP协议，还要查看identity info的信息;

1.相同点:

1,都是应用层的协议

2,都是基于tcp,并且都是可靠的协议

2.不同点:

1,websocket是持久连接的协议,而http是非持久连接的协议.

2,websocket是双向通信协议,模拟socket协议,可以双向发送消息,而http是单向的.

(***)3,websocket的服务端可以主动向客服端发送信息,而http的服务端只有在客户端发起请求时才能发送数据,无法主动向客户端发送信息.

3.采用websocket的应用场景:

社交订阅;多玩家游戏;协同编辑/编程;点击流数据;股票基金报价;体育实况更新;多媒体聊天;基于位置的应用;在线教育等;

七.输入一个url到浏览器页面所经历的过程

主要步骤:

1.输入网址;2.缓存解析;3.域名解析;4.tcp连接,三次握手;5.页面渲染;

具体流程:

(1)当输入网址后,浏览器获取此url,然后进行解析;先从浏览器缓存-系统缓存-路由器缓存中查看,如果有则从缓存中调取;

缓存:将之前访问的web资源，比如一些js，css，图片什么的保存在你本机的内存或者磁盘当中。 将资源缓存到磁盘或者内存中，等待下次访问时不需要重新下载资源，而直接从磁盘中获取；

如果没有,(2)首先会进行DNS域名解析;所谓DNS解析:域名到ip地址的转换过程,由DNS服务器完成,解析后获取域名相应的ip地址;(3)在域名解析之后,浏览器向服务器发起三次握手建立tcp连接;(4)成功对接上服务器后,发起http请求,服务器收到浏览器发送的请求信息,响应http请求;(5)随后浏览器解析html代码,并请求html代码中的资源(js,css,图片);(6)断开TCP连接;(7)浏览器对页面进行渲染呈现给用户;

八.cookie和session的区别;

1.cookie的介绍以及作用:

cookie用于客户端浏览器存储会话数据,由web服务器创建,最后保存在客户端上的文本数据;

在浏览器端存储数据,为了减轻服务器的压力;

2.原理:浏览器首次请求,没有cookie数据,而是由服务器创建并输出给浏览器,之后,每次请求服务器,浏览器如果有cookie数据,则会携带本服务器的cookie数据给服务器,服务器读取后进行业务逻辑处理;

3.session的介绍以及作用:

服务器端会话对象,存储数据在服务端内存中,服务器会为每一个用户创建独享的session对象内存空间;

会话域数据,在一个会话内是有效的,默认只要不关闭浏览器,session数据有效;注意:(1)关闭浏览器后,重新打开,代表新的会话,读取不到数据;(2)换新的浏览器就代表新的用户，服务器会为新的用户创建新的 session 对象，所以用不了上一个浏览器的session对象;

4.session原理分析:

1.浏览器第一次访问服务器，服务器会获取请求头中的cookie数据,判断cookie中是否含有key=JSESSIONID数据,**如果没有**,则服务器创建一个新的session对象给当前用户使用, 同时会产生新的sessionid并写入cookie中, 使用cookie默认有效期，输出给浏览器保存;**如果有**,服务器会获取JSESSIONID的值,去服务器内存中查找一样sessionid的session对象返回给当前用户浏览器使用;

2.当浏览器关闭了，cookie就过期了，打开浏览器再次访问服务器时，由于服务器没有获取到cookie 中的sessionid，就重新创建新的session对象，与上面相同继续运行；

***重要:，session是根据cookie中的JSESSIONID决定是否创建新的会话对象;

5.二者的区别

(1)存储位置:cookie浏览器端/session服务器端;

(2)大小限制:每个cookie限制4kb,每个服务器限制50个;而session仅受限于服务器内存;

(3)安全性:cookie安全性低;

(4)默认有效期:cookie默认浏览器关闭后就会丢失;session默认有效期为默认距离上一次请求间隔超过30分钟会被销毁;

九.mybatis一级缓存与二级缓存

1.使用缓存的目的:

提高数据查询操作的性能;如果有缓存,每执行一条查询sql语句,只需要数据库执行一次,mybatis就会将执行结果放入到缓存中,以后执行同一条sql语句就可以从缓存中获取, 从而提高性能, 避免反复到数据库中执行多次.

2.一级缓存:

在一个sqlsession会话对象内执行查询操作会进行缓存;

一级缓存的分析:

1. 第1次查询记录，将查询到的数据写入到缓存中 2. 第2次查询的时候，首先从缓存中去读取数据，如果缓存中有数据，直接返回，而不去访问数据库了。 3. 如果这个会话执行了添加，**修改，删除，提交，会关闭清空当前会话的1级缓存。**

3.二级缓存

在一个接口内不同 sqlsession会话对象执行的查询操作都会进行缓存,跨sqlsession缓存;

说明:接口映射配置文件中的namespace,就代表一个名称空间内;

4.配置二级缓存

1.sqlMapConfig.xml开启二级缓存标签作用:;2.缓存的实体类要序列化;

5.mybatis原理分析

(1)传统工作模式:创建SqlSessionFactoryBuilder对象,调用build()方法得到SqlSessionFactory对象,然后由该工厂类创建SqlSession会话对象,默认事务关闭,调用SqlSession中的api,传入statement id 和参数,最终通过jdbc执行sql语句,封装结果返回;

mybatis原理:

创建SqlSessionFactoryBuilder对象,调用build()方法得到SqlSessionFactory对象,然后由该工厂类创建SqlSession会话对象,然后调用getMapper()方法,传入要dao接口字节码文件,通过生成该接口的动态代理对象来实现pojo对象的数据封装并返回;(将sql执行结果封装进resulttype指定的pojo对象中);

(1)初始化配置文件信息,创建configuration对象,将解析的xml数据封装到configuration内部的属性中;然后创建工厂对象,传入configuration对象;

(2)SqlSession(通常与Threadlocal绑定)是一个接口,实现类DefaultSqlsession,它有两个最重要的参数,configuration和executor;

十.序列化与反序列化

1.序列化

以流的方式将对象转换为字节序列的过程;

2.反序列化

把字节序列重构/恢复为对象的过程;

3.使用序列化场景

(1)想要将内存中的对象状态保存到一个文件中或者数据库中时;

(2)想用套接字在网络上传送对象的时候;

(3)想通过RMI传输对象的时候;

当我们需要把对象的状态信息通过网络进行传输，或者需要将对象的状态信息持久化存储，以便将来使用时都需要把对象进行序列化;

4.序列化的原因

由于在内存中或者硬盘上,或者平台环境不同,为了方便数据在任何地方能够保持原有的意义,那么将其转换为字节数据进行传输,就保证此效果;

5.序列化注意点:

(1)静态变量不会被序列化;(2)transient关键字修饰的变量不会被序列化;

十一.springMVC相关知识

1.springMVC介绍

MVC:在控制器中,将数据封装至数据模型中,跳转至视图中展示;

Model:处理业务逻辑以及数据操作/数据封装;　　View:展示数据;

Controller:Servlet以及各种Controller类,作用,接收客户端请求,调用数据模型处理请求、跳转页面或者响应数据;

2.三层架构

(1)表现层:接收请求,响应/页面跳转; [SpringMVC框架,Strusts2框架]

(2)业务层:业务逻辑处理; [Spring框架]

(3)持久层:操作数据库(JDBC技术); [Mybatis框架,hibernate框架]

3.SpringMVC执行流程

第一部分:接收客户端请求,携带请求参数去查找对应的handle[Controller];

具体执行流程:服务器启动时,创建DispatcherServlet对象,加载springmvc的配置文件，创建spring的核心容器;随后,DispatcherServlet控制器,接收客户端请求后,先通过处理器映射器(HanlerMapping),将客户端的请求路径映射成一个执行链接(类似于自定义MVC中mappingPath,解析url,得到mappingPath同methodMap进行匹配,**<在此之前已经将requestMapping上的值和其对应的方法以键值对的形式存放到methodMap中>**根据key,返回对应的value执行请求)返回给控制器;随后DispatcherServlet控制器找到合适的处理器适配器,根据执行链接,适配到对应的处理器;由处理器执行请求;

第二部分:获取参数,处理请求后,进行页面展示;

具体执行流程:处理器处理请求后返回ModelAndView(视图名字)给DispatcherServlet中,然后由DispatcherServlet找到视图解析器,由它解析成真正的物理视图地址View,再返回给DispatcherServlet控制器,最终进行页面展示(response响应);

(1)前端发送request请求,由前端控制器DispatcherServlet接受客户端请求,控制各个其他组件的执行,做出响应,页面跳转;**(2)**处理器映射器:将客户端的请求路径,映射成一个执行链接(自定义MVC中的map集合,携带key,来返回对应的value);**(3)**处理器适配器:根据执行链接,适配到对应的处理器来执行;**(4)**处理器:也就是Controller,真正去执行处理请求;**(5)**视图解析器:将处理器返回的值,解析成具体的视图View;**(6)**视图:负责展示页面;

4.SpringMVC中语法作用

1.方法的返回值(响应数据):

(1)字符串:找视图解析器解析成具体的物理地址;(2)如果找不到,则会将该字符串响应给客户端;

(2)pojo对象或map,将其转换成json字符串响应;

2.@RequestBody注解,将json类型数据封装至pojo对象或map中;

3.@ResponseBody注解,返回响应数据时,将它们转换成json字符串响应;(引入jackson依赖)/如果加@RestController注解,则不需要;

4.@Pathvariable注解获取RestFul风格的url中的值;RestFul风格的url:一个路径表示一种资源，通过不同的请求方式来区分不同的操作;(POST/DELETE/GET/PUT)

5.拦截器:只拦截访问controller中的方法请求;

十二.Spring的理解

**在服务器启动时,加载spring配置文件,spring容器就创建出来了,并且已经将对象交给了容器(创建 spring 容器并且初始化容器中的对象。);这一结果是由于框架启动时,spring会去扫描并识别基础包中所有类的注解,拿到类的全限定名,然后利用反射技术(通过全限定名获取字节码对象,然后new instance()创建出对象),创建出类的对象,并放置在spring核心容器中;

1.IOC(inversion of control)

指容器控制程序对象之间的关系,而不是传统实现中,由程序代码直接操控,控制权由应用代码中转到了外部容器,控制权的转移是所谓的反转;DI(依赖注入):由容器动态地将某种依赖关系注入到组件中;

2.在spring的工作方式中,所有类都会在spring容器中登记,然后spring会在系统运行到适当时候,提供所需要的东西;所有类的创建,销毁都由spring来控制,也就是说控制对象生存周期的不再是引用它的对象,而是spring;

优点:降低了组件之间的耦合,降低了业务对象之间替换的复杂性,能够灵活的管理对象;

2.AOP(面向切面编程)

在aop思想未出现时,解决切面问题思路为以下两种:

(1)静态方法实现;将需要添加的代码抽取到一个地方,然后在需要的地方调用;(缺点:需修改源码,后期不好维护)

(2)继承方案:抽取共性代码到父类,子类在需要的地方,调用父类方法;(缺点:修改源码,继承关系复杂,不好维护)

AOP:就是把我们程序中影响多个类的公共行为抽取出来,封装为一个可重用的模块,在需要执行的时候,使用动态代理的技术,不修改原功能,再此基础上增强;

aop实现原理:(1)使用JDK的动态代理。针对有接口实现的情况。 它的底层是创建接口的实现代理类,实现扩展功能。也就是我们要增强的这个类,实现了某个接口,那么我就可以使用这种方式了. (2)使用 cglib 的动态代理,针对没有接口的方式,那么它的底层是创建被目标类的子类,实现扩展功能.

优点:在不修改源码的基础上,对现有方法进行增强;

十三.Mybatis相关知识

1.jdbc存在的问题:

(1)每次操作数据库都要操作PreparedStatement,查询使用ResultSet,代码冗余;(2)需手动开启管理事务处理;(3)sql语句与java代码耦合;(4)**最大的问题:解析结果集,封装成java对象,一个字段一个字段的封装;**

2.mybatis的优点:

(1)轻量级框架:对底层其他技术不依赖或轻依赖;mybatis只对jdbc进行封装,不依赖其余任何技术;

(2)解除了sql与程序代码的耦合:sql语句和代码的分离,提高了可维护性;

(3)**mybatis核心优势:ORM框架(对象关系映射框架),将关系型数据库的sql语句映射成技术语言对象;自动完成对象封装;**

(4)Mybatis支持缓存;可提高性能;

十四.JVM相关知识

1.java程序是如何执行的? / java如何实现跨平台

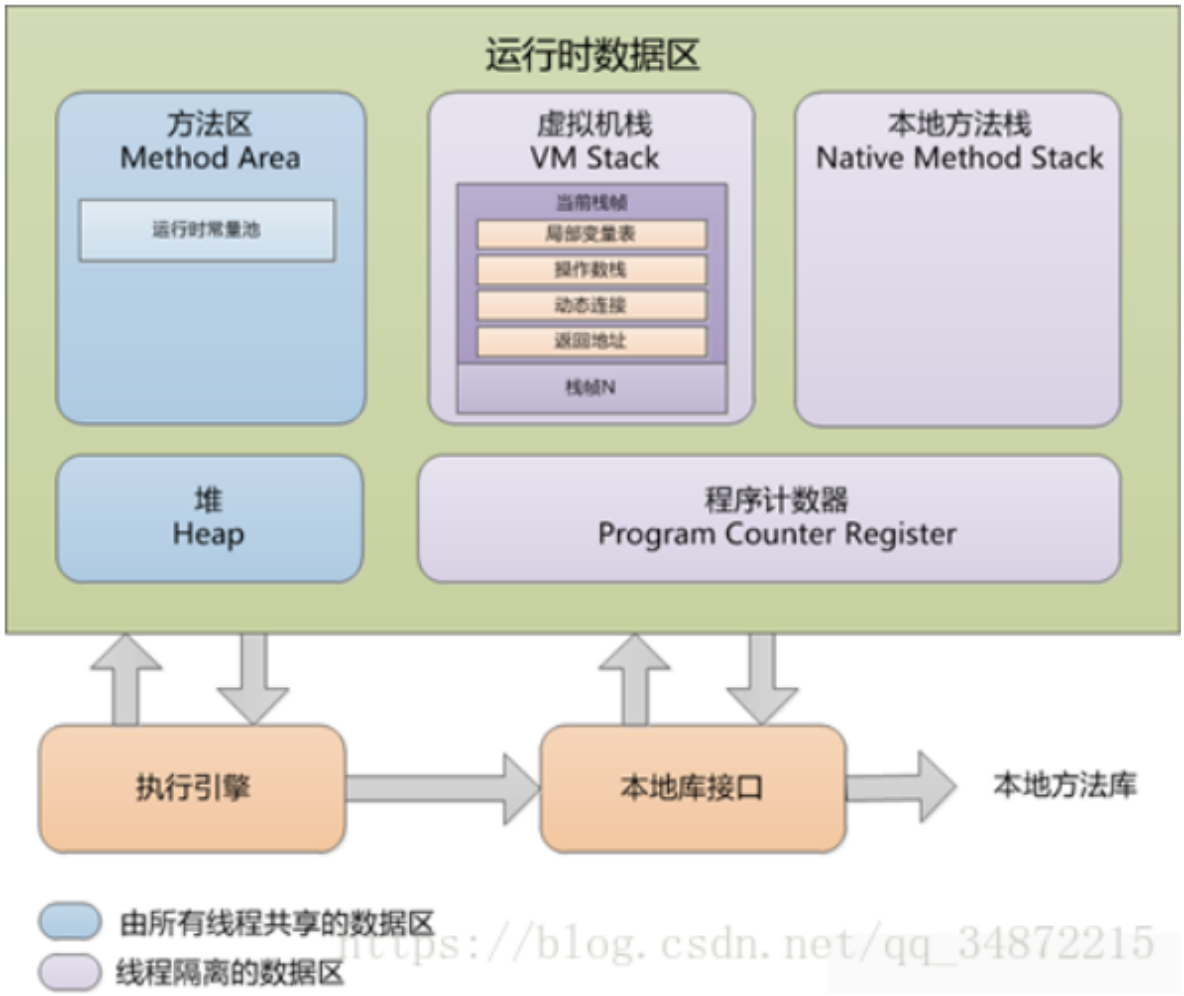
(1)跨平台:不依赖于操作系统,也不依赖于硬件环境;

(2)不同的系统下有不同的JVM(JVM是C语言写的);所以JVM不是跨平台;

(3)Java编译器将java源程序编译成平台无关的字节码文件(class文件),然后由java虚拟机JVM对字节码文件解释执行;该**字节码与系统平台无关,是介于源代码和机器指令之间的一种状态**;在后续执行时,采取解释机制将java字节码解释成与系统平台对应的机器指令;这样,既减少了编译次数,又增强了程序的可移植性,因此,称为"一次编译,多处运行".

(4)JDK(java开发工具包):包含jre(java运行环境)以及编译工具;JRE:包含核心类库和JVM;JVM:java虚拟机进行解译操作;将字节码文件解释成与系统平台对应的机器指令;

2.JVM内存区域



(1)java堆:Java虚拟机启动时就建立java堆,它是java程序最主要的内存工作区域,几乎所有的对象实例都存放在堆中,**堆空间是所有线程共享**的;

(2)java栈:每个虚拟机线程都会有一个私有的栈,一个线程的java栈在线程创建的时候被创建;栈中保存着局部变量/方法参数/java的调用方法和返回值等; 虚拟机栈和线程的生命周期相同;

(3)本地方法栈:与java栈类似,最大不同是本地方法栈用于本地方法调用;java虚拟机允许java直接调用本地方法(本地方法是C语言编写);

(4)方法区:存放类信息,常量信息,常量池信息,字符串常量以及数字常量等;

十五.collection集合相关知识点

1.集合和数组的区别

(1)二者都是引用数据类型的一种,均可存储多个元素;

(2)不同点:数组的长度是固定的;而且,数组既可以存储基本类型数据,也可以存储引用数据类型的数据;集合长度是可以变化的,只能存储引用数据类型的数据;

2.List集合特性(单列集合)

(0)List接口特点:有序集合(存储元素和取出元素顺序是一致的)/允许存储重复元素/含有带索引的方法(set(index,element) get(index) add(index,element) remove(index)); add方法默认在末尾添加;

(1)ArrayList:排列有序,可重复,底层使用数组;查询快/增删慢;线程不安全;初始容量是10;扩容条件:*50%

+1; 根据创建方式不同,容量也不同;

(2)LinkedList:排列有序,可重复,底层使用双向循环链表结构;查询慢,增删快;线程不安全;

栈:先进后出; 队列:先进先出;

单向链表:链表中只有一条链子,把元素链接在一起,不能保证元素的顺序(**HashSet**);

双向链表:链表中有两条链子,有一条链子专门记录元素的顺序,能保证元素的顺序(**LinkedList**,**LinkedHashSet**);

(3)Vector:排列有序,可重复;底层使用数组,查询快,增删慢;线程安全(效率低);当容量不够时会默认扩展一倍;

注意:**数组增删慢**原因(往数组中添加或删除元素,必须创建一个新数组,把之前数组中的数据复制到新的数组中);而**链表查询慢**原因(链表中元素的地址是不连续的,每次查询元素,都必须从链表的开头进行查询/增删则对链表没有影响);

Vector集合是同步技术;效率比arrayList集合低;是可实现可增长的对象数组;

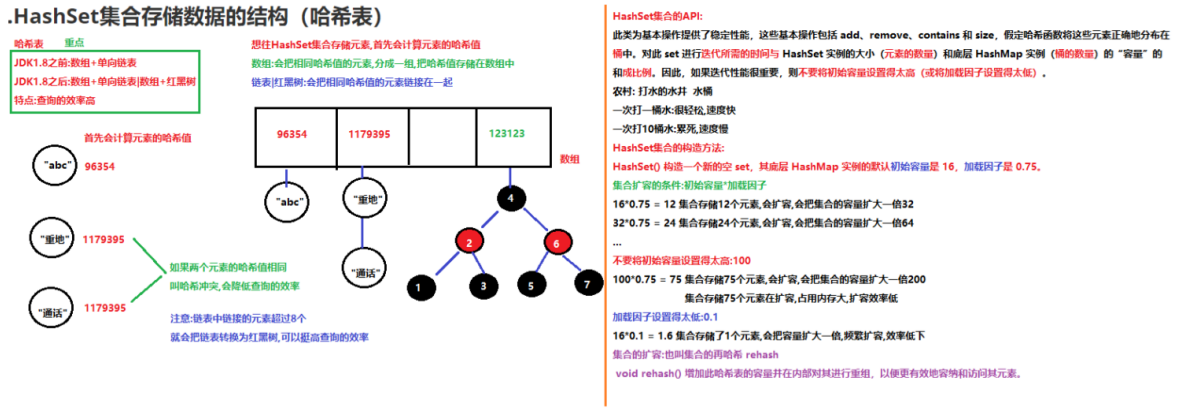
3.Collections类中的方法

sort/shuffle排序方法中的参数必须传递List接口下的实现类(ArrayList/LinkedList),不能使用set接口下的集合;

4.set接口的特性(单列集合)

(1)不允许存储重复的元素;(2)不包含带索引的方法;(3)此类允许使用null元素

(3)HashSet集合:无序集合(存储和取出的元素顺序可能不一致),不允许存储重复元素;(不能使用普通的for循环遍历set集合);底层是一个哈希表jdk1.8之前:数组+单向链表/1.8后:数组+单向链表|数组+红黑树(提高查询效率)

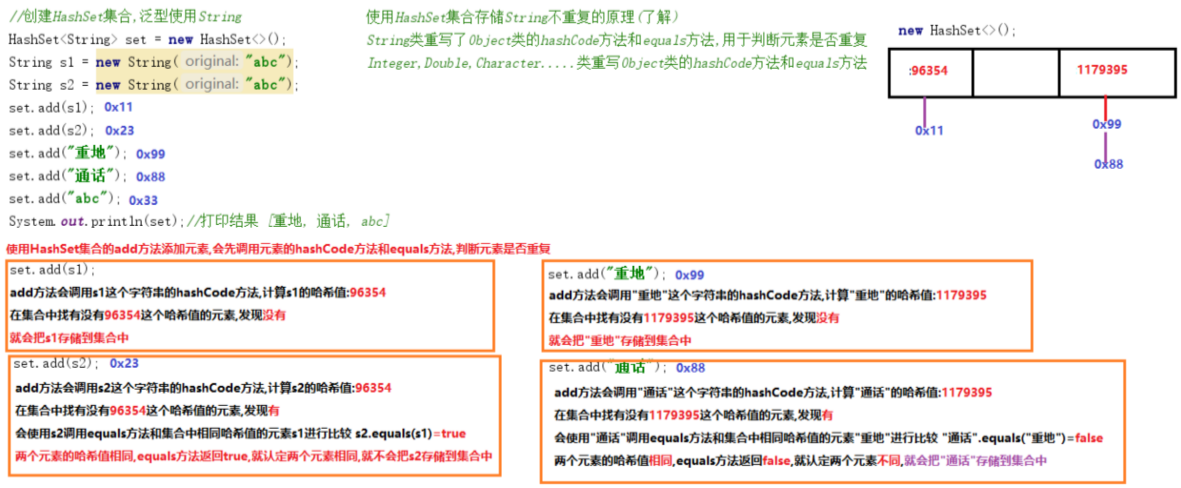


(4)往set集合中存储元素,首先会计算元素的哈希值;[**数组**]:会把相同的哈希值元素分成一组,将哈希值存储到数组中;[**链表**|红黑树]:会把相同哈希值的元素链接在一起;

哈希表结构:数组+单向链表(数组+红黑树);**当链表长度大于8时,会使用红黑树;**

(4-1)集合扩容的条件:**初始容量*加载因子**;hashSet构造一个新的set,其底层HashMap实例的默认初始容量为16;加载因子为0.75; 如果迭代性能很重要,则不要将初始容量设置太高(或者将加载因子设置太低);集合存储达到扩容条件就会自动扩大一倍;

(5)使用HashSet集合存储String不重复原理就是重写了Object类中的hashCode和equals方法;用于判断元素是否重复;[Integer....都重写了]



(6)**存储自定义类型元素:必须重写hashCode和equals方法,保证元素唯一;**

重写hashCode方法:是为了避免逻辑上冲突,降低哈希冲突率;也就是说更快的找到重复的元素;(如果不重写,那么自定义new person后,存入map集合后,又new一个一模一样的人,那么此时二者的地址值是不同的,则会产生不一样的哈希值,就会存入集合中,相同对象,不同的hash值,就get不到);

重写equals方法:是为了当出现哈希冲突时,用于判断对象是否重复;

5.哈希值

(1)哈希值:是一个十进制的整数,由操作系统随机给出,而我们打印的对象地址值使用的就是哈希值(逻辑地址);但是对象在内存中的实例存储位置(物理地址)并不是哈希值;

(2)String类的哈希值:规则(相同的字符串,返回的哈希值是一样的,但不同的字符串,计算出的哈希值有可能是一样的);

(3)String类的底层是一个数组;(1.9之前是字符数组/1.9之后是字节数组);

(4)计算哈希值公式:h=31*h+val[i];

6.LinkedHashSet集合特点:

(1)不允许存储重复的元素;(2)没有带索引的方法;(3)底层是1.8之前哈希表+单向链表(1.8之前是数组+单向链表+单向链表|1.8之后数组+单向链表|红黑树+单向链表);结构是:一个双向链表,可保证迭代顺序,是一个**有序集合**;

7.TreeSet集合特性:

(1)基于set接口的红黑树实现;默认根据元素的自然顺序进行排序;如果指定比较器,则根据比较器规则进行排序;

(2)不允许存储重复元素;(3)没有带索引的方法;(4)底层是一个红黑树;(5)可以根据比较器产生的规则对元素进行排序;

十六.Map集合相关知识点

1.Map集合的特点

(1)Map集合是一个双列集合;每个元素包含两个值,一个key,一个value;

(2)Map集合中的key是不允许重复的,value可以重复;(key重复,则会替换之前的value)

(3)map集合中一个key只能对应一个value;

(4)map集合中key和value数据类型是可以相同的,也可以不同;

2.map集合的分类

存储自定义类型时,key元素必须重写hashCode和equals方法,保证key唯一;

(1)**HashMap**<K,V>实现Map接口:底层是一个哈希表结构和HashSet是一样的;/是一个无序的集合;

(2)**LinkedHashMap**<K,V>继承了**HashMap**集合:该集合的底层是哈希表+单向链表;/是一个有序集合;

(3)**TreeMap**<K,V>实现Map接口:底层是一个红黑树结构;/集合中自带了一个比较器,里面存储的key是有序的;默认是升序或者可根据比较器自定义排序的规则;

(4)**HashTable**<K,V>实现Map接口:此类实现一个哈希表:

3.map集合遍历的方式

(1)键找值:把所有Key取出来,存储到一个set集合中,然后根据key获取value;特点:**遍历2次,第一次是获取iterator时,第二次是从hashmap中取出key所对应的value,效率低(普遍使用);**

(2)键值对方式:获取entry对象,存储到一个集合中,遍历获取每一个对象,然后通过getKey和getValue方法获取键与值;特点:**将HashMap键值对作为一个整体对象进行处理,只遍历一次,将key和value都放到entry中,效率高(当容量大时,推荐使用);**

4.对比HashMap和HashTable集合的特点:

(1)**HashMap**:允许存储null键和null值;/不同步,效率高(不安全);/底层是一个哈希表;

(2)**HashTable**:不允许存储null键和null值(它的equals()方法需要对象);/是同步的,效率低(安全);/底层是一个哈希表;

(3)HashTable线程安全,效率低,由于内部使用synchornized保证线程安全,每次同步执行的时候要锁住整个结构,在线程竞争激烈的情况下HashTable效率下降很快;

5.为什么用concurrentHashMap集合

(1)在多线程环境中HashMap的put方法有可能导致HashMap形成环形链表;从而造成程序死循环;

(2)HashTable通过使用synchornized保证线程安全,但在线程竞争激烈的情况下效率低,当一个线程访问HashTable时,其他线程只能阻塞等待该线程操作完成;

(3)concurrentHashMap,无论是读操作还是写操作都能保证很高的性能;在进行读操作时几乎不需要加锁,而在写操作时通过分段锁技术,只对要操作的数据段进行加锁而不影响客户端对其他数据的访问;

(**当put元素时,并不是对整个hashmap进行加锁,先通过hashcode来找到它要put的那个分段,然后对该分段进行加锁,所以,在多线程put时,只要不是放在一个分段中,那就实现了真正的并行插入,但在统计size时,需要获取所有分段锁进行统计**); ** **细化锁的粒度**;

(4)理想状态下,可支持16个线程执行并发操作,及任意数量线程的读操作;

(5)存储结构:1.7中使用分段锁(reentranlock+segment+hashentry); 1.8使用CAS+synchronized+node+红黑树;

6.拉链法导致的链表过深问题为什么不用二叉查找树代替,而选择红黑树,为什么不一直用红黑树?

选择红黑树是为了解决二叉查找树的缺陷,二叉查找树在特殊情况下会变成一条线程结构,遍历查找会很慢;而红黑树在插入新数据后可能需要通过左旋,右旋,变色等操作来保持h平衡,引入红黑树是为了查找数据快,解决链表查询深度的问题;但是如果链表长度很短的话,不需要引入红黑树,引入反而会慢;

7.红黑树

是一种特殊的二叉查找树;满足二叉查找树的特征:任意一个节点所包含的键值,大于等于左孩子的键值,小于等于右孩子的键值;

其余特性:

(1)每个节点或者是黑色,或者是红色;(2)根节点是黑色;(3)每个叶子节点(为空的叶子节点)是黑色;(4)如果一个节点是红色,那么其子节点必须是黑色的;(5)从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点;(确保没有一条路径会比其他路径长出2倍,因此,红黑树是相对接近平衡的二叉树);

红黑树的基本操作是添加,删除和旋转;旋转的目的是让树保持红黑树的特性;

8.HashMap的工作原理

(1)PUT过程

1.在进行数据存放时,首先会对key计算哈希值,然后计算下标;

2.如果没有,则直接放入桶中;

3.如果发生哈希冲突,则以链表的方式链接在后面;

4.如果节点存在,则进行value替换;

4.如果链表长度达到阈值(>8时,则转换为红黑树,低于6时,则又转回链表);

5.如果桶满,那么就进行扩容(16初始容量*加载因子0.75),扩容2倍后进行重排;

(2)(GET过程)

1.当调用get方法时,首先根据key对应得哈希值找到桶位置;

2.找到后,会调用keys.equals方法找到链表中正确的节点;

3.最终找到值对象;

(3)HashMap是在bucket中储存key和value,作为**Map.Node**;

(4)存储结构:

1.java1.8中,如果链表长度超过8,则将链表转为红黑树;如果低于6,那么就将红黑树转回链表;

2.发生哈希冲突时,java1.7会在链表头部插入,而java1.8会在链表尾部插入;

3.java1.8中,entry被node代替;

9.堆和栈的区别

(1)栈内存(存什么/生命周期):存储的都是局部变量(定义在方法中,局部变量是线程私有的);先加载函数才能进行局部变量的定义,所以方法先进栈,然后再定义变量,变量有自己的作用域,一旦离开,就会被释放;栈内存的更新速度很快,因为局部变量的生命周期很短;

(2)堆内存:存储的都是实体对象,用于封装数据,如果一个数据消失,该实体不会消失;堆里的实体虽然不会被释放,但会被当成垃圾,java有垃圾回收机制不定时回收;

(3)区别:栈存储的是局部变量,堆存储的是实体;//栈内存更新速度要快于堆内存,因为局部变量生命周期很短;//栈内存存放的变量生命周期一旦结束就会被释放,而堆内存存放的实体会被垃圾回收机制不定时的回收;

十七.JAVA面向对象知识

0.说说面向对象

面向对象:基于面向过程;

面向过程:当我们遇到一件事情时,我们自己分析事情的解决步骤,重视解决事情的过程;

面向对象:我们不自己完成,而是找一个能够帮助我们完成此事情的对象,调用对象的功能完成.重视的是对象;

封装:隐藏对象的属性和实现细节,仅对外提供公共访问的get/set方法;好处:隐藏类的实现细节,让使用者只能通过程序员规定的方法来访问数据,限制不合理操作;

继承:子类继承父类,子类就会自动拥有父类的非私有的成员变量与成员方法;提高了代码的复用性,使得类与类之间产生了关系;java中只支持单继承; 注意:构造方法不能被继承,是创建本来对象使用的;在子类加载时,会调用父类构造方法;

多态:父类的引用变量指向子类的对象; 前提:(1)必须有子父类关系或者实现关系;(2)在子类/实现类中重写父类/接口中的抽象方法,否则多态没有意义;有多个子类,那么创建哪个子类就调用哪个子类重写后的方法;(例如:**在controller层中注入service对象,用到多态**)

1.java中的数据类型:

(1)基本数据类型:byte(1字节)<short(2字节)<int(4字节)<long(8字节)<float(4字节)<double(8字节); char(2字节)

整数类型:默认int; 浮点数:默认double;

(2)引用数据类型:类class,Interface以及[]数组;

2.包装类和基本类型(自动装箱和自动拆箱)

(1)java5引入自动装箱和拆箱机制;**在Java8中,Integer缓存池的大小默认为-128-127**;

自动装箱:调用integer的valueOf方法; 自动拆箱:调用Integer的intValue方法;

(2)new Integer(123) 每次都会新建一个对象; 而Integer.valueOf(123)会使用缓存池中的对象,多次调用会取得同一个对象的引用;(3)超过127范围后,每次调用valueOf生成integer对象都是new,引用不同;

3.equals与==的区别

(1)==比较的是栈内存中存放的对象的堆内存地址;用于判断两个对象的地址是否相同,是否指向同一对象;

(2)equals:是Object类中的方法,而它里面的equals方法返回的是==的判断,如果对该方法进行重写的话,则比较的是两个对象的内容是否相等;

4.final关键字的作用

(1)被final修饰的类不可以被继承;(2)被final修饰的方法不可以被重写;(3)被final修饰的变量不可以被改变;如果修饰引用类型,那么表示引用不可变,但引用指向内容可以变(**不是在原内存地址是修改数据,而是重新指向一个新对象**);(4)被final修饰的常量,在编译阶段会存入常量池中;(5)此外,被修饰的方法,jvm会尝试内联,提高运行效率;

5.string和stringbuffer,stringbuilder的区别及应用

三者都可以存储和操作字符串;但对于(1)**string**类来说,由于其被final修饰,每次操作都会创建新的对象,消耗时间最长,性能最差;(2)**StringBuffer**操作字符串(每次操作对象都是同一个对象)且类中方法大多加了synchronized关键字,所以线程安全,一般在多线程操作字符串时使用,但是耗时比StringBuilder长;(3)**StringBuild**也是对同一个对象进行操作,耗时最短,但线程不安全;一般在单线程操作大量数据时使用;

6.抽象类(abstract class)和接口(interface)有什么异同?

相同点:(1)抽象类和接口都不能实例化;(2)一个类如果继承了某个抽象类或者实现了某个接口都需要对其抽象方法全部进行实现;否则该类仍然需要被声明为抽象类;

不同点:(1)抽象类中可以定义构造器,可以有抽象方法和具体方法;但接口中不能定义构造器且其中的方法全部是抽象方法;(2)抽象类中的成员可以是private,default,protected,public;而接口中的成员全是public;(3)抽象类中可以定义成员变量,而接口中只能定义常量;

有抽象方法的类必须被声明为抽象类,而抽象类未必要有抽象方法;

7.泛型

泛型:编写的代码可以被不同类型的对象所重用;就是说,我们不需要根据元素类型不同而定义不同类型的集合进行存储;只需要将其声明为泛型,那么在使用时就可通过具体的规则来控制存储的数据类型;

8.动态代理

实现动态代理的两种方式:

(1)JDK原生动态代理;

(2)CGLIB动态代理;

区别:(1)JDK代理,是JAVA自带API,不需要引入依赖,且基于接口(被代理者必须实现某接口[**获取被代理者的字节码对象,以及类加载器和被代理者实现的所有接口;通过此方法Proxy.newProxyInstance**]);
(2)CGLIB代理,则通过继承的方式进行代理(让被代理者成为enhance的父类),无论其有没有实现接口,都可以进行代理[**创建enhance对象,设置被代理者为父类**],其实就是产生这个类的子类,来增强方法;

最后,调用动态代理对象,进行方法执行;

9.常见设计模式

1.单例模式(Singleton):确保有且只有一个对象被创建;

2.构建模式(Builder);

3.抽象工厂模式:提供一个创建一系列或相互依赖对象的接口,无须指定具体的类;

4.工厂方法模式:定义一个用于创建对象的接口,让子类决定要创建的具体类;

5.观察者模式:定义对象间的依赖关系,当一个对象的状态发生变化时,所有依赖于它的对象都得到通知并被自动更新;

6.模板方法模式:子类可以不改变一个算法的结构就可以定义该算法的某些特定步骤;

7.装饰者模式:动态地给对象添加一些额外的功能;

8.代理模式:为对象创建代理者,控制对此对象的访问(Spring-AOP采用动态代理);

10.设计模式的原则(六大原则)

(1)开闭原则:对扩展开放,对修改关闭;(2)里氏代换原则;(3)依赖倒转原则:针对接口,依赖于抽象不依赖于具体;(4)接口隔离原则:降低类之间的耦合;(5)迪米特原则(一个对象应该对其他对象保持最少的了解);(6)合成复用原则;

11.object对象的方法

(1)clone()创建此对象的一个副本;(2)equals()方法,比较引用地址;(3)finalize()当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法;(4)getClass()返回此object运行时的类;
(5)hashcode(),返回该对象的哈希值;(6)notify()唤醒在此对象监视器上等待的单个线程;(7)notifyAll()唤醒在此对象监视器上等待的所有线程;(8)toString()返回该对象的字符串表示;(9)wait(),使得当前线程等待;

12.jdk1.8新特性

十八.JAVA异常知识

1.运行时异常

RuntimeException,既没有Throws声明抛出,也没有try-catch捕获;Java编译器不会检查;常见的运行时异常:ClassCastException(类型转换异常);IndexOutOFBoundsException(数组越界);NullPointerException(空指针异常);ConcurrentHashMap(并发修改异常);

2.检查时异常

java编译器会检查它;此类异常,必须通过Throws声明抛出或者try-catch捕获处理,否则不能通过编译;例如:IO exception,FileNotFoundException,SqlEXCeption;

3.错误Error

特点:和运行时异常一样,编译器不会对错误进行检查;当资源不足,约束失败以及其他程序无法继续运行时,会产生;程序本身无法修复这些错误; Error和Exception都继承了Throwable类;

4.Throw与throws的区别

(1)位置不同:throws用在函数上,后面跟异常类;而throw用在函数内,后面跟的是异常对象;

(2)功能不同:throws用来声明异常,让调用者知道该功能可能出现异常;throw抛出具体的问题对象,执行到throw,功能已经结束,跳转到调用者,并将具体的问题抛给调用者;

(3)throws表示出现异常的可能性,并不一定;而throw则是一定抛出了异常;

(4)两者都是消极处理异常的方式,只是抛出,真正的处理异常由调用者处理;

十九.数据结构

1.常用的数据结构:数组,栈,链表,图,散列表,队列,树,堆;

1.数组

数组可以再内存中连续存储多个元素,而再内存中的分配也是连续的,数组中的元素通过下标访问;适合频繁查询,增删少的情况;

2.栈

一种特殊的线性表,仅能在线性表的一端操作; 先进后出(弹夹);

3.队列

队列可在一端添加元素,在另一端取元素;先进先出;使用场景:在多线程阻塞队列管理中适用;

4.链表

链表是物理存储单元上非连续,非顺序的存储结构;数据的逻辑顺序是通过链表的指针地址实现,每个元素包含两个节点,一个存储元素的数据域,另一个就是指向下一个结点地址的指针域;根据指针指向,链表形成不同结构;单向链表,双向链表,循环链表;

优点:增删快,可任意加减元素;只需改变指针域的指向地址就行; 缺点:含有大量指针域,占用空间较大;每次查询都要从链表头部重新查找,非常耗时;

5.散列表

哈希表,根据Key/value直接进行访问的数据结构;通过key和value映射到集合中的一个位置,很快找到对应的元素;

6.堆

可以被看做是一颗树的数组对象;(堆总是一颗完全二叉树;堆中某个节点的值总是不大于或不小于其父节点的值);

(1)将根节点最大的堆叫做最大堆;(2)将根节点最小的堆叫做最小堆;因为堆有序,一般用来做数组中的排序,称为堆排序;

2.常见的树有哪些,并介绍

(1)二叉树;(2)二叉查找树(3)**平衡二叉树**(4)B树(5)**B+树**(6)B-树

3.排序算法

(1)冒泡排序;(2)插入排序;(3)**快速排序**;(4)选择排序;(5)希尔排序;(6)**归并排序**;(7)堆排序;(8)桶排序;

二十.多线程&&并发

1.Java中的锁;

(1)乐观锁/悲观锁;

乐观锁(CAS[Compare and swap比较与交换]):不会上锁,但在更新时会判断在此期间别人有没有更新过此数据;采用版本号机制; 无锁算法:CAS算法三个操作数(需要读写的值V/进行比较的预期值A/要写入的新值B),当A==V时,将V修改为B,否则返回V;只有一个线程能更新变量的值,其余会失败,但失败的线程不会被挂起,而是被告知这次竞争失败,并可以再次尝试;

每次去拿数据都认为别人没有修改数据,但是当进行更新操作时,会先判断一下在此期间有没有人修改;

悲观锁(synchronized关键字/LOCK锁):每次去拿数据都会认为别人会修改,每次在拿数据时都会上锁,别人想要操作必须阻塞等待拿到锁;传统的mysql数据库中用到了此锁机制:行锁,表锁,读锁,写锁,都是在操作之前先上锁;

LOCK的三种实现类:Reentrantlock,Readlock(读锁),writelock(写锁);

总结:悲观锁适合写操作多的场景,保证数据正确;而乐观锁适合读操作多场景,不加锁使得读性能大幅提升;

乐观锁,在更新数据时会判断数据有没有被修改,一般使用 数据版本机制 实现;(版本号/时间戳)

```
 update table set xxx=#{xxx}, version=version+1 where id=#{id} and version=#
{version};
```

(2)公平锁/非公平锁

公平锁:线程会直接进入等待队列;等待队列中除第一个线程外其余都会阻塞;等待锁的线程不会饿死;但整体吞吐率相对非公平锁要低,且CPU唤醒阻塞线程的开销比非公平锁要大;**(按照申请锁的循序来获取锁**)

非公平锁:进来就会尝试占有锁,如果失败,就采用类似公平锁方式;等待队列中的线程可能会饿死;但整体吞吐率高;

(不按照锁定顺序来获取) Synchronized

Reentrantlock,可以设置是否为公平锁,默认非公平锁,吞吐量大;

(3)共享锁/独享锁

共享锁:该锁可被多个线程持有;独享锁:只能被一个线程持有,其他只能等待;

(4)分段锁:是一种锁的设计,并不是具体的锁(ConcurrentHashMap[CAS+局部synchronized])

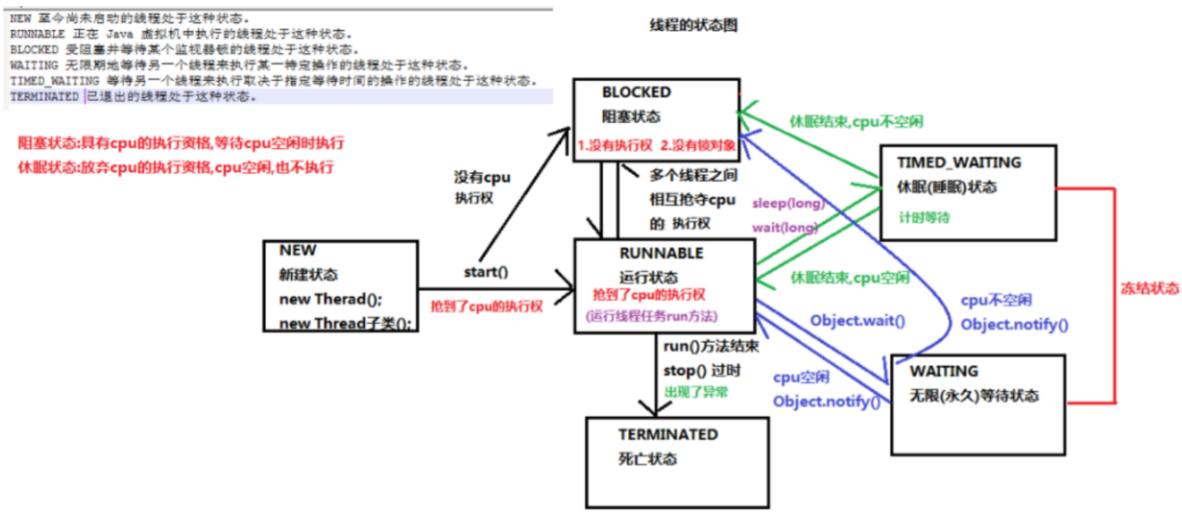
(5)互斥锁/读写锁

互斥锁:一次只能一个线程拥有互斥锁,其余会阻塞(Reentrantlock);

读写锁即是互斥锁,又是共享锁;read模式共享,write是互斥; 并发读非常高效,读写,写写都互斥;(ReadwriteLOCK)

读写锁特点:1.多个线程可以同时读2.写着必须互斥3.写着优于读者;

2.JAVA线程状态和生命周期



(1)新建(NEW):表示线程新建出来还没有被启动的状态;

(2)运行状态(RUNNABLE):正在运行的线程;就绪(ready) 运行(running);

(3)阻塞(BLOCKED):没有cpu执行权/没有锁对象; 表示线程正在获取有锁控制的资源;(等待CPU空闲时执行)

(4)等待状态(WAITING):无限期等待另一线程来执行;(<1>当一个线程执行了object.wait()方法,那么就会进入'此时,必须等待另一线程执行object.notify唤醒<2>当一个线程执行Thread.join()方法)

(5)超时等待状态(TIMED_WAIT):等待另一个线程来执行取决于当前线程指定的睡眠时间; (Thread.sleep()方法)

(6)死亡状态(TERMINATED):已退出的线程都处于这种状态; cpu空闲,也不执行;

3.synchronized和lock的区别

** 通过new Reentrantlock(),获取lock对象,然后调用lock()/unlock()方法;

(1)lock是一个接口,而synchronized是java的一个关键字;

(2)synchronized异常会释放锁,lock异常不会释放,一般try catch包起来,finally中写入unlock,避免死锁;

(3)Lock可以提高多个线程进行读操作的效率;

(4)synchronized关键字,可以放在代码块,实例方法,静态方法以及类上;

(5)lock一般使用reentrantlock类作为锁,配合lock()和unlock();

4.synchronized与Reentrantlock区别

(1)synchronized依赖JVM实现,而Reentrantlock是JDK实现的;synchronized是内置锁,只要在代码开始的地方加关键字,代码结束会自动释放,Lock必须手动加锁,手动释放锁;

(2)Reentrantlock扩展性好,灵活;而synchronized自动化,但扩展性低;

(3)两者都是可重入锁,互斥锁;

(4)synchronized是非公平锁,Reentrantlock可以指定是公平锁还是非公平锁;

(5)Reentrantlock通过AQS来实现线程调度;

抽象同步队列:其中包含三个部分;1.采用一个int类型的变量来表示同步状态(0/1);2.另一个变量来表示目前正在使用的线程;3.双端队列(遵循FIFO先入先出原则),主要用于存放阻塞的线程;

公平锁:当一个线程释放锁资源,那么会清除当前其余两个变量的状态;显示没有线程正在使用;并唤醒后面排队的节点,该节点会重新将变量状态修改为被使用状态;一旦有新的线程来,新线程看到队列有排队的,就会自动排在最后面;

非公平锁:当一个线程释放锁资源后,并修改状态为没有线程在使用,这时,新来的线程进入,当它看到变量状态为可取锁时,就直接获取锁,并修改变量状态为正在使用;

是用一个Int类型的变量来表示同步状态,并提供了一系列CAS操作来管理这个同步状态;是一个双端队列,遵循FIFO(先进先出)原则,主要作用用来存放在锁上阻塞的线程,当一个线程尝试获取锁时,如果已经被占用,那么当前线程就会被构造成一个Node节点加入到同步队列的尾部,队列的头节点是成功获取锁的节点,当头节点线程释放锁时,会唤醒后面的节点并释放当前头节点的引用;

5.synchronized与threadLocal区别

(1)都可以解决多线程中访问相同变量时的冲突问题;

(2)synchronized同步机制,提供一份变量,让不同的线程排队访问;

(3)threadLocal类,为每一个线程都提供了一份变量,因此可以同时访问而互不影响;

(4)threadLocal比直接使用synchronized同步机制解决线程安全问题更简单,方便;且程序拥有更好的并发性;

6.synchronized与volatile区别

volatile关键字:只能修饰变量,可以解决变量的可见性(一个线程修改值后,其他线程会立即看到),有序性,不能解决原子性;

synchronized关键字:不能修饰变量,可以修饰方法,代码块,使用范围比volatile广,可以解决:可见性/有序性/原子性;将私有内存和公共内存中的数据做同步;

7.Thread类中的run()方法和start()方法区别

(1)调用start()方法来启动一个线程,使线程处于就绪状态,此时,可以被JVM调度执行,在调度过程中,JVM通过调用线程类的run()方法来完成实际的业务逻辑,当run()方法结束后,此线程就会终止;

(2)如果直接调用run()方法,则会被当做一个普通的函数,程序中仍然只有主线程一个线程;即start()方法能够异步调用run方法;

(3)只有通过调用线程类的start()方法才能达到多线程目的;

8.事务的隔离级别及引发的问题

(1)4个隔离级别;

(2)a.读未提交(READ UNCOMMITTED),事务中的修改,即使没有提交,对其他事务也是可见的;

b.读已提交(READ COMMITTED),一个事务能读取已经提交的事务所做的修改,不能读取未提交事务所做的修改;也就是事务未提交之前,对其他事务不可见;---->oracle sql server默认隔离级别

c.可重复读(REPEATABLE READ),保证在同一个事务中多次读取同样数据的结果是一样的; ---> mysql默认;

d.串行(SERIALIZABLE),强制事务串行执行;

(3)多个事务,不同隔离级别引起的问题;

a.读未提交:可能会出现脏读,不可重复读,幻读

b.读已提交:可能出现不可重复读,幻读

c.可重复读:可能出现幻读

d.没问题

脏读:另一个事务读取到了其他事务未提交的数据;

不可重复读:一个事务中,读取到了另一个事务已经提交的数据,多次读取数据不一致;[在同一个事务中,数据应当保持一致性,不读取其他事务修改后的数据]

幻读:1号事务在进行写操作并提交后,2号事务看不到,但当2号事务执行同样写操作后,提交会报错;原因是已经该操作已执行过;对于2号来说,明明没执行成功,却提示已执行过;

9.介绍下线程池

(1)线程池就是预先创建一些线程,它们的集合称为线程池;

(2)线程的创建和销毁比较消耗时间,而线程池可以很好地提高性能,在于系统启动时创建一定数量的线程,当程序给到任务时,线程池就会启动线程去执行,执行结束后,该线程不会死亡,而是再次返回线程池中成为空闲状态,等待下一个任务;

(3)executors是jdk1.5之后的一个类,提供了一些静态方法,方便用于生成一些常见的 线程池;(单线程化single线程池/固定大小的线程池fixed/cached可缓存的线程池/schedule可周期性执行任务的线程池);

(4)使用ThreadPoolExecutor自定义线程池;(corePoolSize核心池大小//maximumPoolSize线程池最大线程数//keepaliveTIME保持线程的存活时间//线程工厂//异常捕捉器);

补充:这几种线程池的区别

(1)单线程化线程池(newSingleThreadExecutor)只创建唯一的工作者来执行任务;可保证顺序的执行各个任务;

(2)固定大小的线程池:创建指定数量的线程池;每当提交一个任务就会创建一个工作线程,如果工作线程数量达到最大数,则将提交的任务存入到池队列中; 当线程池空闲时,它不会释放工作线程,还会占用一定的系统资源;

(3)可缓存线程池:如果线程池长度超过处理需要,可灵活回收空闲线程;若无可回收,则新建线程;如果长时间没有往线程池中提交任务,那么该工作线程会自动终止(1min),终止后,如果有新的任务,那么会重新创建一个工作线程;在使用时,注意控制任务数量;

(4)可周期性执行任务的线程:该线程池支持定时周期性的任务执行;

10.同步与异步区别

(1)同步发了指令,会等待返回,才发下一个;异步,则发出指令后,不用等待返回,随时可以发下一个请求;

(2)同步可避免出现死锁以及读脏数据的发生;

(3)异步可以提高效率;

(4)实现同步的机制:临界区/互斥/信号量/事件;

11.如何自定义线程池

(1)通过ThreadPoolExecutor创建;

(2)7个参数:核心池大小(corePoolSize)//线程池最大线程数//keepalivetime保持时间(额外线程存活时间)//tmeunit时间单位//任务队列(workqueue)//线程工厂(threadFactory)//异常捕捉器handler

12.如何异步获取多线程返回的数据

(1)通过callable+future.callable负责执行返回值,future负责接收;callable接口对象可交给ExecutorService的submit方法去执行;ExecutorService的submit方法接收Runnable和Callable

(2)通过callable+futureTask;callable负责执行返回值,futureTask负责接收;futureTask同时实现了Runnable和callable接口;可以交给ExecutorService的submit方法和thread去执行;

(3)通过completionService,jdk1.8之后提供了完成服务,可实现这样的需求;

(4)实现Runnale接口任务执行结束后无法获取执行结果;

(5)注意:completionService和future的区别:future获取结果,一个一个的取,一个取完了在取另一个,就会等待;而completionService任意一个线程有返回值,就立马取出;

submit方法:会在线程池中获取一个线程执行任务,执行完成,会将线程归还给线程池;

13.线程的创建方式

(1)继承Thread类实现; [MyThread继承Thread类(重写run方法),然后new MyThread()创建线程]

特点:调用start()方法并不意味着立刻执行run方法,只是使该线程处于可运行状态,具体什么时候执行,由CPU决定;

java不支持多继承,这种方式扩展性差;

(2)实现Runnable接口方式; [实现Runnable接口,重写run方法,然后 new runnaleimpl实现类,在将实现类传入new Thread()中

]

(3)实现callable接口方式; [实现callable接口,重写call方法,然后创建New MyCallable,再创建线程池,采用线程池submit方法,传入线程任务并执行,且有返回值]

(4)前两者常用,但需要有返回值则实现callable接口;

(5)callable需要配合线程池使用;callable比Runnable功能复杂些;callable的call方法有返回值并且可抛异常(可知道线程执行过程中出了什么错误),而Runnable的run方法没有返回值,也没有抛异常;

callable运行后可以拿到一个future对象,这个对象表示异步计算结果,可通过future的get方法获取call方法的返回结果;但调用Get方法时,当前线程会阻塞,直到call方法返回结果;

13.countDownLatch多线程协作(同步辅助类)

(1)countDownLatch允许一个或多个线程等待其他线程完成操作;注意:必须保证多个线程使用的是同一个countDownLatch对象;

场景:线程1要执行打印动能,A和C,线程2执行打印B,但要求1在打印完A之后,要2打印B之后才能打印C;

(2)countDownLatch是通过一个计数器来实现,每当一个线程完成了自己的任务后,可以调用countDown()方法让计数器减1,当计数器到达0时,调用countDownLatch;然后阻塞状态解除,继续执行下面代码;

await()方法,使线程处于阻塞状态;

(3)使用:创建countDownLatch对象,可设置计数器初始值,然后执行线程任务; new countDownLatch(int count)

14.工作中哪些地方使用了多线程

(1)自己做并发测试时;(2)多线程下单抢单;(3)多线程写入mysql;(4)多线程写入redis;导入ES索引;(5)poi多线程导入导出;

15.数据并发操作可能的问题

(1)丢失的修改;(2)不可重复读(3)读脏数据;(4)幻读;

16.消息等待通知wait/notify具体应用

(1)一个线程修改了一个对象值,另外一个线程需要感知到这个变化;

(2)Java中我们使用的对象锁及wait()/notify()方法进行线程通信;

(3)等待方遵循的原则:a.获取对象锁b.不满足条件就调用wait()方法;c.条件满足就继续执行;

(4)通知方原则:a.获取对象锁b.改变条件,然后notify;

17.线程池中submit()和execute()方法的区别

(1)execute()参数Runnable;

(2)submit()参数(Runnable)或(Runnable和结果T)或(Callable);

(3)execute(Runnable)没有返回值,可以执行任务,但无法判断任务是否成功完成;

(4)submit(callable)有返回值,返回一个future类的对象;

(5)future对象:get方法,获取线程返回结果;也接收任务执行时候抛出的异常;isdone方法,可判断线程是否执行完成;

18.事务特性

(1)事务特性指的是ACID;原子性(Atomicity),一致性(Consistency),隔离性(Isolation),持久性(Durability);

(2)原子性:事务包含的操作要么全部成功,要么全部失败;

一致性:强调数据的一致性,A与B两者总钱数5000,不论AB之间来回操作,其事务结束后两者的钱数仍然是5000;

隔离性:多个并发事务是相互隔离的,事务之间不能干扰;

持久性:一旦事务提交,那么对数据库中的数据改变是永久性的;即使在数据库系统遇到故障情况下也不会丢失提交事务的操作;

19.进程和线程的区别

(1)程序被载入到内存中并准备执行,它就是一个进程;

(2)单个进程中执行的每个任务就是一个线程;(3)一个线程只能属于一个进程,但一个进程可以拥有多个线程;

二十一.数据库

0.Mysql数据库索引

(1)没有索引的时候:

需要进行全表扫描,数据库必须一行一行的查找直到最后一行,就算在查到第一个符合条件的行后,也不能停止查询;

(2)有索引的时候:

就不需要进行全表扫描,快速找到所需要的数据; 此处可类比书的索引(目录),如果没有目录,那么查询就很快;

(3)什么是索引:

索引就帮助数据库高效获取数据的数据结构;它对数据库表中一个或多个列的值进行排序;

(4)索引结构:主要有 **B+树索引** 和 **哈希索引**

B+树索引:对于Mysql数据库来说,Innodb存储引擎和MyISAM存储引擎都使用B+树索引,但其实现方式不同;**MyISAM** 它的其叶结点data 域存放的不是实际的数据记录;而是数据记录的地址,指针;先找到对应的索引文件中的数据地址,然后根据该地址指针找到原始表数据;其主键索引和辅助索引区别并不大,只是主键索引要唯一;而对于**Innodb**来说,其叶子结点的数据域存放的就是实际的数据记录;其数据文件本身就是索引文件(对于主索引,此处会存放表中所有的数据记录,辅助索引会引用主键,通过主键到主键索引中找到对应数据行);

B树和B+树的区别:(1)B树其关键字个数等于m-1;B+树是m;(2)对于B树来说,如果中途命中那么就直接返回[其非终节点包含需要查找的有效信息];而B+树,需要到终结点去找;且叶子节点本身根据关键字的大小自小而大的顺序链接;

索引类型:主键索引/唯一索引/普通索引/全文索引和复合索引

(1)普通索引:仅加速查询;(2)唯一索引:加速查询+列值唯一(可以有null);(3)主键索引:加速查询+列值唯一(不可以为Null)+表中只有一个;(4)组合索引[要遵循最左前缀原则]:多列值组成一个索引,专门用于组合搜索;

(5)全文索引:对文本内容进行分词,进行搜索;通常使用倒排索引来实现;倒排索引同B+树索引一样,是一种索引结构;

(6)主键索引:唯一,非空,自增;

哈希索引:只有memory引擎支持;无序的结构;其弊端就是仅仅满足等值查询,范围查询就不能满足;

(5)索引的缺点:1.索引需要占用物理和数据空间,随着数据量的增加而增加,其创建和维护耗费的时间也越来越多;2.性能损失,当在表中进行更新操作时,其索引也会有相应的操作,会较低更新的速度;因此要尽量避免在改动较多的列上创建索引;

(6)判断是否应该建索引:较频繁作为查询条件的字段应该创建索引;唯一性太差的字段不适合单独创建索引;

(7)数据库优化:1.合理的设计库与表(字段类型的选择);2.合理建立和利用索引; 对于索引来说,要判断哪些字段适合加索引,哪些字段不适合加索引;

1.Mysql哪些字段适合建立索引?

(1)表的主键,外键必须有索引;(2)数据量500左右的表应该有索引;

(3)在where从句,group by从句,order by从句以及on从句中的列添加索引;

(4)索引应该建立在选择性高的字段上;

(5)索引应该建立在小字段上,对于大的文本字段甚至超长字段不要建立索引;

(6)复合索引的建立需仔细分析,尽力考虑用单字段索引代替;

(7)不适合建立索引:表记录少//经常插入,删除,修改的表,不要建太多索引;//删除无用的索引;

2.mysql的索引失效情况

(1)索引列为null;(2)mysql查询只使用一个索引,多个索引,后面失效(顺序);

(3)Like%前置,模糊查询,不走索引,全表查询,但如果是尾部,则不会失效;(4)not in 索引失效,in 不会造成索引失效;

(5)对索引列进行函数操作;(6)where条件是or,不论左右列是否有索引,都会失效;

(7)范围查询时,右边的列,不能使用索引;(8)字符串不加单引号,造成索引失效;

3.mysql数据库的存储引擎

(1)mysql的存储引擎:对数据库文件的一种存取机制,如何实现存储数据,如何为存储的数据建立索引以及如何更新,查询数据等技术实现的方法;

(2)mysql存储引擎总共有9钟;常用的数据引擎有myisam,innodb,memory,archive;查询支持引擎的命令:show engines;对于mysql5.5及以上,默认是innodb;之前是myisam;

(3)MyISAM和innodb的区别:

<1>MyISAM存放方式:不支持事务,不支持行级锁,只支持并发插入的表锁;主要用于高负载的select;

索引方式:使用B+TREE索引,但具体实现与innodb不同;

优缺点:优势在于占用空间小,读取数据快;缺点:不支持事务的完整性和并发性;

<2>InnoDB存放方式:支持自增长列,但自增长列值不能为空;支持外键,支持事务以及事务相关功能;支持mvcc行级　　锁;mysql5.6以上才支持全文索引;

索引方式:InnoDB存储引擎使用B+TREE;

优缺点:优势在于提供了良好的事务处理,崩溃修复能力和并发控制; 缺点:占用空间较大,读写效率较差;

4.存储过程;

(1)存储过程就是一段用于完成特定功能的代码;

(2)create procedure demo-in-parameter

5.数据库三范式

(1)列不可拆分;具有原子性;

(2)在1的基础上,有主键,非主键列需要完全依赖主键,不能只依赖于主键的一部分;

(3)在2的基础上,非主键列需要直接依赖于主键,不能存在传递依赖;即(非主键A依赖于非主键B,非主键列B依赖于主键的情况)

(4)反三范式:利用空间换时间;降低范式,增加字段,减少了查询时的关联,提高查询效率;

6.视图(view)

视图也被称为虚拟的表,是一组数据的逻辑表示,其本质是对应于一条select语句,结果集被赋予一个名字;视图名字;

7.触发器怎么写

(1)触发器一旦定义,无需用户调用,任何对表的修改操作均有数据库服务器自动激活相应的触发器;

(2)主要作用是实现主键和外键不能保证的复杂的参照关系和数据的一致性,从而保护表中数据;

8.多列索引的最左原则

(1)(A,B,C)创建了复合索引,相当于创建了(A),(A,B),(A,B,C);

(2)如果索引了多列,要遵守最左前缀法则;指的是查询从索引的最左前列开始,并且不跳过索引中的列;

<1>遵循最左原则(按顺序),如果跳过第一个索引,进行查询,则都不使用索引;

<2>如果跳过中间B索引,那么第一个索引生效,第二个C不生效;

<3>如果语句中没有A,那么其余两个都不会采用索引;

<4>当三个条件都在,且A在第一个,那么B和C无关顺序,索引会生效;

9.表设计,如何进行表设计;

(1)合理的数据库命名规范;(2)合理的字段类型选择规范;

(3)合理的表结构规范;<1>水平拆分:把一张表中的数据拆分到不同的数据库进行存储;<2>把一张表拆分成多张表;

<3>垂直拆分:拆分成不同的数据库,比如会员数据库,订单数据库,支付数据库,消息数据库等;

10.常用的聚合函数

(1)AVG() 平均值;(2)COUNT() 返回指定条件的行数;(3)MAX() 返回指定列的最大值 (4)MIN() 最小值;
(5)SUM() 总和;(6)ROUND() 把数值舍入为指定的效数位数; (7)FORMAT() 对字段显示进行格式化;

11.数据库约束:

(1)主键约束:(PRIMARY KEY,非空和唯一的结合),唯一标识;有助于更快找到;

(2)唯一约束(unique),保证某列的每行必须有唯一值;

(3)默认约束(default),没有给列赋值时得到默认值;

(4)外键约束(foreign key),保证一个表中的数据匹配到另一个表中的值;

(5)非空约束,不能存储null值;

(6)检查约束,保证列中的值符合指定条件;

12.数据库连接查询;

若一个查询同时涉及两个或两个以上的表，则称之为连接查询;

内连接:符合条件的左表与右表的数据查出来;

外连接: 在内连接的基础上保证左表或者右表的数据全查出来;

(1)内连接,a和b同时符合条件的行;(2)外连接,(左外连接left join,右外连接right join,全连接union/union all);(3)交叉连接,笛卡尔积;

<1>左外连接:查询结果保留左表的所有行,右表中不匹配的行默认填充为null;

<2>右外连接:查询结果保留右表的所有行,左表中不匹配的行默认填充为null;

<3>全连接:包括左右表所有行;union(去除重复的数据)/union all(不去除);

(4)where 和 on 的区别:

1.在使用left join/right join 时,on和where条件区别;

1、select * form tab1 left join tab2 on (tab1.size = tab2.size) where tab2.name='AAA'
2、select * form tab1 left join tab2 on (tab1.size = tab2.size and tab2.name='AAA')

<1>on条件在生成临时表时使用,不论on中的条件是否为真,都会返回左/右表中的记录,不匹配的就默认null填充;

<2>where条件则时在临时表生成好后,再对临时表进行过滤;条件不为真的就全部过滤;

2.对于内连接inner join来说,条件放在on和where中,返回的结果集都相同;前者是先求笛卡尔积然后按照on后面的条件进行过滤，后者是先用on后面的条件过滤，再用where的条件过滤。

join过程可以这样理解：首先两个表做一个笛卡尔积，on后面的条件是对这个笛卡尔积做一个过滤形成一张临时表，如果没有where就直接返回结果，如果有where就对上一步的临时表再进行过滤;

13.Mysql主从复制面试之作用和原理(数据库安全问题)

(1)主从复制

用来建立一个和主数据库完全一样的数据库环境,称为从数据库;主数据库一般时准实时的业务数据库;

(2)优点:做数据的热备,后备数据库,主数据库服务器故障后,可切换到从数据库继续工作,避免数据丢失; 架构的扩展,多库存储,降低磁盘I/O访问的频率; 读写分离:报表使用slave,前台使用master;保证速度;

(3)主从复制原理

<1>数据库有个bin-log二进制文件,记录了db的更新事件(更新,插入,删除)语句;

<2>将主数据库的bin-log文件的sql语句复制过来;<3>让其在从数据的relay-log重做日志文件中再执行一次这些sql语句即可;

(4)具体需要三个线程来操作

<1>binlog输出线程:每当有从库连接到主库的时候,主库都会创建一个线程然后去发送Binlog内容到从库;在从库里,当复制开始时,从库就会创建两个线程进行处理;

<2>从库I/O线程;当start slave语句在从库开始执行之后,从库会创建一个i/o线程,该线程连接到主库并请求主库发送binlog里面的更新记录到从库上;从库I/O线程读取主库的binlog输出线程发送的并更新到本地文件,其中包括relay log文件;

<3>从库的sql线程:从库创建一个sql线程,这个线程读取从库I/O线程写到relaylog的更新事件并执行;

14.mysql的慢查询

(1)慢查询日志;用来记录mysql中响应时间超过阈值的语句; 具体环境中,运行时间超过long-query-time值的sql语句,则会被记录到慢查询日志中;默认值是10s;默认情况下,mysql数据库并不启动慢查询日志,需要手动来设置这个参数;如果不是调优,一般不建议启动该参数;慢查询日志支持将日志记录写入文件和数据库表;

(2)慢查询相关参数:slow-query-log:是否开启慢查询日志; log_queris_not_using_indexed;未使用索引的查询会被记录到慢查询日志中;long-query-time,如果运行时间正好等于是不会记录下来;

(3)mysqldumpslow工具:日志分析工具;

常用命令:<1>mysqldumpslow -s r -t 10 日志文件路径(得到返回记录集最多的10个sql)
<2>mysqldumpslow -s c -t 10 日志文件路径(得到返回访问次数最多的10个sql);使用这些命令时,要配合more使用,避免出现刷屏的情况;

(4)explain 可以帮助我们分析 select 语句，让我们知道查询效率低下的原因;

二十二.Redis数据库

1.redis存储的(Value)数据类型,以及如何选择;

(1)String,字符串,最简单的k-v存储; **短信验证码**;

(2)字符串类型的List,列表,因为是有序的,适合存储一些有序且数据固定的数据; **消息队列**

(3)字符串HASH,散列表,包含键值对的无序散列表; **商品详请,个人信息详情,新闻详情**;

(4)字符串集合SET; 可对两个**set进行交集/并集/差集**操作; **查找两个人的共同好友**,**图片清理**

(5)有序的字符串集合ZSET,增加了score参数,自动根据score的值进行排序,适合 **应用排行榜**;

2.在项目中哪些地方使用

(1)垃圾图片的定时清理:set结构; 上传图片,保存套餐,如果只做了上传,没有提交套餐,那么此图片就是垃圾图片;

上传成功,存入set A,保存成功,存入set B,通过取差值,找到垃圾图片,然后清理;

(2)短信验证码5分钟过期:String 结构; (3)购物车数据: HASH结构,Key:用户名 value:购物车数据;

(4)秒杀; LIST结构,做队列;

3.说下redis

(1)redis是C编写的高性能非关系型数据库;

(2)redis键只能为字符串,值支持5钟类型:String/list/hash/set/zset;

(3)redis数据保存在内存中,读写速度非常快,广泛用于缓存方向,每秒/10w次读写操作,性能最快的nosql;

(4)支持数据持久化,支持AOF和RDB两种方式;

RDB:快照模式;每隔一段时间对redis中的数据进行一次持久化; 文件名:dump.rdb;空间占用少,持久化效率高;但不利于重构;

AOF:采用append模式,记录的是数据的变化过程;各种命令; 数据安全,但文件体积大; aof文件;

(5)支持主从复制,主机会自动将数据同步到从机,可进行读写分离;

(6)经常用来做分布式锁和分布式事务;

(7)缺点:不能用作海量数据高性能读写,受限于内存大小; 较难支持在线扩容;

不具备自动容错和恢复功能; 主机从机的宕机都会导致前端部分读写请求失败,需要等待机器重启或者手动切换前端的IP才能恢复;

主机宕机前有部分数据未能及时同步到从机,切换IP后会导致**数据不一致**;.

4.redis的缓存降级

(1)缓存降级的目的是保证核心服务可用,即使有损;有些服务无法降级(加入购物车/结算);防止数据库发生雪崩问题;因此,对于不重要的缓存数据,可以采取服务降低策略;Redis出现问题,不去数据库查询,而是直接返回默认值给用户;

5.缓存击穿/穿透/雪崩问题

(1)击穿:redis没有数据,数据库有;一般是高并发时,访问同一资源,此过程中,该资源过期失效,那么大量请求涌入数据库,瞬间给数据库造成极大压力; (秒杀/限时抢购);解决方案:在活动期间,设置该资源的过期时间为永久;避免该情况发生;

(2)穿透:redis和数据库都没有的数据;如通过id=-1或者很大的值(一定不存在的对象)不断发送请求,程序每次就会去查询数据库,每次查询结果为空,如果有人恶意攻击,那么就会对数据库造成很大压力;解决方案:(1)将用户id作为key,查询不到结果,返回null,并存入redis中,并设置缓存过期时间很短,那么下次访问则返回Null给它;避免数据库遭到攻击;(2)增加接口层校验,用户校验;(3)采用布隆过滤器:将所有可能存在的数据存到一个足够大的bitmap中,对于不存在的数据就会被它拦截;

(3)雪崩:某一时间段,redis中数据大面积过期失效,那么一瞬间大量请求,都落到了数据库上,对于数据库而言,就会产生周期性的压力波峰; 解决方案:将缓存的数据设置不同的失效时间;热门数据时间设置长一些,冷门就短一些;

6.redis缓存预热

(1)缓存预热就是系统上线后,将相关的缓存数据直接加载到缓存系统;这样就可以避免在用户请求的时候,先查询数据库,然后再将数据缓存的问题.

(2)解决:直接写个缓存刷新页面,上线时手工操作一下; 数据量不大,可以再项目启动时自动进行加载; 定时刷新缓存;

7.redis的过期键的删除策略

(1)redis中同时使用了惰性过期和定期过期的策略;

(2)3种:定期过期/定时过期/惰性过期;

(3)惰性过期:只有当访问一个Key时,才会判断该key是否已过期,过期则清除;该策略可以最大化节省CPU资源,但对内存非常不友好;极端情况,可能会出现大量的过期key没有再次被访问,也不会被清除,占用大量内存;

(4)定期过期:每隔一定时间,会扫描一定数量的数据库的expires字典中一定数量的key,并清除其中已过期的Key;

(5)定时过期:每个设置过期时间的key都需要创建一个定时器,到过期时间就会立即清除;该策略对内存友好,但会占用大量CPU资源去处理过期数据;

(6)定期,定时是主动; 惰性是被动删除;

8.redis淘汰策略

(1)8种;noeviction:当内存使用超过配置的时候会返回错误;不会驱逐任何键; alikeys_lru:加入键时,如果过限,通过lru算法驱逐最久没有使用的键;alikeys_random:加入键时过限,则从所有key中随机删除; alikeys_lfu:所有键中驱逐使用频率最少的键;volatile_lru:加入键时,首先从设置了过期时间的键集合中驱逐最久没有使用的键; volatile_random:从过期键的集合中随机驱逐; volatile_ttl:从配置了过期时间的键中驱逐马上要过期的键; volatile_lfu:从所有配置了过期时间的键中驱逐使用频率最少的键;

(2)大致分为:lru/ttl/lfu/random

lru:最近最少使用;ttl:马上要过期;lfu:使用频率最少;

(3)默认是noevicition;对于写请求不再提供服务,直接返回错误;evicition驱逐;

9.redis主从复制

(1)master/slave;一个主数据库可以有多个从数据库;

(2)实现读写分离;master具有读写权限,而slave只有读权限;当写操作导致数据发生变化时会自动将数据同步给从数据库;从数据库只读,并接收同步过来的数据;

(3)可实现容灾备份;master出现故障,利用"sentinel"哨兵机制,把slave其中一个提升为master,让系统继续执行;

(4)slave node主要用来进行 横向扩容;扩容可以提高读性能;

(5)搭建:slave node 指定mater node的ip和端口; 主节点不用改;配置master服务器可跨网络访问;然后在slave的redis.conf中修改 port 为 6380
最后添加 slaveof 192.168.175.128 6379

(6)主从复制的原理

a.slave启动成功会连接到master后发送一个sync同步命令;

b.master在执行完修改命令后,将传送数据文件给slave;

c.全量复制:slave服务在获取数据库文件后,将其存盘并加载到内存中;

d.增量复制:master继续将新的所有收集到的命令依次传给slave.,完成同步;

e.只要重新连接master,一次完全同步将会自动执行;

缺点:延时,由于写操作都在master上,然后同步更新到slave上;

redis高可用性两种方案:redis+cluster // redis+sentinel

10.说一下redis的红锁

(1)redlock是一种算法,redis distrubuted lock,可实现多节点redis的分布式锁;

(2)redisson完成了对redlock的算法封装;

(3)三种特性:互斥访问;(永远只有一个client拿到锁;)避免死锁:不会出现死锁,即使锁定资源的服务器崩溃或者分区,仍能释放锁;容错性:只要大部分redis节点存活(一半以上),就可以正常提供服务;

11.redis的分布式锁实现

(1)分布式锁实现很多种:数据库/redis/系统文件/zookeeper等;

(2)针对单机redis实例,一个Master; 通过setnx获取锁,lua释放锁; 通过redission框架;

(3)针对多个redis; n个redis master; 使用redlock算法/通过redission;

12.说一下redis的哨兵机制

为了解决主从复制架构中出现的宕机情况;

管理多个redis服务器;任务:(1)监控:会不断检查主节点和从节点是否正常运行;(2)提醒:当被监控的某个redis出现问题,sentinel可通过API向管理员或者其他应用程序发送通知;(3)自动故障迁移:当一个master不能正常工作时,

哨兵会开始一次自动故障迁移操作,会将失效master其中一个slave升级为新的master;当客户端试图连接失效的master时,集群会向客户端返回新master的地址;

一般在一个架构中运行多个哨兵;

13.redis集群搭建

(1)redis-sentinel(适合读多写少)/redis-cluster(分布式环境)两种

(2)哨兵模式;当master宕机了,redis本身以及客户端都没有实现自动进行主备切换;

(3)redis-cluster是分布式集群解决方案;3.0版本推出;一组由多个redis组成,推荐6实例;

(4)redis-cluster结构:所有节点彼此互联;使用二进制进行传输;每个节点存储的数据都不一致,且每个节点要保证有副本,作为备用;

二十三.spring框架

1.spring框架

(1)IOC;控制反转; 对象统一由spring核心容器创建,并管理;

xml配置文件:Bean标签:注册bean,把对象交给spring管理; id属性:作为bean的唯一标识;class属性:类的全限定名;

scope属性:singleton单例,不论获取多少次,都是同一个;创建出来存到spring容器中; prototype多例,每获得一次,就创建一个新的对象,创建出来不会存到容器中; request:针对web应用;spring创建对象时,会将此对象存储到request作用域; session:针对web应用;spring创建对象时,会将此对象存储到session作用域;

2.bean的作用范围和生命周期

(1)单例对象: 一个应用只有一个对象的实例;它的作用范围就是整个引用;

生命周期:对象出生:当应用加载,创建容器时,对象就被创建了; 对象活着:只要容器在,对象一直活着; 对象死亡:当应用卸载,销毁容器时,对象就被销毁了;

(2)多例对象:每次访问对象时,都会重新创建对象实例;

生命周期:对象出生:当使用对象时,创建新的对象实例; 对象活着:只要对象在使用中,就一直活着; 对象死亡:当对象长时间不用时,被gc回收;

(3)Bean生命周期:init method/destory method|| @postconstruct初始化/@predestory销毁

3.spring中工厂的类结构图

(1)Classpathxmlapplicationcontext:它是从类的根路径下加载配置文件;

(2)filesystemxml:从磁盘路径上加载配置文件,配置文件可以在磁盘的任意位置;

(3)anotationconfig:当我们使用注解配置容器对象时,需要使用此类来创建spring容器; 用来读取注解;

在工厂初始化的时候,就会创建配置的所有单例Bean,存到spring容器中;

4.实例化bean的三种方式

1.无参构造方法2.静态化工厂3.实例工厂实例化

5.DI依赖注入

(1)依赖注入:交给spring创建的bean中可能有一些属性(字段),spring在创建时,同时会把bean的一些属性给赋值;

(2)动态地将某种依赖关系注入到组件中;

6.依赖注入方式(4种)和实现方式(2种)

(1)依赖注入方式:set方法注入:property(最常用,支持注解+xml); 构造器注入:指带有参数的构造方法注入;construct

静态工厂的方式注入:通过调用静态工厂的方法来获取自己需要的对象;只支持xml;

实例工厂的方法注入:获取对象实例的方法不是静态的,先new工厂,在调用普通实例方法,只支持xml;

(2)实现方法: 注解(@Autowired;@resource@required) 配置文件

7.spring5和4的区别

(1)jdk版本要求;5的话基于jdk1.8,低于的不能使用;

8.spring注解开发

java配置类中@bean使用注解开发,默认创建的对象是方法名(首字母小写); 此注解只能写在方法上;

配置组件扫描,让spring扫描该基础包下的所有子包注解;在需要spring创建对象的类上使用注解,有注解,则会生成这个实例,如果不指定value的话,就默认使用类名的名字,第一个字母小写;

(1)@component==

(2)@controller@service@repository;

9.spring中的注解

(1)@value:注入基本数据类型和string类型; 第三方使用xml配置开发,自己写的使用注解开发;

10.spring整合mybatis

(1)配置文件datasource;有内置,数据源;(2)sqlsessionfactory;加载核心配置文件
(3)mapperscannerconfigurer;扫描dao接口创建代理对象; 创建代理对象的目的是执行sql语句;

11.纯注解开发

(1)@configuration;指定当前类是一个spring配置类;获取容器时,使用annotationapplicationcontext;
(2)@import导入其他配置类;@propertysource用于加载.properties文件中的配置;

12.AOP面向切片编程

在程序运行期间，不修改源码,在需要执行的时候，使用动态代理的技术对已有方法进行增强。

spring内部,发现如果目标对象没有实现接口,就会默认采用cglib代理;如果实现了接口,默认采用java动态代理;aop典型应用就是 事务管理;

13.spring事务 (开启/提交/回滚)

(1)编程式(硬编码方式)事务:一步一步实现事务管理;

(2)声明式事务:AOP思想,事务代码不需要我们编写;xml文件方式:/配置切入点(pointcut)以及配置切面(aspect)前置通知,异常通知,最终通知;注解方式:@Transactional方式;

(3)事务的话,xml要配置事务规则以及事务AOP;

事务规则:tx:advice配置事务属性;tx:attributes指定方法;参数:read-only是否时只读事务,**默认false**;isolation指定事务隔离级别,**默认使用数据库的默认隔离级别**;propagation:指定事务的传播行为;timeout:指定超时时间;**默认永不超时**;rollback-for:用于指定一个异常,当执行产生该异常时,事务回滚;产生其他异常,事务不回滚;**默认任何异常都回滚**;no-rollback-for:用于指定一个异常,当产生该异常时,事务不回滚,产生其他异常时,事务回滚,**默认任何异常都回滚**;

事务AOP:aop:config配置事务AOP;aop:pointcut定义切入点;该在哪些类和哪些方法中加入事务管理;aop:advisor将切入点和事务的建议绑定到一起;

(3)基于和命名的声明式事务,可以充分利用切点表达式的强大支持,使得管理事务更加灵活;

(4)注解方式,在需要实施事务管理的方法或者类上添加@Transactional;指定事务规则就可以实现事务管理;

(5)声明式事务是建立在AOP上的.本质就是对方法前后进行拦截,在目标方法开始之前加入一个事务,在执行完后根据执行情况提交或回滚事务;

(6)最大的优点就是不需要通过编程方式管理事务;不需要在业务逻辑代码中掺杂事务管理的代码,只需在配置文件中做相关的事务规则声明;便可以将事务规则应用到业务逻辑中;

(7)**事务的传播行为的作用**:一般将事务设置在service层,当我们调用其中一个方法时,它能够保证我们的这个方法,在执行对数据库更新操作保持在一个事务中,要么全部成功,要么全部失败;如果你的service层的方法最终,除了调用dao层方法之外,还调用了本类的其他的service方法,那么在调用其他service方法的时候,必须保证两个service处在同一个事务中,确保事务的一致性; 传播特性就是解决这个问题的;

事务传播行为的取值:propagation_required;默认值; 如果当前没有事务,就新建一个事务;如果已经存在一个事务中,那就加入到这个事务中; supports:如果当前没有事务,就以非事务方式执行;madatory:如果当前没有事务,就抛出异常;

14.使用spring框架的好处

(1)轻量级;(2)IOC.通过控制反转实现了松散耦合;(3)AOP,不需要再业务逻辑代码中掺杂事务管理的代码;(4)容器:spring创建并管理应用中对象的生命周期;(5)事务管理:提供一个持续的事务管理接口,可通过配置文件或者注解实现;很方便;(6)提供了全局异常处理;只需要声明一个全局异常处理器就可以捕获所有异常信息;

15.spring中应用到的设计模式

(1)工厂设计模式:使用工厂模式通过BeanFactory/applicationcontext创建bean对象;

(2)代理者模式:aop声明式事务;

(3)单例设计模式:bean默认都是单例;(4)模板方法模式:jdbctemplate;

<1>饿汉式:私有空参构造方法; 定义一个私有的静态变量并进行初始化赋值; 定义一个公共的静态方法,返回对象;

<2>懒汉式:私有空参构造方法: 定义一个私有的静态变量不赋值; 定义一个公共的静态方法,判断有没有创建对象,没有则创建,有则不创建,返回创建好的;

(5)适配器模式:aop增强或通知就使用了;还有mvc中的dispatchservlet;

适配器模式:将一个接口转换成客户希望的另一个接口,使得接口不兼容的那些类可以一起工作;mvc中使用:是因为dispatchservlet要根据请求信息调用映射处理器,解析请求对应的处理器,最终生成链接,再进行访问;mvc中controller种类众多,不同类型的,需通过不同方法对请求进行处理;如果不利用适配器,那么直接获取后,我们需要自行判断;

16.Mybatis中的 #{}和${}的区别

(1)#{参数名} 底层sql语句的执行是预编译占位符的方式执行,安全; 防止sql注入;

(2)${参数名} 底层sql语句的执行是拼接字符串方式执行,不安全,有sql注入攻击;

(3)#{参数名}预编译处理,将sql中的#{}替换为?,再调用preparedstatement的set方法来赋值的时候都会加上2个单引号;

(4)where查询/like查询时,选择#{}; order by/from表名,只能使用${};

(5)#{}可以对数据类型进行自动转换.而${}将所有数据都当作字符串处理;

(6)statement和preparedstatement的区别:statement在执行同一条sql语句多次的时候,每条sql语句执行都要经历3个阶段(发送sql/数据库执行编译/执行命令);preparedstatement首先创建preparedstatement对象时,就传入sql语句模板给数据库去预编译;在执行同一条sql语句多次的时候,都不需要编译,只需要传递占位符的值去执行即可;

17.mybatis模糊查询Like语句编写

(1)传入参数中有%,sql语句使用#{};(2)传入参数中有%,sql语句不建议使用${};(3)sql语句中使用%,sql使用#{};要注意选择合适的引号select * from foo where bar like "%"#{value}"%";(4)sql语句中使用%,sql使用#{}.配合concat函数;

二十四.springmvc框架

1.springmvc介绍

是一种基于java实现的mvc设计模型的请求驱动类型的轻量级WEB层框架;参数绑定/调用业务/响应/分发转向等;

2.spring执行流程

服务器启动,读取web.xml中的配置创建spring容器并初始化容器中的对象;浏览器发送请求,被dispatchservlet拦截,通过映射器将请求路径映射成一个执行链接,然后适配器根据此链接匹配到相应的处理器去执行请求;处理完请求后,将方法的返回值,借助视图解析器找到对应的结果视图,最终进行渲染并展示页面;

3.注解

(1)@requestmapping:建立请求URL和处理方法之间的对应关系;可作用在类和方法上;

(2)表单类型请求参数的绑定:基本类型或string类型,其参数名必须和控制器中方法的形参保持一致; list和map集合接收参数前面必须写@RequestParam; pojo类型,则参数名和类型与pojo属性保持一致;

(3)json类型的请求参数;采用@RequestBody;将json类型的请求参数封装到POJO对象或者Map中;

(4)@PathVariable:获取restful风格的url上的参数;restful同一请求路径,但根据请求方式的不同达到不同的效果;

POST//PUT//GET//DELETE

(5)@requestHEADER;获取请求头信息;

(6)@ResponseBody;将对象转为指定格式json.xml;响应给客户端;

4.返回值

(1)controller:返回字符串可指定逻辑视图名,通过视图解析器解析为物理视图地址;

(2)ModelAndView:该对象可用作控制器方法的返回值; 数据模型和视图;

二十五.springboot//spring cloud

1.springboot介绍

根据spring框架中存在的问题,(1)jar包冲突;(2)配置太多(3)容器要额外找;它提供了一些自动配置的依赖包,自动嵌入servlet容器,提升了开发效率;

2.注解

(1)@value,获取配置文件中的值;@Value("${name}");(2)@ConfigurationProperties(prefix = "person");

(3)@ComponentScan:扫描当前类所在包以及子包;(4)@Import导入其他配置,让spring容器进行加载和执行;(通常用于加载配置文件中的Bean)(5)@Condition;(6)@EnableAutoConfiguration开启自动配置;

3.redis的序列化机制

出现乱码,是由于默认的序列化机制导致;默认使用redistemplate操作数据时,Key value都需要实现序列化;要解决此问题,将key的序列化机制改为字符串序列化机制;

4.版本控制原理:

由于项目继承于spring-boot-starter-parent,而他又继承于spring-boot-dependencies，dependencies中定义了各个版本;通过maven的依赖传递特性从而实现了版本的统一管理;

5.自动配置执行的原理及流程

(1)添加起步依赖后,会自动创建该对象并放置在spring容器中;

(2)切换服务器,必须导入依赖,并排除当前依赖,默认tomcat;

Springboot启动时会通过@EnableAutoConfiguration[**@import selectimports()方法可获取要加载bean的全路径的字符串数组,spring通过反射创建对象**]注解找到META-INF/spring.factories配置文件中的所有自动配置类,将其加载到Spring容器中;而所有导入的bean对象所需的属性都是由该类中@EnableConfigurationProperties注解开启配置属性，而它后面的参数是一个ServerProperties类,该类上通过@ConfigurationProperties(prefix=""以开头的)注解与配置文件中对应的属性进行绑定;此外,每一个自动配置类都是在某些条件下才会生效,以注解的形式体现;@ConditionalOnBean：当容器里有指定的bean的条件下。@ConditionalOnMissingBean：当容器里不存在指定bean的条件下。@ConditionalOnClass：当类路径下有指定类的条件下。@ConditionalOnMissingClass：当类路径下不存在指定类的条件下。@ConditionalOnProperty：指定的属性是否有指定的值，比如@ConditionalOnProperties(prefix="xxx.xxx", value="enable", matchIfMissing=true)，代表当xxx.xxx为enable时条件的布尔值为true，如果没有设置的情况下也为true。

6.springcloud Eureka

(1)注册中心(可以是个集群);服务的管理/注册/发现,状态监管/动态路由;

(2)服务调用者无需自己寻找服务,Eureka会自动匹配服务给调用者;

(3)Eureka与服务之间通过心跳机制进行监控; 提供者定期通过HTTP方式向Eureka刷新自己的状态;

(4)Eureka会将对应的服务提供者地址列表发送给消费者,并且定期更新;

(5)服务续约,如果关闭自我保护机制,超过设定时间没有发送心跳,那么就会从服务列表移除;

(6)消费者启动时,会检测是否获取服务注册信息配置,每隔一定时间,会重新获取并更新数据;

(7)失效剔除:每隔一段时间将清单中没有续约的服务剔除;(关闭自我保护)

(8)自我保护:Eureka会统计服务实例最近15分钟心跳续约比例是否低于85%,如果低于触发自我保护机制;自我保护机制下,不会剔除任何服务,保证大多数服务依然可用;默认是打开;enable_self_preservation设置;

(9)集群:配置多个server,让彼此之间相互注册,当服务提供者向其中一个eureka注册服务时,那么就会共享到其他上;

7.Eureka与zookeeper的区别

(1)CAP定理:数据一致性,服务可用性,分区容错性(断网);

(2)zookeeper注重**一致性**,;假如一个master节点故障,剩余就会重新选举,(30/120s)选举期间整个集群都不可用,那么注册服务在此期间处于瘫痪状态;而Eureka注重**可用性**;各节点都是平等的,只要有一台还在就可以保证可用性,只是可能查询到的不是最新的;

(3)Eureka各节点平等,zookeeper有leader和follow角色;

(4)zookeeper采用过半数存活原则,而Eureka采用自我保护机制来解决分区问题;

8.spring cloud ribbon(默认是轮询)

(1)解决集群服务中,多个服务高效率访问的问题;有助于控制HTTP客户端行为,为ribbon配置服务提供者地址列表后,就可以基于负载均衡算法,自动帮助服务消费者请求;

9.spring cloud hystrix 熔断器

(1)作用:用于隔离访问远程服务,第三方库,防止出现级联失败(雪崩效应);

(2)微服务中,一个请求可能需要多个微服务接口才能实现,会形成复杂的调用链路;如果某服务出现异常,请求阻塞,用户得不到响应,容器中线程不会释放,那么越来越多的请求堆积,最终会导致服务器资源耗尽;

(3)线程隔离模式和服务降级模式//信号量模式

1.线程隔离:指的是hystrix为每一个依赖服务调用一个小的线程池,如果线程池用尽,调用会立即被拒绝,默认不采用排队;

2.服务降级:优先保证核心服务,而非核心服务不可用;触发原因:线程池用尽,请求超时;

(4)熔断器3个状态:闭合状态,所有请求正常访问;

打开状态,所有请求都会被降级,会对请求情况计数,当一定时间失败请求达到阈值,则触发熔断;默认阈值是50%,请求次数最低不少于20次;(请求20次,出现失败比例达50%)

半开状态:打开状态过会儿会进入休眠时间(5秒),休眠时间过后进入半开,会判断下一次请求的返回状态,如果成功,则切回关闭状态;如果失败,切回打开状态;

(5)如何实现一个局部熔断:

定义一个局部处理熔断的方法failback();在指定方法上使用@hystrixcommand(fallbackmethod="failback");

创建回调类,编写熔断时要返回的内容,fallbackmethod指定回调类;

(6)实现全局方法熔断

定义一个全局处理熔断的方法defaultfailback();在类上使用@DefaultProperties(fallbackmethod="defaultfailback");创建回调类,编写熔断时要返回的内容,fallbackmethod指定回调类;

10.spring cloud Feign

(1)声明式webservice客户端,集成了ribbon负载均衡和hystrix熔断器功能;支持请求压缩;可进行日志级别设置;

(2)通过动态代理,帮我们生成实现类; @Feignclient声明feign客户端,并指明服务名称;接口定义方法,feign会根据注解帮我们生成url地址(拼接可访问地址);

(3)使用:引入open-feign依赖; 创建feign接口,指定要调用的服务名称; 最后开启@EnableFeignClients启用Feign;

(4)Feign支持四种级别:NONE:不记录任何日志;默认值 BASIC:仅记录请求的方法,url以及响应状态码; HEADERS:在BASIC基础上,额外记录了请求和响应的头信息; FULL:记录所有请求和响应信息;

11.spring cloud Gateway

(1)Gateway网关的介绍

为微服务提供统一的路由管理,可以在路由管理基础上进行一系列的过滤,做一系列的监控操作以及限流;解决跨域，安全，权限校验，维护不方便的问题;动态路由

<1>核心功能:过滤(Filter)+路由(route); 断言predicate:路由转发规则(条件);

<2>路由地址通过lb协议+服务名称时,gateway会将服务名解析为实际的主机和端口,并通过ribbon进行负载均衡;

<3>过滤:针对请求,做一些额外的处理,再转发;常用于请求鉴权,服务调用时长统计,修改请求或响应header,限流,去除路径等;

(2)过滤器的分类:全局过滤器:作用在所有路由上; 局部过滤器:配置在具体路由下,只作用于当前路由上[前缀prefixpath/去除前缀StripPrefix];

(3)介绍下Zuul: 也是微服务网关,对请求提供路由及过滤功能;

12.spring cloud Config

(1)配置中心,方便配置文件集中管理,支持配置文件放在配置服务的本地,也支持配置文件放在远程仓库Git;本质是一个微服务,要注册到服务中心;两个角色,一个server(管理配置信息)/一个client(获取配置信息);

(2)bootstrap.yml相当于项目启动的引导文件，内容相对固定;application.yml文件是微服务的常规配置参数，变化比较频繁;

(3)修改配置文件,需要重启,才会重新加载更新内容;

13.spring cloud Bus

(1)基于rabbitMQ实现,默认使用本地的消息队列服务;用轻量的消息代理将分布式的节点连接起来; 消息总线整合,实现配置中心的配置自动更新,不需要重启微服务;本质:利用了MQ的广播机制;

(2)加入@RefreshScope刷新配置注解;

(3)实现消息分发:请求地址访问配置中心的消息总线,接收到请求后,向消息队列发送消息,其余微服务会监听消息队列,接到消息队列中的消息后,会重新从配置中心获取最新配置信息;

二十五.zookeeper

1.zookeeper介绍

管理服务,维护和监控存储数据的状态变化; 应用场景**:注册中心/ 配置中心**(数据发布/订阅即所谓的配置中心：发布者将数据发布到ZooKeeper一系列节点上面，订阅者进行数据订阅，当数据有变化时，可以及时得到数据的变化通知，达到**动态获取数据**的目的)

2.watch机制

订阅-发布功能(观察者模式);观察者会订阅一些主题,当主题发生变化时,就会自动通知观察者; 采用一种推拉结合的模式;一旦主题改变,只会发送事件的节点信息,不会包括具体的变更内容;

3.zookeeper集群搭建

通常由奇数servers组成;为了保证leader选举,能得到多数支持;

方式(1):伪分布式集群搭建:在一台机器上启动多个zookeeper,在启动每个zookeeper时,分别使用不同的配置文件来启动,设置不同的参数;

方式(2):多台机器各自配置,将各自互相加入服务器列表;

角色:leader:写操作的唯一处理者;保证集群事务处理的顺序性;follower:跟随者;读操作请求;转发事务请求给Leader,参与集群选举; observer:观察者角色:针对访问量大的,观察zookeeper集群二点最新状态变化并将这些状态同步过来;对于读操作可独立处理,对于写,转发给leader;

4.负载均衡算法

zookeeper集群,每台服务器启动时都会去zookeeper的servers节点下注册临时节点,每台客户端都会去节点下取得可用的工作服务器列表;根据一定的负载均衡算法来得出一台工作服务器,与之建立连接;

二十六.Dubbo(默认随机)

1.RPC:远程服务调用;组件:服务调用者/客户端存根/服务端存根/服务提供者; RPC框架,要求传递的参数和实体类要实现序列化;

2.zookeeper(树形结构)中存放dubbo服务结构;作为它的注册中心;

3.@Reference是dubbo的注解;表示订阅服务;@service表示提供服务;

4.dubbo协议:TCP,长连接,单连接;NIO异步传输; RMI:TCP,短链接,多连接;

5.负载均衡

将请求分摊给多个操作单元上进行,共同完成任务;随机//轮询/最少活跃调用数;

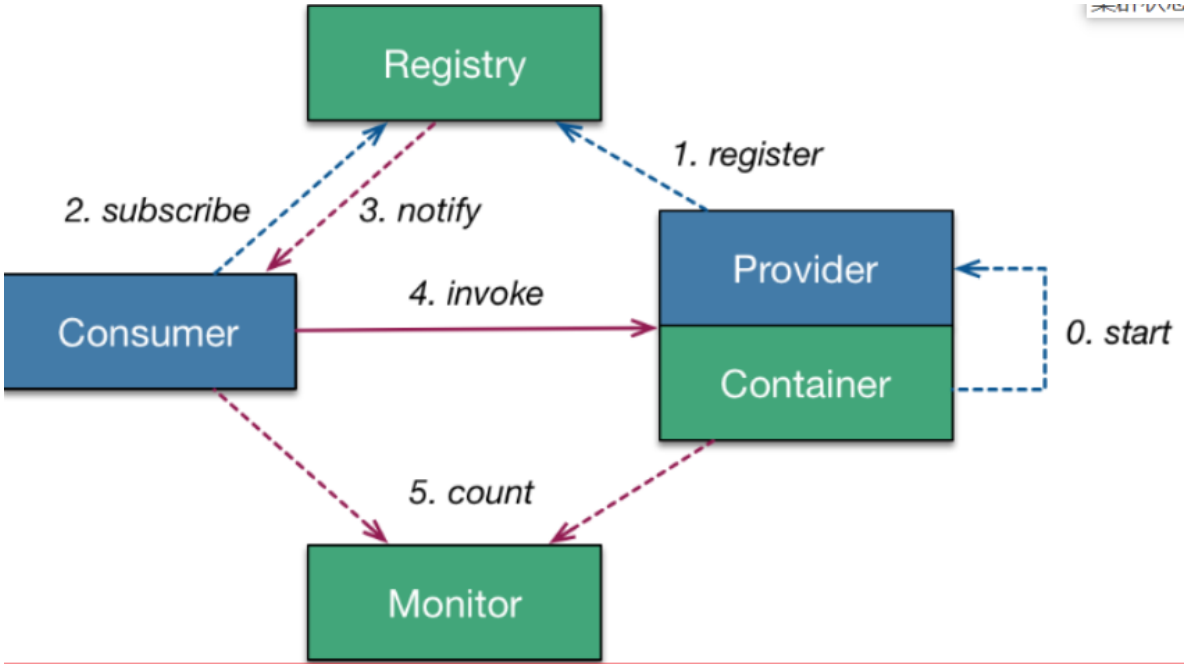
(1)随机,调用量越大分布越均匀;(2)轮询,存在慢的提供者累计请求问题;如果集群中有一台比较慢的服务器,每次请求调用都会卡在这里,久而久之,所有请求都会卡在这台机器上;(3)最少活跃调用数,越慢的提供者其调用前后计数差会越大;(4)一致性Hash;当某一台提供者挂掉时,原本发往该提供者的请求,基于虚拟节点,平摊到其他提供者,不会引起剧烈变动;

6.配置中心

项目中多个提供者,配置了相同的数据源,如果此时修改,必须同时修改,麻烦; 可以将配置信息存储在zookeeper中,如果要修改数据源配置,使用watch机制,同时对提供者数据源信息进行更新;

7.dubbo组件

(1)服务提供者:暴漏服务的服务提供方,启动时注册服务地址;(2)消费者:调用远程服务,启动时订阅服务地址;(3)注册中心(基于长连接推送变更数据);(4)monitor:统计服务的调用次数和调用时间的监控中心,每分钟发送一次;(5)容器:负责启动,加载,运行服务提供者;



8.dubbo的<服务启动/依赖检查 配置>

(1)它指的是,dubbo缺省会在启动时检查依赖的服务是否可用,不可用时会抛出异常,阻止spring初始化完成,以便上线时发现问题,默认"check=true";可以通过check=false关闭检查;

(2)可以关闭单个服务的检查;在dubbo:reference配置check=false,也可以关闭所有服务的检查dubbo:consumer配置check=false;

(3)对于check配置,二者配一个就行;(4)dubbo:registry中也有check配置,默认为true,只是检测注册中心是否存在;

二十七.RabbitMQ

1.MQ消息队列是应用程序之间的通信方法;

任务异步处理:将不需要同步处理的并且耗时长的操作由消息队列通知消息接收方进行异步处理;提高了应用程序的响应时间;

应用程序解耦合:MQ相当于一个中介,生产方通过MQ与消费方交互,它将应用程序进行解耦合; 流量消峰;

2.实现MQ两个协议:AMQP,JMS;

AMQP:高级消息队列协议,是一个进程间传递异步消息的网络协议,直接定义网络交换的数据格式;

两者区别:JMS定义了统一的接口,对消息操作进行统一;AMQP通过规定协议来统一数据交互的格式;

JMS限定了必须使用java语言;AMQP只是协议,不规定实现方式,跨语言;

JMS只有(订阅模式/点对点消息模式);

3.Kafka,RocketMQ,RabbitMQ;

Kafka:快速持久化,高吞吐,10w/s,高堆积;支持hadoop数据并行加载; 50w

RocketMQ:(**)保证严格的消息顺序;丰富的消息拉取模式;理论上不丢失消息;支持事务消息;亿级消息堆积能力;毫秒 延迟; 10w

RabbitMQ:使用erlang语言编写; 微妙 延迟; 吞吐量 5w

3.1消息队列模式

(1)工作模式:多个消费端共同消费同一个队列中的消息;[3个角色 生产者/消费者/队列]

(2)订阅模式:多了交换机角色,生产者将消息发送给交换机,交换机负责转发;交换机不具备存储消息的能力;要有队列跟其进行绑定;(Fanout广播;Direct定向;Topic通配符); 交换机将消息交给每一个绑定的队列;

(3)路由模式:

```
1.队列与交换机的绑定，不能是任意绑定了，而是要指定一个RoutingKey（路由key）
2.消息的发送方在 向 Exchange发送消息时，也必须指定消息的 RoutingKey。
3.Exchange不再把消息交给每一个绑定的队列，而是根据消息的Routing Key进行判断，只有队列的
Routingkey与消息的 Routing key完全一致，才会接收到消息
```

(4)通配符模式:都是根据rountingkey将消息路由到不同的队列;只是以一大类(以**开头/结尾)来区别;

4.RabbitMQ的使用步骤:

(1)导入springboot-amqp整合起步依赖;(2)配置RabbitMQ信息,以及路由交换机,队列和路由规则;在合适的地方发送消息给指定的队列名称(rabbitTemplate.convertAndSend);然后再合适的地方编写监听器接收消息;(@RabbitListener监听队列;@RabbitHandler处理监听事务);

5.开发遇到的问题

高版本的rabbitmq配合低版本的erlang，会出现资源占用后不被正常释放的问题;

6.生产者可靠性消息投递

MQ投递消息流程:(1)生产者发送消息给交换机;(2)交换机根据rountingkey转发消息给队列;(3)消费者监控队列,获取队列中信息;(4)消费成功,删除队列中消息;

(1)提供了两种方式用来控制消息的投递可靠性; confirm模式:生产者发送消息到交换机的时机 return模式:交换机转发消息给queue的时机;

(2)消息从product到exchange会返回一个confirmCallback;消息从exchange到queue投递失败会返回一个returncallback;

7.消费者确认机制(ACK)

(1)三种方式:自动确认;手动确认;根据异常情况确认;
(2)自动确认方式:当消息一旦被Consumer接收到，则自动确认收到，并将相应 message 从 RabbitMQ 的消息缓存中移除。但是在实际业务处理中，很可能消息接收到，业务处理出现异常，那么该消息就会丢失;

(3)手动确认方式:需要在业务处理成功后，调用channel.basicAck()，手动签收，如果出现异常，则调用channel.basicNack()等方法，让其按照业务功能进行处理，比如：重新发送，比如拒绝签收进入死信队列等等;

8.消费端限流

(1)在并发量大的情况下,生产方不停发送消息,消息会在队列中堆积很多,当消费端启动时,会瞬间涌入,那就有可能宕掉;所以采用在消费端进行限流操作,每秒钟放行多个消息;这样就可以进行并发量控制,减轻系统的负载,提供系统的可用性;

9.TTL(Time to live 消息过期时间)

当消息到达存活时间后,还没有被消费,会自动清除;

(1)针对某一队列;如果过期,就清除队列中的全部;(2)针对特定;队列中消息过期,则清除特定消息(**注意**:针对某一个特定的消息设置过期时间时，一定是消息在队列中在队头的时候进行计算，如果某一个消息A 设置过期时间5秒，消息B在队头，消息B没有设置过期时间，B此时过了已经5秒钟了还没被消费。注意，此时A消息并不会被删除，因为它并没有再队头);

10.死信队列

(1)当消息成为死信后,可以被重新发送到另一个交换机(Dead Letter Exchange),死信交换机DLX;

(2)成为死信的三种条件:队列消息长度到达限制(最大消息长度); 消费者拒接消费消息; 原队列存在消息过期设置,消息过期未被消费;

(3)死信交换机,可在任何队列上被指定,实际上就是设置某个队列的属性; 当队列中有死信时,rabbitmq会自动将这个消息重新路由到另一个队列中;

设置参数:x-dead-letter-exchange和x-dead-letter-rounting-key;

11.延迟队列

当消息进入队列后不会立即被消费,只有到达指定时间后,才会被消费; 可以根据TTL+死信队列的方式进行达到延迟的效果; 比如:订单业务:用户下单30分钟后未支付,则取消订单;

12.幂等性问题(重复消费)

1.多次请求某一个资源,对于资源本身应该具有同样的结果;超时//网络抖动问题;

(1)通过版本号或者时间戳解决;CAS (2)数据库锁; (3)[消息表]对消息进行判定,消息是有状态的,接收到消息,消费完立即将消费的状态改掉;如果第一次操作完,那么就改掉状态,等到第二次来的时候,一看到消息状态,就会弃用(在事务中,写一个**存储过程**,业务逻辑判断);

2.发生重复消费:要判断该消息的具体事情;[Select不会/update/insert/delete]

例:update set a=? where id =? 情景:第一次执行和第二次执行的区别;(涉及消息的时序性问题) 假如在第一次执行完后将a=4,来了一个线程,将a改为5,第二次执行后,又将a改为4;就出了顺序问题;

13.rabbitmq集群搭建和通信

14.rabbitmq如何保证数据不丢失

(1)保证生产者不丢消息回调接口保证rabbitmq不丢消息;(3)保证消费端不丢消息;

(1)保证生产者不丢消息:确切说写消息,可以开启confirm模式,如果消息写入mq中,那么就会回传一个ack消息;如果不能处理,则会回调一个nack接口,通知消息接收失败,可以重试;

(2)保证rabbitmq不丢消息:

1.开启rabbitmq的持久化(持久化queue和message),消息写入后会持久到磁盘,如果rabbitmq挂掉,恢复之后会自动读取之前存储的数据,一般不会丢;

2.如果没有持久化,就挂掉,可能会导致少量数据会丢失;

3.持久化message,再发送消息时将消息设置为持久化,这样rabbitmq就会将消息持久化到磁盘上;

(3)保证消费端不丢消息:依靠ACK机制(消费端收到消息要有应答/响应),关闭rabbitmq的自动确认通知,使用手动通知,等消费者服务执行完毕没有异常在ack;

(4)rabbitmq发送消息有重试机制(断网)和应答模式(消费端); 默认情况下,rabbitmq会自动实现重试机制,默认5s重试一次,重试策略可修改:最大重试次数以及重试间隔次数;

(5)rabbitmq:在使用amqp,发送异常,消息重回队列;

(6)当投递失败,路由失败,消费失败时,会导致消息失败;

(6)终极保证消息不丢失办法:发送方与消费端两者都有一个**消息表**,用于记录判定**发送**的消息和**接收**的消息是否一致;判定的话根据一个**监控系统**(查询两张表的数据对比),通过定时任务来实现[每个5s生成一个动态图,便于观察];来通过人工干预(增或减)解决; 对于消息的发送和接收同样要做**日志记录**,防止丢失,搭建日志服务平台(ELK),做日志的实时分析; rocketMQ支持**顺序消息/事务消息**;底层采用queue队列保证顺序;

15.MQ事务消息(分布式事务)

在分布式事务中,(发送方)写消息和发送消息要保证原子性,要么都成功,要么都失败;

[1]MQ发送方,向MQ服务器发送消息(状态HALF),服务器就会响应接收成功,而这并不代表真正的成功,MQ发送方就会去执行本地事务,然后将执行结果(commit/rollback)提交给服务器; 服务器收到后,在进行判定,commit就投递消息,rollback就删消息不投递;

注意:执行超时,如果执行超时,就会在服务器中呈现**unkown**状态; 对此,MQ服务器有个**超时机制**,如果是**unkown**状态就会进行回调,回查事务状态;**发送方**就会去检查**本地事务状态**;询问次数有**上限**,到达上限后,就删除(**做日志**,进行记录,然后人工干预);

16.处理消息堆积

(1)消息堆积主要是生产者和消费者不平衡导致;

(2)处理思路:提高消费者的消费速率,保证消费者不出现消费故障;

(3)避免消息堆积:a.足够多的内存资源;b.消费者数量足够多;c.避免消费者故障;

(4)解决:修复consumer,使其恢复正常消费;临时新建一个topic,将其队列增加10或20倍;然后编写临时处理分发程序,从旧的topic中快速读取到新的topic中,再启动更多consumer消费临时新的topic的消息;直到堆积的消息处理完成,再还原到正常的机器数量;

二十八.ElasticSearch(java开发)

1.介绍

es是一个高扩展的分布式全文检索引擎;它可以近乎实时的存储,检索数据;可处理PB级别的数据,通过简单的RESTFUL api来隐藏LUCENE的复杂性; 相当于数据库;

2.es对比solr

(1)Solr利用zookeeper进行分布式管理;而es自身带有分布式协调管理功能;

(2)Solr支持更多格式的数据;而es仅支持json文件格式;

(3)Solr提供的功能更多;而es本身更注重于核心功能;

(4)Solr在传统的搜索应用中表现好于es,但在处理实时搜索应用时效率明显低于es;

3.es对比DB

(1)db:databases(库)--tables(表)--rows(行)--columns(字段);

(2)es:index(库)--types(表)--documents(行)--field(字段);

field四个属性:数据类型/分词/索引/存储(默认不存储);

4.es的原理:

(1)es底层使用的是lucene;(2)lucene使用的是倒排索引的方式进行加快检索速度;

5.为什么用es不用mysql索引

(1)数据库的索引是B+tree;es基于的是倒排索引;

(2)es可以处理分词后的全文搜索;而mysql数据库有时候查询关键字时,Like%word%,会进行全表查;es只需要查word这个词包含的文档id;

(3)es可以处理海量数据的搜索;

二十九.JVM

1.JVM虚拟机组成

包含了两个子系统和两个组件;子系统分别为类加载和执行引擎;组件为运行时数据区和本地接口;

(1)类加载:根据指定的全限定名类名来加载class文件到JVM的内存区域中的方法区;

(2)执行引擎:执行class中的指令;

(3)本地接口:与其他编程语言交互的接口;

(4)JVM内存区域;

2.JVM内存区域

组成:PC寄存器,Java虚拟机栈,本地方法栈,(这三个是各线程独享的),堆和方法区是共享的;

3.GC机制

五个内存中,有3个是不需要进行垃圾回收的;本地方法栈,程序计数器,以及虚拟机栈; 其生命周期和线程是同步的,随着线程的销毁,其占用的内存会自动释放;只要方法区和堆区需要进行垃圾回收,回收不存在任何引用的对象;

判断是否是垃圾数据: 根搜索算法:从一个叫GC ROOTS的根节点出发,向下搜索,如果一个对象不能达到GC ROOTS的时,说明该对象不再被引用,可以被回收;

4.Minor GC机制和Full GC机制

堆内存中主要分三块:分别是 新生代/老生代和持久代

新建的对象都是从新生代分配内存,而新生代大致分为Eden区和survivor区,eden区不足时,会放置在survivor中;当新生代eden区满时,就会触发MInor GC;

老生代或者持久代里面的垃圾回收机制称为FULL GC;速度比MInor GC慢10倍;当老生代满时,会触发FULL GC,会同时回收新生代和老生代;当持久代[方法区]满时,也会触发FULL GC,会导致CLASS Method元信息卸载;

GC算法:新生代主要用复制算法,将不需要清除的对象复制到一块区域,在清除其他无用对象;缺点是需要额外的空间和移动;

老生代:主要使用标记压缩算法;将不需要清除的对象进行标记,清除未标记的对象,然后把活对象向空闲空间移动;在更新引用对象的指针;

5.线上CPU过高的问题解决

(1)通过JDK自带jvisualvm监控工具故障处理分析问题;(2)通过堆页面查找保留前10个对象;(3)重点关注前几个对象,分析问题,通过工具查询到sql和代码;对sql进行分析优化;

6.JVM的主要组成部分和作用

(1)JVM的包含两个子系统和两组件

首先,jdk中的编译工具将java文件编译为class文件;通过**类加载器**负责将字节码文件,加载到**JVM内存区域**,随后由**系统执行引擎**将其解析为底层系统指令;再由CPU去执行,该过程中需要调用**本地接口**来实现整个程序的功能;

7.类加载机制:

全盘委托机制:当一个类加载器负责加载某个class时,该class所依赖和引用其他class,将由该类加载器负责载入;

双亲委派机制:当某个特定的类加载器在接到加载类的请求时,首先将加载任务委托给父类加载器,如果无法完成,则自己去加载;

缓存机制:缓存所有加载过的类,当程序需要使用某个类时,会先去缓存中查找,只有不存在时,系统才会去加载该类,修改class信息后,jvm必须重启;

三十.IO

1.IO

以流向来说,分为输入流和输出流; 按操作单元划分,可分为字节流和字符流//字符:读流和写流;

2.阻塞IO

在读写数据过程中会发生阻塞现象,当用户发出IO请求后;内核会去查看数据是否就绪,如果没有就绪就会等待数据就绪,而用户线程就会处于阻塞状态,用户线程交出CPU;当数据准备好之后,内核会将数据拷贝到用户线程,并返回结果给用户线程,此时,用户才解除阻塞状态;典型的阻塞IO模型例子为:data=socket.read();如果数据没有就绪,就会一直阻塞在read方法;

3.非阻塞IO模型

当用户线程发起read操作后,并不需要等待,而是马上得到一个结果;如果结果是一个error,它就知道数据还没有准备好,于是它可以再次发生read操作,一旦内核中的数据准备好了,并且又再次收到了用户线程的请求;那么它马上就将数据拷贝到用户线程,然后返回;事实上,用户线程需要不断地询问内核数据是否就绪;非阻塞IO不会交出CPU,而会一直占用CPU;这样使得CPU占用率非常高;因此在一般情况下很少使用while循环方式来读取数据;

4.多路复用IO模型

(1)JAVA NIO实际上就是多路复用;多路复用模型中,会有一个线程不断地去轮询监听多个端口的情况,只有当socket真正有读写事件时,才真正调用实际地IO读写操作;在多路复用IO模型中,只需要使用一个线程就可以管理多个socket,系统不需要建立新的线程;这样就减少了资源占用; 因此适合高并发/高频段业务环境;

(2)如果不使用多路复用,那么服务器端就需要开很多线程处理每个端口的请求,在高并发下,会造成系统性能下降;

(3)多路复用IO为什么比非阻塞IO模型的效率高? 主要是因为非阻塞IO中,是用户线程去进行不断地询问socket状态,而多路复用,轮询每个端口状态是内核进行,其效率比用户线程要高;

5.异步IO模型

当用户线程发起read操作后,立刻就可以去做其他事情;且内核收到异步读取请求后,会立刻返回结果,说明read请求已经成功发起,因此,不会对用户线程产生任何block;然后,内核准备好数据后,会

6.selector NIO核心类

它能够检测多个注册通道上是否有事件发生,如果有事件发生,那么就进行相应地响应处理;这就使得只有在连接真正有读写事情发生时,调用真正地实际地IO操作;大大减少了系统开销;

三十一.Docker

将应用程序和开发环境一起打包;然后上传公司仓库,测试人员或运维从上面pull下来直接run; 解决了环境差异问题;

(1)docker pull docker.io应用程序名(下载镜像)

(2)docker run -p 端口号:端口号 --name 应用程序名 -d 应用程序名(容器安装)

(3)docker exec -it 应用程序名 /bin/bash(登录应用)

三十二.Linux

(1)cat: 查看文件内容; (2)more:分页查看文件;(3)tail 用于显示文件后几行的内容; tail -n 文件名:数字 [ctrl+c退出]

(2)创建文件touch; mv 移动文件; cp拷贝文件; rm -rf 不询问直接;删除目录;

(3)打包压缩[tar-zcvf] 解压[tar-xvf]

(4)重启 reboot // ps -ef 查看进程// kill -9 pid // ps -ef | grep vi :查看所有vi的进程; grep是筛选

(5)chmod[参数] (6)rpm -qa 查询所有安装过的软件包 rpm -e --nodeps 包名 (7)启动tomcat ./startup.sh