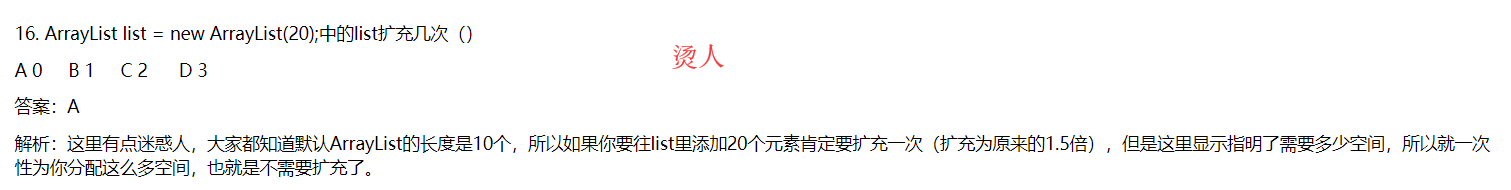
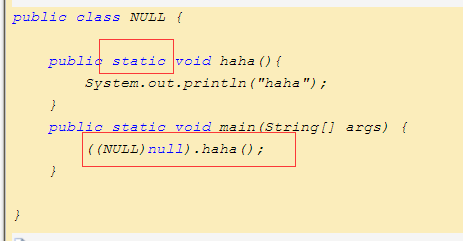
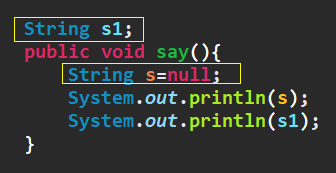


性能比较：new最快，反射，clone，序列化





这样是能够输出的，这里必须是static方法，在java中可以将null强制转换为java类型，但是这里null任然为空，这里能输出结果的原因是static方法(类的方法，不是static的就回报空指针异常)，这里并为创建对象



方法内部的String必须初始化，why：s1是成员变量，存在堆中，在分配堆内存时，JVM会对属性进行初始化操作，局部变量在栈中，生命周期短，如果由JVM初始化，增加开销，所以，局部变量必须手动初始化

接口和抽象类的区别：

抽象类的方法不能是private，这样子类将不能重写此方法，默认public

接口中的方法默认是public abstract，其他修饰符都不可以：private，protected，static，final

接口的定义就是不同类直接的相同行为，如果使用private修饰符就限制了接口的使用范围

接口中声明的属性是常量 static final

接口中不能包含静态代码块和静态方法

抽象类中可以有静态代码块和静态方法

java内存空间：程序计数器，java虚拟机栈，本地方法栈，堆，方法区

程序计数器：

用于记录(保存)当前线程正在执行的字节码的地址

程序计数器每个线程私有，这是一个唯一不会出现OutOfMemoryError的内存区域

java虚拟机栈：

java虚拟机栈使用描述java方法运行过程的内存模型

java 虚拟机栈会为每一个即将运行的 Java 方法创建一块叫做 “栈帧” 的区域

用于存储方法的局部变量表，基本数据类型，引用数据类型，返回值等

方法执行完毕，方法对应的栈帧将会出栈，并释放内存空间

java虚拟机栈线程私有

本地方法栈：

本地方法栈和java虚拟机栈类型，只不过本地方法栈是本地方法(native)的内存模型

堆：

用来存放对象

线程共享，java中只有一个堆

在JVM启动时创建，GC的主要场所

堆：先进先出

栈：先进后出

方法区：

存放被虚拟机加载的类信息，常量，静态变量

线程共享，方法区是堆的一个逻辑部分，整个JVM只有一个方法区

Class对象在方法区中

常量池：

一个类中的public static final变量，类被编译为class文件之后，类的所有

信息存储在class文件中，当class文件被加载到内存中，常量就会存放在常量池中

运行时可以向常量池中添加新常量

java中new一个对象的过程：

1，jvm将class文件加载到方法区中，

根据方法区中该类的信息，确定该对象的大小，一个类的所有对象大小都是一样的

JVM 在一个类被加载进方法区的时候就知道该类生产的每一个对象所需要的内存大小

2，在对内存中划分一块对应大小的内存给对象

3，为对象的成员变量进行赋初值(默认大小)

4，调用构造函数进行初始化

对象属性存放的位置：

局部变量(方法内部)：

当调用对象的方法时，JVM会创建一个方法栈，局部变量是基本数据类型时

变量名和值都保存在栈中，引用类型时，变量名保存在栈中，值保存在堆中

类变量：

基本数据变量名和值都保存在对象所在的堆中

引用类型变量名存放在栈中，对应的值存放在堆中

成员变量存在于堆内存中，随着对象的产生而存在，消失而消失。

局部变量存在于栈内存中，随着所属区域的运行而存在，结束而释放。

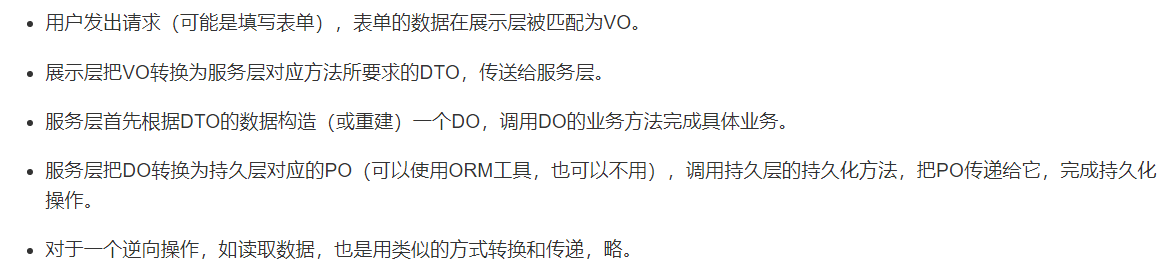
栈中存放的数据只有在执行方法的时候才会开辟，方法执行完被释放

VO：view Object，页面和Controller交换的对象

PO：和数据库交互的对象

DO：从现实世界中抽象出来的有形或无形的业务实体

DTO：Controller和Servlet交换数据的对象



对于绝大部分的应用场景来说，DTO和VO的属性值基本是一致的，而且他们通常都是POJO，因此没必要多此一举。但不要忘记这是实现层面的思维，对于设计层面来说，概念上还是应该存在VO和DTO，因为两者有着本质的区别，DTO代表服务层需要接收的数据和返回的数据，而VO代表展示层需要显示的数据。

网关：跳出当前网络(局域网)

在JS可以使用JSP以及框架的表单式输出内容

# ：id . : class

MySQL：

在Linux下，数据库和表名区分大小写

表的别名区分大小写

列名不区分大小写

Windows下不区分大小写

$.post和$.ajax的区别：

post的作用就是取代复杂的ajax，如果需要在出错时执行函数，请使用ajax

$.post(url,data,function(res){

});

$.ajax({

type:post/get

url

data

success:function(res){

},

error:function(res){

}

});

字符串的不能被修改的好处：

·hashcode，String的hashcode常在Java中被用到，String不变，那么在创建的时候就可以确定String的hashcode，以后

就可以直接使用，如果String可变，那么每次改变后都会重新计算hashcode

·String不可变，线程安全

·简化其他对象的使用，如果Set集合中存放的String，如果对Set集合中的某个元素进行修改后，就会影响到Set集合中不能

出现重复元素的特性(有点牵强)

HashSet<StringBuilder> hs = new HashSet<StringBuilder>();

StringBuilder sb1 = new StringBuilder("aaa");

StringBuilder sb2 = new StringBuilder("aaabbb");

hs.add(sb1);

hs.add(sb2); //这时候HashSet里是{"aaa","aaabbb"}

StringBuilder sb3 = sb1;

sb3.append("bbb");//这时候HashSet里是{"aaabbb","aaabbb"}

System.out.println(hs);

在向HashSet中添加值的时候，sb1，sb2不相等，但是后面动态的修改了sb1所指向的值，但是HashSet并未知道

这里HashSet的K是StringBuilder的hashcode

String s1=new String("123");

String s2=new String("123");

System.out.println(s1.hashCode());

System.out.println(s2.hashCode());

System.out.println(s1==s2);

hashcode相等，s1==s2为false

hashcode计算的是底层数组的每个元素的值

为什么Map结构的K为String

String不可变，hashcode不会变

每个Thread中都有一个ThreadLocalMap，用于存放线程私有的属性

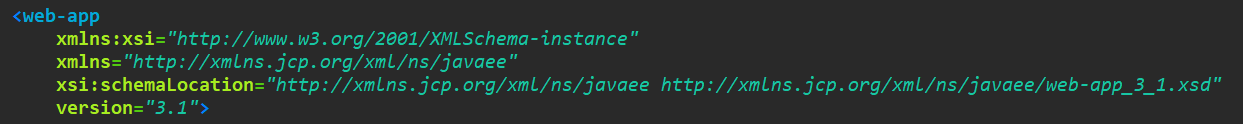
ThreadLocalMap以ThreadLocal为键，属性为值

ThreadLocalMap内部保存对应属性是通过Entry保存的，ThreadLocalMap内部

维护了一个Entry数组，在查找的时候，根据ThreadLocal计算hash(通过开放定址法解决hash冲突),

找到Entry数组的下标，取出Entry中的属性值

ThreadLocal与线程同步机制不同，线程同步机制是多个线程共享同一个变量



在xml文件中有很多的标签，而这些标签又会受到一些约束(标签的位置，标签的子标签有哪些等)，常见的约束文件有schemaLocation和DTD约束

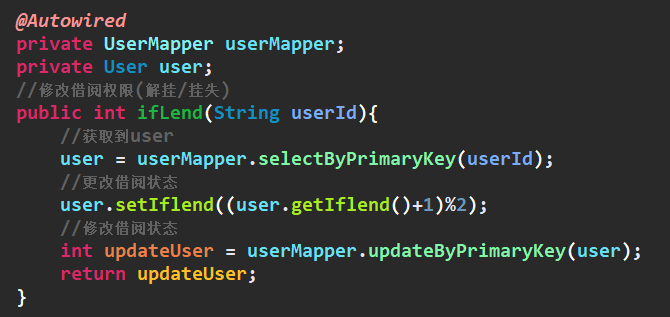
在xml文件中就的引入对应的约束文件，这里以schemaLocation约束为例：

上面这个是web.xml的约束文件，在xml中通过使用xmlns引用约束，所以这里有两个约束，一个是w3c的约束，一个是javaee的约束，任何一个xml文件，必须引入w3c的约束文件，xsi的作用是给引入的约束取别名，这里的xsi：schemaLocation就是w3c约束下的一个属性，这里属性的作用是告知约束对应的具体位置，xmlns只是声明了有这个约束，约束的位置必须由w3c的约束的schemaLocation属性指定

事务commit后，需要与数据库建立连接

MyBatis整合Spring后sqlSession的获取和创建：

Mybatis整合Spring使用的是SqlSessionFactoryBean，Mybatis的Mapper接口的代理对象也是由Spring进行创建的



在service层在可以直接使用Mapper时，先会去判断当前Mapper是否获取了sqlSession，如果没有，则先获取sqlSession，再执行方法，当Mapper中的方法执行完毕后，会自动关闭session

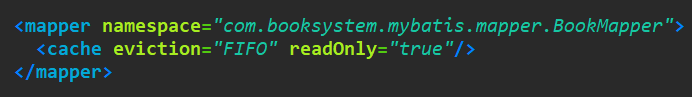
因此，在SSM中每执行一次Mapper的方法，都会执行session的创建和关闭操作

<https://blog.csdn.net/u011686226/article/details/52704158>

Mybatis的缓存在commit或写操作后都会被清空

Mybatis中的二级缓存中除了在setting中开启二级缓存，还要在Mapper的标签(select)中声明useCache为true(默认为true)，这样才会执行二级缓存

Mybatis的二级缓存是基于Mapper的，即每个Mapper文件中共享一个二级缓存，可使用<cache>标签指定缓存的策略

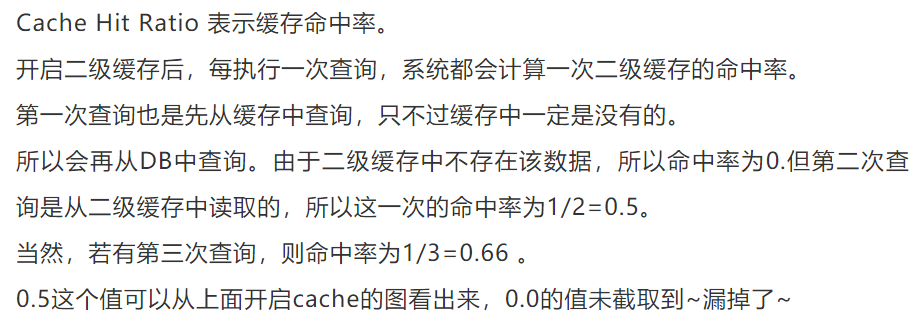


声明第三方缓存的时候，任然是在cache标签中指定

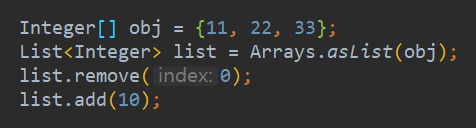
如何Mapper之间向共享缓存，可以使用cache-ref标签来引用共享的Mapper

<cache-ref namespace="cn.chenhaoxiang.dao.UserMapper"/>

在二级缓存中有个命中率(Cache Hit Ratio 表示缓存命中率)



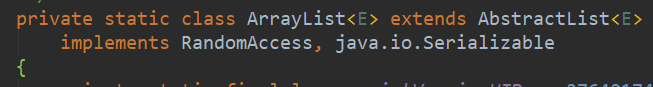
JDK动态代理基于接口，cgLib基于类的继承，JDK动态代理的性能优于cgLib代理

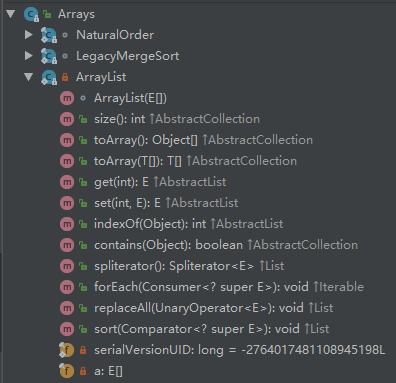




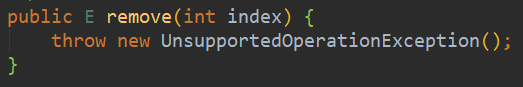
在使用Arrays的asList将数组转化为List的时候，使用remove方法的时候回抛异常

原因：Arrays.asList转化的List是Arrays类下的ArrayList类(class java.util.Arrays$ArrayList)，而不是ArrayList(java.util. ArrayList)





这个ArrayList只实现了这几个方法，而remove方法是AbstractList类的remove方法



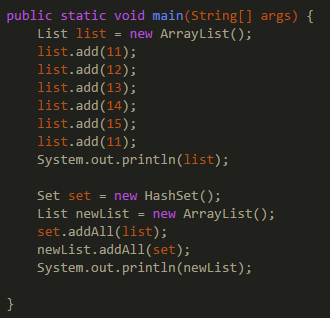
所以会抛出上面的异常

转化为正常的ArrayList(java.util. ArrayList)

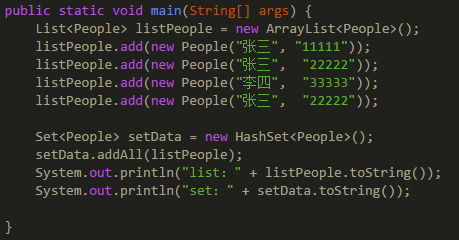
ArrayList<Integer> copyArrays=new ArrayList<Integer>(Arrays.asList(obj));

Collections.addAll(new ArrayList<Integer>(), obj);

List去重操作：

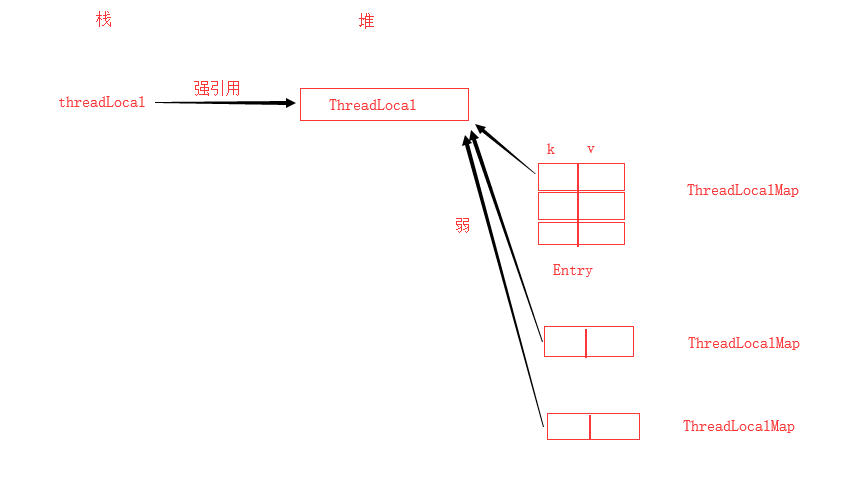


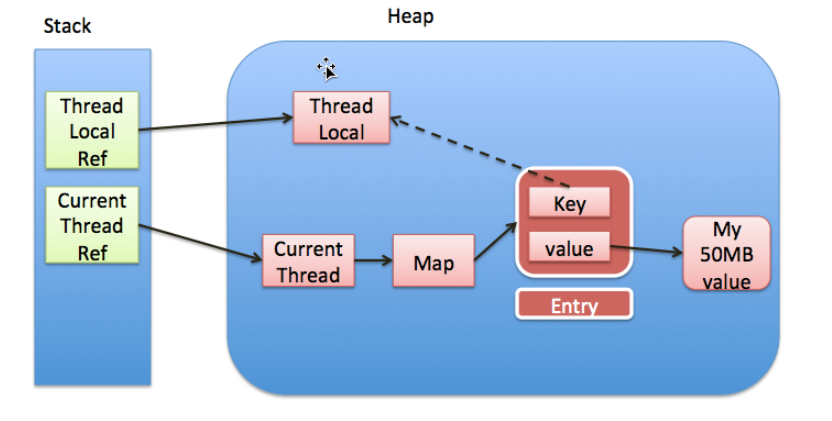
先将List转化为set，利用set无重复元素的特点去重，然后再讲set转化为List(这里的addAll是AbstractCollection的方法，set，list都是Collection的实现类)



但是如果add不是基本数据类型而是引用数据类型就不可以，此时应该重写equals和hashCode方法

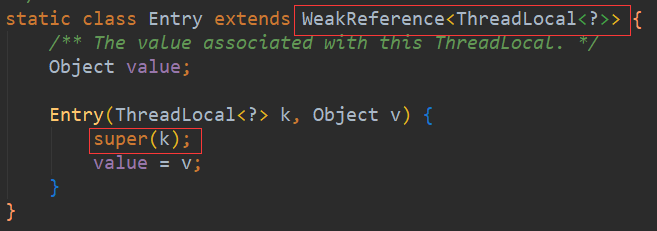
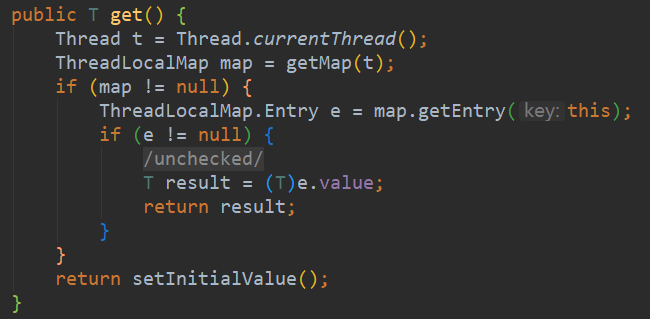
字符串在拼接的时候才要StringBuilder，返回时再toString(返回的是一个new String对象)





ThreadLocal实现线程值私有过程：

每个线程都有一个ThreadLocalMap，ThreadLocalMap中通过ThreadLocal做为K存在一个Entry中，值私有是通过ThreadLocalMap实现的，一个ThreadLocalMap可以存放多个的Entry(一个线程Thread可以有多个ThreadLocal变量)



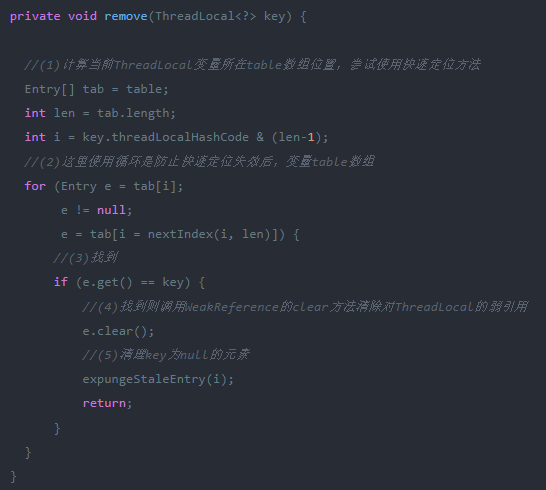
每次调用ThreadLocal的get方法时，实际是先获取当前线程的ThreadLocalMap，然后在到Map中获取V

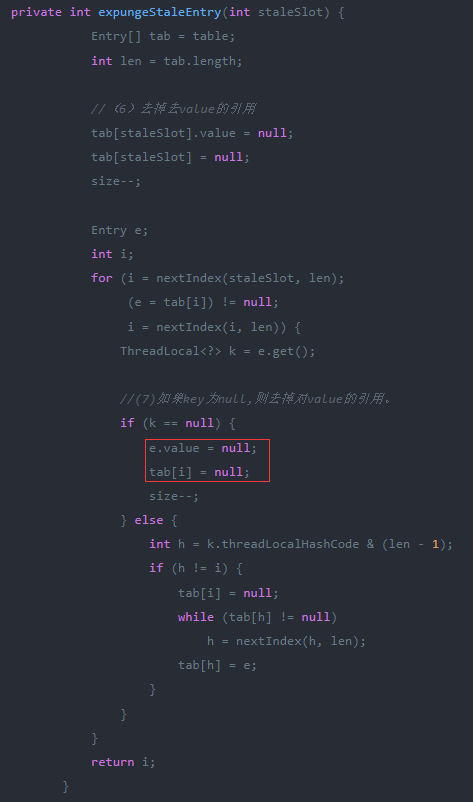
内存溢出：

当执行threadLocal = null 后，Entry将永远不会被访问，因为获取操作是通过threadLocal的get方法获取，Entry将长时间存在内存中，直到当前线程的生命周期结束，但是在这个过程中及很有可能导致内存溢出，所以当不需要再访问Entry时，需要手动调用remove方法，将Entry和Entry中的V设置为null，帮助GC回收

Entry的K为什么是弱引用：

帮助回收ThreadLocal，当没有强引用指向ThreadLocal时，ThreadLocal就被回收(注意一个对象被回收的条件是：不可达，且无强引用指向，执行GC并不是把Entry中的K和ThreadLocal之间的弱引用断开，而是如果ThreadLocal无强引用指向，回收ThreadLocal对象)，一般我们不能手动设置threadLocal = null，引用其他线程还对它有引用，只能通过remove方法来防止内存溢出，而且设置threadLocal = null也不能防止内存溢出





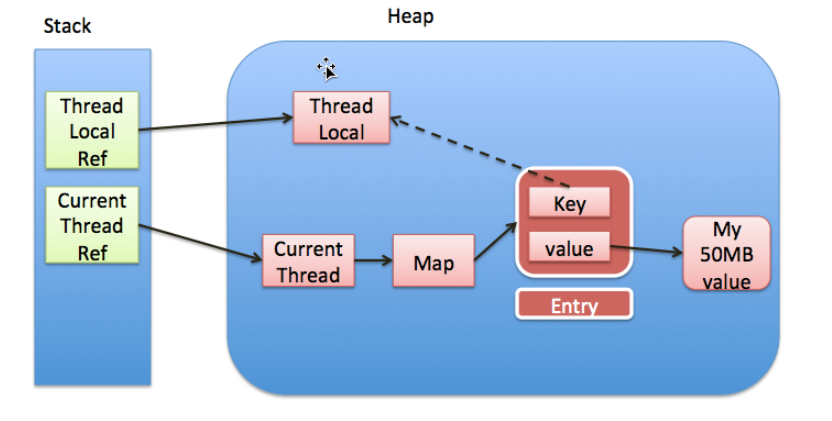
在调用ThreadLocal的get，set，remove方法时，都会执行expungeStaleEntry方法，去清除k为null的Entry，只有ThreadLocal为弱应用，Entry中的K才为null，才能帮助回收

内存溢出的原因：

当我们设置threadLocal = null时，Entry将不会被访问，导致内存溢出(线程死亡之前)，设置threadLocal = null后将会回收ThreadLocal，在设置为null之前需要执行remove(一个ThreadLocalMap中可以保存多个ThreadLocal)，ThreadLocal声明周期过长，导致ThreadLoaclMap中保存了大量无用数据，导致内存溢出(可以手动remove，但是既然remove了也可以结束ThreadLocal的生命周期了)

为什么是弱引用：

当前线程将threadLocal置null后，引用弱引用，其他线程ThreadLocalMap对ThreadLocal的引用为null，在执行set，get，remove方法时将回清除Entry中K为null的Entry，这也说明了为什么采用弱引用，从这里也可以看出另外一个导致内存溢出的操作：线程长时间不执行set，get，remove方法，当线程对threadLocal置null后，其他线程没执行set，get，remove方法清除空K的Entry导致内存溢出



内存溢出的根本原因是Entry，threadLocal = null后，只会回收ThreadLocal对象(Entry指向ThreadLocal为弱引用)，但是Entry任然存在，Entry对应的V将不会被访问，导致OOM，当线程再次执行set，get时(其他的ThreadLocal)会试图请求没有引用的Entry，帮助清除Enrty防止OOM

日志:容易出错的地方，和第三方对接时可以打

减少hash冲突的方式：使用不可变的、声明作final的对象

不可变性使得能够缓存不同键的hashcode，使用String，Interger这样的wrapper类作为键是非常好的选择，不可变，hash不变

string是fianl的，底层数组实现，创建时数组长度固定，所以对String append会新创建String

这个问题主要是CAS的操作机制导致，自旋CAS如果长时间不成功，会导致CPU的大量执行开销。好在JVM虚拟机对于自旋这种情况有一定干预，会设置自旋次数，当超过该次数，就会将线程挂起，让出CPU资源。

IDEA快捷键：

Ctrl+F ；查找

Ctrl+N ：查找类，和Eclipse中的Ctrl+Shift+T类似

Alt+Enter：快速补全，类似Eclipse中的Ctrl+1

Ctrl+ALt+o：导入包

Ctrl+Alt+L：格式化代码格式

redis.set 默认永不失效

redis中的incr和incrby方法：原子操作，set是非原子操作，而inrby是原子操作(好比多线程中的Atomic)

incr每次加1,incrby加到指定值，decrBy 减

hsetnx：当hash中的值不存在时才添加

maven常用命令：

mvn ：compile 编译源代码，生成target文件

mvn ：package 将程序打包，但是只是打包到target下

mvn ：install 将程序打包，安装到本地仓库，供其他项目使用

mvn ：clean 清空target文件，然后再重新执行package/install

mvn ：deploy 上传到私服

错误：

当前代码是否是最新的

在dubbo中所有Bean必须实现Serializable接口，因为服务之间要通信，需要将Bean序列化，然后再序列化显示

如果项目之间有依赖，修改后，必须对项目进行install，然后再重新导入maven依赖

从git拉下代码后，从新引入maven依赖

在将代码上传到git上后，对项目进行打包

修改代码后，不仅要在上传到git上，还要对项目进行打包，防止其他项目引入的不是最新的

注意细节比如输入参数

通过查看源码BeanUtils的copyProperties是先根据属性名判断是否相等，如果相等再调用对应的set和get方法

对字符串，集合一定要进行判空(size==0||=null)

在执行对象的方法是一定要确保对象是否为空

@NotNull注解的判断机制是根据属性的get/set方法判断，不能将@NotNull注解直接放到方法的参数上

使用@RequestParam(value = "id", required = true)对方法参数校验

在使用Mybatis的批量操作时 <foreach collection="list" item="info" >

这里的collection是指添加的集合，但是这里必须写list，不能写Mapper方法的方法参数名称

或者在方法参数名称中使用@Param注解

post请求，在方法参数前添加@RequestBody注解(JSON)

@NotEmpty：String，集合不能为null或长度不能为0

@NotBlank：String不为null或长度不为0，去除尾部的空格

@NotNull：不能为null

方法在重载时，如果一个方法参数为List<A>,一个方法参数为List<B>，则方法不能重载

git log 查看以前的日志

git reflog 查看以后的日志

git reset --hard 版本号 回滚操作

在定义对象的类型的时候，应该考虑数据库中的类型，前端传入参数类型(这里主要是String和Integer,Long之间的问题)

主要是注意转换的时候类型之间的变换

时间yyyy-mm-dd验证的正则表达式

" [0-9]{3}[1-9]|[0-9]{2}[1-9][0-9]{1}|[0-9]{1}[1-9][0-9]{2}|"

+ "[1-9][0-9]{3})-(((0[13578]|1[02])-(0[1-9]|[12][0-9]|3[01]))"

+ "|((0[469]|11)-(0[1-9]|[12][0-9]|30))|(02-(0[1-9]|[1][0-9]|"

+ "2[0-8]))))|((([0-9]{2})(0[48]|[2468][048]|[13579][26])|(("

+ "0[48]|[2468][048]|[13579][26])00))-02-29) "

对数字字符串的验证的正则表达式

"\\d+(\\d+)+"

在验证String是否为Null使用@NotBlank，对Long，Integer验证用NotNull

Linux中 ~ 表示桌面，/ 表示根路径

Linux中搜索命令 find / -name 文件名(根路径下查找) find /etc -name 文件名(etc目录下查找)

Controller方法尽量少，Service中的方法尽量多，细

一个方法包含方法名，方法体，参数及其**注解**

Get请求不能和@RequestBody使用

在使用Gson将对象转化为字符串时，如果对象中有null值，如name = null，那么，在转化的时候不会转化name

解决方式：

GsonBuilder gsonBuilder = new **GsonBuilder**();

Gson gson = gsonBuilder.**serializeNulls**().create();

gson.toJson(new Object());

com.alibaba.dubbo.rpc.RpcException: Fail to start server 可能是启动了两次

maven下载失败：

删除后缀为.lastupdate的文件，重新下载

mvn install 只会在当前分支上打包

git pull 会将远程的所有分支更新到对应分支上

Cookie失效时间是按第一次获取的时间计算，而不是最后一次访问时间

mvn打包后，出现了Spring相关的，但没有自身的相关，可能在pom文件中引入了spring-boot-maven-plugin插件，删除就OK了

当数据库中通过0，1声明状态时，是否需要为此字段添加一个枚举类型的status：

比如删除，是否显示这种只有两个状态的值时，就没必要添加

比如状态很多时，就需要添加，这样前端也可以获取直接使用，因为状态多，在编写代码的时候也不方

便，所以可以声明一个status，其实最好声明一个status，后期万一有新增状态呢，两个状态可以直接使用true/false代替

多个状态就一定要用枚举的status？如果改变不频繁就没必要

在对数据进行写操作时，是否有必要对数据进行合法性校验：

在更新操作时，是否判断更新数据在数据库中存在，有没有根据ID查询吗？

这个要根据实际的情况考虑，一般不用校验，如在更新操作时根据主键ID更新

where id = #{id,jdbcType=INTEGER}

如果id不存在，也就不会更新，返回值为0，此时再判断更新操作是否成功

具体还是要看业务逻辑和Mybatis的mapper文件的书写

判空是有必要的，如果为空，就没必要走数据库，判空也花不了多少时间

Service层是否可以同层之间调用？

当两个Service之间在业务逻辑上关联不大时，是不能同层调用的，需要在AO层调用

当两个Service之间在业务逻辑上关联大时，本来两个Service就是由一个Service拆分而来的，就可以同层调用的

注意命名：要见名知意，在service中不要使用select，get 要见名知意，如getXxx

而在mapper1文件就可以使用select，insert命名

service层主要是读写数据操作

webservice层只是对前台数据进行转换

AO层业务逻辑处理层

@RequestBody不能在Get请求中使用，@RequestBody的作用是将请求参数封装为对象，操作对象是写操作，而get请求是读操作

@ModelAttribute和@RequestBody的区别：

@ModelAttribute作用于方法，类和方法参数上

@RequestBody作用于方法参数上

@RequestBody注解将JSON字符转化为对象

@ModelAttribute将K-V的键值对转化为对象

函数式编程主要是起一个约束的作用

数据库连接和远程操作需要连接，这个连接操作应该放到service层中，连接操作属于服务，应该直接

暴露给使用者，即使有权限，也应该将连接操作放在service中，使用者提供token来判断是否有权限

这个属于服务，是供别人使用，不应该再登录，即使有权限，登录也应该在service中

连接操作的url也放在service层中

token的作用：

防止重复提交，对token设置一个失效时间，防止重复提交

用户是否有相应的权限

token超时时间：如果token无超时时间，别人只要获取到token，就可以在未登录的情况下及可以进行相应的操作

签名的作用：

确保当前数据的来源：在数据来源里添加一个数字签名，确定信息来源

确保信息未修改：在传输过程中应对重要数据进行签名，虽然可以加密，可以确保数据不会以明文

传输，但是确保不了数据是否被修改，重要数据+秘钥=签名，接收端接收到数据后，通过接收

到的数据+秘钥(发送端和接收端都一样)生成签名，和发送端判断是否一致

但是这种方式最大的弊端是接收方必须要知道秘钥，相对来说不安全

非对称加密：公钥/私钥 发送方通过公钥对数据进行加密，接收方通过私钥解密，公钥是公开的

任何都可以获取，一个公钥对应一个私钥，最常用的非对称加密算法是RSA，秘钥的长度约长，越安全

数据摘要：对数据进行加密后得到的一个摘要，也加数据指纹

特点：

无论数据有多长，计算出的信息摘要的长度总是固定的

不可逆，不能通过数据摘要计算出数据

只要不是特别重要的信息都没必要加密传输，如果对整段数据进行加密，太耗时(此时的加密是对称加密，即可通过密文解密为原文)，但是应该确保数据在传输过程中没有被修改过，可以对数据进行MD5加密，获取数据摘要，在对摘要进行加密

公钥/私钥：如果信息是别人发给我，切只能我才能看到，此时应该使用公钥加密，私钥解密

如果我想向大家发信息，但是只能部分人才能看到，此时私钥加密，公钥解密

使用私钥加密最常用的是数字签名：公钥加密，私钥解密；私钥进行数字签名，公钥解密验证

数字签名：确保信息的来源，数据未被修改过

来源：公私钥对应，私钥解锁，只有对应的公钥才能解锁，确定数据的来源

修改：公钥解锁后，可以验证数据是否被修改过

数字证书：如果A拥有B的公钥，但是A拥有B的公钥被换成C的公钥，此时C用自己的私钥给A发送消息，A就能接收到C的信息(因为公钥已经被换掉)，后来A和B找证书中心(CA)，为公钥作认证。CA使用私钥为公钥和一些进行加密，生成数字证书，以后B在给A写信的时候，在签名的同时，附数字证书，A收到信后，用CA的公钥解开数字证书，再获取B的公钥原来是公钥在自己手上，可能被串改，现在是通过签名证书获取对应的公钥数字证书权威性高，拿到公钥时会告知私钥的主人是谁非对称加密算法慢如果某些信息不能在网上明文传输，采用对称加密涉及到秘钥会被获取的情况，但是如果采用非对称加密耗时，性能低可以在进行信息传输的过程中，首先建立连接，通过非对称加密的方法将秘钥发送给接收方，然后信息采用对称加密进行传输，这样相对来说快一点

https://www.cnblogs.com/franson-2016/p/5530671.html

MD5 SHA : 获取信息摘要

AES 对称加密/RSA 非对称加密

AES加密模式:

ECB(点码本模式)：先将明文分为若干块，再对每小段进行加密

CBC(分组链接模式)：先将明文分成若干块，然后每一小段与初始块或者上一段的密文段进行异或运算后，再与密钥进行加密。

MD5 HSA1 sha-256

MySQL清空表中记录；

delete from table\_name;

truncate table table\_name;

truncate是整体删除，delete是逐条删除，truncate的速度比delete快

truncate不写log，delete写log，可恢复

如果是truncate删除，新加记录后，主键会从1开始(主键自增)‘

如果是delete删除，新加记录后，主键依旧接着原来被删除记录+1

git查看分支的不同

git diff branch1 branch2 --stat 显示差异文件夹

git diff branch1 branch2 文件路径 显示指定文件的差异

git diff branch1 branch2 显示所有文件差异

git diff 本地和远程分支比较时，先git checkout到远程分支(此时才会下载此分支)

然后再会比较

有些通信只能通过服务之间访问，在浏览器中直接发起请求是不会响应的

不同通过ip访问，但是能通过域名访问

通过ip不能直接方法就是本机不是服务器，而通过域名能访问

Nginx可以通过域名将请求分发给不同的服务

ip为127.0.0.12下A，B，C三个服务，请求域名为a时访问A服务，b访问B服务

Nginx可以拿到请求的域名来分发请求

如果直接输入IP就可能找不到服务

center层：只做简单的CRUD

分页查询(支持模糊)，AO层自己使用，一般不对外，如有导出操作，可使用in语句

单个详情(必须为主键，或者唯一字段)，精确查询，对内，对外，确保查询结果唯一

添加，

删除(必须为主键，或者唯一字段)，where 条件为主键或唯一字段

更新，where 条件为主键或唯一字段

ao层：一般就是对center层再次封装，所有的业务逻辑应该放ao层，webservice只做简单的数据转换

层级之间不能同层调用

center层与center层

center只做简单的CRUD操作，不涉及业务逻辑，如果业务逻辑改变

将会修改center层

ao层与ao层

虽然ao层最调用的都是center，但是如果ao1的方法调用了ao2的方法，而不是之间调用

ao2对应的center层方法，如果ao2的方法修改会导致ao1的方法修改，而center层一般不会

做什么修改

一般对外提供的基础数据，应该有一个status状态，标志是否可用，如物流公司，提供商等

在查询时，对delete，status等状态值的判断，应该放到ao层，不应该在center中限制查询

条件

参数RQ

尽量为每一个webservice中方法对应唯一一个RQ，方便程序阅读，以及接口之间的联调

在死循环时，采用for(;;)，而不采用while(true)

编译后for比while的指令更少，占用寄存器内存更少

打日志：调用第三方接口，容易出错的地方

//dubbo实现异步RPC，在标签中有很多属性值可以设置

@Reference(version = "0.0.0",async = true)

java开发Linux常用命令

查询的时候没必要将对应的所有信息都取出来：

有些信息没必要

有些信息拉出来后，任然要在AO中处理(通过id获取详情)

写操可以在center中一起操作(保证事务)

读操作在AO中读

在center中写操作时的where条件是id(主键)和业务逻辑键(唯一)

但是在AO层中，使用主键作为where的查询条件

本地分支很多，远程分支以删除，本地分支还存在：

git remote show origin ：先查看远程分支还存在哪些

$ git remote show origin

\* remote origin

Fetch URL: https://gitlab.xgimi.com/sale/sale-aftersale-center.git

Push URL: https://gitlab.xgimi.com/sale/sale-aftersale-center.git

HEAD branch: master

/\* 远程分支：tracked表示还有的，stale旧的，远程分支以不存在 \*/

Remote branches:

develop tracked

master tracked

refs/remotes/origin/feature\_20180908 stale (use 'git remote prune' to remove)

refs/remotes/origin/feature\_20181106\_ld stale (use 'git remote prune' to remove)

refs/remotes/origin/feature\_20181205\_aftersale\_goods stale (use 'git remote prune' to remove)

refs/remotes/origin/feature\_function\_20181210 stale (use 'git remote prune' to remove)

refs/remotes/origin/feature\_install\_20181101 stale (use 'git remote prune' to remove)

Local branches configured for 'git pull':

develop merges with remote develop

feature\_20181205\_aftersale\_goods merges with remote feature\_20181205\_aftersale\_goods

feature\_install\_20181101 merges with remote feature\_install\_20181101

master merges with remote master

Local refs configured for 'git push':

develop pushes to develop (local out of date)

master pushes to master (up to date)

git remote prune origin ：删除本地的origin下的分支(origin远程分支)

git remote show origin

git remote prune origin

remotes/origin/表示远程分支下原来以及现在存在的分支(git pull时会将远程分支保存到本地，origin表示

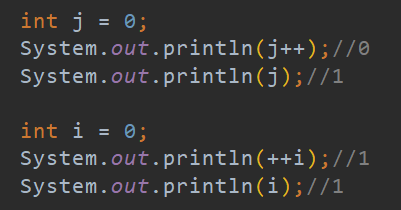
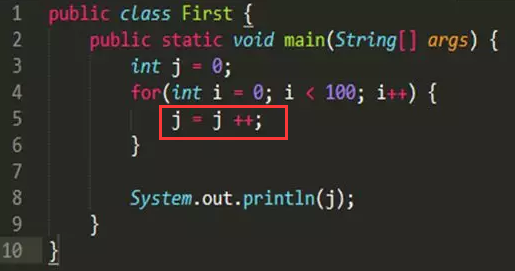
本地只存在分支，但是分支中无具体的内容)

git remote show origin ： 查看远程分支和本地的origin的特点

git remote prune origin ：删除本地的origin下远程无的分支

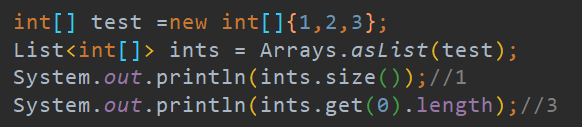
git remote \* origin：就是查看本地和远程的区别

linux下载命令：wget -c https://nginx.org/download/nginx-1.10.1.tar.gz



输出结果为0，注意，这里不是j++；而是j=j++；j++(先将j的值存入到堆中，然后在对变量区的j+1)；

++j(先对变量区的j+1，再将变量区的j存入堆)，输出和读取都是读取堆中的值，语句执行完后，再将变量区中的值存入堆中



通过Arrays.asList转换出来的list是Arrays的内部类(ArrayList)，而ints里存放的是test的引用，而并非是1,2,3，Arrays.asList的作用是将参数封装为List，所以ints的大小为1，传入的参数只有一个

Arrays.asList(T... a);

取得从 1970 年 1 月 1 日 0 时 0 分 0 秒到现在的毫秒数：

System.currentTimeMillis()

new Date().getTime()

获取java类字节码的方法：

Class.class

object.getClass()

Class.forName()

isSuccess不推荐：RPC在解析的时候会将isSuccess解析为success

Boolean success / Boolean success生成get/set的区别



阿里规范规定：POJO的属性使用包装类，局部变量基本数据类型

在分布式系统中，调用服务时，出现请求超时，会重新请求，这样会导致多次请求，不能保证幂等性

幂等性：指一次和多次请求某一个资源应该具有同样的副作用。幂等就是防止数据重复提交

Get方法是幂等的，通过url执行get得到的结果一致

Delete/put是幂等的

Post非幂等，post是写操作

实现幂等性：

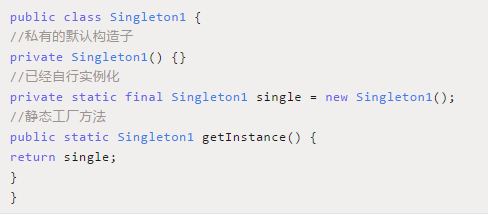
全局唯一ID：请求时，传入ID，接收方校验ID是否存在，判断是否以收到请求(推荐使用)

去重表：写操作时，会有一个唯一键，在数据库中对唯一键创建唯一索引，如果写操作重复执行，会抛出异常，操作回滚



单例模式：

饿汉模式：在类初始化的时候就创建



懒汉模式：在使用的时候才会创建(加锁避免重复创建)

安全失败(fail-safe)：你在迭代的时候会去底层集合做一个拷贝，所以你在修改上层集合的时候是不会受影响的，不会抛出ConcurrentModification异常。在java.util.concurrent包下的全是安全失败的。在java.util包下的都是快速失败(fail-fast)。

Executor框架：线程池

每次执行任务创建线程 new Thread()比较消耗性能，创建一个线程是比较耗时、耗资源的。

调用 new Thread()创建的线程缺乏管理，疯狂创建线程导致OOM

Executor和Executors的区别：Executor是线程池接口，Executors是创建线程池的工具类

如何实现多线程之间的通讯和协作：中断和共享变量

线程如果出现了运行时异常会怎么样：如果这个异常没有被捕获的话，这个线程就停止执行了。并且如果这个线程持有某个某个对象的监视器，那么这个对象监视器会被立即释放。