# 千万不要乱该代码

ccc2021.9.3

Stack stores variables

New objects are in heap ram

Ram hardware use pointer to jump to the index object array number.

Visualgo.net

冒泡排序

内层循环每次控制比较前两个元素中大的，并往后移动比较，大的浮现到最后 并限制跳出为

那个找到最大的

外层循环每次控制从下一个元素开始浮现最大的到最后

时间复杂度O(n2)理想O（n）

选择排序 内层循环每次拿第个元素和后面所有的比较如果小于，就交换

外层再拿下一个元素和后面所有的比较

2020.9.4

Int arr[][] = new arrary [3][];

Change two varibles

a = a+b;

b = a-b;

a = a-b;

计算机中只有加法

2021.9.6

原码

最高位为一 表示负数 负数有补码 先取反再加一

还原减一再取反 或者再取反加一一次 变回原码 符号位不参与 1000 0000 表示负128 但是加减的时候符号位会参与

如果 case 语句块中没有 break 语句时，匹配成功后，从当前 case 开始，后续所有 case 的值都会输出。

**Test.java 文件代码：**

public class Test { public static void main(String args[]){ int i = 1; switch(i){ case 0: System.out.println("0"); case 1: System.out.println("1"); case 2: System.out.println("2"); default: System.out.println("default"); } } }

以上代码编译运行结果如下：

1

2

default

for(;;) 无限循环

1/10.0 transfer to a double digit

每行输出三个 定义一个个数的标记 满三 输出换行 然后重置为零 用if 语句

刷题 刷的是思考的方式和思路

Thread start 3 methods：Thread Runable lamda

Synchronize

其实是指令系统分成堆栈型和寄存器型。不光这两种，指令系统共有四种分类，堆栈型，累加器型，寄存器-存储器型和寄存器-寄存器型。分类的依据是操作数的来源。堆栈型默认的操作数都在栈顶，累加器型默认一个操作数是累加器，寄存器-存储器型的操作数可以是寄存器或者内存，寄存器-寄存器型除了访存指令，操作数都是寄存器。

Jvm 堆栈

安卓 davli vm 寄存器型

2021.9.7

Break只能跳出内层循环 无法跳出外层循环

Print（“（”+i+“，”+j+“）”）（1，2）

Continue 跳出本次循环

Return 返回方法的返回值

终止当前方法：

Local variable 局部变量的作用域只在方法内部，不包含默认值，如果没有使用，只定义是可以的，如果用，要给初始值。

Global variable 全局变量的作用域在整个类体内

成员变量初始值 int 0， Boolean false ，string null

String s=null;//定义了一个对象实例s，但未将该实例指向任何内存空间

String s="";//定义了一个对象实例s，指向了一个空串



一个类中可以定义多个同名方法；

Construction method overload

But the args of number,type,oreder must be different

In one class

Two methods invoke each other will create StackOverFlow mistake ，cuz it’s a dead loop. 方法是压栈执行的

Static modify args means it’s static

You don’t have to create an object to get it.

Creating an object is a kind of movement.

Static initiate before the Object was created.

Static variable is shared by all objects,kind of like an public variable,but it’s recommended to invoke by class.

Global variables are put in heap,while static variables are put in static area in method area.

Static variable can’t be defined in the method.

Usually，the method in tool class is defined as static

2021.9.8

CodeBlocksDemo:

定义在类中 使用括号括起来的代码块叫做构造代码块

每次代码块运行 会将构造代码快中的代码添加到构造方法的前面

This（a）调用本类中a的构造方法

构造代码块会添加到每个构造方法中 使用this（）的构造方法不会添加

执行顺序 静态block》构造block》（ 创建对象的时候才会用到）usual block》

Package:

解决包重名的问题

For better management for class

对应到文件系统就是多级目录

统一功能的代码放到统一目录下

一般定义package会放置在java文件的第一行

Package域名的倒写

Package com.mashibing

完全限定名 包名加类名 44min

使用不同包的同个类名，一个导入，另一个使用完全限定名

频繁使用某个包的某个类，可以使用静态导包

Import static java.lang.Math\*

Encapsulation

Class modify mark only two kinds of：public default

字符串比较一般使用equals

**Java中的字符串不可以修改 字符串存在于常量池中**

**+ 建议使用stringbuilder**



**Stringbuffer 可以修改**

2021.9.10

插入排序：默认左手拿扑克牌，外层循环从1到最后，控制每次从

右边拿一个比较，需要一个temp来记录这个拿过来的需要插入的

值，左手fori=1，i《length，I++

Temp， 内层从j=i-1，开始》=0 递减，依次和temp比较，如果

大于，则和temp交换位置。

for (int i = 1; i < arr.length; i++) {

int temp = arr[i];//outside loop ,control the step moving forward;

for (int j = i-1; j >= 0; j--) { // inside loop ,control the comparison step moved;

if (temp < arr[j]) { //if the comparison success, insert the args into the right position.

arr[j + 1] = arr[j];

//temp = arr[j];

if (j == 0) {

arr[0] = temp;

}

}

else{

arr[j+1] = temp;

break;

}

}

}

Equals方法 默认比较哈希值

将重复代码提取到父类当中

2021.9.11

java 中是单继承关系 如果子类调用的方法和多个父类中的方法重名，无法

确定调用哪个方法

Super 是直接父类对象的引用

This（） 和super调用父类构造方法的话，一定要放在第一行

构造方法中，This（）和super不能同时使用

父类中定义为私有的方法和属性在类外都不能被调用

创建对象时，该类的父类的属性和方法也会随该对象

创建到（堆内存） 中，因此子类的构造方法中默认会有super

这个指针

2021．9.12

子类默认的构造方法中都会默认使用super（），父类一定要有无参构造方法

定义类的时候 ，最好将无参构造方法写上

创建子类对象的时候一定会优先创建父类对象

所有java 对象都具备同一个老祖宗类，称之为Object

Java中的多继承 可以通过接口来实现

方法重写

Override toString()方法，返回对象的属性值而不是地址

toString() 类 只能被重写

重写的方法权限不能比父类方法中的更严格

Private 方法没有办法被继承

构造方法不能被继承 因此不能被重写

当子类中存在对父类的继承时，若所调用的子类方法中无Super（），则默认调用父类中的无参构造方法。

静态方法可以继承 但不能重写

Super this 都不能出现在静态方法中

抽象类 限制实例化

抽象类中的某些方法需要子类更丰富地实现，可以定义为抽象方法，

可以不写方法的实现，即没有大括号

抽象方法必须在子类中实现,除非子类也是抽象类

Final variable can’t be changed. Final method can’t be override.

Final class can’t be extended.

被final 修饰的叫做常量

Constant 称之为常量

定义一个类 中包含大约十个以上的object中的方法

== 比较两个内存地址是否一致

Equals 比较两个内容是否一致

自定义类必须重写equlas 方法 默认情况下,它的实现是比较对象内存地址,因此它的工作方式与==运算符相同。

抽象类可以写构造方法

多态

2021.9.13：

Native 开头的方法 是内核 封装中的 调用操作系统

通过反射方式访问类的私有属性？

Hashcode 需要无序放置 hashcode() 方法 返回一个对象的int值 哈希算法

算出来的int值

如果两个对象equal 哈希值必相同 否则哈希值无要求

2021.9.14

【XX收快】您的XX快递已到（原菜鸟驿站）65号店取货吗A10-8-9856，请9点至21点取货，电话xxx-xxxxxxxx）

大家重点看看这个取货码：A10-8-9856（哈希键值）。这可不是一堆乱码。这串数值，就是驿站员能快速找到你快递的关键。

A10指的是货架编码，8指的是第8层，9856指的是订单后四位。因此每个驿站小哥只需要瞄一眼，就能瞬间知道你的快递所在位了。

因此，哈希算法最核心的用处，就是高速存取。在区块链技术中，它才真正大展身手。以下是它在区块中的具体作用：

2.hashCode()和equals() 的关系

和equal()方法作用类似，hashCode()在java中的也是用于比较两个对象是否相等。我们应该都大概听过一句话：重写equals()方法的话一定要重写hashCode()。但是从来也不知道是为啥，这里就说明一下这点。

分两种情况：①首先一种情况是确定了不会创建散列表的类，我们不会创建这个类的HashSet、HashMap集合之类的。这种情况下，这个类的hashCode()和equals() 完全没有任何关系，当我们比较这个类的两个对象时，直接用的就是equals()，完全用不上hashCode()。自然，也没啥必要重写。②另一种情况自然就是可能会需要使用到散列表的类了，这里hashCode()和equals()就比较有关系了：在散列表的使用中，经常需要大量且快速的对比，例如你每次插入新的键值对，它都必须和前面所有的键值对进行比较，确保你没有插入重复的键。这样如果每个都用equals，可想而知，性能是十分可怕的。如果你的equals再复杂一些，那就凉凉了。这时候就需要hashCode()了，如果利用hashCode()进行对比，只要生成一个hash值比较数字是否相同就可以了，能显著的提高效率。但是虽然如此，原本的equals()方法还是需要的，hashCode()虽然效率高，可靠性却不足，有时候不同的键值对生成的hashcode也是一样的，这就是哈希冲突。

在使用时，hashCode()将与equals结合使用。虽然hashcode可能会让不同的键产生相同的hashcode，但是它能确保相同的对象返回的hashcode一定是相同的（除非你重写没写好），我们只需要利用后面这点，一样可以提高效率。

在散列表中进行对比时，先比较hashCode()，如果它不相等，那说明两个对象肯定不可能相同，就可以直接进行下一个比较了。但如果hashCode()相同，因为哈希冲突的缘故我们无法直接判断两个对象是相同的，就必须继续比较equals()来获取可靠的结果。

注（怕大家看不懂因果逻辑，简单说明下）：

因为相同对象一定是相同hashcode，所以相同对象一定不会有不同的hashcode，所以两个对象如果是不同的hashcode，那么这两个对象一定是不同的。

所以如果这个类可能会创建散列表的情况下，重写了equals方法，就必须重写配套的hashcode方法，他们两个在散列表中是搭配使用的。

————————————————

版权声明：本文为CSDN博主「Hal白夜」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/weixin_38958597/article/details/112759067>

哈希提高存储效率 依据key 找value 但是不同的value可能有相同的key 需要equal来补充对比，提高准确性，hashcode（）默认输入对象地址 计算出key toString hashcode 权是31

另一种情况自然就是可能会需要使用到散列表的类了，这里hashCode()和equals()就比较有关系了：

   // it checks if the argument is of the

        // type Geek by comparing the classes

        // of the passed argument and this object.

        // if(!(obj instanceof Geek)) return false; ---> avoid.

        if(obj == null || obj.getClass()!= this.getClass())

            return false;

hashmap 由数组和链表组成

hash()map 先比较hashcode 再比较其中的值

多态1.方法调用的不同实参是形参（是一种抽象类类）的子类，可以自动识别。

2．形参（父类）向下兼容不同实参（子类）

最后实现的是子类的重写的方法 .

多态规范（也是前提）

* 有继承
* 父类引用指向子类对象(反之不行)等号右边归属于等号左边
* 子类方法重写父类方法

使用父类作为形参实现多态

使用父类作为方法的返回值实现多态

Public void pet(int type){

If type==1{

Return new dog();

If type == 2

Return new cat();

为了维护性和扩展性

方便代码逻辑的编写

引用类型的转换跟基本类型的转换类似，父类没有办法转换成子类

Pet isntanceOf Dog

算法正确性

循环第一步是正确的

每一步正确

最后返回的结果是正确的

2021.9.21

自定义类需要重写equals 方法，否则equals 默认使用的是==方法比较，返回总是false

面向接口编程

内部类可以访问外部类的私有属性

外部类不能访问内部类的私有属性,但是创建内部类的对象后可以访问

内部类不能定义静态属性

当外部类和内部类具有相同名字的属性或方法，内部类需要调用

外部属性或方法 需要使用 Outer.this.outerMethod or attribute

2021.9.24

JDK 11.0.7 第七次更新

CS BS

BS客户端与服务器实现了解耦

只需要服务器更新即可

Php 适合构建小项目，

打印三角形 平行四边形 对称 空心 正方形状 长方形的 精髓在于

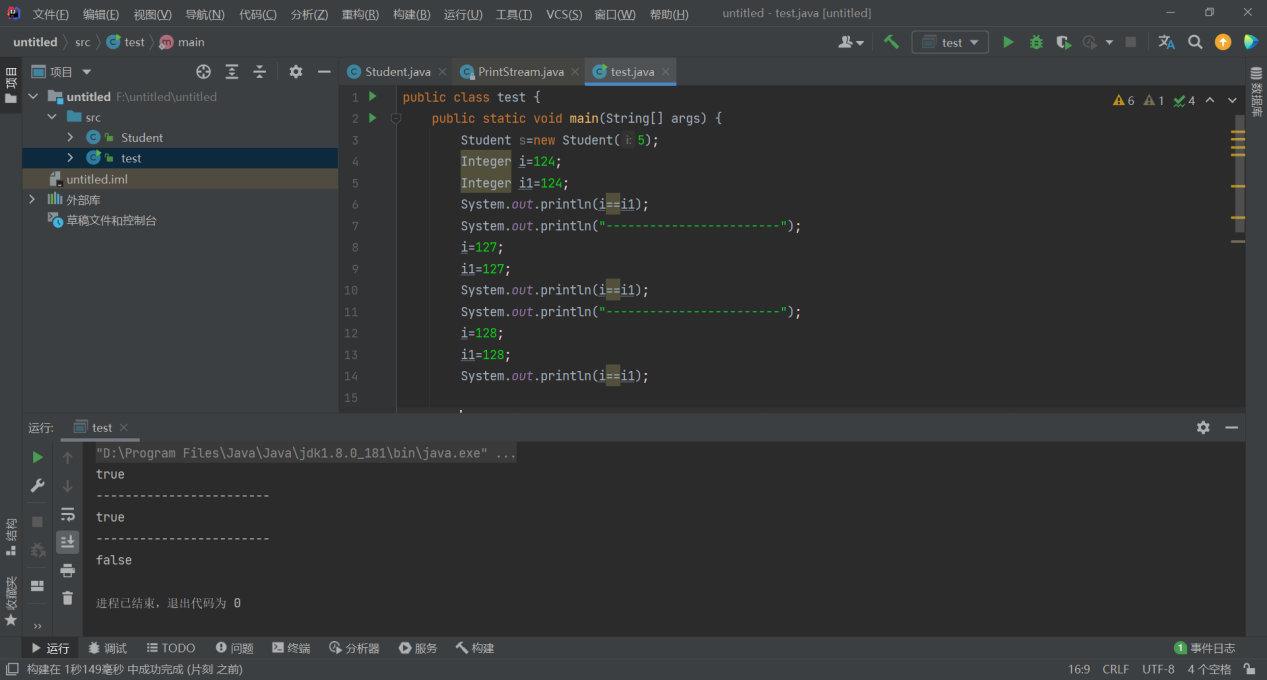
如何打印空格 for 循环中并列两个for 循环 一个打空格，另一个打 星星

关键在于递增还是递减 空心用或判断是哪一行的i 或哪一列的j

2021.9.28

Integer 反转 取出 压入 rev=rev\*10 +digit

注意最大rev不能超过max/10



Java integer 常量池到127

2021.10.10循环边界的判定不用从程序的最头开始考虑 从边界点考虑

2021.10.13 多维数组的 下标是层层引用 最后一维是真正的数据

Why apple find acron to make cpu？At the beginning

数组在内存中是一块连续的空间

2021.10.14

选择排序 涂抹掉之后要填上 本质上还是交换

数组默认值为0

创建方法快捷键 alt+enter itar for 循环快捷代码

数组末端加入新数据，考虑扩容（新建数组将原来的数据复制

到新数组中和限容量）设置size 的值

10.16.2021

Alt insert create getter and setter method.

在 Java 中，无论是 explicit 还是 implicit 方式，都要求在子类的构造方法中调用其父类的构造方法。如果父类无构造方法（其实是一个默认无参的构造方法），那么子类的构造方法中会自动进行调用；如果 父类有自己的构造方法（这时父类不会有默认无参的构造方法），那么在子类的构造方法中，必须要调用父类的某个构造方法，而且必须是在构造方法的第一个语句 中进行调用

因此 默认一定要把无参构造方法写上。

学生管理 系统 StudentOs 是一个类

If else if 语句是互斥的

Jvm编译器都是在操作系统之上的

2021.10。19

Link 引入css 文件

Meta标签 description Meta key word 会在百度排名 seo 优化

 ul 是 Unordered List

Td table data

Br blank row 换行

Hr horizontal rule

Ol order list

Form表单提交的数据都是给自己的服务器

Input type password encrypt text unencrypt

Checkbox 复选框 勾选

Label 样式标签

Input标签必须要有name 属性

Style 定义html 属性

2021.10.21

Css选择器有三种使用方式

Style

Display：block 块元素

Display：inline行内元素 宽高无意义

Div division 分格块儿

Padding 内间距 顺时针

Margin 外间距

Position：static absolute

Position:relative子元素的定位参照父元素

Static 参照浏览器定位

Fixed 跟随滚动条的定位

相对浏览器的固定定位

把图片都放在一张图片上可以提高加载速度

每张图片都是http 请求 四次握手

Javascript 1992年右。。公司

<script type = “text/javascript”>

</script>

2021.10.24

Html

ID容易区分，每个ID的样式只可以用一次，你可以知道它是用在什么地方，

classname可以用在任何地方，套在任何一个DIv中都可以，缺点是后期维护代码很难找

Javascript 在script 标签里面

Css 在style标签里面

<>指的是元素

Css文件要引入<link rel = “stylesheet”herf = “/Mycss.css”

Document

2021.11.7

B+tree B-tree

查找算法都是在有序的数据结构的基础上进行区间确定（即已经划分好了的区间）

主键和唯一性的列作为外键

外键约束删除主表 外键表对应的元组 如何 选择 on delete cascade

三、设置级联操作

在主表数据发生改变时，关联的从表数据应该如何处理

主表更新：使用关键字on update标识

主表删除：使用关键字on delete标识

允许的级联操作：（指的是当主表更新或者删除时，从表相应的字段有哪些操作）

方式一：cascade关联操作，如果主表被更新或者删除，那么从表也会执行相应的操作

方式二，从表设置set null ，表示从表不指向任何主表记录（推荐操作，因为删除数据要慎重！！）

alter table student add foreign key (class\_id) references class (class\_id) on delete set null;

要想设置null，此外键就得允许为空，若此外键不允许为空，就不能这样做

方式三：拒绝主表的相关操作：restrict

三种方式中，restrict是默认的操作，在二标题中就有体现

修改外键：

先删除，再创建，通过修改表完成

alter table student drop foreign key 主表id;//主表id通过show create table tb\_name中的constraint字段

删除外键需要通过指定外键名称达到目的（注意只是删除了外键这个约束，而不是删除此字段，所以从表中的数据不会有丢失）

索引 加快查询速度 影响增删改的速度

一对多 在多方设置外键关联 即一方的主键

多对多 需要再创建一个表

2021.11.9 链接 jdbc

导入mysql jar 包

创建对象 链接

查询建立一个ResultSet 接受 while .next 定义变量接受 并打印

pdm don't need to create fk, it will help you create it

因此，我们看String类的concat方法。实现该方法第一步要做的肯定是扩大成员变量value的容量，扩容的方法重新定义一个大容量的字符数组buf。第二步就是把原来value中的字符copy到buf中来，再把需要concat的字符串值也copy到buf中来，这样子，buf中就包含了concat之后的字符串值。下面就是问题的关键了，如果value不是final的，直接让value指向buf，然后返回this，则大功告成，没有必要返回一个新的String对象。但是。。。可惜。。。由于value是final型的，所以无法指向新定义的大容量数组buf，那怎么办呢？“return new String(0, count + otherLen, buf);”，这是String类concat实现方法的最后一条语句，重新new一个String对象返回。这下真相大白了吧！

// connect mysql

Class.*forName*(*driver*);

Connection connection = DriverManager.*getConnection*(*url*, *user*, *password*);

Statement statement = connection.createStatement();

String sql = "select username from user where username = '"+input+"'";

//use result set to receive the query result

ResultSet resultSet = statement.executeQuery(sql);

//judge if there is the input name

if(resultSet.next()){

System.*out*.println("user has been used");

}

else{

System.*out*.println("username can be used");

}

resultSet.close();

statement.close();

connection.close();

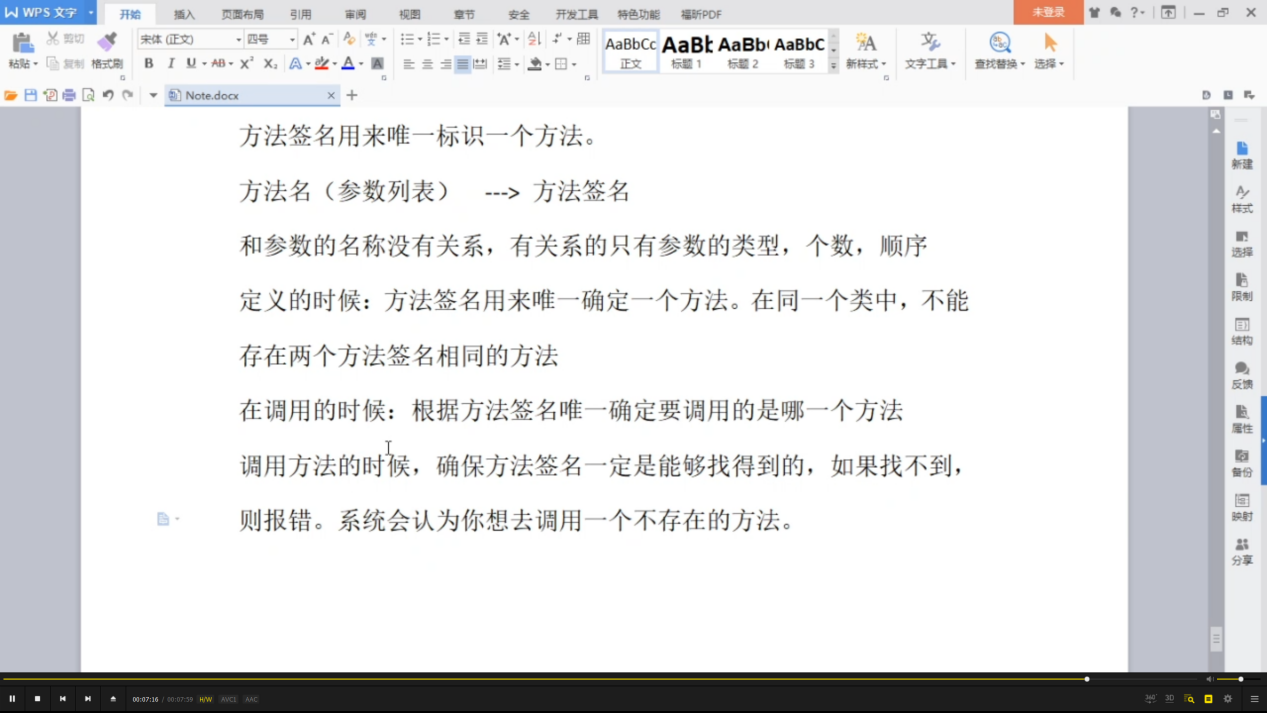
byte、short、char在运算时，会自动向上转型为int型，所以返回的结果是int类型，而接收的类型均不是int，由于

类型不匹配，所以报错

2021.11.16

For 中的continue 跳转到迭代因子上

Return end the method and doesn’t excute the following statementsxxxxxx

方法 值 传递 数据交换

重载有注意滥用问题 多参数 不同类型 多个匹配 又不匹配 会造成重载滥用

调用函数时一定注意传参的 方法 形参类型

2021.11.17

设定器 方法 访问器方法 Boolean type like

isGender

Static

Virables don’t need to

Create object to call on

增强for 循环不能更改数组元素

For（int ages:age){

Sout}

11.18.2021

类中有abstract方法 则类也必须是abstract类

2018.11.19

内部类： 外部通过创建内部类对象才能访问内部类

内部类可以访问外部类的私有属性 相当于继承外部类 （多继承）

11.20.2021

静态内部类不能使用外部类非静态属性方法

匿名内部类 靠接口 实现对象的创建 直接付给一个引用 类的名字会是$1 $2 $3...

局部内部类像局部变量一样 除了final 不能有权限修饰符

局部内部类的作用域只能在方法体当中 相当于隐式继承外部类 还可以继承其他的类

Java 贪吃蛇

node 节点 蛇一个个节点组成

每次重绘 最后一个节点移到第一个 body 节点数组

键盘控制 重绘的 节点 坐标 上下1 3

左右0 2 if（direction%2 ！= dir%2){

Change direction

1. 绘制窗口
2. 绘制蛇 while（isRunning） isEat isOver
3. StringBuffer thread safety concerned slow
4. StringBuilder no safety concerned fast
5. SimpleDateFormat parse format method
6. Date 存储从1970年开始的毫秒数
7. LocalDate
8. 11.24.2021
9. Arraylist 添加一个元素 容量为10 每次扩容容量为1.5倍
10. Vector 类似 arraylist ， vector 线程安全
11. Vector 每次扩容原容量的2倍
12. **手写 arrayList 手写hashmap**
13. **注意检索LinkedList 会返回null 或者和抛出异常**
14. **递归删除文件夹**

**This（） super() 只能放在构造方法的第一行**

1. **Java 泛型**
2. **Set 无序不可重复 通过equal 方法决定 ，以及只能有一个null 值**
3. **Treeset 二叉树中的红黑树 平衡二叉树**
4. **Treeset是treemap 的key**
5. Lambda comparator
6. 让自定义的引用类型 重写comparable方法 指定当前数据的比较规则
7. Treeset implement comparable override compareTo method inner 比较器
8. Treeset construct method can send a comparator to be as a outter 比较器
9. Map遍历的三种方式1.keyset()2.values()3.Set<Map.Entry<k,v>>
10. Hashmap默认位桶16 扩容因子0.75 大于0.75 则扩容为原来的2倍
11. 扩容之后散列在多个位桶的格子中，提高查找效率
12. Hashtable 线程安全的哈希表 与哈希map 非常像 不能存储null 作为key 和value
13. 默认11 加载 因子0.75 每次扩容2倍加一
14. join（） 加入另一个线程 先执行
15. Yield 礼让
16. Wait 等待 sleep 抱着线程资源睡觉
17. 死锁 两个线程 一起执行到 刚好共同霸占对方的资源 情况下 互相等待 形成死锁

在一个对象中的同步方法 调用另一个对象 的同步方法 （嵌套同步） 先吃饭再买单 先买单再吃饭

http 基于tcp面向连接 udp 面向

2021.12.3

工厂方法 用接口实现 抽象出来 同类方法的不同参数可以实现为一个接口 便于以后判断 流程语句 代码的维护 把需要创建的对象 集合到一个工厂类里面 避免创建者与具体的对象的操作耦合、满足单一职责，每一个业务逻辑实现都在所属自己的类中完成、满足开闭原则，无需更改使用调用方就可以在程序中引入新的产品类型

12.4 迭代器 .next() 是第一个元素哦 有一个只存第一个元素地址的head node

反射 jvm loader everyclass has a object in heap call class Class

类名.getClass() 对象.class() ClassForName(包名） 可以创建对象 获取方法 调用 方法

注意方法的形参 是类型的类对象

21.12.5

Xml 先构建文件，用Document再加元素 addElement

写入一个文件 用XMLWriter

XMl 移除元素 使用remove 但是要先获取父元素

正则表达式 （） 表示分组 group是括号 （）

Myabatis xml DTD 规范 规定 xml 文件只能这样写

12.6.2021

Java IO stream 字节流 inputStream read outputStream write

OutputStreamWriter (传入输出流,编码格式) 字节和字符的桥梁

InputStreamReader (传入输入流，编码格式）

Java设计模式

只扩展 不修改

代码块儿的执行步骤

**A的静态代码块**

**A的构造代码块**

**A的构造函数**

**多线程 实现runnable 接口 重写run 方法 放进线程中.start()**

**Sychronized() sychronize 表示同步 即同一时间只能有一个线程对sychorized中的代码进行执行，**

**原因是给代码块儿上锁**

**Sychronize修饰方法**

**Cas 轻量级锁 自旋锁 无锁**

**Aba采用版本号加一**

**Cas 本身必须要具备原子性 lock cpu的单核 changeandswap**

**偏向锁 经常使用的 vector stringbuffer 贴上标签**

Sychronize 的锁 this 指的是执行该代码块的类的对象实例作为锁。

同一个对象启动的不同线程互斥。

不同对象（锁是不同对象的实例）启动的不同线程并不互斥。

互斥执行指的是不同线程，同个锁的互斥。

进入sychronize 代码块儿 首先要访问锁中的monitor

分析多线程就是考虑多个thread同时执行这一块儿sychronized 的代码 ABC线程

锁 monitor object 存储在对象头里面

1. 当synchronized作用在实例方法时，监视器锁（monitor）便是对象实例（this）；
2. 当synchronized作用在静态方法时，监视器锁（monitor）便是对象的Class实例，因为Class数据存在于永久代，因此静态方法锁相当于该类的一个全局锁；
3. 当synchronized作用在某一个对象实例时，监视器锁（monitor）便是括号括起来的对象实例；
4. 如果monitor的进入数为0，则该线程进入monitor，然后将进入数设置为1，该线程即为monitor的所有者；
5. 如果线程已经占有该monitor，只是重新进入，则进入monitor的进入数加1；

如果其他线程已经占用了monitor，则该线程进入阻塞状态，直到monitor的进入数为0，再重新尝试获取monitor的所有权

Jvm 是一个specification 阿里 都可以实现

2021.12.7

Java 对象 layout markword classobj instance date padding

Org.openjdk.jol.info.ClassLayOut 测试一个对象有多少个字节 ClassLayout.parseInstance(t).toPrintable());

//list 转成array

String[] strArray = strList.toArray(new String[strList.size()]);

out目录配置项目编译之后的文件

Properties pro = new Properties();

pro.load(Thread.*currentThread*().getContextClassLoader().getResourceAsStream("db.prperties"));

Jdbc 使用properties

同一个事务（原子性，可回滚性，A,B转钱）放入同一个try catch 语句中

setAutoCommit（false）则关闭每个语句是一个事务的情况，表示开启组事务，语句仍会默认执行提交，但通过conn.commit

提交。

Conn.rollback() 回滚。

Try 中的变量提前声明 否则finally中的语句编译不识别

事务默认自动提交

Conn.commit()提交事务

Conn.rollback() 回滚事务

String str = ("insert into t\_user values('"+name+"',"+pwd+")");构建 sql 语句

其中单引号在外面 ，相当于sql 语句中的字符串标志，而双引号是Java中字符串的标志 加号理解为转义为变量实际值

先不管需要更新的是否是字符串，先将SQL语句写出来。

String sql="insert into user(username,password,email)values("+username+","+password","+email")";

然后我们找出其中的字符串，添加上单引 号，

PrepareStatement 预处理块儿 提前编译

一下午的时间败给了 commit；

检查你再创建Oracle数据表时、插入、删除、修改某些数据时有没有在最后要写上commit或者点击commit；

Oracle数据库在进行增删改操作时不会自动提交事务！！！

不使用预处理块儿会sql注入

2021.12.8

A Statement object represents a primitive statement in which a single method is applied to a target and a set of arguments - as in "a.setFoo(b)".

Statement may use reflection.

Java bean setter getter java bean

[太阳公司](https://www.zhihu.com/search?q=%E5%A4%AA%E9%98%B3%E5%85%AC%E5%8F%B8&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":31625054}" \t "https://www.zhihu.com/question/_blank)在设计Java语言时，也懂得这个道理。所以Java标准库中，绝对不会出现public int size这样的代码，而一定会一开始就写成：

**private** **int** size**;public** **int** **getSize()** **{** **return** size**;** **}**

int minNum = Arrays.stream(nums).min().getAsInt();

Stream return an intstream .min return a optional stream

xml实现不同系统之间的数据交互。

1.1.2. 作用

1. 配置应用程序(servlet 配置,框架环境配置等)和网站

2. 数据交互

3. Ajax基石

2.注解的作用

如果说注释是写给人看的，那么注解就是写给程序看的。它更像一个标签，贴在一个类、一个方法或者字段上。它的目的是为当前读取该注解的程序提供判断依据。

测试工程师可以根据注解来执行选定的代码

2021.12.11

一定要注意大小写

注意mybatis resource 中的目录是/

# interface





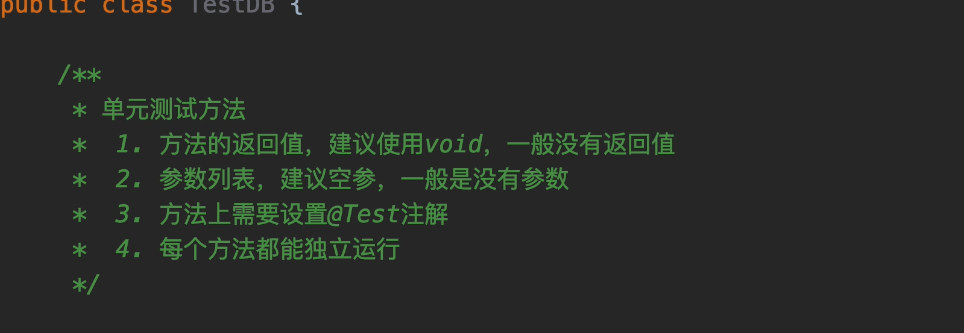
# 获取类对象的方法

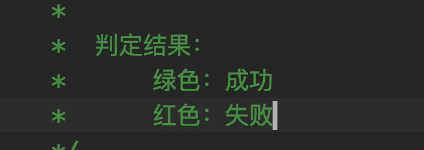
因此可以得出结论. Class.forName() 默认会对第一次加载的类初始化. 而 .class不会, 至于getClass() 你都能拿到对象了...可肯定已经初始化过了..

# 自定义异常

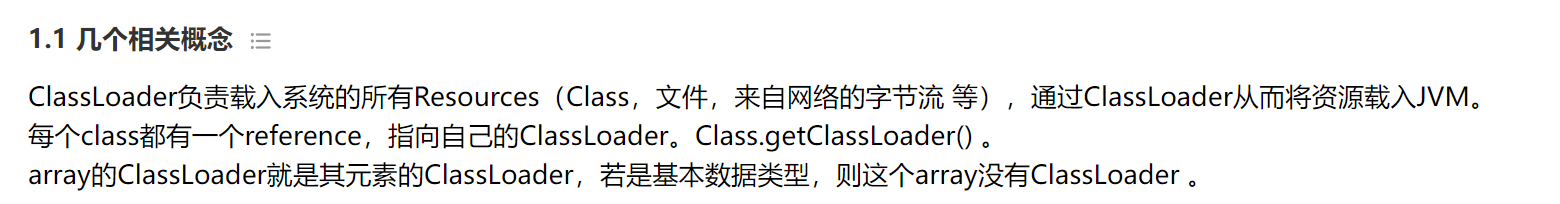


# 单元测试





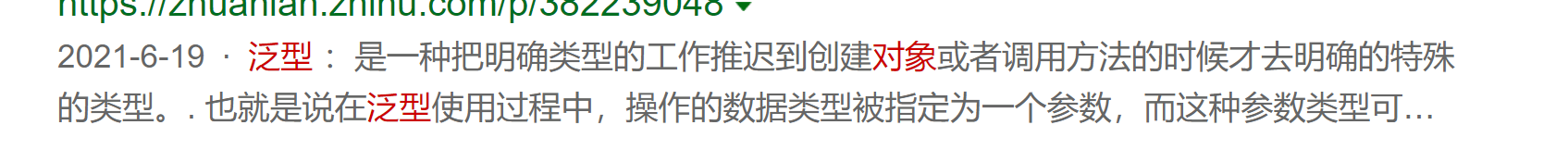
# classloader



区别：. （1）Class.forName除了将类的.class文件加载到jvm中之外，还会对类进行解释，执行类中的static块。. （2）而classloader只干一件事情，就是将.class文件加载到jvm中，不会执行static中的内容，只有在newInstance才会去执行static块。. #Class.forName (name,initialize,loader)带参数也可控制是否加载static块。. 并且只有调用了newInstance ()方法采用调用构造函数，创建类的对象。

# 泛型

所谓****泛****型，就是允许在定义类、接口时通过一个标识表示类中某个属性的类型或者是某个方法的返回值及参数类型。



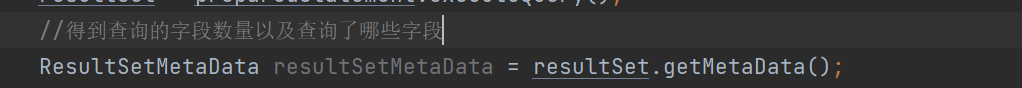


# Lombok

虚拟机内存 static 方法 都在方法去 code segment

# Jdbc结果集

Resultset 维护一个表 .next 返回一个Boolean 按行读取 getInt getObject 参数为列



# Substring

### **语法**

*stringObject*.substring(*start*,*stop*)

|  |  |
| --- | --- |
| **参数** | **描述** |
| *start* | 必需。一个非负的整数，规定要提取的子串的第一个字符在 stringObject 中的位置。 |
| *stop* | 可选。一个非负的整数，比要提取的子串的最后一个字符在 stringObject 中的位置多 1。  如果省略该参数，那么返回的子串会一直到字符串的结尾。 |

### **返回值**

一个新的字符串，该字符串值包含 *stringObject* 的一个子字符串，其内容是从 *start* 处到 *stop*-1 处的所有字符，其长度为 *stop* 减 *start*。

### **说明**

# 多态 ArrayList

## 父类是统领 更关注 父类 降低耦合

feed(Cat cat){ ... }

feed(Dog dog){ ... }

如果Cat、Dog都是Animal的子类，那你就可以写成

feed(Animal animal){ ... }

Animal dog1 = new Dog(); //父类引用指向子类对象

Animal cat1 = new Cat();

feed(dog1);

————————————————

版权声明：本文为CSDN博主「云灬」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/weixin\_45564920/article/details/100647620

有人说这是面向接口编程，可以降低程序的耦合性，即调用者不必关心调用的是哪个对象，只需要针对接口编程就可以了，被调用者对于调用者是完全透明的。

****让你更关注父类能做什么，而不去关心子类是具体怎么做的****，你可以随时替换一个子类，也就是随时替换一个具体实现，而不用修改其他。

## 增加方法参数的扩展性

比如最基本的一个方法 equals，他是Object类的一个方法，完整写法是 public boolean equals(Object obj)

注意这里的参数类型使用的是Object 而Object又是所有类的父类，所以你在调用这个方法的时候，这个参数可以传入Object的子类对象（即任意对象），所以这个方法适用于任何对象。

如果没有多态 这个方法就不能这么写，参数只能写一个具体的类，那么这个方法的适用范围就只是这一个类。。

————————————————

版权声明：本文为CSDN博主「云灬」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/weixin_45564920/article/details/100647620>

# List 多态

为什么一般都使用 List list = new ArrayList() ,而不用 ArrayList alist = new ArrayList()呢？

问题就在于List有多个实现类，如 LinkedList或者Vector等等，现在你用的是ArrayList，也许哪一天你需要换成其它的实现类呢？，这时你只要改变这一行就行了：List list = new LinkedList(); 其它使用了list地方的代码根本不需要改动。假设你开始用 ArrayList alist = new ArrayList(), 这下你有的改了，特别是如果你使用了 ArrayList特有的方法和属性。 ,如果没有特别需求的话,最好使用List list = new LinkedList(); ,便于程序代码的重构. 这就是面向接口编程的好处

注意事项

list只能使用ArrayList中已经实现了的List接口中的方法，ArrayList中那些自己的、没有在List接口定义的方法是不可以被访问到的

list.add()其实是List接口的方法

但是调用ArrayList的方法如 clone()方法是调用不到的

# 空方法体

什么都不做 不向使用父类方法中的内容

# Request.setAttribute 内容

将对象set 进去

取出来要用

El 表达式

resultInfo.result.userName}

第一个result 是setattribute 进去的键值 的值 是一个对象 第二个result 是这个对象的属性，是一个user 对象，第三个对象是userName 属性 应该是树的一个数据结构

# static 内存

Static 方法 方法区中 维护了这个方法的地址，调用的时候再调到内存

Static 方法最快

# Acid 原则

原子性： 要么全部成功，要么全部失败

一致性：转账， 只存在转账前后的状态,不存在中间态， 指令一步一步执行，中间不能读到数据。

隔离性：事务之间隔离，避免dirty read photom read unrepeatable reead

四个隔离级别： 不可重复读，seriliazable， 可重复读，

Interger 值传递



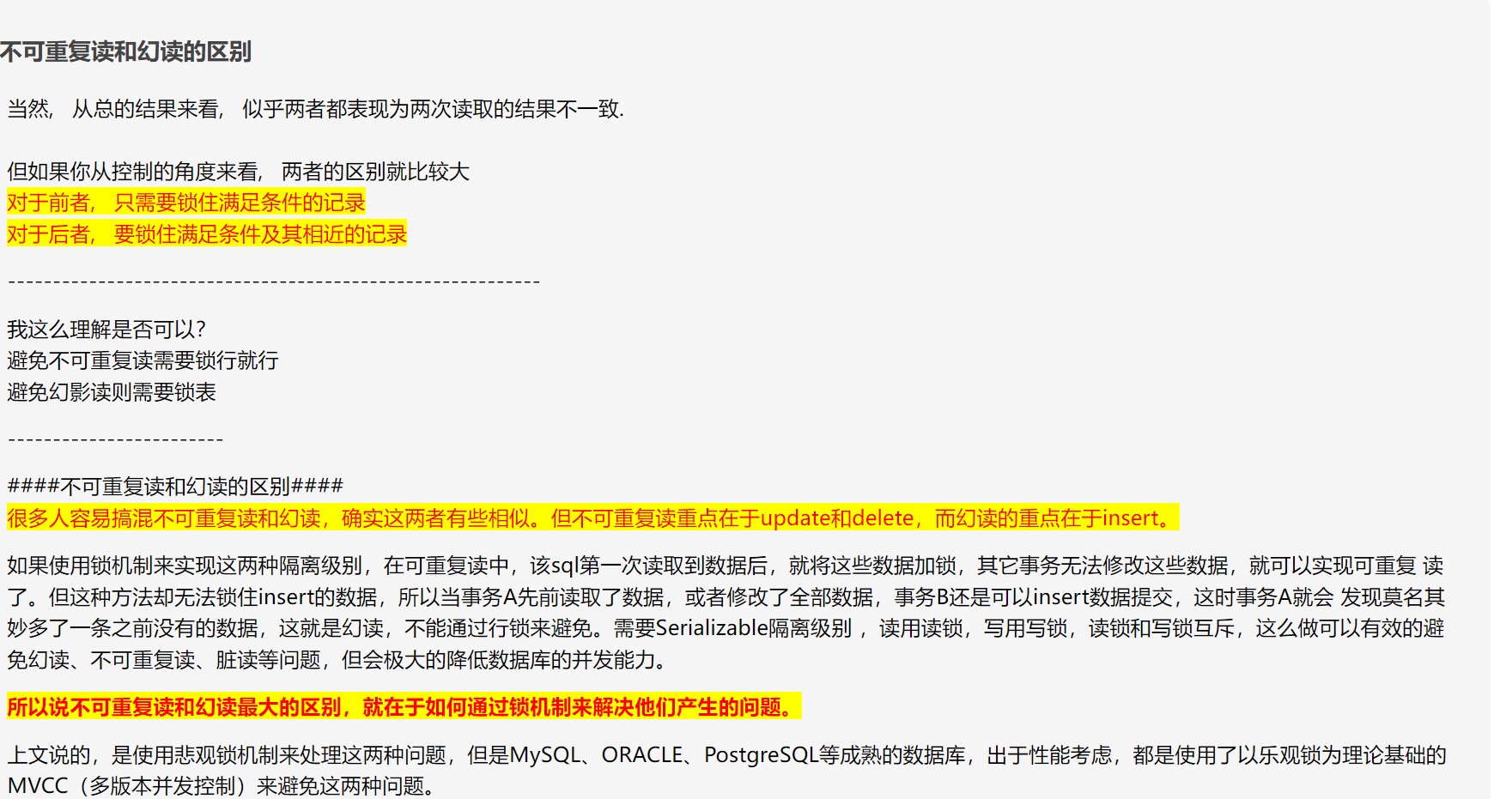
脏读 读到了另一个事务未提交的脏数据

幻读，

上面session B的修改结果，被session A之后的select语句用“当前读”看到，不能称为幻读。幻读仅专指“新插入的行”。 现在你知道了，产生幻读的原因是，行锁只能锁住行，但是新插入记录这个动作，要更新的是记录之间的“间隙”。因此，为了解决幻读问题，InnoDB只好引入新的锁，也就是间隙锁(Gap Lock)。

A事务在读，B事务在中间insert，A事务就在读的前后两次结果集不一样，幻读。需要range lock

不可重复读，同幻读， A两次返回的值不一样。b事务update 需要read lock



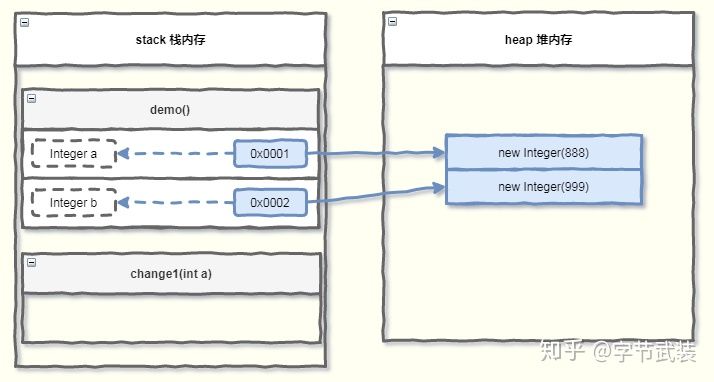
# Integer

值传递 swap 会new 一个对象

**public** **void** **demo3\_1()** **{**

Integer a **=** 888**;** *// @1 实际上调用的是Integer.valueOf(888);*  Integer b **=** change1**(**a**);** *// @2 值传递*

Assert**.**assertTrue**(**a **==** 888**);** *// 通过* Assert**.**assertFalse**(**"a == b"**,** a **==** b**);** *// 通过, a != b***}private** Integer **change1(**Integer a**)** **{** *// @3* a **=** 999**;** *// @4 其实是new Integer(999)* **return** a**;}**



# 并发 并行

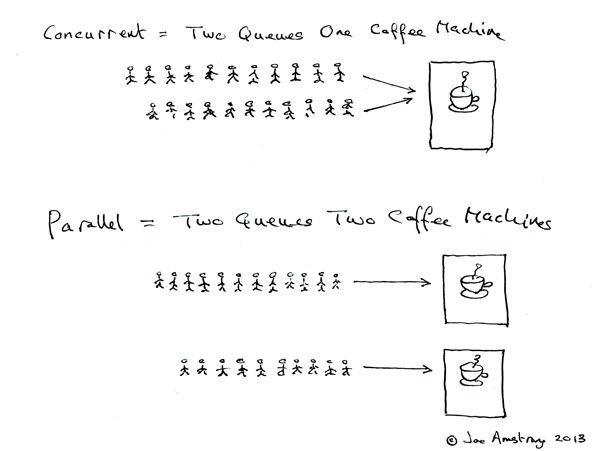


# 枚举

Java 枚举 相当于一个自带属性的累 有限个数







一个队列表示一个可以拆分的任务

一个队列可以看成一个进程

**要点**：[串行](https://www.zhihu.com/search?q=%E4%B8%B2%E8%A1%8C&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1559913485}" \t "_blank)还是并发，这都是**任务安排者**视角看到的东西。前者要求你看到前一个任务结束了，下一个任务才能安排；而后者呢，你可以同时提交许多任务，执行者（们）之间会相互协调并自己安排执行顺序（但未必合理，比如可能出现死锁）——总之，你把任务安排下去就不用管了。

相比之下，“并行”是**任务执行者**视角的东西，和前两者所处平面不同

作者：invalid s

链接：https://www.zhihu.com/question/33515481/answer/1559913485

来源：知乎

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

# Java 逃逸分析

逃逸分析（Escape Analysis）简单来讲就是，Java Hotspot 虚拟机可以****分析新创建对象的使用范围，并决定是否在 Java 堆上分配内存****的一项技术。 在方法中创建对象之后，如果这个对象除了在方法体中还在其它地方被引用了，此时如果方法执行完毕，由于该对象有被引用，所以 GC 有可能是无法立即回收的，此时便成为 内存逃逸现象 。

当变量（或者对象）在方法中分配后，其指针有可能被返回或者被全局引用，这些变量就会逃逸到堆中，造成内存浪费

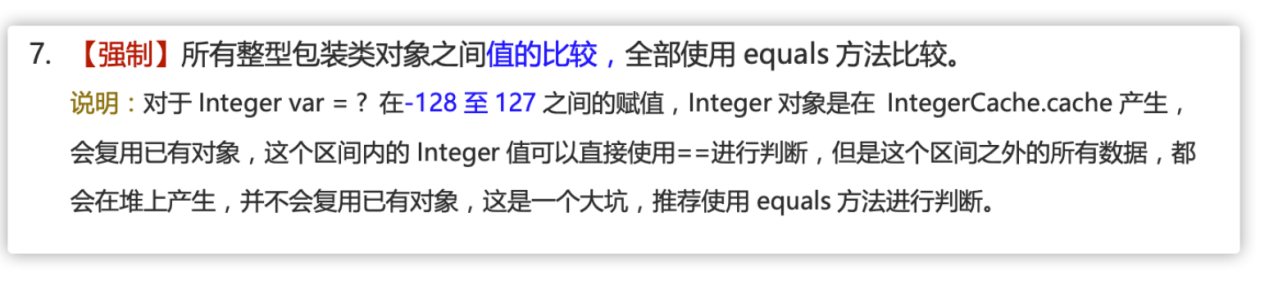
我们知道Java对象总是在堆中分配的，因此Java对象的创建和回收对系统的开销是很大的。Java语言被批评的一个地方，也是认为Java性能慢的一个原因就是Java不支持运行时栈分配对象，缺少像C#里面的值对象或者C++里面的struct结构。

另外为什么要在逃逸分析之前进行内联分析呢？这是因为往往有些对象在被调用过程中创建并返回给调用过程，调用过程使用完该对象就销毁了。这种情况下如果将这些方法进行内联，它们就由两个方法体变成一个方法体了，这种原来通过返回传递的对象就变成了方法内的局部对象，就变成了非逃逸对象了，（目的）这样这些对象就可以在同一栈上进行分配了。 除能将堆分配对象变成栈分配对象，逃逸分析还有其他两个优化应用。一是同步消除。我们知道线程同步的代价是相当高的，同步的后果是降低并发性和性能。逃逸分析可以判断出某个对象是否始终只被一个线程访问，如果只被一个线程访问，那么对该对象的同步操作就可以转化成没有同步保护的操作，这样就能大大提高并发程度和性能。 二是矢量替代。逃逸分析方法如果发现对象的内存存储结构不需要连续进行的话，就可以将对象的部分甚至全部都保存在CPU寄存器内，这样能大大提高访问速度。

基本数据的

⚠️注意 ： ****基本数据类型存放在栈中是一个常见的误区！**** 基本数据类型的成员变量如果没有被 static 修饰的话（不建议这么使用，应该要使用基本数据类型对应的包装类型），就存放在堆中。

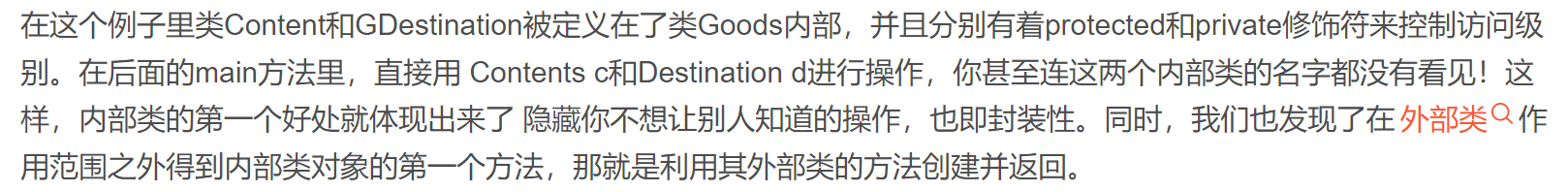
# 包装类 缓存机制



# 权限修饰符

Protect 保护父子 排除

Default 只在本包有用



静态内部类

静态内部类是指被声明为static的内部类，他可以不依赖内部类而实例，而通常的内部类需要实例化外部类，从而实例化。静态内部类不可以有与外部类有相同的类名。不能访问外部类的普通成员变量，但是可以访问静态成员变量和静态方法（包括私有类型）

成员内部类

一个 静态内部类去掉static 就是成员内部类，他可以自由的引用外部类的属性和方法，无论是静态还是非静态。但是不可以有静态属性和方法

局部内部类

定义在一个代码块的内类,他的作用范围是所在代码块，是内部类中最少使用的一类型。局部内部类跟局部变量一样，不能被public ，protected，private以及static修饰，只能访问方法中定义final类型的局部变量

匿名内部类

匿名内部类是一种没有类名的内部类，不使用class，extends，implements，没有构造函数，他必须继承其他类或实现其他接口。匿名内部类的好处是使代码更加简洁，紧凑，但是带来的问题是易读性下降。

内部类的使用时机

1、实现事件监听器的时候（比方说actionListener 。。。采用内部类很容易实现）；

2、编写事件驱动时（内部类的对象可以访问外部类的成员方法和变量，注意包括私有成员）；

3、在能实现功能的情况下，为了节省编译后产生的字节码（内部类可以减少字节码文件,即java文件编译后的.class文

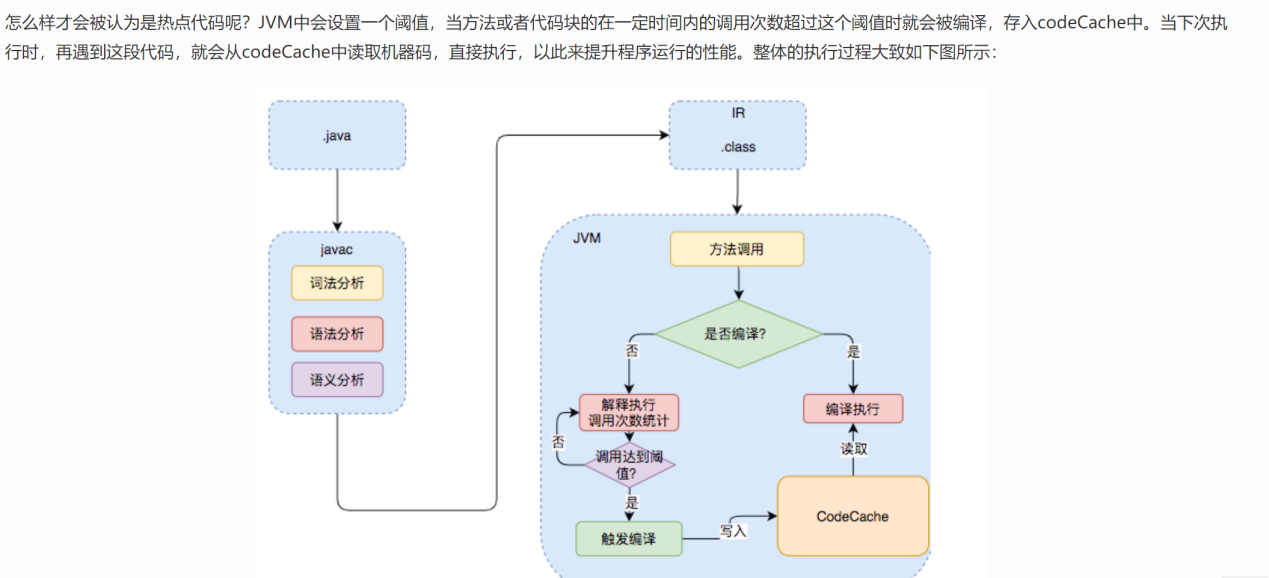
# Java 值传递

栈中直接复制值，所以不会影响全局变量 local variable private variable

而堆中 值传递的是引用 所以会改变 私有变量 即是元数据



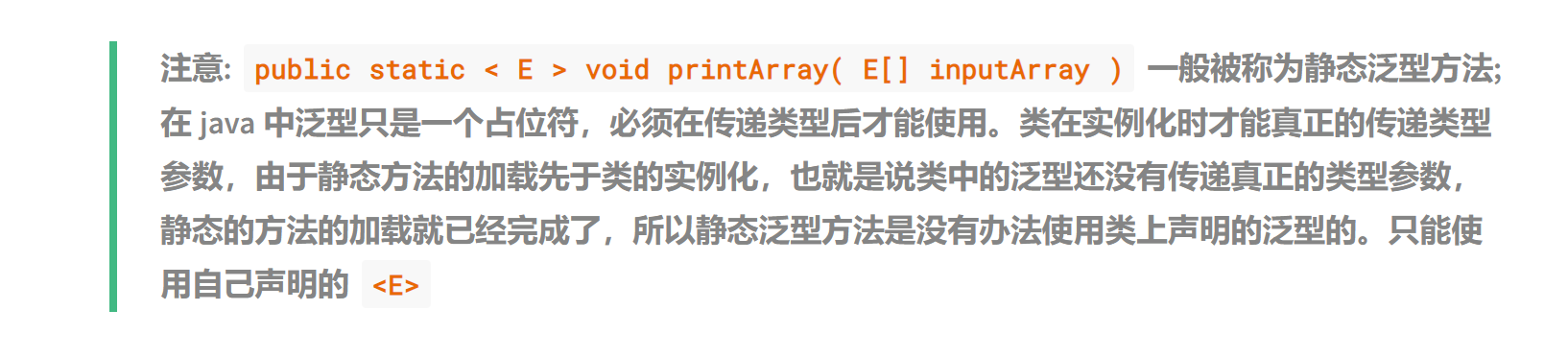
# Java编译过程



# Try finally return

Try中return会被finally中return 替换掉

Try 中return finally 会在return 后执行



# 类的加载过程IMG_256

a).Class.forName除了将类的.class文件加载到jvm中之外，还会对类进行解释，执行类中的static块。

　　b).而classloader只干一件事情，就是将.class文件加载到jvm中，不会执行static中的内容，只有在newInstance才会去执行static块。

　    c).Class.forName(name,initialize,loader)带参数也可控制是否加载static块。并且只有调用了newInstance()方法采用调用构造函数，创建类的对象

# 同步和异步

#### FuncA 调funcB

#### b执行完 A 才能继续执行 此为同步 必须在一起一步一步同步执行

两个不能线程中的函数也可以进行同步调用两个不能线程中的函数也可以进行同步调用FuncA 阻塞式I/O

异步 funcB和 funcA 余下的步骤一起执行 funB 直接返回

被调函数执行完后很自然的就是调用方继续执行，那么异步调用下调用方怎知道被调函数是否执行完成呢？

这就分为了两种情况：

调用方根本就不关心执行结果调用方需要知道执行结果第一种情况比较简单，无需讨论。

第二种情况下就比较有趣了，通常有两种实现方式：

一种是通知机制，也就是说当任务执行完成后发送信号来通知调用方任务完成，注意这里的信号有很多实现方式，Linux中的signal，或者使用信号量等机制都可以实现。

另一种是就是回调，也就是我们常说的callback，关于回调我们将在下一篇文章中重点讲解，本篇会有简短的讨论。

b

Big decimal

****为什么浮点数**float**或**double**运算的时候会有精度丢失的风险呢？****

这个和计算机保存浮点数的机制有很大关系。我们知道计算机是二进制的，而且计算机在表示一个数字时，宽度是有限的，无限循环的小数存储在计算机时，只能被截断，所以就会导致小数精度发生损失的情况。这也就是解释了为什么浮点数没有办法用二进制精确表示。

# 不同接口可以有同名方法

Queue 方法

Dqueue 是双向的

* Priorityqueue 是由二叉堆实现的 是非线程安全的 不支持nullPriorityQueue 利用了二叉堆的数据结构来实现的，底层使用可变长的数组来存储数据
* PriorityQueue 通过堆元素的上浮和下沉，实现了在 O(logn) 的时间复杂度内插入元素和删除堆顶元素。
* PriorityQueue 是非线程安全的，且不支持存储 NULL 和 non-comparable 的对象。
* PriorityQueue 默认是小顶堆，但可以接收一个 Comparator 作为构造参数，从而来自定义元素优先级的先后。
* PriorityQueue 利用了二叉堆的数据结构来实现的，底层使用可变长的数组来存储数据
* PriorityQueue 通过堆元素的上浮和下沉，实现了在 O(logn) 的时间复杂度内插入元素和删除堆顶元素。
* PriorityQueue 是非线程安全的，且不支持存储 NULL 和 non-comparable 的对象。
* PriorityQueue 默认是小顶堆，但可以接收一个 Comparator 作为构造参数，从而来自定义元素优先级的先后。

实现 NavigableMap 接口让 TreeMap 有了对集合内元素的搜索的能力。

# Comparator

Arrays.sort(T[],Comparator<? super T> c); Collections.sort(List<T> list,Comparator<? super T> c);

# Lambda（函数） 表达式

表达式中的参数是一个形参 是方法参数

匿名内部类 传入一个函数的专一功能

lambda 可以直接省掉传入参数的类型

传入匿名的接口引用对象 并将接口中的方法重写

package com.isea.java;

public class TestLambda {

public static void main(String[] args) {

Thread thread = new Thread(new Runnable() {

@Override

public void run() {

System.out.println("Hello");

}

});

thread.start();

}

}

//写成这样

package com.isea.java;

public class TestLambda {

public static void main(String[] args) {

new Thread(() -> System.out.println("Hello")).start();

}

}

([Lambda参数列表，即形参列表]) -> {Lambda体，即方法体}

拷贝小括号，写死右箭头，落地大括号，大括号中写上业务逻辑

package com.isea.java;

import java.util.ArrayList;

public class TestLambda {

public static void main(String[] args) {

ArrayList<String> list = new ArrayList<>();

list.add("AAAAA");

list.add("BBBBB");

list.add("CCCCC");

list.add("DDDDD");

//形参的类型是确定的，可省略；只有一个形参，()可以省略；

list.forEach(t -> System.out.print(t + "\t"));

//或者更简洁的方法引用：list.forEach(System.out::println);

//打印结果：AAAAA BBBBB CCCCC DDDDD

}

}

public void forEach(Consumer<? super E> action)

ForEach 的用法

## **forEach() 方法**

forEach() 方法是 Java 8 为所有集合新增的方法。该方法定义在 java.lang.Iterable 接口中。

java.lang.Iterable 接口是 Java 5 引入的，目的在于为实现该语句的对象提供 「 for-each 循环 」 语句。换句话说，所有实现了该接口的对象都可以使用 for 语句进行迭代。

**[← Java 8 接口静态方法](https://www.twle.cn/c/yufei/java8/java8-basic-interface-static-method.html)[Java 8 可选值 java.util.Optional 类 →](https://www.twle.cn/c/yufei/java8/java8-basic-optional-class.html)**

# Java 8 集合遍历 forEach() 方法

Java 8 为所有的集合添加了一个新的方法 forEach() ，该方法以只读形式遍历集合所有的元素并为每一个元素执行一个动作。

我们先来看一个范例

#### **ForEachTester.java**

import java.util.Arrays;

public class ForEachTester{

public static void main(String[] args)

{

Arrays.asList("你好", "简单教程", "简单编程").forEach(System.out::println);

}}

上面这段代码中，我们使用了 forEach 方法遍历列表的每一个元素，并把元素传递给 System.out.println() 方法打印输出到屏幕上。

对于这个功能，如果我们使用 for ( : ) 语法，则会又有些小复杂

import java.util.Arrays;import java.util.List;

public class ForEachTester{

public static void main(String[] args)

{

List<String> list = Arrays.asList("你好", "简单教程", "简单编程");

for( String it : list)

{

System.out.println(it);

}

}}

## **forEach() 方法**

forEach() 方法是 Java 8 为所有集合新增的方法。该方法定义在 java.lang.Iterable 接口中。

java.lang.Iterable 接口是 Java 5 引入的，目的在于为实现该语句的对象提供 「 for-each 循环 」 语句。换句话说，所有实现了该接口的对象都可以使用 for 语句进行迭代。

当然了，我们今天的重点是 forEach() 这个方法。该方法的原型如下

default void forEach(Consumer<? super T> action)

从函数原型中可以看出，该方法是 java.lang.Iterable 接口的默认方法，所有子类可以不用实现，也没必要实现。

该方法对 Iterable 中的的每个元素执行给定的操作 ( action )，直到处理完所有元素或操作抛出异常为止。除非实现类另有指定，否则操作会按迭代顺序执行（ 如果指定了迭代顺序 ）。且操作抛出的异常将转发给调用者。

该方法的的实现类似于

for (T t : this){

action.accept(t);}

除非实现类另有指定，否则操作会按迭代顺序执行（ 如果指定了迭代顺序 ）。且操作抛出的异常将转发给调用者。

该方法的的实现类似于

for (T t : this){

action.accept(t);}

## **范例一：forEach() 方法迭代列表**

我们写一个范例使用 forEach() 方法和方法引用来实现一个列表的迭代

#### **ForEachTester.java**

import java.util.Arrays;import java.util.List;

public class ForEachTester{

public static void main(String[] args)

{

List<String> list = Arrays.asList("你好", "简单教程", "简单编程");

for( String it : list)

{

System.out.println(it);

}

}}

运行结果如下

[yufei@www.twle.cn helloworld]$ javac ForEachTester.java && java ForEachTester

你好

简单教程

简单编程

## **范例二： forEach() 方法迭代一个哈希表**

我们写一个范例使用 forEach() 方法和 Java 8 提供的 lambda 表达式来迭代一个哈希表

#### **ForEachTester.java**

import java.util.HashMap;import java.util.Map;

public class ForEachTester{

public static void main(String[] args)

{

Map<String, Integer> items = new HashMap<>();

items.put("A", 10);

items.put("B", 20);

items.put("C", 30);

items.put("D", 40);

items.put("E", 50);

items.put("F", 60);

items.forEach((k,v)->System.out.println("Item : " + k + " Count : " + v));

items.forEach((k,v)->{

System.out.println("Item : " + k + " Count : " + v);

if("E".equals(k)){

System.out.println("Hello E");

}

});

}}

运行结果如下

[yufei@www.twle.cn helloworld]$ javac ForEachTester.java && java ForEachTester

Item : A Count : 10

Item : B Count : 20

Item : C Count : 30

Item : D Count : 40

Item : E Count : 50

Item : F Count : 60

Item : A Count : 10

Item : B Count : 20

Item : C Count : 30

Item : D Count : 40

Item : E Count : 50

Hello E

Item : F Count : 60

## **范例三：forEach() 方法迭代一个流**

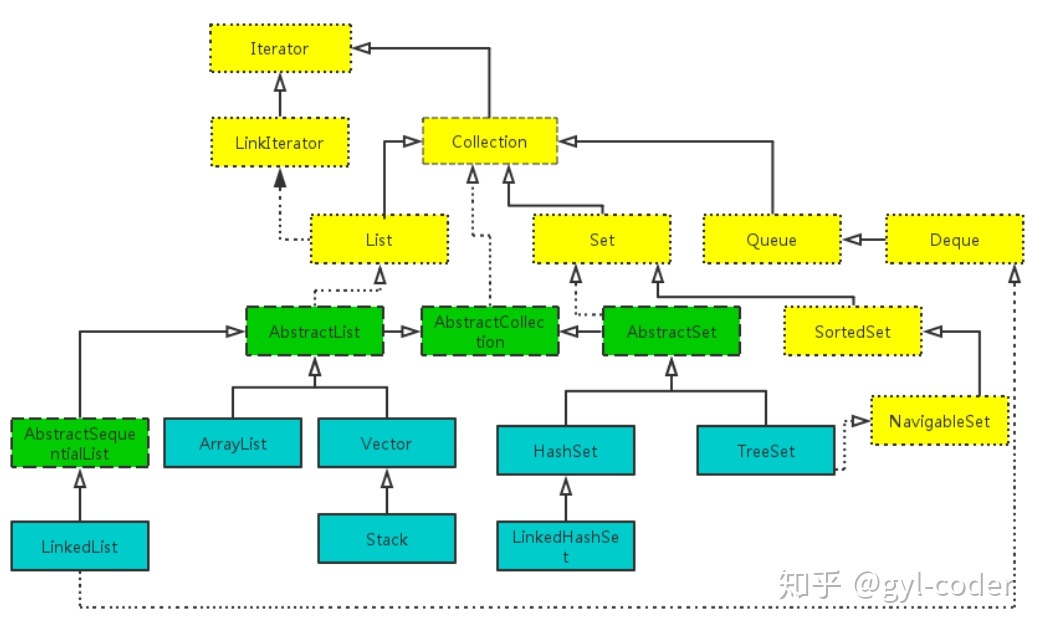
我们写一个范例使用 forEach() 方法迭代一个流，输出流中的所有元素

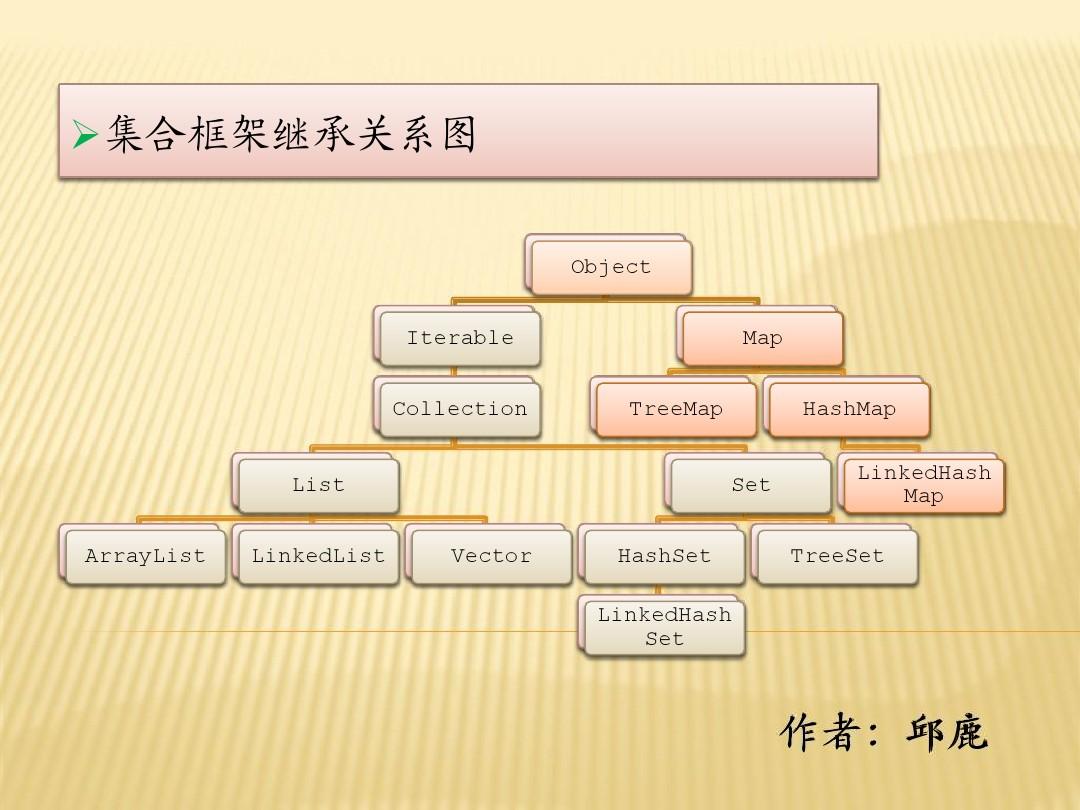
#### **ForEachTester.java**

**[← Java 8 接口静态方法](https://www.twle.cn/c/yufei/java8/java8-basic-interface-static-method.html)[Java 8 可选值 java.util.Optional 类 →](https://www.twle.cn/c/yufei/java8/java8-basic-optional-class.html)**

# Java集合







# 函数式接口

什么是函数式接口？

即SAM（Single Abstract Method ）接口，有且只有一个抽象方法的接口（可以有默认方法或者是静态方法和从Object继承来的方法，但是抽象方法有且只能有一个）。 JDK1.8之后，添加@FunctionalInterface表示这个接口是是一个函数式接口，因为有了@functionalInterface标记，也称这样的接口为Mark（标记）类型的接口。举例子：

@FunctionalInterface

public interface Runnable {

public abstract void run();

}//来自源码

1. 消费型接口：Consumer< T> void accept(T t)有参数，无返回值的抽象方法；

2. 供给型接口：Supplier < T> T get() 无参有返回值的抽象方法；

3. 断定型接口： Predicate< T> boolean test(T t):有参，但是返回值类型是固定的boolean

4. 函数型接口： Function< T，R> R apply(T t)有参有返回值的抽象方法；

除了这四个之外，在java.util.function包下还有很多函数式接口可供使用。

# 双冒号

请注意，此时的接口Comparator是一个用

[@Functional](//www.zhihu.com/people/d508d8faab94ce24f178d83882036702" \t "_blank)

来注解的，所以你可以用一个Lambda表达式来写方法定义，来创建Comparator的实例：

Arrays.sort(rosterAsArray,

(Person a, Person b) -> {

return a.getBirthday().compareTo(b.getBirthday());

}

);

在Person.compareByAge函数中，其instance已存在，所以我们要比较两个Person实例的年龄大小，用Lambda expression可以这样写：

Arrays.sort(rosterAsArray,

(a, b) -> Person.compareByAge(a, b)

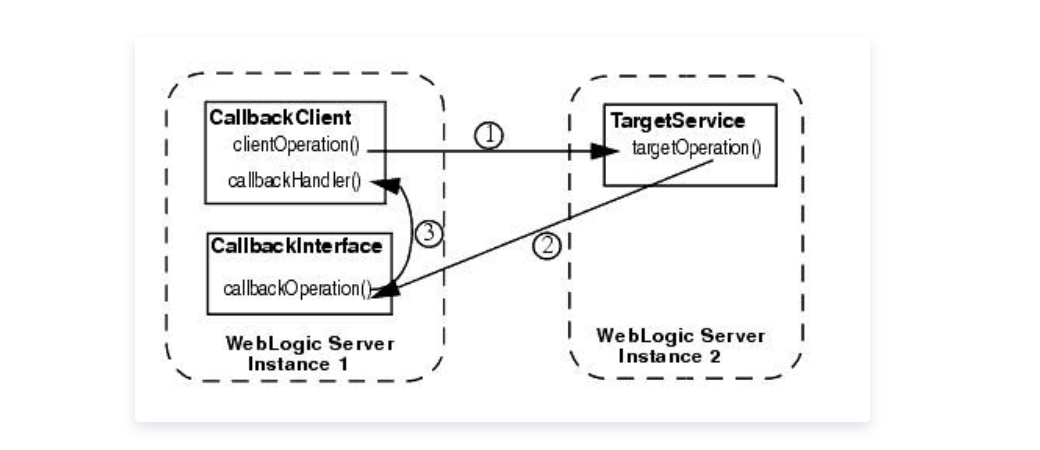
);

由于咱们的Lambda表达式调用的是一个已经存在的方法，即，我们做的就是***方法引用***，所以你就可以用双冒号::来改造一下：

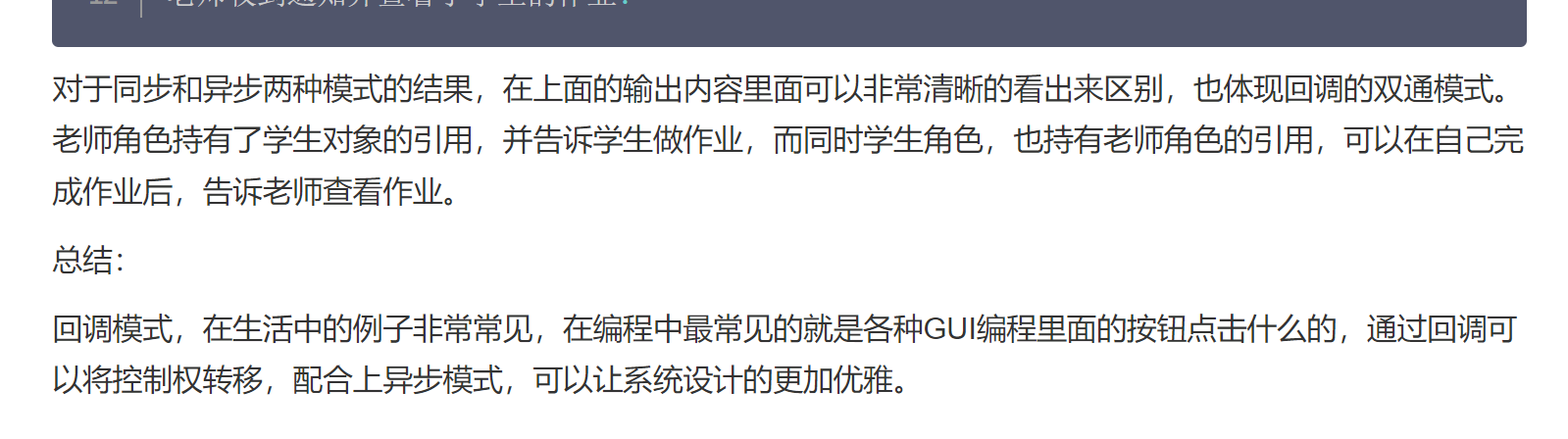
Arrays.sort(rosterAsArray, Person::compareByAge);

双冒号 就是方法引用，双冒号的最后一个方法表示入参

# 回调函数



老师写回家 修改完回家 和先回家用手机批改 调给学生



https://cloud.tencent.com/developer/article/1373683

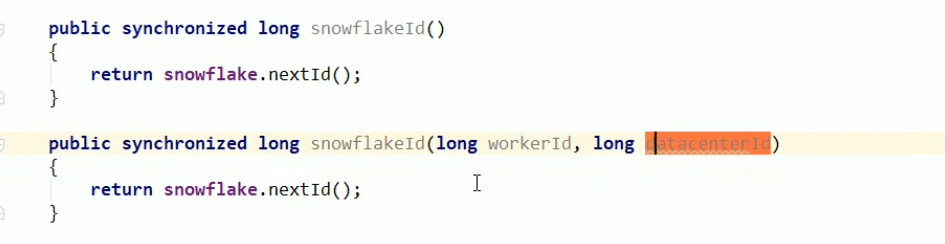


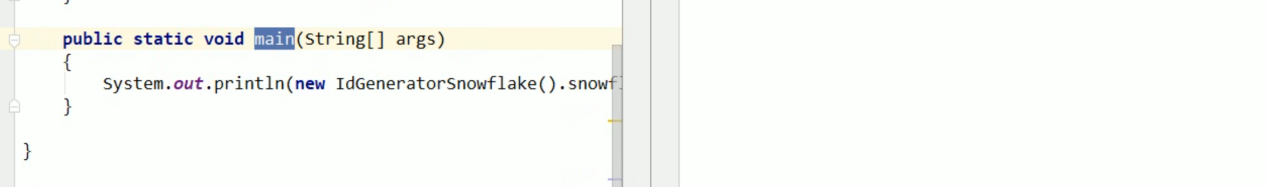
# PostConstruct 注解

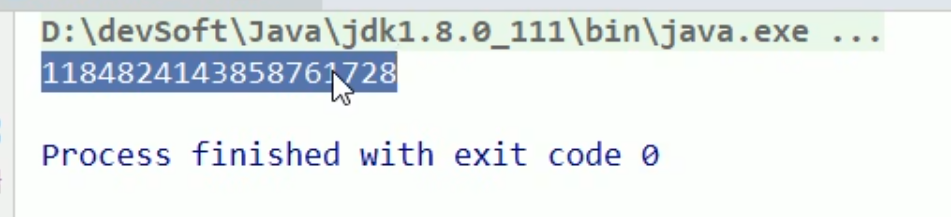
@PostConstruct应用场景：

如果想在生成对象时候完成某些初始化操作，而偏偏这些初始化操作又依赖于依赖注入，那么就无法在构造函数中实现。为此，可以使用@PostConstruct注解一个方法来完成初始化，@PostConstruct注解的方法将会在依赖注入完成后被自动调用。

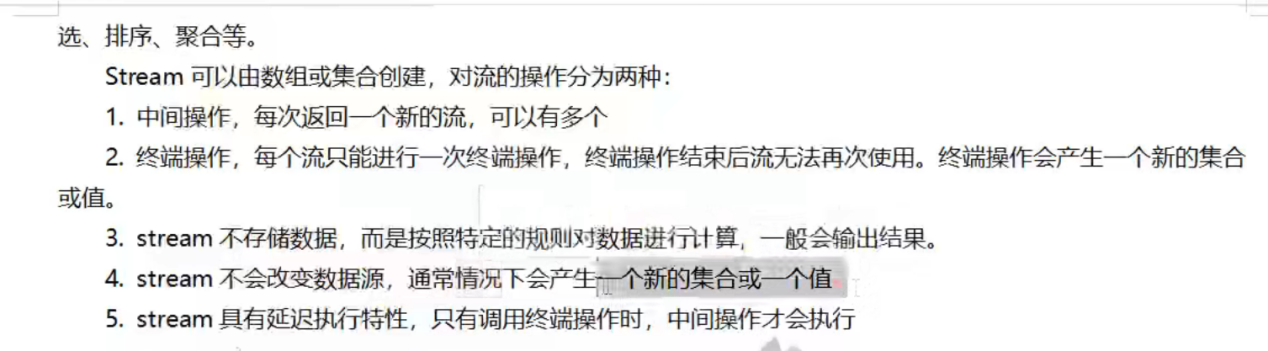


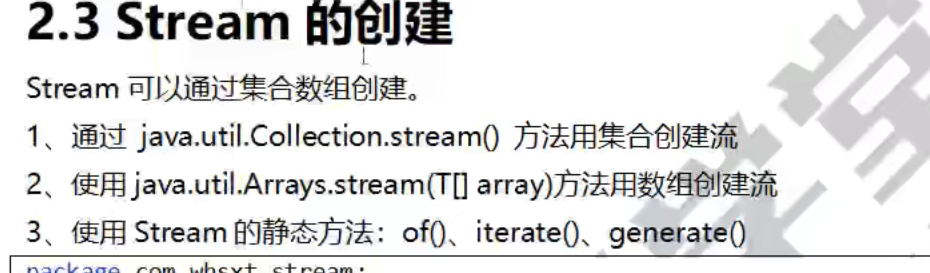




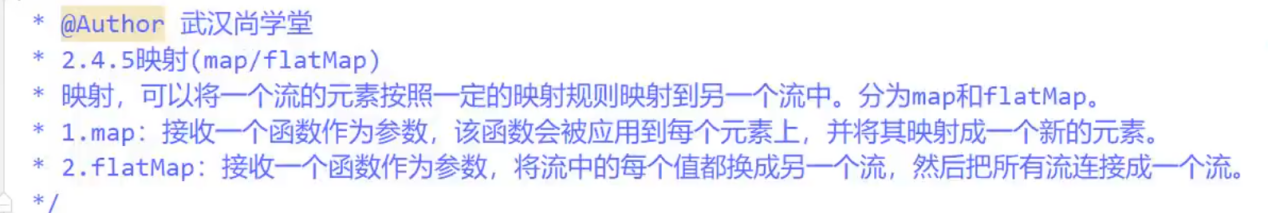


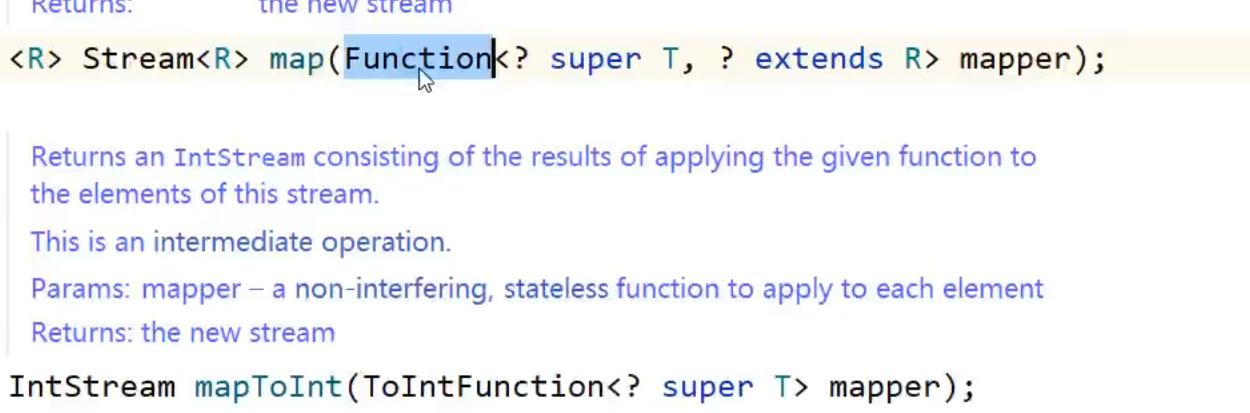
# Stream

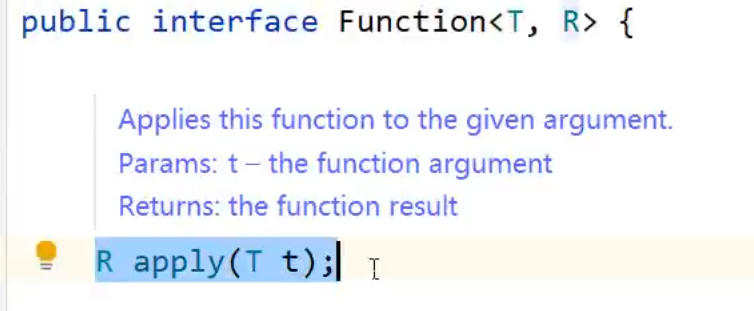




# Stream.map

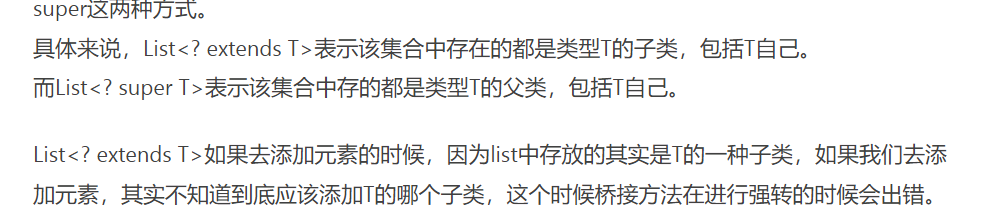






相当于对每个元素 进行规则映射

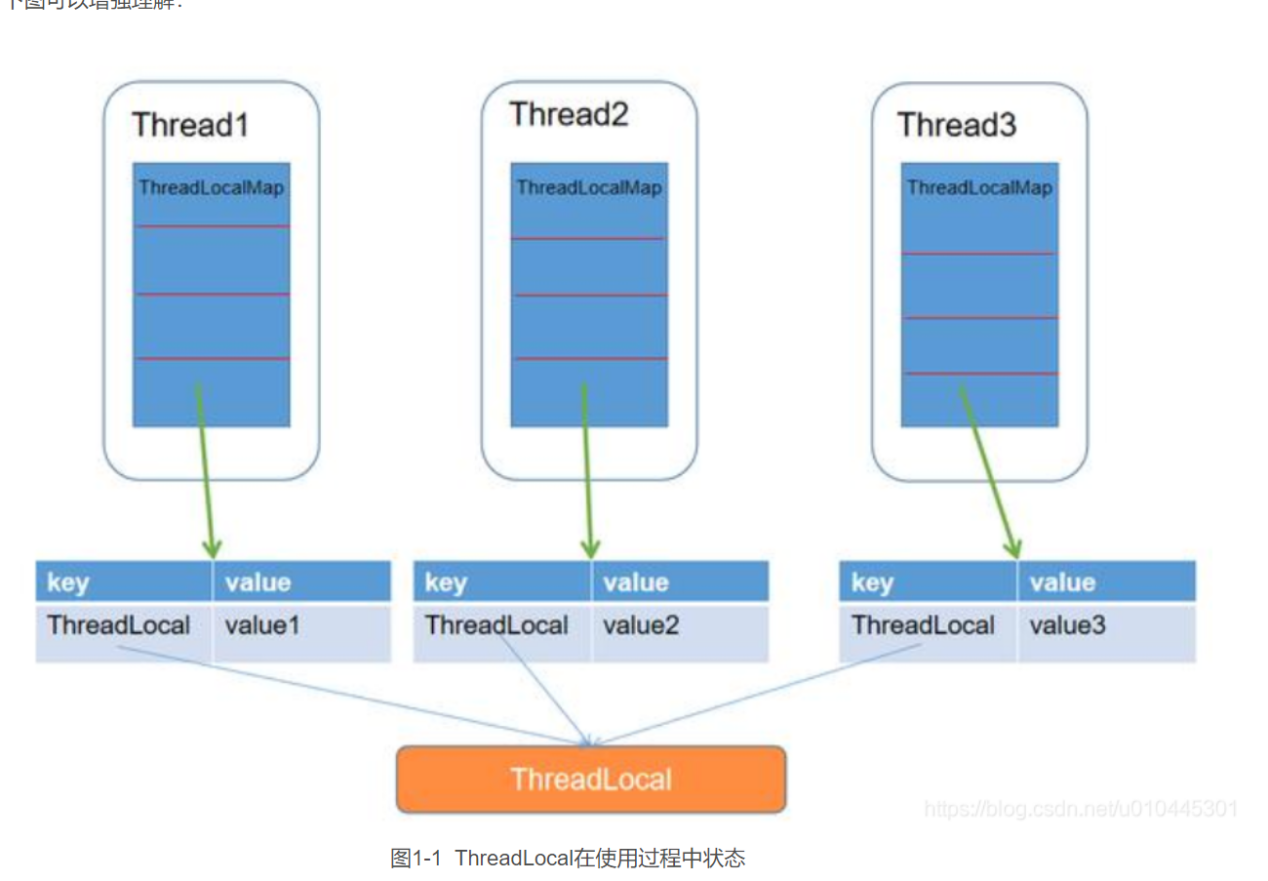
# ? Super T



# Stream.sorted()



# Threadlocal





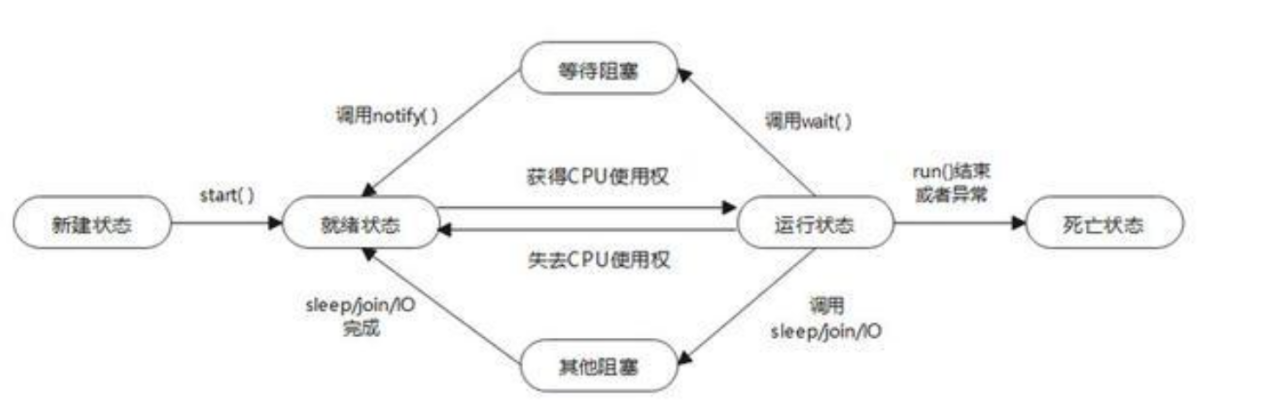
“很多项目中需要在代码中使用当前登录用户的信息,但是又不方便把保存用户信息的session对象传来传去,这种情况下,就可以考虑使用 ThreadLocal。ThreadLocal是一个依附于本地线程的变量,按照我的理解,每次对服务器请求,都会使用到一个线程,ThreadLocal的作用就是在这个线程的使用过程中只为这个线程所用。”

# 并发的原子性

对于涉及共享变量访问的操作，若该操作从其执行线程以外的任意线程来看是不可分割的，那么该操作就具有原子性。原子操作是多线程环境下的一个概念，它是针对访问共享变量的操作而言的。原子操作的“不可分割”包括以下两层含义：

* 读写某个共享变量的操作从其执行线程以外的任何线程来看，该操作要么已经执行结束要么尚未发生。
* 访问同一组共享变量的原子操作是不能够被交错的。

# 线程生命周期



# Yield

让出cpu 时间片儿 让其他线程先执行一会儿

## JOIN

Join 英文名字 链接 连片

理解为有限执行

1. join（）优先执行T

因此在tA.join()当中的wait(0)方法是让main线程陷入了无尽的等待中。正是因为如此，在tA.join()之前的代码都会正常从上往下执行，而在tA.join()之后的代码都随着main线程陷入等待而无法继续执行。这样便达到了网上说的 “t.join()方法会使所有线程都暂停并等待t的执行完毕后再执行”。

## 到此，关于join的讲解尚未结束。之前有说到，join方法中调用wait(0)让当前线程陷入无尽的等待。那么有wait等待就会有相应的notify或者notifyAll唤醒，那么唤醒的地方又在哪里？

这个问题我也跟踪过join的方法，但始终发现不了在哪里唤醒等待的线程。所以在此引用网上的结论：join源码中，只会调用wait方法，并没有在结束时调用notify，这是因为线程在die的时候会自动调用自身的notifyAll方法，来释放所有因为该锁陷入等待的资源和锁。

# 控制反转 和依赖注入

控制反转（Inversion of Control，缩写为IoC )，是面向对象编程中的一种设计原则，可以用来减低计算机代码之间的耦合度。其中最常见的方式叫做依赖注入（Dependency Injection，简称DI），还有一种方式叫“依赖查找”（Dependency Lookup）。通过控制反转，对象在被创建的时候，由一个调控系统内所有对象的外界实体，将其所依赖的对象的引用传递给它。也可以说，依赖被注入到对象中。

控制反转针对的是依赖对象的获得方式，也既依赖对象不在是自己内部生成，而是由外界生成后传递进来。

依赖注入 由容器来控制

# 依赖倒置

Service 类依赖了很多的bean bean 又依赖了其他bean 造成庞大 复杂，使用ioc

不用关心 这写bean 的构造函数是什么样的，使用service 接口 ，可以简化修改

# 线程安全

“**线程安全是多线程编程时的计算机程序代码中的一个概念。**在拥有共享数据的多条线程并行执行的程序中，线程安全的代码会通过同步机制保证各个线程都可以正常且正确的执行，不会出现数据污染等意外情况。”

# 有限状态机

有限状态机(Finite State Machine) 缩写为 FSM。以下简称为状态机。

状态机有 3 个组成部分：状态、事件、动作。

状态：所有可能存在的状态。包括当前状态和条件满足后要迁移的状态。

事件：也称为转移条件，当一个条件被满足，将会触发一个动作，或者执行一次状态的迁移。

动作：条件满足后执行的动作。动作执行完毕后，可以迁移到新的状态，也可以仍旧保持原状态。动作不是\* 必需的，当条件满足后，也可以不执行任何动作，直接迁移到新状态。

状态机的表示

状态转移图

我们举个例子，马里奥共有三种状态，可执行三个动作，如图中所示。

从图中可以看到，

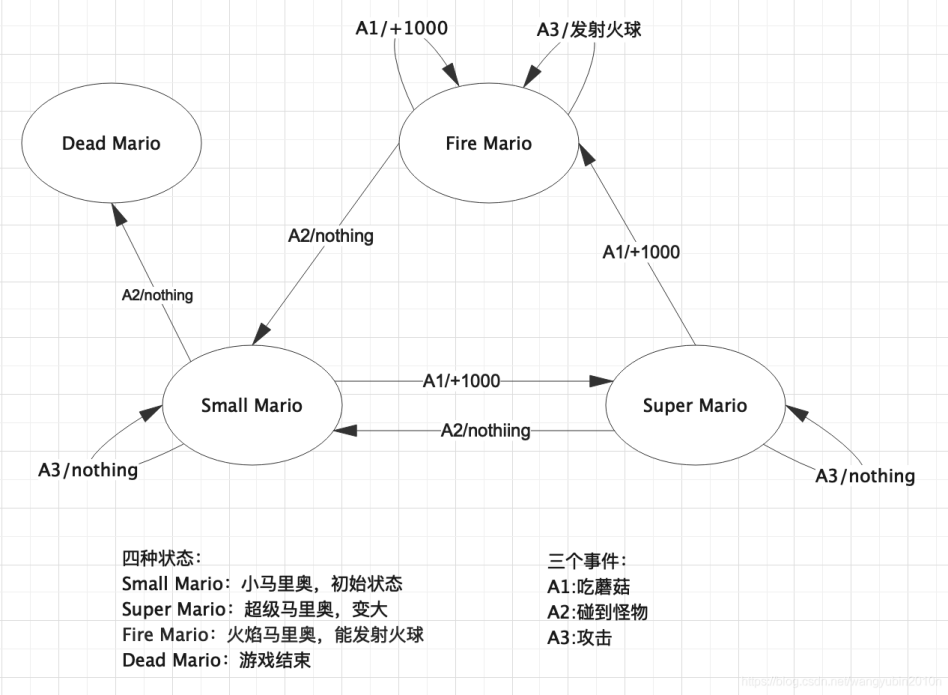
在Small Mario状态下，A3，没有触发状态转移，也没有触发任何动作。

在Fire Mario状态下，A3，没有触发状态转移，触发了发射火球动作。

在Super Mario状态下，A2，触发状态转移到Small Mario，没有触发任何动作。

在Small Mario状态下，A1，触发状态转移到Super Mario，触发增加1000积分。

可见：动作的执行，和状态的转移，是相对独立的。并不是必须有状态转移，或必须触发动作。状态转移和动作之间也没有必然联系。



# Dispatcher servlet 和 spring mvc 的关系

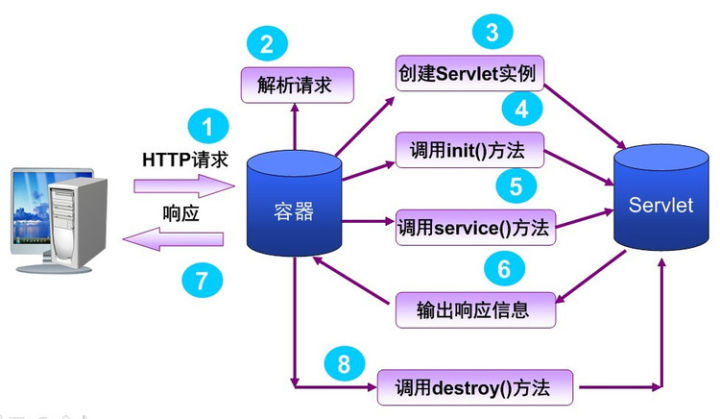
总结 ： 请求处理是tomcat 中 调用，调用dispatcher service 类中的service方法。

可以看出Tomcat与SpringMVC其实就是通过request，response交流的。

SpringMVC将结果写入response后，Tomcat则会根据http协议格式，提取response内容写入Buffer，再将Buffer写入SocketChannel。

# main方法的作用

The main method is used to specify the starting point of the program. This is the starting point of our program from where the JVM starts execution of the program. Can you have methods in main Java.

如果使用spring mvc，那么tomcat只是解析http报文，然后将其转发给dispatchsetvlet，然后由springmvc根据其配置，实例对应的类，执行对应的逻辑，然后返回结果给dispatchservlet，最后由它转发给tomcat,由tomcat负责构建http报文数据。



**3**.SpringMVC框架如何让请求进入某个业务方法

根据【[tomcat与springMVC是如何协同工作的](https://zhuanlan.zhihu.com/p/361185535)】一文可知道，tomcat会调用SpringMVC的DispatcherServlet类的service方法。

源码后面再说，大家根据日常实际开发经验也大概能猜出，该方法中SpringMVC会根据请求路径去匹配对应@RequestMapping配置路径的Controller，再根据后面的路径去匹配该Controller中对应@RequestMapping配置路径的方法。

而在Tomcat中，是通过读取web.xml文件来知道给定的实现类，比如

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param></filter><listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class></listener><servlet>

<description>spring mvc servlet</description>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>

classpath:spring-mvc.xml

</param-value>

</init-param>

<load-on-startup>1</load-on-startup></servlet>