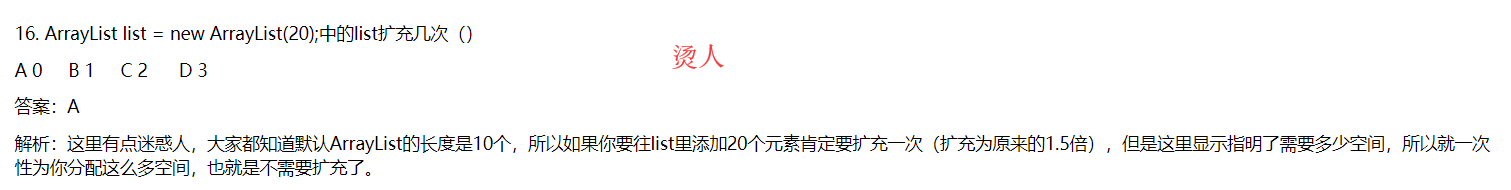
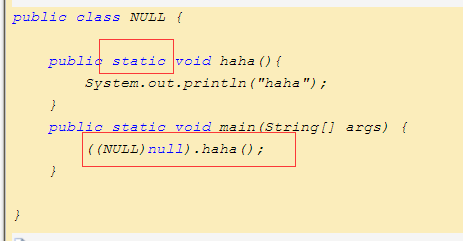
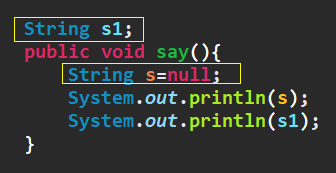


性能比较：new最快，其次，反射，clone，序列化





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，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，声明周期短，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。

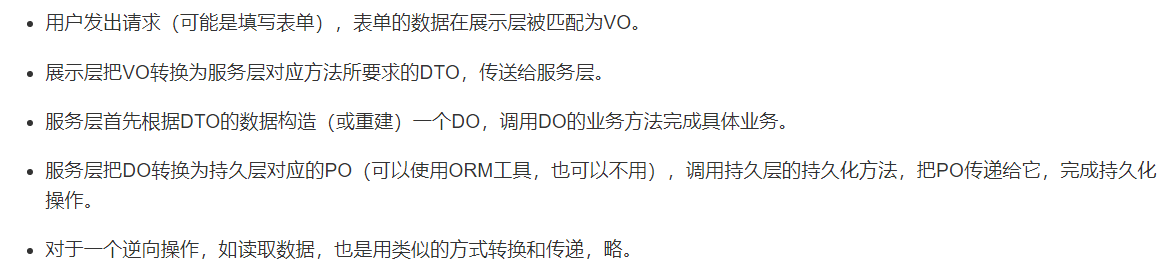
栈中存放的数据只有在执行方法的时候才会开辟，方法执行完被释放

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

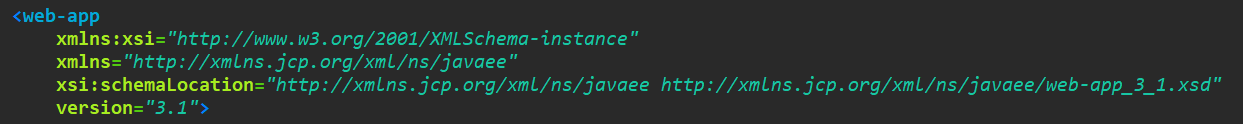
ThreadLocalMap以ThreadLocal为键，属性为值

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

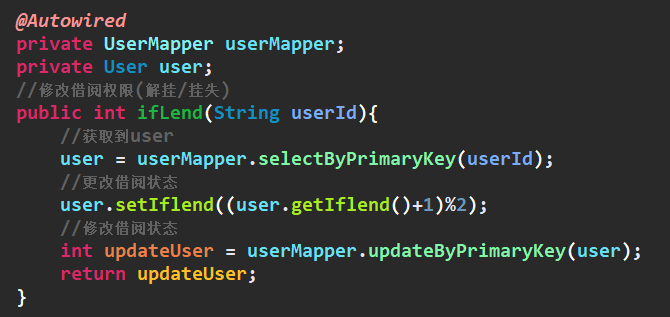
在xml文件中就的引入对应的约束文件，这里以schemaLocation约束为例：

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

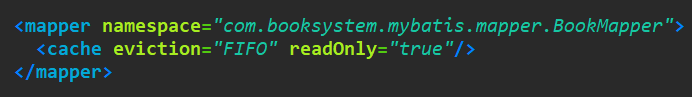
因此，在SSM中每执行一次Mapper的方法，都会执行session的创建和关闭操作

<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)

