# https://github.com/Snailclimb/JavaGuide学习java网站

# 面向对象

## 局部变量和成员变量的区别

1. 在方法中定义的变量 在类中定义的非静态变量信息
2. 先赋值在使用 可以不赋初始值，系统自动赋值 0 null 0.0 false
3. 方法中使用 整个类中使用
4. 完成即销毁 随类产生销毁

## 对象调用

1. 当在别的类中调用时 静态：直接类名.方法名 非静态：实例化后用 引用名.方法名
2. 当在本类中 静态方法只能调用静态的成员变量和方法，调用非静态需要在方法里实例化， 不是静态的本类中可以直接调用。

# 2、面向对象---封装

## 1、简介

面向对象的设计过程？ 发现类，发现类的属性，发现类的方法

对象和类的区别？ 类是对象的类型，对象是类的实例

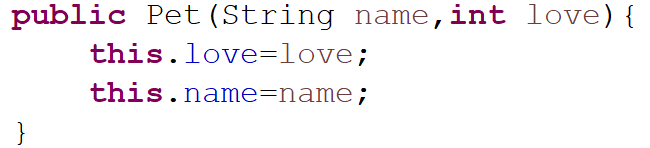
什么事封装？ 隐藏了对象属性和实现的细节，仅对外部提供共有的方法来访问操作

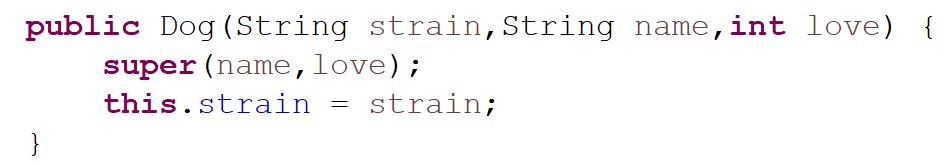
## 2、构造方法

构造方法的作用为 创建对象的时候执行一些初始化操作 结构体:

权限修饰符 类名(){}

把公共属性弄到父类构造方法里，公共方法写到父类方法，子类构造就调用父类构造方法并把独特的属性加到里面

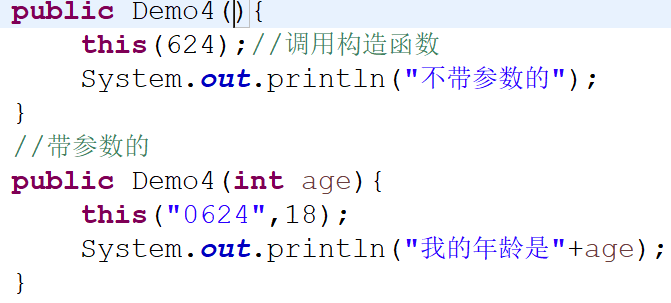




注意点：如果类没有构造方法，系统会自动创建一个不带参数的构造函数，一旦创建了构造，默认构造函数就不在存在，要想使用，必须手动创建

## 3、this的用法

this为当前对象：

1. 在方法内获取同名的类变量信息 this.属性名
2. 通过函数调用本类中函数信息，这时候this可以省略
3. 通过构造调用其他构造函数  
   

注意点：

1. 构造函数不能调用自己本身，
2. 构造函数不能相互调用
3. 只能在一个构造函数中调用一次构造函数，如果调用多个可以依次调用
4. This调用构造函数的时候只能放在第一行
5. 静态方法不能使用this和super关键字

# 3、面向对象---继承

## 1、结构体

修饰符 class 子类名 extends 父类名{}

## 2、注意点

1、私有的属性和方法不能被子类继承

2、java不能多继承

## 3、重写：

在子类中必须有相同的方法名和参数列表，返回值类型相同或是其子类

可以大于或者等于父类的方法权限

## 4、重载：

在同一个类中，函数名称一致，参数列表不同（参数类型，参数数量，参数次序），和返回值无关，这种函数的体现叫做函数的重载。

如果存在多继承关系那么会一直默认执行到object的无参构造为止； 从父类构造往子类执行

## 5、super的用法

1、可以使用super关键调用父类被子类隐藏的属性信息 super.属性名

2、super 调用父类被子类隐藏的方法 super.方法名 ();

3、super 可以子类的构造函数调用父类的构造函数 super(参数列表);

1. super关键用在继承关系的子类中

2、子类会默认调用父类的不带参数的构造函数，必须调用一个构造函数，否则报错

1. super 调用构造函数得放到第一行
2. this和super不能同时使用调用构造方法,因为他俩调用构造都得在第一行

# 4、面向对象---多态

## 1、权限修饰符

本类中 同一个包中 不同包子类 其他包的其他类

public 可用 可用 可用 可用

protected 可用 可用 可用 不可用

默认的(default) 可用 可用 不可用 不可用

private 可用 不可用 不可用 不可用

public:

修饰：方法、属性、类

private：私有的

修饰：方法和属性 ，也可以修饰类(只限制内部类)

默认的：

修饰： 类、方法、属性

protected：受保护的

修饰： 方法 属性

## 2、多态简介

是具有表现多种形态的能力的特征

### 1、向上转型

1、使用父类的引用可以指向子类的对象 Person per=new Student(); 向上转型自动转

2、此时通过父类引用变量调用的方法是子类重写或继承的方法而不是父类的方法

3、父类引用变量无法调用子类特有的方法

### 2、向下转型：

子类引用指向父类只能转换为父类指向的真实子类类型

Pet p=new Dog();

Dog d=(Dog)pet; 正确

Penguin pgn=(penguin)pet; 出现异常

必备条件：

1、要有继承

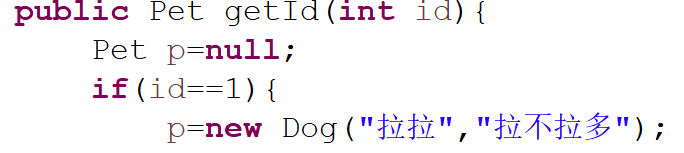
2、要有重写

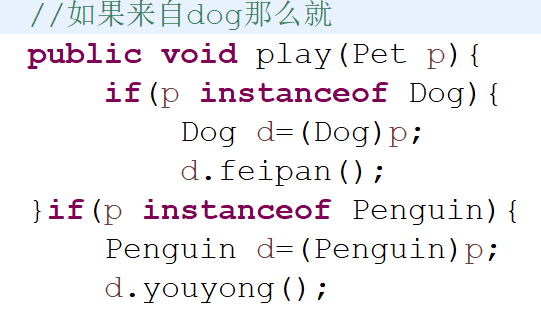
3、父类的引用指向子类的对象

### 3、强制转换类型 instanceof 判断 一个对象是否来自那个类

people p=new Student();

p instanceof people





# Static final 与abstract

## 1、 static

主要修饰： 成员变量、方法、初始化代码块

不能修饰局部变量

修饰属性：

在本类中，可以直接使用

不同类中：类名.属性名

修饰方法：

在本类中，直接调用

在不同类中，通过类名.方法名(参数列表)

静态方法必须通过静态方法重写

初始化代码 结构体：

static {

内容

}

静态变量只有一份存储空间，每个实例化对象都是公用静态属性的

静态块只会初始化一次重复调用不会重复执行

静态方法没啥用只是不创建对象的前提下调用

一旦加载类信息或实例化,静态块会立即执行，静态对象引用会被初始化

初始化顺序：先初始化静态的，父类构造，然后普通的，然后执行构造函数里的

## 2、final: 必须在定义时指定初始值

主要用来修饰 类、方法、属性

当final修饰变量的时候，一般指的是常量信息是和static结合使用 static final Strign SEX=”男”;

不可以被更改

用final修饰的方法，不能被重写

final 修饰类：代表是一个完整的类，不能被继承，但是里面的属性是可以用的

## 3、abstract:

用来修饰 类 方法

类 不能被实例化

方法 没有内容，必须被重写

1. 在抽象类中可以定义普通或者抽象方法
2. 抽象类不能被实例化 要想用可以用子类继承
3. 父类中的抽象方法在子类（普通类）必须被重写, 子类(抽象类) 可以不重写

abstract 和 private方法 错误：子类无法继承私有方法

和 static方法 错误：抽象方法还没实现呢，不能直接调用

和 final方法或类 错误：矛盾一个必须重写方法，一个不能重写

# 接口

## 1、简单介绍接口

良好的接口定义一定来自于需求，而不是程序员绞尽脑汁想出来的

每个对象都只能满足某些请求，这些请求由对象的接口定义

接口是什么：接口表示一种约定，表示一种能力

has—a(拥有) is—a(属于)

防盗门拥有锁 属于门

## 2、接口的作用：弥补java单继承的缺陷，利于代码扩展和维护

结构体：

修饰符 interface 接口名{

属性: 全局静态常量，必须在定义时指定初始值：： public static final 默认自动修饰

方法： 全局抽象方法 public abstract 默认自动修饰

void 方法名();

}

通过类实现接口：

修饰符 class 类名 implements 接口名,接口名2,接口名3{ }

通过接口实现接口：

修饰符 interface 接口名 extends 接口名1,接口名2{ }

## 3、注意点：

同时继承类和接口extends位于implements前

接口不能实例化，不能构造方法，不能继承类

类实现接口可以同时实现多个接口，多个接口之间逗号分隔

通过接口继承接口可以多继承

## 4、接口和抽象类区别：

相同点：

1、在java中接口和抽象类都不能被实例化

2、抽象类和接口都可以含有抽象方法

3、如果通过类(普通类)实现接口或者继承抽象类都需要重写抽象方法

不同点：

1、抽象类可以含有普通方法，接口不能有普通方法默认都是public abstract

2、抽象类中属性一般任意都可以，接口中属性都会自动用public static final 修饰

3、一个类可以继承一个抽象类，而类实现接口可以一次实现多个接口

# Jdbc

## 1、jdbc API

jdbc(java database connectivity)

Class.forName("com.mysql.jdbc.Driver"); 加载数据库驱动程序

con=DriverManager.getConnection("jdbc:mysql://localhost:3306/student?useUnicode=true&characterEncoding=utf-8", "root", "123456"); 建立连接

DriverManager:负责加载各种不同驱动程序，并根据不同的请求，向调用者返回相应的数据库链接（Connection）；

Connection:数据库连接，负责与进行数据库通讯，sql执行以及事物处理都是在某个特定的Connection环境中进行的；

pst=con.prepareStatement(这里面写查询和更新语句);

然后查询用rs=sm.executeQuery; 返回查询的结果集 ResultSet.next方法判断当前记录是否为空

更新用int no=sm.executeUpdate； 对于更新语句(插入，更新，删除)完成后返回一个受操作影响的行数；

Rs.getxxx(1顺序,“或者字段名字”)方法读取结果集中当前记录的字段信息

语法都可以加上？ 比如：where name=？

然后用pst.setxxx方法(1,"这里写需要查的名字"); 1就代表第一个问号

PreparedStatment:用于执行SQL查询和更新(比Statment效率高，提高了可读性和可维护性和安全性避免了sql注入攻击);

SQLException:代表在数据库连接的创建和关闭和SQL语句的执行过程中发生了例外情况

关闭对象的时候从re pst到 con

con.isClosed(); 是否被关闭

con.setAutoCommit(false) 设置不自动提交

con.rollback; 回滚

con.commit; 提交

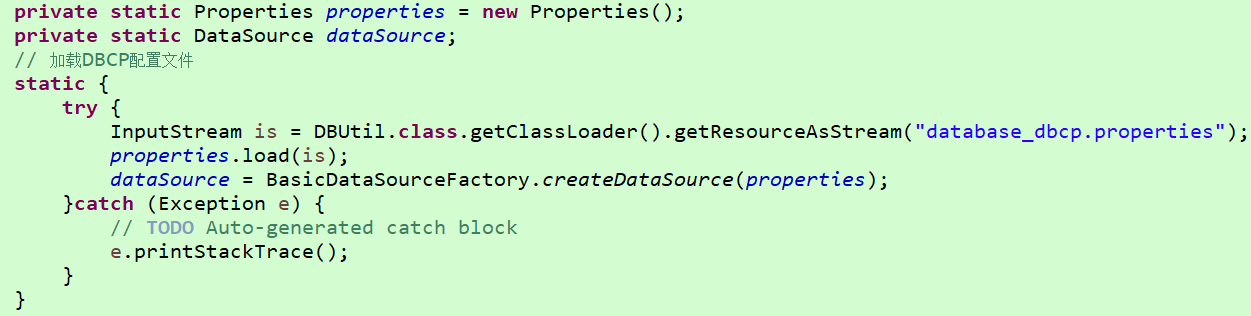
## 2、注意：

常用类关闭了是不等于null的，可以用isclosed判断

实体类最好实现serializable接口，支持序列化机制(将对象写入byte流里，并且可以从其他地方把该对象读出来，在网络进行传输) 定义属性解决版本之间序列化问题 ctrl 1解决黄异常会直接生成序列化

|  |
| --- |
| public int executeUpdate(String sql,Object... prams){  int a=0;  PreparedStatement pst=null;  try {  pst=c.prepareStatement(sql);  for (int i = 0; i < prams.length; i++) {  pst.setObject(i+1, prams[i]);  }  a=pst.executeUpdate();  } catch (SQLException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return a;  } |

|  |
| --- |
| public ResultSet executeQuery(String sql,Object... prams){  ResultSet rs=null;  PreparedStatement pst=null;  try {  pst=c.prepareStatement(sql);  for (int i = 0; i < prams.length; i++) {  pst.setObject(i+1, prams[i]);  }  System.out.println(pst.toString());  rs=pst.executeQuery();  } catch (SQLException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return rs;  } |



# 集合框架

## 1、简介

集合框架是为表示和操作集合规定的一种统一的标准体系结构包括三大块内容：

collection 可重复无序

set接口继承Collection接口， 存储一组不可重复、无序的对象；

List接口继承Collection接口， 存储一组可重复、有序的对象；

Map接口存储一组成对的键值对象，key不要求有序，不允许重复。value不要求有序，但允许重复

Iterator接口是负责定义访问和遍历元素的接口

## 2、list

### ArrayList

ArrayList<泛型> 名字=new ArrayList<泛型>(); 重复，有序 遍历元素随机访问效率高

boolean add(Object o) 在列表的末尾顺序添加元素，起始索引位置从0开始

void add(int index,Object o) 在指定的索引位置添加元素。内容直接会往后退一位

int size() 返回列表中的元素个数

Object get(int index) 返回指定索引位置处的元素。取出的元素是Object类型，使用前需要进行强制类型转换

boolean contains(Object o) 判断列表中是否存在指定元素

boolean remove(Object o) 从列表中删除元素

Object remove(int index) 从列表中删除指定位置元素，起始索引位置从0开始

### LinkedList

LinkedList<泛型> 名字 = new LinkedList<泛型>() 重复，有序 增删元素效率高

void addFirst(Object o) 在列表的首部添加元素

void addLast(Object o) 在列表的末尾添加元素

Object getFirst() 返回列表中的第一个元素

Object getLast() 返回列表中的最后一个元素

Object removeFirst() 删除并返回列表中的第一个元素

Object removeLast() 删除并返回列表中的最后一个元素

Remove 方法默认删第一个数据(linkedlist独特的)

Vector和array都是采用数组方式存储数据但是Vector为线程安全

### 提升

s.indexOf("a") a在哪个位置

set(1,”a”) 将位置1的内容替换成a

Collections.sort(); 对集合进行排序 只能为lsit

## 2、Set

不能重复(重复后一个不添加进去)，无序，最常用是测试归属性，可以很容易用contains询问某个对象是否在set中 使用散列的速度都特别快，**去重复，过滤，随机选出，redis里set有随机弹出几个数据的方法**

HashSet<String> (散列函数storage元素)

TreeSet 按照比较结果升序保存对象

LinkedHashSet 按照插入顺序保存对象,也使用了散列

s1.containsAll(s) s1如果包含s里所有内容则返回true

## 3、Map: 查询很快

HashMap<String,Dog> dog=new HashMap(); 键不可重复，如果重复最后一个将替代前一个，无序

TreeMap 保证了映射按照升序排列key 所以我们必须保证所传的建必须能排序

LinkedHashMap 按照插入顺序保存健(保留了hashmap的查询速度)

map遍历只能单一遍历建或者值 先用set或collection接口接收集合在迭代器遍历

Object put(Object key, Object val) 以“键-值对”的方式进行存储

Object get (Object key) 根据键返回相关联的值，如果不存在指定的键，返回null

Object remove (Object key) 删除由指定的键映射的“键-值对”

int size() 返回元素个数

Set keySet () 返回键的集合

Collection values () 返回值的集合

boolean containsKey (Object key) 如果存在由指定的键映射的“键-值对”，返回true

Hashtable与HashMap的区别

HashMap不是线程安全的，HashTable是线程安全。

HashMap允许空（null）的键和值（key），HashTable则不允许。

HashMap性能优于Hashtable。

## 4、Iterator迭代

Iterator接口: 为集合而生，专门实现集合遍历，它隐藏了各个集合实现类的内部细节；

boolean hasNext()； 判断是否存在另一个可访问的元素

Object next(); 返回要访问的下一个元素，创建迭代器没用泛型的话这个需要强转

remove() 删除当前元素,迭代的那个集合 那么集合里的这个元素就没了

例如：Iterator it1=dogList.iterator(); Dog d=(Dog) it1.next();

ListIterator(1) 用于list 双向移动,而且可以获取前一个后一个索引,set替换,可以传参表示从下标1开始迭代

相对于iterator多了这4个方法 和set

a.previousIndex();

a.nextIndex();

a.hasPrevious();

a.previous();

一般a.next() 后 在 a.previousIndex(); 代表的为当前元素在集合里的下标

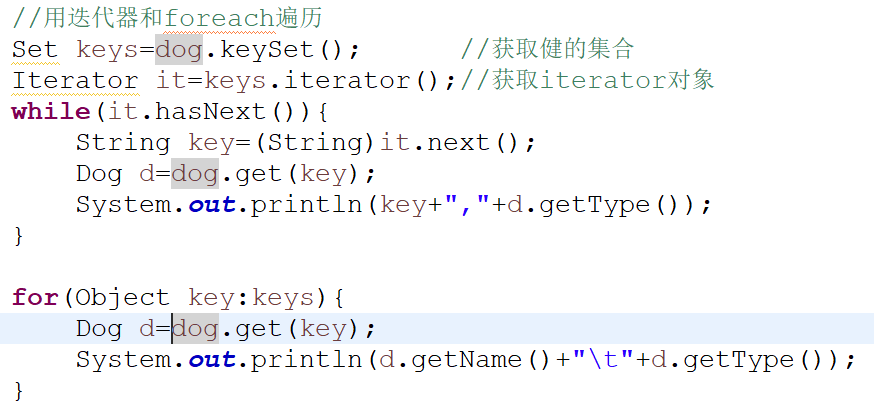
只是循环的话用foreach, 循环并且打算删除 用iterator,list打算set和删除用listiterator.

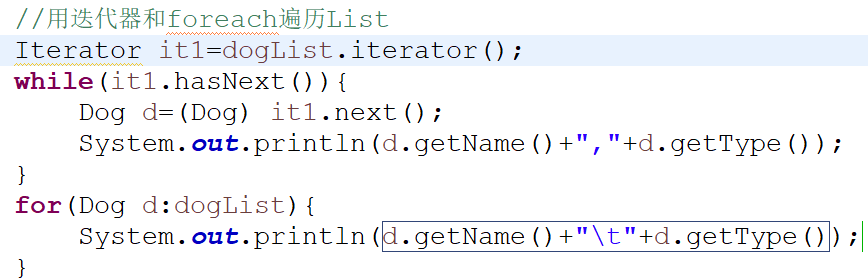
## 5、数组和集合的主要区别：

数组可以存储基本数据类型对象，而集合中只能存储对象(可以以包装类形式存储基本数据类型)

jdk有Arrays类专门用来操作数组，collections专门用来操作集合

用isEmpty()判断集合里是否有值





# 异常

## 简介

异常就是在程序运行过程中所发生的不正常事件

所有异常类型都是Throwable类的子类 Error Exception

运行时异常包括RuntimeException及其所有子类不要求程序必须对它们进行处理

非运行时异常必须捕获或者声明

## 各种情况

正常运行catch被忽略

try语句执行过程中遇到异常，而抛出的异常在catch里没有被声明，那么程序立刻退出

try语句执行过程中遇到异常，与catch里声明的异常相匹配try其余代码忽略

try结构体catch和finally可选其一，但至少出现一个。

## 其他

catch里面加入用户自定义的处理信息或者调用异常对象的方法来输出异常信息：

e.printStackTrace:输出异常的堆栈信息。

e.getMessage:返回异常信息描述字符串

finally无论如何都执行，不执行的唯一情况在catch里面写system.exit(1)退出虚拟机

如果catch里有return执行顺序是先执行finally在返回执行return

在使用多重catch时顺序一定要是从子类到父类要不会报错

## Throws与throw 这里看一下

Throws ：声明异常，如果调用者不打算处理该异常，则可以继续throws让上一级处理，main方法声明将会交给虚拟机处理。

throw： 手动引发一个异常 throw new Exception(); 括号里面没东西，异常也没东西

用于在程序中抛出异常 声明方法内抛出的异常

抛出一个异常对象（1） 跟异常类（多个）

位于方法内部 必须跟在方法参数列表后

手动抛出 非运行异常和exception时 必须throws或try否则会报错，

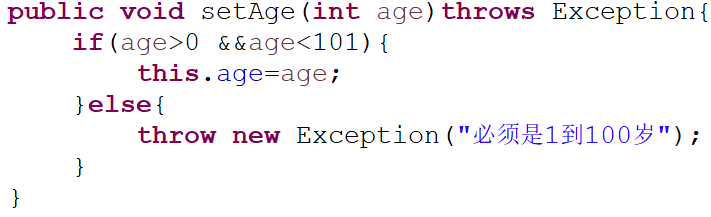
## java自定义异常

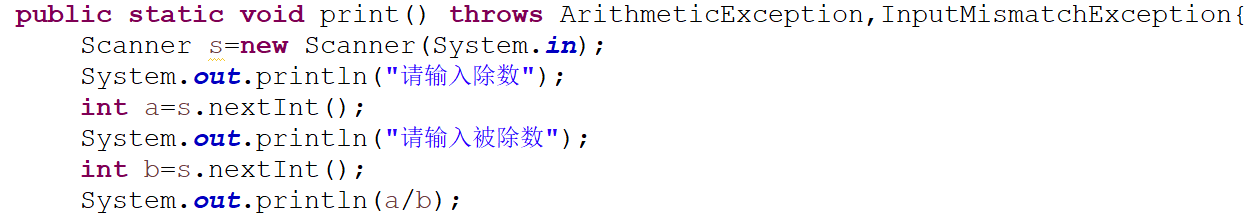
1. 定义一个类继承Throwable或其子类
2. 添加构造方法

public MyException(){}

public MyException (String name) { super(name); } 异常就有名字了

然后按照其他异常照常抛出就行了





# 线程

## 1、简介

进程：执行中的程序

线程：每个进程中间同时运行了多个线程用来完成不同工作，则称为多线程

好处：充分利用cpu资源，带来良好的用户体验。

原理：分成多个时间片，分配给不同线程，执行完之后就让出使用权给下一个持有时间片的线程使用， 时间片可能连续分配给同一个线程

## 2、使用方式

Thread.currentThread() 调用当前线程引用

调用start和run方法区别：多个线程同时启动状态，线程是交替执行的，而调用run方法是从上刀下执行；

### 1、通过继承Thread类来创建线程

public class MyThread extends Thread{//重写run方法}

如果想赋值线程名的话就写构造方法传个name

public MyThread (String name){ super(name); }

### 2、实现Runnable接口创建线程

public class Demo3 implements Runnable { public void run() {} }

Demo3 d=new Demo3(); Thread t=new Thread(d,[线程名]);

## 3、线程的状态：

新建状态：线程被创建之后便处于新建状态

就绪状态：新建状态的线程调用start进入就绪状态，阻塞状态解除阻塞之后进入就绪状态

运行状态：处于就绪状态的线程获取cpu资源就变成运行

阻塞状态：正在运行的线程因为一些原因让出cpu资源

终止状态：正常终止、强制终止：stop、异常终止

## 4、线程的扩展

线程的优先级：最大是10最小是1

t.setPriority(); 设置线程优先级

t.getPriority(); 返回线程优先级

线程的休眠：Thread.sleep();休眠以毫秒为单位

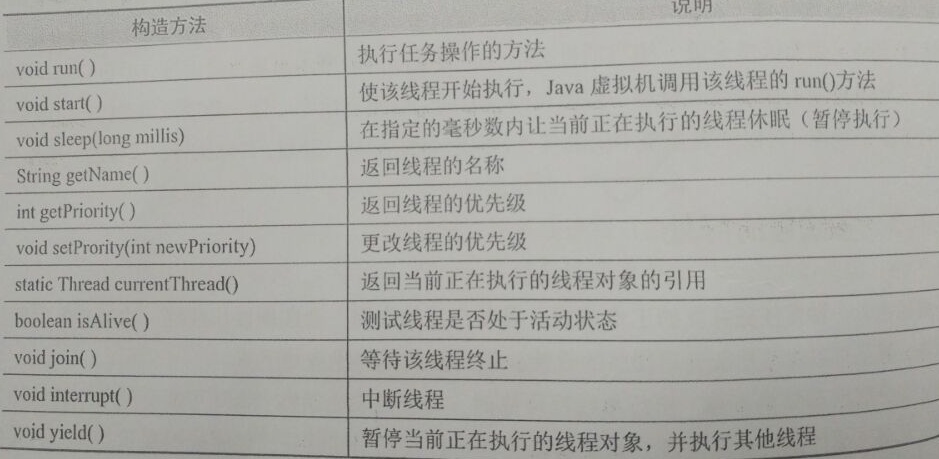
线程的强制运行：t.join() 等t线程执行完毕在执行往下继续执行

线程的礼让：Thread.yield() 让出cpu资源，然后在共同继续抢

线程同步的实现：synchronized方法

public synchronized void sale(){}把需要锁住的内容写到方法里然后在run方法里调用

synchronized(this){需要同步的代码}



## 5、重点：

主线程的引用通过Thread t=Thread.currentThread();获取当前线程的引用传给调用者

Thread t=new Thread();只是普通的一个线程对象没什么意义，但也能用里面的方法，需要传值才行。

当线程进入同步方法的时候，会获得同步方法所属对象的锁，一旦获得对象锁，则其他线程不能再执行被锁对象的其他任何同步方法。只有在同步方法执行完毕之后释放了锁，其他线程才能继续执行。

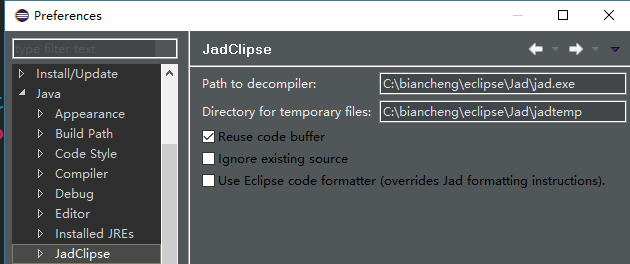


# eclipse反编译工具



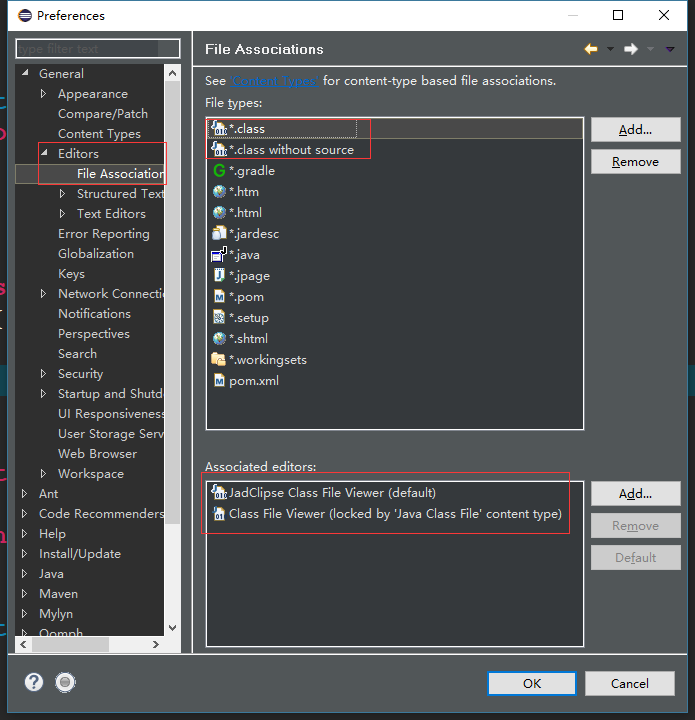
第二个放到eclipse plugins 第一个新建一个Jad文件夹解压到里面在新建一个





确保class 和class without source 都是jadclipse (defaulet) 如果没有就点击旁边add添加

和设置default



# log4j

1. 导入jar包
2. src导入配置文件，然后就能用了，会自动生成日志txt

private static Logger logger=Logger.getLogger(当前类名.class.getName());

然后logger.debug();

会自动创建jbit.log文件(异常信息文件)刷新一下就成

xml在所需的东西xml文件夹有案例

# Io流

## 简介

1. 流是指一连串流动的字符，是以先进先出的方式发送信息的通道，程序和数据源之间是通过流联系起来的。
2. 流可以分为输入流和输出流，也可以分为字节流和字符流
3. FileInputStream和FileOutputStream以字节流的方式读写文本文件
4. BufferedReader和BufferedWriter以字符流的方式读写文本文件，效率高

## 2、Flie类用于访问文件或目录的属性 file类的常用方法

boolean exists( ) 判断文件或目录是否存在

boolean isFile( ) 判断是否是文件

boolean isDirectory( ) 判断是否是目录

String getPath( ) 返回此对象表示的文件的相对路径名

String getAbsolutePath( ) 返回此对象表示的文件的绝对路径名

String getName( ) 返回此对象表示的文件或目录的名称

boolean delete( ) 删除此对象指定的文件或目录

boolean createNewFile( ) 创建名称的空文件，不创建文件夹

long length() 返回文件的长度，单位为字节, 如果文件不存在，则返回 0L

## 3、字节读：3个read方法

1、while((num=fis.read())!=-1){ System.out.println((char)num); } 返回0-255整数

2、byte [] b=new byte[fis.available()];

fis.read(b);

String s=new String(b);

System.out.println(s);

3、num=fis.read(a,1,2); 从txt读取2个字节的文档添加到a数组1下标开始

Num返回的是实际读取的字节数

## 4、字节写：3个write方法

1、for (int i = 0; i < words.length; i++) { f.write(words[i]); } 单个写入

2、String a="sdsasa\n撒旦";

fis.write(a.getBytes()); 写入数组里的所有字节

3、f.write(words, 4, 2);//从数组第4个字节往后读取两个字节

如果没有文件会自动创建，能写入中文，但是文本用字符流操作效率高

每次都是覆盖文件 多了一个构造，true就不覆盖了

字节流参数传入的都是全路径

## 字符读：FileReader 3个read方法

### FileReader：

第一个同上字节读取

第二个为char数组

第三个也一样只不过是char数组

New FileReader(“sasd/sad.text”)

### BufferedReader (fileReader)：

String a;

while((a=br.readLine())!=null){

System.out.println(a);}

## 6、字符写：FileWriter

### FileWriter：

f.write("123"); 添加123

f.write("123",1,1); 从2开始输出一个长度的内容

### BufferedWriter：

bw.write("大家好");

bw.newLine();

bw.write("我是马志凯");

bw.flush();清空流

## 注意

close() 记得关闭流

int available() 返回流的估计字节数

在读写文件需要对内容按行处理，比如比较特定字符，处理某一行数据的时候一般会选择字符流。只是读写文件，和文件内容无关的，一般选择字节流。

## 二进制文件的读写

//创建对象

fis=new FileInputStream("hancheng1.jpg");

dis=new DataInputStream(fis);

fos=new FileOutputStream("hancheng2.jpg");

dos=new DataOutputStream(fos);

//读取图片1存储到a数组

byte[] a=new byte[fis.available()];

dis.read(a);

//把a数组内容写到图片2

dos.write(a);

或者 while ((int b=dis.read())!=-1) {

dos.write(b);

## 对象序列化

将对象写入byte流里，并且可以从其他地方把该对象读出来

1. Serializable接口 实现序列化
2. 对象输入\输出流 实现持久化，保存到文件里

ObjectInputStream(fis) (类型)ois.readObject() 用一个对象接收并且前面得强转

ObjectOutputStream(fos) oos.writeObject(对象);

写多个对象到一个里面就写多个oos.writeObject 读的话也是多个ois.readObject 是按照写的顺序一个一个读的，而且因为是Object类型所以集合也能写进去

3、transient关键字

被它修饰的属性是不会被写到文件里去的 隐私数据

# 三层架构

表示层 业务逻辑层 数据访问层

数据访问层：公用的修改查询方法,普通sql方法接口和实现

业务逻辑层:调用访问层sql方法写业务逻辑

表示层：调用具体业务逻辑方法来进行显示

dao包 写basedao公共改查方法和普通访问数据库接口

dao.DaoImpl 写实现dao包 普通访问接口 访问层

service 写业务逻辑的接口

service.serviceImpl 写实现service包 业务逻辑的接口 逻辑层

entity包 写实体类

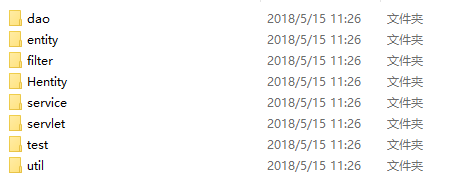
DBUtil 写util公共连接释放方法 和一些工具

filter 和 sevlet包

好处 代码复用 分离开发人员的关注 无损替换 降低了系统间的依赖

关于user增删改差写到一个userDao里 一个页面的增删改查(有可能有user 也有商品表)都写到一个controller里面 每个页面都多串一个method参数

Object... 名字 这样的参数只能有一个并且只能放在最后，在方法里用数组来接收 Integer... 都可以



# 包装类和Object的方法

## **1、包装类**

boolean byte short integer long float double character

## 2、自动装箱与拆箱

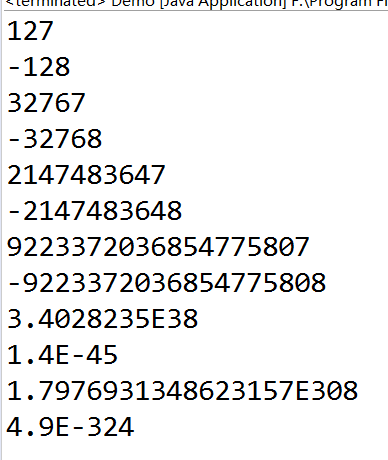
基本数据类型转换封装类叫做装箱，反之为拆箱；

## 3、数值的基本类型

最大值： e38意思是小数点在往后38位

最小值: 1.4e -45意思是小数点在往前45位 然后就是0.000什么

1.4 x 10的-45次方 等于1除10的2次方 0.1\*0.1



equals 默认比较的内存地址 String重写了这个方法所以是比较的值

hashCode 返回该对象的哈西码值 将对象的内部地址转换成一个整数

getClass 返回此Object的运行时类在getname获取class具体在哪个包下，哪个类型

toString 返回该对象的字符串表示

notifyAll 唤醒所有等待的线程

wait 让调用次方法的线程等待(并且释放了锁)，让调用他的线程进入暂停状态并释放锁，除非唤醒或超过时间。

wait(long ) 调用唤醒或超过指定时间则不等待

# 日期类和格式化

## Data：

Date d1=new Date(); d1.toLocaleString() 输出时间

d1.after(d2) 测试d1是否在d2日期之后

d1.before(d2) 测试d1是否在d2日期之前

## SimpleDateFormat

SimpleDateFormat s = new SimpleDateFormat("YYYYMMdd hh:mm:ss");



String ss=s.format(d1); 根据格式转换把date转换成string类型

Date parse = new SimpleDateFormat("yyyy-MM-dd").parse(d); parse方法将符合这个格式的字符串转化为date类型

## 当前时间+30天结果

|  |
| --- |
| Calendar now = Calendar.*getInstance*(); now.add(Calendar.***DAY\_OF\_MONTH***, +30); String endDate = **new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**).format(now.getTime()); System.***out***.println(endDate); |

# 泛型

## 1、泛型是什么

泛型是将类型参数化

## 2、泛型的作用

提高类型的安全并能简化类型转换的过程，在转换过程中，所有类型转换都是自动隐式的；

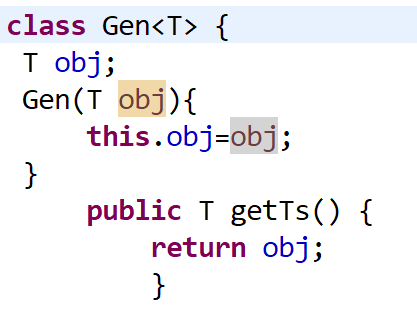
把运行期的错误提前到了编译期

如果没有泛型还得自己强转，而且不安全有可能出现类型转换错误。

## 3、泛型的使用

不确定传入的数据是什么类型，传入之后就都必须使用那个类型的时候用泛型

泛型定义了A类 那么只要是A类或其子类都可以添加



不确定传入的数据是什么类型但是知道在哪个范围就用泛型边界，并且想用范围里免得方法，要不他不知道你具体是什么类型所以没有相应的方法

|  |
| --- |
| **public class** Test<T **extends** Number> {  T [] **ts**;  Test(T [] ts){  **this**.**ts**=ts;  }  **public int** arerage(){  **double** sum = 0;  **for** (**int** i = 0; i <**ts**.**length** ; i++) {  *//如果不加泛型边界这一行就会报错* sum+=**ts**[i].doubleValue();  }  **return** (**int**) (sum/**ts**.**length**);  } ｝ |

# 反射

## 简介

反射就是把java类中的各种成分通过class文件映射成一个个的Java对象，在运行状态中，对于任意一个类，都能够知道这个类的的所有属性和方法,对于任意一个对象,都能够调用它的任意方法和属性

例如：一个类有：成员变量、方法、构造方法、包等等信息，利用反射技术可以对一个类进行解剖，把个个组成部分映射成一个个对象。

## 常用方式

class.forName是获取class对象引用的一种方法，传入全类名

你想在运行时使用类型信息，就必须获得class对象引用 用forName或.class 因为你不需要为了获得class引用而持有该类型的对象 如果拥有了对象则可以调用getClass来获取引用了

类名.class 不做类的初始化工作返回class对象

forName 做类的静态初始化(就是执行静态块)，返回Class的对象

开发的话用forName 因为他传入的为字符串，可以把字符串配置到配置文件中

只forName会执行静态块和static User u = new User()这种new出来的属性, 不执行构造方法

而class不会做这些，他必须newInstance实例化了 才会执行静态和构造

## 常用的一些方法

getName 全限定类名 getSimpleName getCanonicalName 不包含包名,全限定类名

isInterface 是否表示接口

getInterfaces 返回class对像所包含的接口组成的class数组

getSuperclass 返回父类

newInstance 返回该class的object引用 必须带有默认的构造器,不能为接口和抽象类

普通类类型可以被重新赋值为任何其他class对象,而泛型类只能赋值声明的类型 Class<Initable>

如果想稍微放松一下限制那么可以

Class<? extends Number> 本身和子类都行

Class<? super b> 本身和父类都可以

在用class创建的object之后如果要转换先做检查 if( o instanceof Person1 ){ Person1 p = (Person1)o }

不用获取构造方法实例化(就是直接class.newinstan)的话必须有无参构造

## 通过反射获取构造方法

getConstructors 返回该类的所有public的构造方法

getDeclaredConstructors 返回所有构造，不包括private

getConstructor 获取一个,如果想获取String,int这个构造，传对应的class对象

Constructor con = c.getConstructor(); c为该对象的class引用

Object o = con.newInstance();

Constructor c = c.getDeclaredConstructor(String.class,int.class); 加上Declared可以获取所有不包括私有

c.setAccessible(true); 取消访问检查,这样才能获取私有

Object o = c.newInstance("的",2);

Person1 p = (Person1)o;

私有得加上getDeclaredConstructor并调用setAccessible方法，多参得传class对象并且newInstance时传入参数

## 通过反射获取成员变量

getFields 返回该类的public成员变量

getDeclaredFields 返回所有成员变量

getField("b") 获取单个成员变量

Object obj = con.newInstance();

Field b = c.getDeclaredField("b"); 加上这个可以获取私有的

b.setAccessible(true); 取消访问检查

//将obj对象的b属性设置为指定的值

b.set(obj, 15);

b.get(obj); 获取属性值

私有得加上getDeclaredField并调用setAccessible方法

## 6、通过反射获取成员方法

getMethods 返回该类的public成员方法(包括父类的公共方法)

getDeclaredMethods 返回该类的所有成员方法

getMethod 获取单个成员方法

加上这个可以获取私有方法 参数1 :方法名， 参数2 :参数的class对象

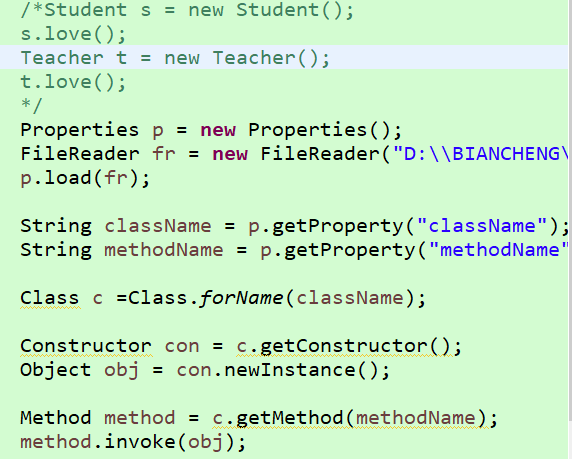
Object obj = con.newInstance();

Method m1 = c.getDeclaredMethod("show");

m1.setAccessible(true); 取消访问检查

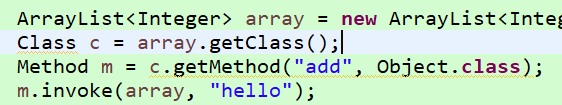
调用obj对象的m1方法，第二个参数为方法的参数

m1.invoke(obj);



## 7、通过反射越过泛型检查：

list里add方法传入的实际为object类型，泛型只是给编译器看的，所以可以通过反射来实现向Integer泛型集合添加字符串



# 动态代理

动态代理：在程序运行过程中产生的这个对象。所以，动态代理其实就是通过反射来生成一个代理

在Java中java.lang.reflect包下提供了一个Proxy类和一个InvocationHandler接口，通过使用这个类和接口就可以生成动态代理对象。JDK提供的代理只能针对接口做代理。我们有更强大的代理cglib

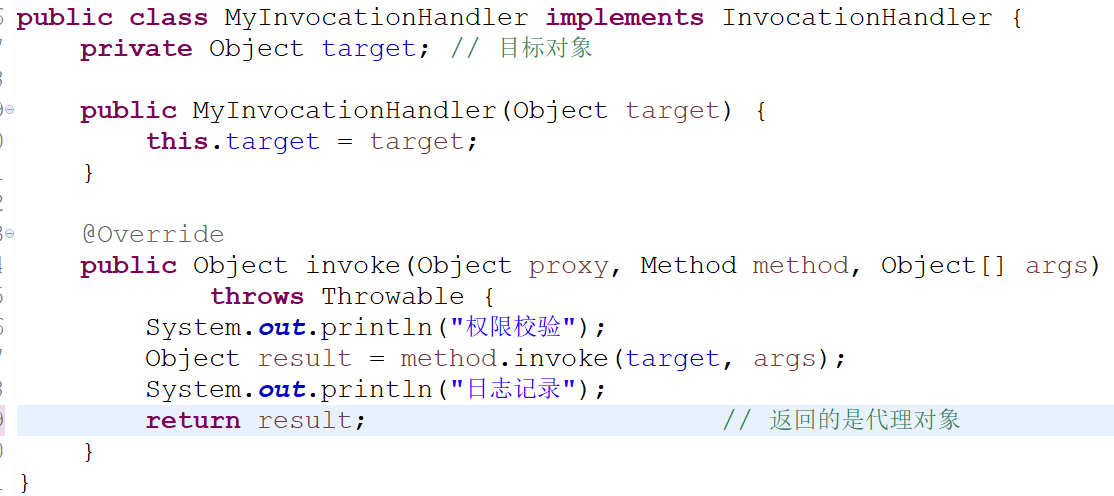
Proxy类中的方法创建动态代理类对象

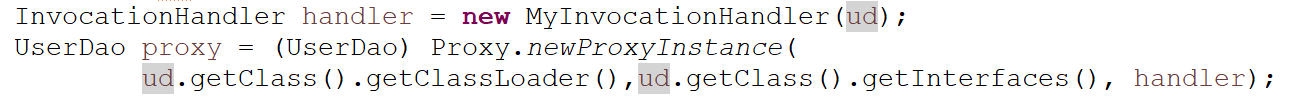
public static Object newProxyInstance(ClassLoader loader,Class<?>[] interfaces,InvocationHandler h)

最终会调用InvocationHandler的方法

InvocationHandler

Object invoke(Object proxy,Method method,Object[] args)

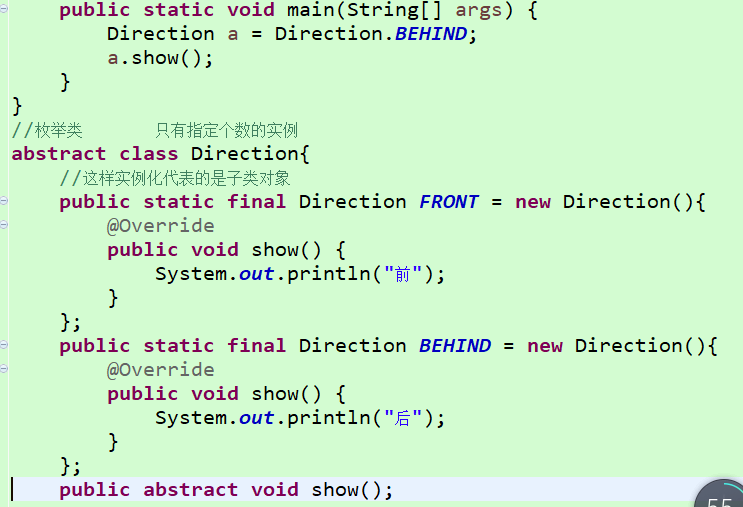




# 枚举

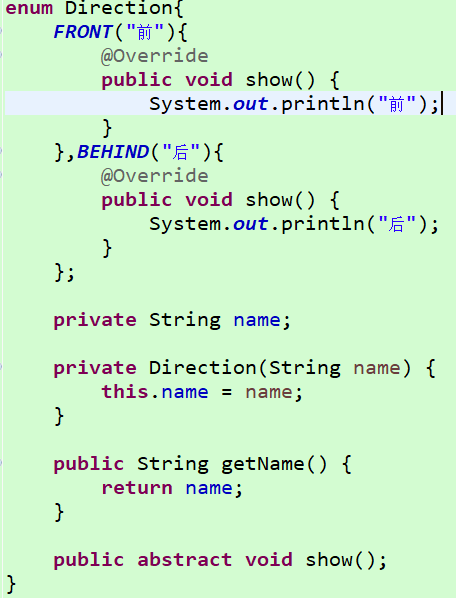
枚举是指将变量的值一一列出来，变量的值只限于列举出的值的范围：例如一周7天

## 原始枚举



## 枚举类

1. 需要几个实例在enum里写名字就行了
2. 如果构造需要带参的话，那就(”“)传参数并在下面写构造和属性等
3. 需要抽象方法那就加个{}重写就成了呗



## 扩展

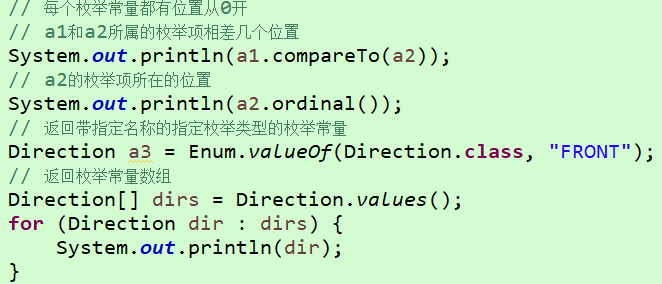
枚举类重写了toString方法返回的是枚举常量名称、

枚举类的第一行必须是枚举项

枚举类可以有构造器，但必须是private

可以在switch用 switch (对象名) case BEHIND:

A a = new A(){}; 这个括号里就代表的是其子类对象



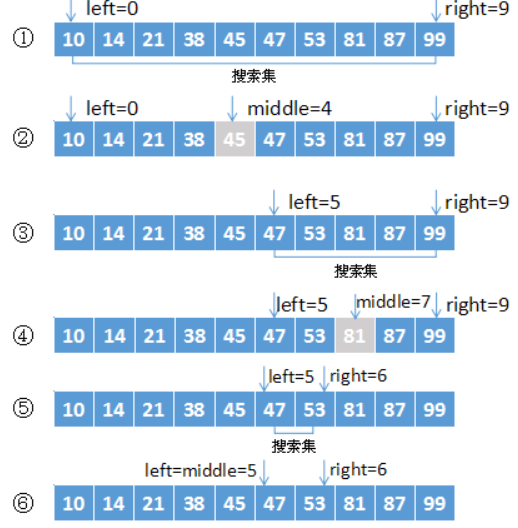
返回带指定名称那个 同时会执行其他比如back的构造方法

# 二分查找算法

二分查找也称折半查找（Binary Search），它是一种效率较高的查找方法。但是，折半查找要求线性表必须采用顺序存储结构，而且表中元素按关键字有序排列

将表中间位置记录的[关键字](https://baike.baidu.com/item/%E5%85%B3%E9%94%AE%E5%AD%97" \t "https://baike.baidu.com/item/%E4%BA%8C%E5%88%86%E6%9F%A5%E6%89%BE/_blank)与查找关键字比较，如果两者相等，则查找成功；否则利用中间位置[记录](https://baike.baidu.com/item/%E8%AE%B0%E5%BD%95/1837758" \t "https://baike.baidu.com/item/%E4%BA%8C%E5%88%86%E6%9F%A5%E6%89%BE/_blank)将表分成前、后两个子表，如果中间位置记录的关键字大于查找关键字，则进一步查找前一子表，否则进一步查找后一子表。重复以上过程，直到找到满足条件的[记录](https://baike.baidu.com/item/%E8%AE%B0%E5%BD%95/1837758" \t "https://baike.baidu.com/item/%E4%BA%8C%E5%88%86%E6%9F%A5%E6%89%BE/_blank)，使查找成功，或直到子表不存在为止，此时查找不成功

|  |
| --- |
| **public static int** binarySearch(**int**[] arr, **int** x) {  **int** low = 0;  **int** high = arr.**length**-1;  **while**(low <= high) {  **int** middle = (low + high)/2; //先找出中间  **if**(x == arr[middle]) { //如果是的话就返回位置  **return** middle;  }**else if**(x <arr[middle]) { //如果小于的话就让最大位置变成当前位置-1(在继续查找小的那半个数组)  high = middle - 1;  }**else** { //如果大于就让最小位置变成当前位置+1，这样的话(9+5） /2 正好为后半表的中间位置  low = middle + 1;  }  }  **return** -1;  }  *//递归实现二分查找* **public static int** binarySearch(**int**[] dataset,**int** data,**int** beginIndex,**int** endIndex){  **int** midIndex = (beginIndex+endIndex)/2;  **if**(data <dataset[beginIndex]||data>dataset[endIndex]||beginIndex>endIndex){  **return** -1;  }  **if**(data <dataset[midIndex]){  **return** *binarySearch*(dataset,data,beginIndex,midIndex-1);  }**else if**(data>dataset[midIndex]){  **return** *binarySearch*(dataset,data,midIndex+1,endIndex);  }**else** {  **return** midIndex;  }  }  **public static void** main(String[] args) {  **int**[] arr = { 6, 12, 33, 87, 90, 97, 108, 561 };  System.***out***.println(**"循环查找："** + (*binarySearch*(arr, 87) + 1));  System.***out***.println(**"递归查找"**+*binarySearch*(arr,3,87,arr.**length**-1));  } |



# 阶乘，递归函数

## 递归函数

函数自己调用自己的体现，就叫做函数的递归

把问题分解成为规模更小的、具有与原问题有着相同解法的问题。比如二分查找算法，就是不断地把问题的规模变小（变成原问题的一半），而新问题与原问题有着相同的解法。

|  |
| --- |
| **public static int** multiply(**int** num) {  **int** result;  **if** (num < 0) {  System.***out***.println(**"请输入大于0的数！"**);  **return** -1;  } **else if** (num == 0 || num == 1) {  **return** 1; 必须有一个明确的约束条件  } **else** {  result=*multiply*(num - 1) \* num; 每次递归都是为了让规模变小  }  **return** result; } |

程序在计算5的阶乘的时候，先执行递推，当n=1或者n=0的时候返回1，再回推将计算并返回。由此可以看出递归函数必须有**结束条件**。

### 什么时候使用递归

可以通过递归调用来缩小问题规模，且新问题与原问题有着相同的形式。   
存在一种简单情境，可以使递归在简单情境下退出。

****循环能干的事,递归都能干;递归能干的循环不一定能干**对于我们,能用循环解决的,尽量不适用递归.**

//获取5\*4\*3\*2\*1的结果

int result=1;

for (int i = 5; i >0; i--) {

result\*=i;

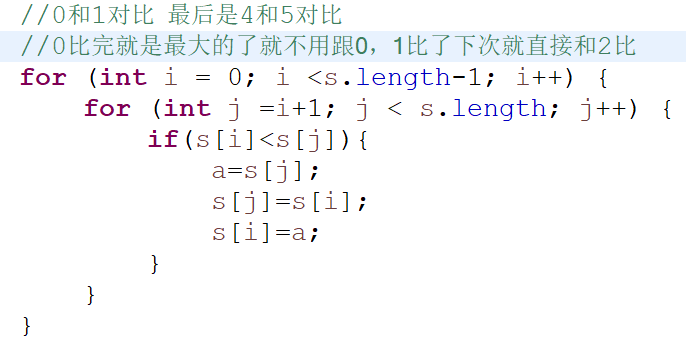
}

System.out.println(result);

}

//result=1;然后1\*5的递减for循环 就得到了结果

# 冒泡排序



# Java内存区域

java程序在运行时，需要在内存中分配空间，为了提高效率，就对空间进行了不同的区域划分，因为每一块区域都有特定的处理数据方式和内存管理方式。

栈 特别小的简单变量没必要存到堆如局部基本类型变量存储到栈，用完立即回收(方法执行完毕)

堆 存储new出来的东西，每个变量都有默认值，使用完毕变成了垃圾但是并没有立即回收，会在垃圾回收器空闲的时候回收

方法区 (面向对象部分讲)

# 内部类

内部类：把类定义在其他类的内部，这个类就成为内部类

内部类的访问特点：

1、可以直接访问外部类的成员，包括私有

2、外部类要想访问内部类必须创建对象

内部类的位置：

1、成员内部类

别的类访问： Outer.Inner inner = new Outer().new Inner();

加了private就不能被别的类访问了，可以加校验成功后才能访问内部》》》

静态内部类只能访问外部静态成员，里面可以写静态和非静态方法

Outer.Inner inner = new Outer.Inner();

一个为了数据安全性，一个为了方便访问数据

内部类局部、成员和外部类都有同一个变量那么用Outer.this.num访问最外部的，也可以new Outer().num

2、方法内为局部内部类

可以在局部里面创建内部类对象，通过对象调用内部类方法，来使用内部类功能

从内部类访问方法内局部的变量必须用final修饰，因为局部变量随着方法调用完

就消失了，而堆内存的内容并不会小时，所以加final修饰，既然是常量，你消失了，我在内存中

存储的事数据20，所以我还是有数据在使用

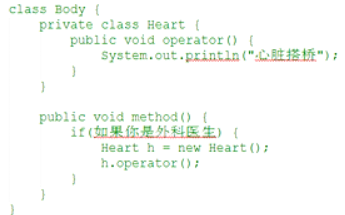
3、匿名内部类

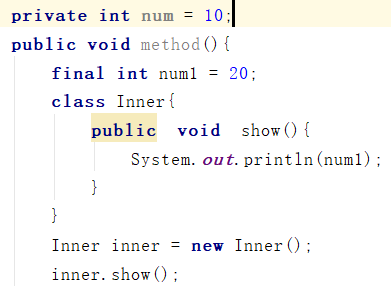
前提：存在一类(抽象类也行)或接口

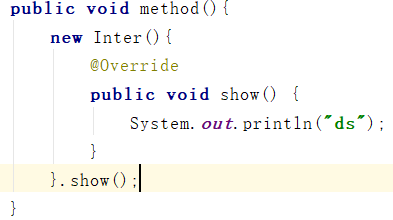
格式：new 类名(){ 重写的方法 }

本质是一个继承了该类或者实现了该接口的子类匿名对象，只能调一次方法

很多个方法会很麻烦，所以既然是子类对象可以赋值给父接口







# 重点

## string和stringbuffer方面

String是一个常量，是不可变的，所以对于每一次+=赋值都会创建一个新的对象， StringBuffer和StringBuilder都是可变的，当进行字符串拼接时采用append方 法，在原来的基础上进行追加，所以性能比String要高，又因为StringBuffer 是线程安全的而StringBuilder是线程非安全的，所以StringBuilder的效率高StringBuffer.

对于大数据量的字符串的拼接，采用StringBuffer,StringBuilder.

得用类型接收的用\*表示 []表示可选

String s=”Hello”; String s=new String(); String s=new String(“Hello”);

s.equalsIgnoreCase(String a); 忽略大小写做比较(忽视) \*\*\*\*

字符串1.concat(字符串2); 连接字符串(用字符串接收) \*\*\*\*

s.indexOf(); 查到返回下标否则返回-1 \*\*\*\*

s.lastIndexOf(); 同上，从后往前执行 \*\*\*\*

s.subString(1); 从下标为1开始提取 \*\*\*\*

s.subString(1,5); 1开始到5之前结束 \*\*\*\*

s.trim(); 过滤吊字符串前后多余的空格 \*\*\*\*

s.split(“ ”,[5]); 有空格就进行拆分,5限制返回数组的个数 \*\*\*\*

s.toCharArray 转换为char类型数组 \*\*\*\*

StringBuffer sb=new StringBuffer();

String s=sb.toString; 把sb转换为String \*\*\*\*

sb.append(); 可以将任何类型追加到字符串后

sb.insert(i,”,”); i的下标加一个逗号

String.valueOf(变量); 把变量转换为String \*\*\*\*

String.replace("a","b") b替换a方法需要用String接收

concat需要用String变量接收而append不需要；

System.arraycopy(a,0,c,0,3); 数组copy方法

a从0 copy 到c 从0 copy3个

s.toCharArray 转换为char类型数组

1. getBytes() 转换为byte类型数组

String.valueOf(变量); 把变量转换为String

int b=Integer.parseInt(a) String转int

String b1=String.valueOf(a1); int转String

## 2、求数字游戏

求数字游戏都可以这样：列求1234 1234/1000=千位 1234%1000=剩下3位

## 3、随机数

Math.random() 获取 0-1 之间随机数

规则： int num=(int)(Math.random()\*(想产生几个随机数)+从几开始) 例：

15-20： int num=(int)(Math.random()\*(6)+15);

Math.round(a)舍入函数

## 4、修饰符

static 成员变量 方法

final 类 属性 方法

abstract 类 方法

public 类 属性 方法

private 内部类 属性 方法

默认 类 属性 方法

protected 属性 方法

final修饰的属性不能被更改，方法不能被重写，类不能被继承，但是里面的属性是可以用的，必须在定义时指定初始值。

abstract类不能被实例化，方法没有内容必须重写

private修饰的方法属性不能被继承

## 5、If else

else if 条件一样只会输出第一个省略剩下的

## 6、构造方法

如果父类没有无参构造方法子类也不调用有参的就会报错；

继承构造函数是从祖先往下执行

子类可以直接用父类的属性和方法,super只是用在有相同方法名或属性名的时候用来区分

## 7、float

float类型后面的参数必须加f

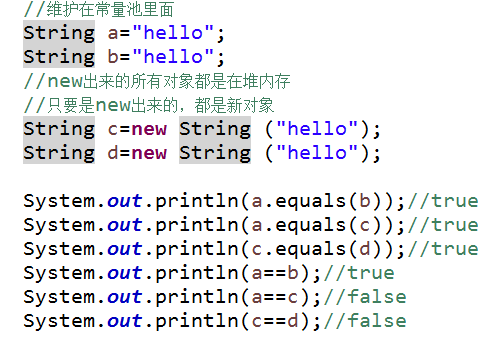
## 8、new == 和equals

通过new等建立的赋值是指向同一个，而八大数据类型赋值是纯赋值

基本类型赋值就是赋值 引用类型赋值是指向同一个引用

String==比较地址equals比较内容 引用类型==和equals比较的都是内存地址(以为string重写了equals)

引用类型赋值是引用乱指     但是用引用里面的内容是公共一起改的



## 9、环境变量

java mysql tomcat  都是path到bin  home到bin上一层 CATALINA\_HOME

## 10、异常链

异常链：两个或者多个不同的异常出现在同一个程序中，并且会发生嵌套抛出，我们称之为异常链。

## 死锁

案例项目有素材  
 死锁：两个或两个以上的线程被无限的阻塞，线程之间相互等待所需的资源。

## 12、泛型数组

泛型数组可以objs=(T[])new Object[10]; 这样创建

## 13、多线程synchronized

syn锁住的是所属对象 那单独的引用而不是 所属对象的其他引用也被锁住

## 14、集合

all.clear(); 移除元素所有元素(所有集合都有这个方法)

bb=a1.toArray(bb); 有参a1集合赋值给b数组返回的类型就是bb类型,无参返回Object,必须都是包装类

Collections.addAll(q, el); 参数1为集合，参数2为需要给集合添加的数据直接写或传集合都行 比引用.addall方便因为不仅可以传集合 添加数据aslist和这个 优先这个效率高(这个直接输1567就行了 另一个得addall(aslist 在1567)) 只适用于set，list

List<Integer> f=Arrays.asList(1,5,7,8); 像数组一样赋值，返回list 也可以传数组 返回的list不能增删(用于查和传数据)

aa.addAll(a); 把a集合里所有东西传入aa 大量数据这个效率高

反正就是能用Collections就用，不能的话就用addAll。

## 其他

Jar文件运行都是文本，可以在jar包目录打开cmd java -jar 名字.jar 运行

每个对象都只能满足某些请求，这些请求由对象的接口定义

开发程序要把对象当成服务提供者，程序向用户提供服务，将通过调用其对象提供的服务来实现

方法的参数传递是对象类型传递的就是为引用，方法内改变会影响实际值

两个用于高精度计算的类BigInteger BigDecimal 传字符串类型的数字

在使用任何引用之前必须为其指定一个对象

成员变量自动默认值，但是最好明确的对变量进行初始化

方法返回值为void的话return则只用于退出方法

静态变量只有一份存储空间，每个实例化对象都是公用静态属性的

静态块只会初始化一次重复调用不会重复执行

静态方法没啥用只是不创建对象的前提下调用

一旦加载类信息或实例化,静态块会立即执行，静态对象引用(必须有new User())会被初始化

每个文件必须有个同名类，只有同名类才可以写main方法

基本类型存储了实际的数值，而并非指向一个对象引用

String和Integer用equals如果内容一样会返回true因为大多数类库都重新实现了equals以便用来比较内容。

重新toString的话输出引用才不是地址 直接输出集合名就有结果不用循环测试了

toBinaryString（int i）toHexString（int i）toOctalString（int i）

返回int变量的二进制 十六进制 八进制字符串

10的-2次方等于1除10的2次方 0.1\*0.1

小转大自动转，大转小保证数据在小范围内否则丢失数据 Math.round(a)舍入函数

方法参数为普通double 那么传入char int float long等都会自动向上转型反之则强转

参数为普通对象那么可以传入其子类对象

float是比long大的，long可以直接赋值float,float类型赋值long需要强转

成员变量对象不初始化都默认为null

初始化顺序：先初始化静态的，父类构造，然后普通的，然后执行构造函数里的(静态的和普通的对象如果为User u;他是默认赋值null的 ，为new User(); 他才会实例化)

静态块只会初始化一次

toString需要的话必须重写否则输出的为地址

package 包名; 必须是第一句代码

import day05.\*; 包名.类名; 可以自己创建自己的工具类减少重复的程序代码

private修饰构造函数相应的构造函数(修饰无参，那么就不能用无参构造实例化拉) 如果不希望别人对类有访问权限可以把所有构造都弄成private 写一个静态方法返回当前对象 就跟get方法一样可以获取当前对象

private static Print1 p=new Print1();

public static Print1 getPrint1(){ return p; ｝ Print1 p=Print1.getPrint1();

始终只能创建它的一个对象,单例模式 。SqlSessionFactory就是单例模式

是在好多类都用同一个对象用这个， 在一个类里一直用一个就 直接加static就可以了如sqlSessionFactory

如果可以的话就在使用对象之前在初始化，可以节省空间 例如 返回list 如果rs.next在初始化list

可以直接类名.main进行测试 调用的为在被调用执行完后的main(也可以传参数)

直接继承会把父类所有方法在子类显示,用代理可以只提供某个子集

如果可以的话引用都用接口(有的类又有继承又有实现) list=new ArrayList 需要linked就改一下方法里的new ArrayList就行了 ，不用改返回值

ArrayList随机访问是固定时间，linked则需要在列表移动越靠近表尾的元素就越慢 ,而插入删除却比array快

栈 通常指后进先出的容器 最后加入的元素 第一个弹出栈 经常用于对表达式求值

可以自己写 p230 也有一个Stack类 stack.push() 增加到头部 stack.pop() 删除并返回删除的元素 peek()获取头部

插入到头部移除头部 后进先出

队列 先进先出的容器 常被用作将对象从程序某个区域传输到另一个区域的途径 ，linkedlist实现了Queue接口

Queue<Integer> s=new LinkedList<Integer>();

boolean s.offer() 将一个元素插入到尾部 peek()element() 返回头部,空时候返回null和异常

poll() remove() 移除并返回头部和异常

插入到尾部移除头部 先进先出

优先级队列 是声明下一个弹出元素是最需要的元素

PriorityQueue<Integer> p = new PriorityQueue<Integer>();

Random rand = new Random(); rand.nextInt(20); 0-20随机数

@Override注解意思是这个方法是重写父类的，如果方法不是重写的还声明这个注解方法就会报错

例跟for(u:user)这一行加个断点，一运行项目在当前行或控制台就会显示user大小和里面得值

Jee web开发 jse 桌面开发 jme嵌入式开发

# 基础

## Java基础介绍

1、 Java的特性

完全面向对象的语言。 可移植性：JVM实现了可移植性。

2、Java的工作方式

Java源文件（.Java格式结尾）——经过java编译器的编译——字节码文件（.class格式结尾）——被类装载器装载到Java虚拟机（JVM）——被JVM解释给操作系统——操作系统执行。

3、Java开发系统的构成

Java虚拟机（JVM）：指令集、寄存器、堆栈、垃圾处理器、方法区域组成。

JRE:Java Runtime Environment Java运行环境

JDK：java Development Kit java开发套件

如果只想跑一个java程序安装jre即可。开发java程序就需要安装jdk

## 搭建基础环境

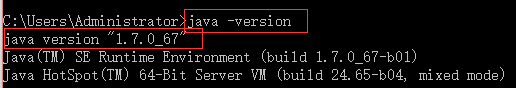
1、安装JDK 建议不要有中文路径

2、配置JDK环境变量

PATH D:\BIANCHENG\JAVA\bin;

3、验证环境搭建成功

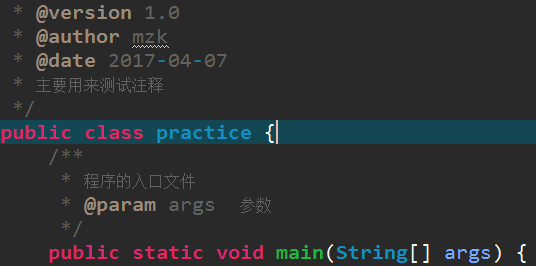
java javac java -version



## 生成api

在class上注释作者、版本、时间等等。 在方法上注释参数。

project—Generate：configure选择javadoc在bin目录里 选择要生成api的文件点击next，起个名字然后在next然后结束 会自动在项目下生成一个doc文档 index html 就是api



## Eclipse

File——switch workspace——重新选择工作空间

右上角java——reset重置界面

修改字体：window——preferences——General

Apprarance——Colore and For——Basic

Text Font

重命名 f2

背景颜色：window——preferences——General——editors——text editors——background color。

window preferences General Appearance 里面选Dark换好看黑色

显示行：编辑区 左侧右击show line number

点击需要的文件右键第二个就把其他class全都关了。

window -show view-navigator 打开导航器 里面可以看到class文件 package explorer 打开包资源管理器

在window preferences里查types 没新开一个工作空间就把所有都弄成utf-8

代码按ctrl全选全选然后 右键 source formatActive 格式化就是变得好看，可读性强

import file system 把需要导入文件直接复制到文件里

项目里的jre system library删除解决方案：右键项目 properties javaBuildPath libraries addlibraries

导出jar包右键包 export java jar file browse选位置就好了

## 八大数据类型

整型

bit(位)是最小数据单位，只能是0或1，

八个二进制位构成一个字节，

一个字节可以存储一个英文和半个汉字

64位表示cpu一次能处理的最大位数

byte 1字节 8位 -128~127

short 2字节 16位 -32768~32767

int 4字节 32位 正负20亿多一点

Long 8字节 64位 特别大 int容量装不下的时候才用long

求整型最大最小多少的方法：Integer.MAX\_VALUE 别的都是原字母

字符型 char 2字节 16位

浮点型

float 4字节 单精度 32位

double 8字节 双精度 64位

布尔型

boolean true 真 false假 一字节

## 命名规范

包名：全部小写单词

类名和接口名：多个组成的单词首字母大写 TestDemo

方法和变量名：驼峰式命名规则 除第一个单词外，所有单词首字母大写 getUserName

常量名：所有的单词字母全部大写，单词之间用下划线分割

## 运算符

### 1、基本算数运算符

+ - \* / %

++ 自加

+- 自减

+= -= \*= /= %=

### 2、模运算符

%是求两个整数相除的余数，

结果的正负取决于被除数。

10%20商0余10

### 算数赋值运算符

i=i+5 = i+=5

### 自增自减运算符

i++ ++i

i++是先用后加 ++i是先加后用

### 比较运算符

返回值是boolean 一般用于判断语句当中

== != >< <= >=

结果全是boolean类型；

### 逻辑运算符

& 逻辑与 |逻辑或 ！逻辑非运算

&&短路与 ||短路或

与 运算符：所有条件都为真，则结果真，有一个假，结果就是假；

或 运算符：所有条件都是假，则结果假，有一个真，结果就是真；

非 运算符：取反

b=condition1&condition2； //先求1，2的值，然后进行判断

b=condition1&&condition2；//首先对1进行判断，如果为真，在判断2，如果1是假的，则不再对2进行判断。

### 三元运算符

布尔表达式？值1：值2

k=i>=j?i:j;//如果i>=j，？是判断，是的话把i赋值给k不是的话：把j赋值给k

## If

结构体1：用在有一种情况的输出下

if（条件表达式）｛

条件成立的时候执行的代码段

｝

结构体2：用在两种情况下输出

if（条件表达式）｛

条件成立执行的内容

｝else｛

条件不成立执行的内容

｝

结构体3：用在多种情况下的判断

if(判断语句){

条件成立执行的代码段

}else if(判断语句){

该判断语句成立执行的代码段

}else if(判断语句){

该判断语句成立执行的代码段

}else if(判断语句){

该判断语句成立执行的代码段

}else{

都不成立执行的代码段

}

if else if else if else对所有else if有效

if if if else else对最后一个if有效

## Switch

switch(表达式){

case 值：

表达式==值，输出的语句块

break

case 值2：

表达式==值2，输出的语句块

break;

default：

默认语句块，都不成立的执行的代码段

break;

}

表达式：

byte 、char、int、short

从jdk7.0开始 String

注意点：

1、key对应value必须唯一

2、默认语句不限制位置，但是建议放置到最后

3、每句结束 break语句，如果没有继续向下执行

if主要用于一个判断区间范围应用

switch:主要用于具体的值信息

## Scanner

Scanner:获取键盘输入的值信息

常用：

1、获取字符串

对象名. next();

一定要读取到有效的数据才会输出，无法获取空格的输入。空格之后的输入都读取不了。

对象名. nextLine();

以enter键为结束，可以获得空格输入。

2、获取整数

对象名. nextInt() 无法获得空格输入

3、获取浮点数 nextDouble

1. 实例化Scanner 为获取键盘输入的信息做准备（需要引入包） import java.util.Scanner;
2. 结束关闭Scanner对象 input.close

## 循环

### While

\* 结构体：

\* while(条件表达式){

\* 循环体

\* }

\* 条件表达式：布尔类型结果

注意点：

1、初始值

2、循环判断条件

3、循环变化变量变化规律

### do-while:

\* 结构体：

\* do{

\* 要循环的内容

\* }while(判断条件);

\* 执行步骤：

\* 1、执行循环内容

\* 2、判断循环条件

\* 如果条件成立，继续执行循环体

\* 如果不成立，结束

\* do-while和while区别：

\* do-while先执行循环体，无论条件成立与否，都会至少执行一次

\* while语句先判断，判断true执行循环体，为false直接结束

### for循环应用

\* for(表达式1;表达式2;表达式3){

\* 循环结构体

\* }

\* 解析：

\* 表达式1：初始化表达式

\* 表达式2：循环判断条件表达式

\* 表达式3：循环变量变化规律

\*

\* 执行步骤：

\* 1、先执行表达式1

\* 2、判断循环条件 表达式2

\* 如果成立true 执行循环内容

\* 如果为false 结束

\* 3、执行循环体

\* 4、执行表达式3

\* 5、循环执行2-4步骤

进入死循环常见方式

\* 1、while(true){

\* 2、for(;;){

5、break、continue 用法

\* break:

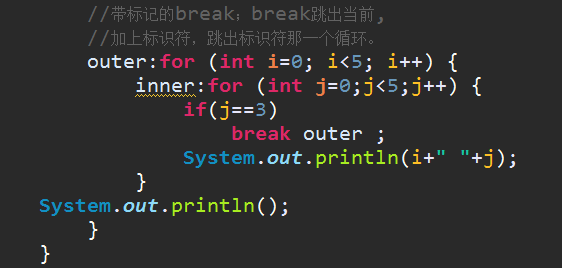
\* 跳出当前循环，不再执行剩下的循环

\* continue:

\* 跳出当次循环，继续执行剩下的循环

return：结束整个程序

for循环需要固定循环次数，while则可以不需要



带标记的break:

标识名：for(){

break 标识名;

## 函数：

\* 又叫做方法

\* 把功能独立的代码段封装成一个小程序，这种封装体现就叫做函数。

\* 1、功能独立的代码段进行封装

\* 2、提高代码的重用性

\* 3、只有被调用了才会执行

\* 4、要严格按照参数列表传值

2、函数的结构体

\* 函数结构体：

\* [修饰符] 返回值类型 方法名(参数列表){

\* 功能代码段；

\* //return 返回值类型；

\* }

会返回值类型：

基本数据类型，用return返回，最后可以syso输出方法，也可以用一个基本类型去接收在运行那个基本类型。

如果没有返回值类型用void代替，

不能通过System输出（直接调用方法就可以）

形参：主要在方法的参数列表，是一个变量信息，主要用来存传进来实际值信息

实参：用来传递给形参的具体值

## 数组

int[] a=new int [5]; 动态

a[1]=1;

a[2]=5;

......

int[] a={1,5....}; 静态

length：获取数组长度。

数组复制之后，两个引用指向同一个数组，不管是哪个引用修改了数组元素，对另一个引用来说，都是修改的。

要想没关系赋值就用for循环

Arrays.toString(a) 内容转化成String类型输出

Arrays.sort(arr)；对数组进行从小到大排序

常见异常：

\* 数组越界：

\* ArrayIndexOutOfBoundsException：越界数组的长度

\* 空指针异常：

\* NullPointerException

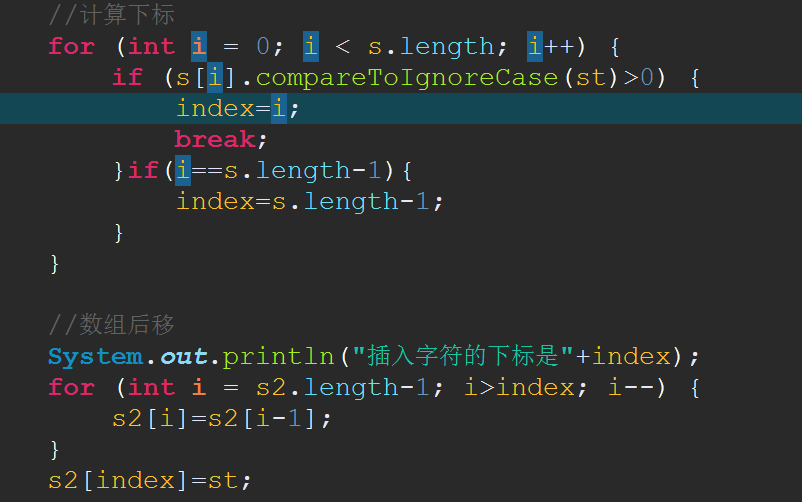
遍历数组：

for (int b : arr) {

System.out.print(b);

}

s[i].compareToIgnoreCase(这里写一个字符串) s[i]>后面的就返回1，小于-1，等于0



## For循环经典案例

