1. 标识符的定义
2. 类和接口：每个单词首字母要大写
3. 方法：首字母小写，其后用大写字母分割每个单词
4. 常量：全部大写，单词之间用下划线分割
5. 变量: 通常使用名词，首字母小写，其后大写字母分隔每个单词
6. 三元运算符

变量=条件 判断 ？ 表达式1： 表达式2

（解析）

如果符合条件判断则执行表达式1 ，否则执行表达式2

三、算术运算符（++，--）

++a 先进行自加在运算

1. - 先赋值运算在自加

四、位运算符（&，| ，^,~）

（1） << 左移 >> 右移 >>> 无符号位右移

（2） 位运算符首先要把数据转换成为二进制补码

（3）当两边是数据做位运算（3 & 4），当两边是布尔值做的逻辑运算（ true & flase）

3的二进制 0011

4的二进制 0100

1. &位与运算：有0则0

结果：0000 结果十进制为0

1. |位或运算：有1则1

结果：0111 十进制为7

（6)^ 位异或运算 ：相同为0，不同为1

结果：0111 十进制为7

1. ~ 按位取反 ：0变1，1变0
2. 一个数据对另一个数据异或两次，该数本身不变

1. 符号位移动

【1】<< 左移 ：左边最高为丢弃，右边补齐0 (把<<左边的数据乘以2的移动次幂)

3<<2 相当于 3\*2\*2

00000011

(00)00001100 得十进制为12

【2】>> 右移 ：最高位是0 ，左边补齐0，最高为1，左边补齐1

（把>>左边的数据除以2的移动次幂）仅限正数

【3】 >>> 无符号位右移 ： 无论最高是0还是1，左边均补齐0

Day4

1. Switch选择语句

【1】、switch语句后的表达式的取值是有限制的：byte,short,int char

Jdk7 版本后可以每句以及String 类型

【2】break :中断结束的标志，可以控制switch语句的结束

【3】当所有case后的值都与表达式值不匹配时，执行default语句

【4】Switch语句先走case，遇见break结束，如果没有break那么执行到末尾在结束

【5】case 后可以接收short ,byte, int,char,string 类型

二、if语句与switch语句运用场景

【1】if语句：针对结果是Boolean类型的判断

针对一个范围的判断

针对几个常量值的判断

【2】针对几个常量值的判断

1. 循环语句

【1】for循环 ：for(初始话语句；判断语句；控制条件语句){

循环体语句；

}

1. 执行初始化语句
2. 进行判断语句：满足则继续，不满足则退出
3. 满足后：执行循环体语句，在执行控制条件语句
4. 回到步骤2 继续执行

【2】for 循环可以嵌套使用

1. while语句

【1】while循环：初始化语句

While（判断条件）{

循环体语句

控制条件

}

1. do……while语句

【1】do……while循环： 初始化语句

Do{

循环体语句；

控制条件语句；

}while（判断语句条件）；

1. for循环的嵌套

【1】for循环第一个循环是行的变换

第二个循环是列的变化

【2】只有当第二个循环的所有满足判断条件的执行完以后，才可以接着进行第一个for循环满足条件的其他语句的执行；（当内循环的语句执行完一次后，跳出内循环，用 System.*out*.println();实现换行）

1. break语句的使用

【1】、用于放在switch语句中实现=程序的结束

【2】用于放在for循环的If语句判断中进行中断程序

【3】嵌套在for循环的内层结构中，实现内层循环的结束，腰想实现外层语句的中断，结构如下： 标签名：语句

wc: **for**(**int** i=1;i<=5;i++) {

nc: **for**(**int** j=1;j<=4;j++) {

**if**(j==3) {

**break** wc;

}

System.*out*.print("\*");

}

System.*out*.println();

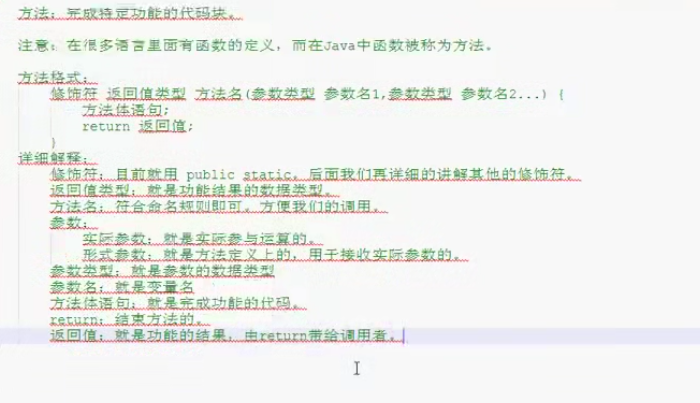
}

1. break：结束单层循环

Continue：结束本次循环

Return : 退出整个main方法

Day5



/\*

\* 构造方法：给对象数据进行初始化

\* 1、方法名与类名相同

\* 2、没有返回值类型，连void都没有

\* 3、没有具体的返回值

\* 重载：方法名与类名相同， 无返回值，参数类型不同

\*{ 如果不给构造方法系统将提供无参构造方法，

\* 但是只要给定了构造方法系统将不提供无参的构造方法，

\* 如果还要使用无参构造方法必须自己给出 }

\*/

\*static (可以修饰成员变量以及方法):

\* 修饰的数据被所有对象共享

\* 随着类的加载而加载

\* 使用情景：如果某个成员变量是被所有对象共享的，就应该定义为静态的

\* (可以通过类名调用,也可以通过对象名调用）

\* 注意事项：在静态方法中是没有this关键字的

\* （静态方法只能访问静态成员方法和静态的成员变量）

（静态比对象先存在）

静态方法：只可以访问静态变量

非静态方法：可以访问非静态以及静态变量,以及方法

\*/

//非静态方法：可以访问非静态以及静态变量,以及方法

//静态方法：只可以访问静态变量

//变量

/\*

\* 成员变量：在类中方法外（有默认值）

\* 局部变量：在方法定义中或者在方法声明上（没有默认值必须初始化定义后才能使用）

当成员变量和局部变量一样时采用就近原则

\*/

/\*

\*基本类型： 基本类型：

引用类型：（类以及数组）必须创建相应类的对象

匿名对象：没有名字的对象

应用场景：调用方法：仅仅只是调用一次的时候

匿名对象作为实际参数的使用

\*/

**class** Student {

**public** **void** show() {

System.***out***.println("study");

}

}

**class** StudentDemo {

//如果看到了一个方法的形式参数是一个类类型（引用类型），需要创建该类的对象

**public** **void** method(Student s) {//调用的时候是将地址值传送到这里

s.show();} }

**public** **class** VariableDemo1 {

**public** **static** **void** main(String[] args) {

//形式参数是引用类型的调用

StudentDemo sc = **new** StudentDemo();

//必须创建类类型的对象

Student s = **new** Student();

sc.method(s);//把s的地址传送到了这里

//类名字的调用

s.show();

//匿名对象

**new** Student();

//匿名对象调用方法

**new** Student().show();

**new** Student().show();//创建了一个与上面不是同一个的新对象

StudentDemo ss = **new** StudentDemo();

//匿名对象可以作为实际参数使用

ss.method(**new** Student());//创建了student的匿名对象

}

}

**DAY9 抽象--接口----多态**

this: 封装 ，对数据进行初始化，，访问本类的内容

super：访问父类的构造函数，访问父类的东西

当变量同名时， 输出局部变量，输出成员变量用this

\*打印父类成员变量用super

\*this：代表本类中的使用

\*super：代表父类存储空间的标识（可以理解为父类引用，操作父类成员）

\*

\*A：调用成员变量

\* this.成员变量 调用本类

\* super.成员变量 调用父类

\* B:调用构造方法：

\* this(...) 调用本类的构造方法

\* super(...) 调用父类的构造方法

\* C:调用成员方法：

\* this.方法名

\* super.方法名

final 标记的类不能被继承

final标注的变量是常量。值不可以被改动

final 修饰的方法不能被重写

final修饰的对象，只能改变对象的内容，不能改变对象的地址值（即就是对象的名字）

多态：同一个对象在不同时期不同的状态

（进行方法的重写）

向上转型: 父 对象 = new 子类（）；

成员变量：编译看左边，结果看左边

构造方法：只跟类有关，默认父类的构造

成员方法：编译看左边，结果看右边

向下转型： 子类 对象名 = （子类） 父类对象；

**接口：**

interface 标记

用implements 实现

接口中的变量只能是常量，就算没有表明，系统也默认是常量

接口中的方法只能被定义成抽象方法，无具体的语句块，无其他的成员方法

接口中无构造方法

类与类：继承关系

类和接口：实现关系，类实现接口中的其他辅助功能，类中是具体存在的共性

接口和接口：继承关系

抽象类：abstract关键字

成员变量：既可以是常量也可以是变量

构造方法：可以无参也可以有参

成员方法：可以抽象也可以非抽象，，，抽象方法中没有实体

抽象类不能实例化：

抽象类可以没有抽象方法，但是抽象方法的类一定是抽象类

抽象类的子类：

是一个具体类：必须重写抽象类的抽象方法

是一个抽象类：

static： 可以标识一个共有的特征变量

可以对象引用，也可以类名进行引用

**DAY10 形式参数---修饰符----权限**

**形式参数**

基本类型：

引用类型：

类：（匿名对象时）方法中定义引用类型的参数，必须定该类的对象（用该类对象调用方法）

抽象类：必须定义该抽象类的子类对象（必须是具体类，并且向上转型）

接口：定义实现接口的类的对象, 用实现类去调用方法

返回类型

基本类型：

引用类型：

类：返回该类的对象

抽象类：返回抽象类的子类对象

接口：返回接口实现类的对象

链式编程：每次调用完后返回的是一个对象，才能继续调用其他的方法

LeiDemo le = new LeiDemo();

le.method().study();

package ：

A：其实就是文件夹

B：作用

a:把相同的类名放在不同的包内

b:把类进行分类管理

C: 定义

package 包名

多级包用 . 分开即可

D: 注意事项

a ：package语句必须在程序的第一条可执行的代码

b ：package语句在第一个Java文件中只能有一个

c ： 如果没有package，默认表示无包名

E ：带包的编译和运行

手动式：

a: 编译一个带包的Java文件夹

b: 通过Javac命令编译该Java文件

c: 手动创建包名

d:把b步骤的class文件放到c步骤的最底层包中

e:回到和包根目录在同一目录的地方，然后运行 （带包运行）

自动式：

a:编写一个带包的Java文件

b:javac编译的时候带上 -d 即可

Javac - d . helloword.java

c: 带包运行

Java cn.itcast.helloword

不同包类的访问：

cn.itcast.Demo d = new cn.itcast.Demo ( );

导包：import 包名

导入是类的名称的级别：import cn.itcast.Demo ;

package import class 有没有顺序关系：

有

package >import >class

package: 只能有一个

import：可以有多个

class：可以有多个，以后建议是一个

权限修饰符：

(同一个包) 本类 同一个包类（子类和父类） 不同包下(子类) 不同包下（父类）

public : ok ok ok ok

private: ok no

protected: ok ok ok

默认： ok ok

修饰符：

权限修饰符：默认，private，protected, public

状态修饰符：static , final

抽象修饰符：abstract

类：

权限修饰符：默认修饰符，public

状态修饰符：final

抽象修饰符：abstract

（用的最多的是 public）

成员变量：final表示常量，static 表示 类名调用 四种权限修饰符都可以

（用的最多的是private）

构造方法：只能使用四种常见修饰符

( 用的最多是public)

成员方法：四种权限修饰符，，static (类名调用) ，final( 不能被重写)，abstract

（用的最多是public）

内部类：把类定义在其他类的内部，这个类称为是内部类

访问特点：

A：内部类可以直接访问外部类的成员，包括私有的

B: 外部类要访问内部类的成员，必须创建对象

C:对象创建格式

《1》 外部类类名. 内部类类名 对象名 = new 外部类类名（）. new 内部类类名（）；

《2》 外部类类名 外部类对象 = new 外部类类名（）；

内部类类名 内部类对象名 = 外部类对象名 . new 内部类类名（）；

内部类对象名 . 内部类方法（）；

D：内部类加了private修饰，只能内部类访问，外部类不能够通过C方式进行创建对象名来访问内部类 ;

成员内部类的修饰符：

private： 通过父类写一个同等的方法。创建内部类的对象，来访问内部类的方法，，，，

在通过外部类创建对象，通过对象来访问同级方法。

static : 内部类用static修饰可以看成是内部类是外部类的成员

注意： 被static修饰的内部类只能访问外部类的静态数据

成员内部类被静态修饰的访问方式是：

外部类名 . 内部类名 对象名 = new 外部类名. 内部类( ）；

注意 ：

1、内部类和外部类没有继承关系

2、通过外部类名限定this对象

外部类名. this. 变量名

局部内部类: 在外部类的方法里定义了一个类

1、可以直接访问外部类的成员

2、在局部位置，可以创建内部类对象，通过对象调用内部类的方法，来使用内部类的特定功能

面试题：

局部内部类访问局部变量的注意事项？

-----局部内部类访问局部变量，必须用final修饰

为什么呢？

--------局部变量随着方法事物调用而调用，随着调用结束而消失，而堆内存（内部类）的内容并不会立即消失

加final修饰后就变成了 常量，，（把值留下了，变量名消失了）

匿名内部类：

前提：存在一个类或者接口

这里的类可以是具体类也可以是抽象类

格式：

new 类名或者接口名（）{

重写方法 ;

}

本质是什么？

是一个继承该类或者实现了该接口的子类的匿名对象

调用方法：多态

**DAY12 String 类**

Scanner类

基本格式：

\* public boolean hasNextxxx();判断是否是某种类型的数据

\* public xxx nextXxx();获取该元素

\*

\* public int nextInt();

\* public String nextLine();输入string类型的数据

\*

\* 先获取一个数值在获取一个字符串，回车默认给了数据给String类

String 类

字符串：就是由多个字符组成的一串数据，也可以看成是一个字符数组

（字符串是常量，一旦被赋值，就不能被改变）

public String(); 空构造

public String(byte[] bytes); //把字节数组转换成字符串

public String(byte[] bytee,int index ,int length); 把字节数组的一部分转换成字符串

public String(char[] value);//把字符数组转换成字符串

public String(char[] value,int index,int length);//把字符数组的一部分转换成字符串

public String(String original );//把字符串常量值转换成字符串

字符串一旦被赋值就不能被修改（指的是值不变，而不是引用）

面试题：

String s1= new String("hello");

String s2 = "hello"; 有什么区别

解答：

==比较对象的地址值，比较引用类型

equals：比较引用类型，也是比较地址值，而string类重写了equals方法，比较的是内容是否相同

前者 中其实有两个对象，堆内存和方法区里分别一个，（先是常量池----然后堆内存指向方法区改变地址值，最后栈内存指向堆内存）

后者直接在方法区常量池中创建一个对象

面试题：

String s1 = "hello";

String s2 = "word";

String s3 = "helloword";

s3==s1+s2;

s3=="hello" + "word";

字符串如果变量相加,先开空间，在拼接

字符串如果是常量相加，先加，然后在常量池中找，如果有直接返回，否则创建

String类的判断功能 :

boolean equals(Object obj);//比较字符串内容是否相同，区分大小写

boolean equalsIgnoreScae(String str);//比较字符串内容是否相同，忽略大小写

boolean contains(String str);//比较大字符串是否包含小字符串

boolean stratWith(String str);//判断字符串是否以某个指定的字符串开头

boolean endsWith(String str);//判断字符串是否以某个指定的字符串结尾

boolean isEmpty();//判断字符串是否为空

注意：

字符串内容为空，字符串对象为空

String s = " "; //对象所表示得到内容为空，但是对象存在

String s = null; // 表示对象不存在

String 类的获取功能：

int length();//获取字符串长度

char CharAt(int index); //获取指定索引位置的字符

int indexOf(int ch); //返回指定字符在此字符串中第一次出现的索引

为什么此处是int类型，而不是char类型？

原因是 'a' 和97 其实都可以代表 ‘a’ index 返回-1 表示不存在该字符

但是如果单写成char类型，，，97就会无法识别

int indexOf(String str); //返回指定字符串在此字符串中第一次出现的索引值

int indexOf(int ch,int fromIndex);//返回指定字符在此字符串从指定位置后第一次出现的索引位置

String substring(int start)；//从指定位置开始获取字符串，默认到末尾结束

String substring(int start ,int end);//从指定位置开始到指定的位置结束

[ strat ，end } 返回的字符串只包含start索引处的字符，但是不包含end所在索引位置的字符

字符串的遍历：

A：如何遍历？

\* char charAt(int index)

B:如何知道字符到底有多少呢？

int length();

字符判断是大写还是小写还是数字：

if(ch>='0'&&ch<='9') {

number ++;//数字

}else if(ch>='A'&&ch<='Z') {

maxCount ++;//大写

}else if(ch>='a'&&ch<='z') {

minCount ++;//小写

}

String的转换功能：

byte[ ] getBytes( )；//把字符串转换成字节数组

char[ ] toCharArray();//把字符串改变成字符数组

static String valueOf(char[ ] chs);//把字符数组转换成字符串

static String valueOf(int i) ;//把int类型的数据转换成字符串

注意：String类型的valueOf能把任意类型的数据转换成字符串

String toLowerCase( );//把字符串全部变成小写

String toUpperCase( );//把字符串全部变成大写

String concat(String str);// 把字符串进行拼接

String的其他功能：

替换功能：

String replace(char old , char new) //将一个字符换成另一个字符

String replace(String old,String new)//将一个字符串换成另一个字符串

去除字符串空格：(只能去除两端的空格)

String trim( )

按字典顺序比较两个字符串：

int compareTo(String str)

int compareToIgnoreCase(String str)

分割字符串：

String str[]=s.split(" ");//将一个字符串分割成子字符串，然后结果作为字符串数组返回

Copy方法

1、 创建一个与指定的字符串具有相同值的字符串的新实例，其语法格式如下。

public static string Copy (string str)； ?

其中：str是指要复制的字符串；返回值是与str具有相同值的字符串。

2．CopyTo方法

CopyTo方法可以将字符串的某一部分复制到另一个数组中，其语法格式如下。

DAY13 --- Arrays工具类

一、Arrays工具类：针对数组进行操作

public static String toString(int[ ] a ) 把其他任意类型转换为字符串

publlic static void sort(int[] a) 对任意类型的数组进行排序

public static int binaySearch(int[] a , int key); 对任意类型的数组进行二分查找（数组必须是有序的）

开发原则：只要是对象就要判断是不是为空

二、基本类型包装类：为了对基本数据类型进行更多的操作

(基本类型) （引用类型）

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Char

boolean Boolean

三、Integer的构造方法：

public Integer(int value); 将一个基本的int类型的数据变成integer包装类型数据

public Interger(String s) ;;将一个string类型的数据变成integer

（条件是原始的string类型的数据必须只能是数字字符组成，不能是英文字符组成）

注意：如果要将int类型或者是string类型的数据转换成integer的话

必须用构造方法，创建对象进行转换

四、int 与 String 转换

\* int -----String

\* 最佳用：String.valueOf(int i)

\*

\* String-----int

\* Interger.parseInt(String s);

五、Integer进制的转换

十进制转换为其他进制： (只能是0 -----26)

public static String toString(int i,int radix); //radix表示想要转换成的进制数

其他进制到十进制

public static int parseInt(String s ,int radix)//将一个其他进制换成是十进制必须要是可以转变的

六、自动装箱和拆箱：

自动装箱：把基本类型转换为包装类类型

注意：对象不能为空，不然调用空指针报错，抛异常

Integer x = null 会抛异常，，每次判断对象，必须对对象进行是否为空判断

自动拆箱：把包装类类型转换为基本类型

七、面试题

针对-128 到 127 的数据，有一个数据缓冲区，如果数据是在此范围内，每次并不创建新的空间

所以对于== 判断或者equals判断都为true

例如：

Integer i =100;

Integer i2 = 100;

i = =12 ; 结果为真

i.equals(i2); 结果也为真

八、character

构造方法：public character(char value)；

成员方法：

public static boolean isUpperCase (char ch);//判断字符是不是大写

public static boolean isLowerCase (char ch );//判断字符是不是小写

public static boolean isDigit(char ch) ; //判断指定字符是不是数字

public static char toUpperCase(char ch);//把指定的字符转成大写

public static char toLowerCase(char ch);//把指定的字符转成小写

**DAY13 StringBuffer类**

StringBuffer:

线程安全的可变字符串

一、StringBuffer 与String的区别？

a: 前者长度和内容可变，后者不可变

b:如果使用前者进行字符串的拼接，不会浪费太多的资源

二、StringBuffer的构造方法：

public StringBuffer(); 无参构造函数

public StringBuffer(int capacity) 指定容量的字符串缓冲区对象

public StringBuffer(String str)；指定字符串内容的字符串缓冲区对象

StringBuffer的方法：

public int capacity (); 返回当前容量 理论值 (默认初始值为16)

public int length() : 返回长度（字符数）， 实际值

StringBuffer的添加功能:

字符的设置和替换：

setCharAt(int index,char ch) ,将指定的字符放在index指出的位置

charAt(int index);获取指定位置的字符

字符的插入：（在指定的位置插入字符或者其他类型的数据）

insert(int offset,char ch)； 在offset位置插入字符ch

public StringBuffer insert(int offset ,String str)

字符的添加：（默认连接到末尾）

public StringBuffer append(String str);

StringBuffer的删除功能：

public StringBuffer deleteCharAt(int index) ; // 删除指定位置的字符

public StringBuffer delete(int start ,int end)；删除指定位置开始指定位置结束的内容，并且返回本身

StringBuffer的替换功能：

public StringBuffer replace(int start ,int end,String str)；//从start开始到end用str替换

StringBuffer的反转功能：

public StringBuffer reverse();

StringBuffer的截取功能：

(返回值类型不同，返回String类型的数据，将截取的数据放在String类型中，本身字符串不变)

public String substring(int start);

public String substring(int start,int end);

面试题：

StringBuffer与数组的区别：

1、两者都可以当成容器。但是StringBuffer能够放不同类型的数据，最终形成字符串

数组能放多个数据，但是只能放同种类型的数据

String与StringBuffer作为参数类型返回：

1、基本类型

2、引用类型：必须创建相应的对象

注意：String类型的参数转变和基本类型是一样的

常量池的内容不变

三、冒泡排序：

1、外层控制次数，内层控制排列

for(int x = 0;x<arr.length-1;x++) {

for(int y =0;y<arr.length-1-x ; y++) {

int temp = arr[y];

arr[y] = arr[y+1];

arr[y+1]= temp;

}

}

syso();

四、选择排序

————\*\*\*\*-————

第一层循环的数据和第二层循环的数据进行比较

————\*\*\*\*————

for(int x=0;x<arr.length-1;x++) {

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

if(arr[x]>arr[y]) {

int temp = arr[x];

arr[x] = arr[y];

arr[y] = temp;

}

}

}

五、高级二分查找

基本查找：从头找到尾 （一般是无序的）

\*高级二分查找：对半找 （前提条件必须是有序的）

\*

\* 思路：

\* A:定义最小索引min，最大索引max

\* B:计算出中间索引mid

\* C:拿中间索引的值与要查找的元素进行比较

\* 相等 ，就直接返回当前的中间索引

\* 不相等

\* 大了，往左边找

\* 小了，往右边找

\* D:重新获取最大索引值以及最小索引值

\* 大了 ，往左边找

\* max = mid-1;

\* 小了，往右边找

\* min = mid+1;

\* E:重新回到B位置

**DAY14 校验---正则表达式----Random\_\_Date--Calendar**

正则表达式：符合一定规则的字符串

正则表达式组成规则：

A：字符

x 字符 x , 举例 ： 'a' 表示字符a

\\ 反斜线字符

\n 换行符 （'\u000A'）

\r 回车符 （'\u000D'）

B:字符类

[abc] a、b或c(简单类)

[^abc] 任何字符，除了a,b,c

[a-zA-Z] a 到z 或者A 到 Z 两头字母包括在内

[0-9] 0 到9的字符都包括

C：预定字符

. 任何字符 如果是 . 字符本身，表示为\.

\d 数字：[0-9]

\w 单词字符[a-zA-Z\_0-9]

表示在正则表达式中组成单词的东西必须有这些组成

D：边界匹配器

^ 行的开头

$ 行的结束

\b 单词边界

就是不是单词字符的地方

举例：hello world？haha;xixi

E:数量词

X? X，一次或一次也没有

X\* X，零次或多次

X+ X，一次或多次

X{n} X，恰好 n 次

X{n,} X，至少 n 次

X{n,m} X，至少 n 次，但是不超过 m 次

正则表达式的功能：

判断功能：

public boolean matches(String regex); 判断字符串是否匹配给定的正则表达式

分割功能：String类的public String[ ] spilt(String regex); 根据给定的正则表达式的匹配拆分此字符串

注意：1、对路径进行分割，应该将\改成\\ 对应正则表达式两个\\对应路径上一个\

2、. 分割 直接写.表示的是任意字符 如果要表示成.的话要写成\\.

替换功能：

String类的public String replaceAll(String regex, String replacement);

注意： 使用给定的replacement 替换此字符串所有匹配正则表达式的子字符串

//定义一个字符串

String string ="caddo12345javaandweb";

//定义新的串接收替换后的字符串

String string2 = string.replaceAll("\\d+", "\*");

//注意\\d+ ， "\*" 表示出现数字一次或一次以上替换一个\*给原字符串

// \\d , "\*" 表示数字出现一次就替换一个\*给原字符串

获取功能：

Pattern 和Matcher类的使用

模式匹配器的基本使用顺序

1、把正则表达式编译成模式对象 public static Pattern compile(String) //将给定的正则表达式编译到模式中

Pattern p = Pattern.compile("a\*b");

2、通过模式对象得到匹配器对象

Matcher m = p.matcher("aaaaab");

3、调用匹配器对象的功能

boolean b = m.matches();

通过Matcher类的 public boolean find( );查找是否存在符合条件的子字符串

通过Matcher类的 public String group( ) ;返回由以前匹配操作所匹配的输入子序列

注意：调用group必须先调用find ，， 先找是否存在再来输出匹配的子字符串

Math类

成员变量 static final double E //自然底数

static final double PI //圆周率

成员方法：

public static int abs(int a) ;绝对值

public static double ceil(double b) ; 向上取整 返回最小的大于b的值

public static double floor(double b) ; 向下取整 返回最大的小于b的整数

public static double pow(int a,int b); a的b次幂

public static double random( ); 返回0.0-1.0的随机数

public static double sqrt(double b) ; 返回b的正平方根

public static int round(其他类型的值) ；返回最接近参数的int值

Random类：

构造方法：

public random() 没用给种子，默认是当前的毫秒值

public random(long seed)给出指定的种子

区别： 给定种子后，每次得到的随机数是相同的

成员方法：

public int nextInt();返回int类型的随机数

public int nextInt(int n)； 返回（0，n）范围内的随机数

System类：

方法：

public static void gc() 运行垃圾回收器 重写Object类的finalize() 方法，进行回收

public static void exit(int status) 终止当前正在运行的 Java 虚拟机。

public static long currentTimeMillis() 返回以毫秒为单位的当前时间

public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

从指定源数组中复制一个数组，复制从指定的位置开始，到目标数组的指定位置结束。

src - 源数组。

srcPos - 源数组中的起始位置。

dest - 目标数组。

destPos - 目标数据中的起始位置。

length - 要复制的数组元素的数量。 最终用Arrays.toString( int[] arr);输出数组

BigInteger :可以让超过Integer范围内的数据进行运算

构造方法：BigInteger(String value )；

public BigInteger[] divideAndRemainder(String value)

//以数组的形式将商和余数放在数组中

Date类：表示特定的时间，精确到毫秒值

构造方法：

Date( ) ; 根据当前默认的毫秒值创建日期对象

Date(long date) ;根据给定的毫秒值创建日期对象

成员方法：

public long getTime( );获取时间，以毫秒为单位

public void setTime(long time) :设置时间

注意：

从Date得到一个毫秒值：

创建Date对象，调用getTime( ）方法

把一个毫秒值转换为Date:

1、构造方法

2、setTime( )方法

DateFormat: (抽象类无实例对象)

进行日期和字符串格式化和解析：可以将Date和String类型进行转换：默认使用simpleDateFormat具体类实现

simpleDateFormat

构造方法：

simpleDateFormat( );默认模式

simpleDateFormat(String pattern ) ; 给定的模式 pattern代表输出的模式规则

规则表示：1、年y 2、月M 3、日d 4、 时H 5、 分m 6、 秒s

例如：yyyy-MM-dd HH:mm:ss 输出当前时间

成员方法：

public String format(Date d) ; 将Date类型转换为以字符串类型输出的模式

public Date parse(String s) ; 将字符串转换成一个Date类型的数据

注意一点，使用SimpleDateFormat需要捕获ParseException异常。

Calendar：类是一个抽象类，它为特定瞬间与一组诸如 YEAR、MONTH、DAY\_OF\_MONTH、HOUR 等 日历字段之间的转换提供了一些方法，并为操作日历字段

//其日历字段由当前日期和时间进行初始化

Calendar rightNow = Calendar. getInstance(); //返回一个子类的对象

成员方法:

public int get(int filed); 返回给定字段的值，日历类中的每个日历字段都是静态的成员变量，且为int类型

注意：month 从0 开始显示

public void add(int filed, int amount)根据给定日历字段和对应的时间，来对当前的日历进行操作

public final void set(int year,int month,int date);设置当前日历的年月日

1、判断电话号码和邮箱

package dh14;

import java.util.Scanner;

//判断电话号码

public class MatchesDemo1 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("请输入要进行判断的qq号码");

String string = sc.nextLine();

boolean b = isNuCheck(string);

System.out.println(b);

}

public static boolean isNuCheck(String string) {

boolean flag = true;

if(string.length()>=5&&string.length()<=15) {

if(string.charAt(0)!='0') {

char[] ch = string.toCharArray();

for(int i=0;i<ch.length;i++) {

if(!Character.isDigit(ch[i])) {

flag = false;

return flag;

}

}

}else {

flag = false;

return flag;

}

}else {

flag = false;

return flag;

}

return flag;

}

}

2、按照不同的规则分割数据

package dh14;

//按照不同的规则分割数据

public class SpiltDemo1 {

public static void main(String[] args) {

String s1 ="aa,bb,cc";

String[] strings = s1.split(",");

for(int i=0;i<strings.length;i++) {

System.out.print(strings[i]+" ");

}

System.out.println();

String s2 = "aa.bb.cc";

String[] strings2 = s2.split("\\.");

for(int i=0;i<strings2.length;i++) {

System.out.print(strings2[i]+" ");

}

System.out.println();

String s3 = "D:\\JSP实验\\第二次实验\\登录验证";

String[] strings3 = s3.split("\\\\");

for(int i=0;i<strings3.length;i++) {

System.out.print(strings3[i]+" ");

}

}

}

3、把字符串中的数字替换成\*

package dh14;

//将字符串中的数字替换成\*

public class ReplaceAllDemo1 {

public static void main(String[] args) {

String string = "java121212andweb";

String string2 = string.replaceAll("\\d+", "\*");//表示用\*替换满足正则表达式的字符串

System.out.println(string2);

//代表一个数字就用一个\*替换

String string3 = "java121212andweb";

String string4 = string3.replaceAll("\\d", "\*");

System.out.println(string4);

}

}

4、获取字符串中由三个字符组成的字符串

package dh14;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

//获取字符串中由三个字符组成的类

public class HuoQuDemo1 {

public static void main(String[] args) {

String string = "ni sha diao ba shi jiu shi";

//创建模式对象

Pattern p = Pattern.compile("\\b\\w{3}\\b");

//将模式对象放在匹配器中

Matcher m = p.matcher(string);

//调用 find 找到 group 输出找到的字符串

while(m.find()) {

System.out.println(m.group());

}

}

}

6、获取任意范围内的随机数

package dh14;

import java.util.Scanner;

//获取任意范围内的随机数

public class RandomDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("请输入起始范围：");

int start = sc.nextInt();

System.out.println("请输入终止范围：");

int end = sc.nextInt();

for(int i=0;i<100;i++) {

int m =(int)(Math.random()\*(end-start))+start;

System.out.println(m);

}

}

}

7、来到世界多少天

package dh14;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Scanner;

//来到这个世界多少天

public class DateCountDemo {

public static void main(String[] args) throws ParseException {

Scanner sc = new Scanner(System.in);

System.out.println("请输入出生日期：");

String string = sc.nextLine();

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

Date date = dateFormat.parse(string);

long l =(new Date().getTime()-date.getTime())/1000/60/60/24;

System.out.println(l);

}

}

8、输出任意年的2月份有多少天

package dh14;

import java.util.Calendar;

import java.util.Scanner;

//获取任意年份的2月份有多少天

public class CalendarDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("请输入要进行判断的年份");

int year = sc.nextInt();

Calendar c = Calendar .getInstance();

c.set(year, 2, 1);

c.add(Calendar.DATE, -1);

System.out.println(c.get(Calendar.DATE));

}

}

9、将字符串“91 27 46 38 50”

\* 用代码实现输出结果为：“27 38 46 50 91”

package dh14;

import java.util.Arrays;

/\*

\* 将字符串“91 27 46 38 50”

\* 用代码实现输出结果为：“27 38 46 50 91”(字符串)

\*/

public class SplitDemo {

public static void main(String[] args) {

String string = "91 27 46 38 50";

String[] strings = string.split(" ");

int[] arr = new int[strings.length];

for(int i=0;i<strings.length;i++) {

arr[i] = Integer.parseInt(strings[i]);

}

Arrays.sort(arr);

StringBuffer buffer = new StringBuffer();

for(int j=0;j<arr.length;j++) {

buffer.append(arr[j]+" ");

}

String string2 = buffer.toString().trim();

System.out.println(string2);

}

}

10、Date---String 之间的转换

package dh14;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

//Date----String之间的转换

public class Date\_String {

public static void main(String[] args) throws ParseException {

Date d = new Date();

String string = dateToString(d,"yyyy-MM-dd");

System.out.println(string);

String s = "2020-01-02";

Date date = stringToDate(s,"yyyy-MM-dd");

System.out.println(date);

}

public static String dateToString(Date d,String pattern ) {

return new SimpleDateFormat(pattern).format(d);

}

public static Date stringToDate(String s,String pattern) throws ParseException {

return new SimpleDateFormat(pattern).parse(s);

}

}

**DAY15 Collection --List 迭代器**

对象数组：

1、数组可以存储基本数据类型，也可以存储引用类型，它存储引用类型的时候的数组就是一个对象数组

2、用数组存储5个学生对象，并遍历数组

class Student {

private String name;

private int age;

//无参构造

Student() {}

//有参构造

Student(String name,int age) {

this.name = name;

this.age = age;

}

//省略getXXX和setxxx()方法

//重写toString() 方法

public void toString() {

//override

}

}

public static void main(String[] args) {

//创建学生类

//创建集合对象 Collection集合为接口，由具体的子类的实现类去调用方法实现

Collection c = new ArrayList();

//创建学生对象

Student s1 = new Student("lili",19);

Student s2 = new Student("mmmm",21);

//将学生对象存储在集合中

c.add(s1);

c.add(s2);

//遍历集合输出

//将集合转换成数组，遍历数组输出 (此时的集合默认是Object类的对象)

Object[] obj = c.toArray();

for(int i=0;i<obj.length;i++) {

Student s = (Student) obj[i]; //将Object类的数组强制转换为Student类的对象用于输出，其方法实质是多态

System.out.println(s);

}

}

二、collection功能概述;

功能:

1、添加功能

boolean add(Object obj) //在集合中添加一个元素

boolean addAll(Collection c) //在一个集合中添加另一个集合的所有元素

2、移除功能

void clear() //清除集合中的所有元素

boolean. remove(Object obj) //清除集合中的一个元素

Boolean. removeAll(Collection c )// 清除集合中的所有元素 (只有当所以元素被清除时才返回true)

3、长度功能

int size() //用于显示集合的长度

5、判断功能

boolean contains(Object obj)//判断集合中是否包含了目标元素，包含一个就返回true

boolean containsALL(Collection c) 集合只有包含了另一个集合的所有元素才能返回true

boolean isEmpty() 判断集合是否为空

6、把集合转换为数组

Object[] toArray()

4、迭代功能(遍历)

Iterator iterator()

用boolean hasNext()方法 判断是否有元素可以进行迭代

用Object next()方法获取每一个元素，并且自动移到下一个位置

注意:

用集合对象去创建迭代对象

迭代器对象是依赖于集合存在而存在的

（了解）

交集

boolean retain(Collection c) //调用此方法的集合是否含有形参集合的元素，，只要含有一个就会返回true

集合转换为数组

Object[ ] toArray( ) ;;将一个集合转换为数组

三、collection 集合的遍历

a:把集合转换为数组

Object[] toArray()

b:迭代器

Iterator iterator();//collection集合下的迭代器方式

四、遍历案例

1、存储字符串并遍历

public static void main(String[] args) {

//创建集合对象

Collection c = new ArrayList();

//创建字符串对象

String s1 = "hello";

String s2 = "java";

//将字符串对象添加到集合中

c.add(s1);

c.add(s2);

//用迭代器遍历集合

//创建迭代器对象（迭代器是一个接口，依赖于集合而存在，所以通过集合对象来创建迭代器对象，调用方法进行实现集合遍历）

Iterator it = c.iterator();

//迭代器输出时，容易抛越界异常 ，因此要进行判断集合中是否还有元素可以取出（因为元素为多个，因此用循环判断输出）

while(it.hasNext()) {

String s = (String) it.next(); // 强转为字符串类型输出

System.out.println(s)

}

/\*

//for循环遍历输出元素 (初始化对象，判断语句，第三个语句可以不写)

for(Iterator it = c.iterator();it.hasNext();) {

String s = (String) it.next(); // 强转为字符串类型输出

System.out.println(s)

}

\*/

}

2、自定义对象遍历

//创建学生类

class Student {

private String name;

private int age;

//无参构造

Student() {}

//有参构造

Student(String name,int age) {

this.name = name;

this.age = age;

}

//省略getXXX和setxxx()方法

//重写toString() 方法

public void toString() {

//override

}

}

public static void main(String[] args) {

//创建集合对象 Collection集合为接口，由具体的子类的实现类去调用方法实现

Collection c = new ArrayList();

//创建学生对象

Student s1 = new Student("lili",19);

Student s2 = new Student("mmmm",21);

//将学生对象存储在集合中

c.add(s1);

c.add(s2);

//通过集合对象创建迭代器对象（多态）

Iterator it = c.iterator();

//判断循环条件输出

while(it.hasNext()) {

Student s = (Student) it.next();

//Object next(); 返回的是一个Object类的对象，进行向下转型输出学生类对象，可以调用学生类特有的输出方法

System.out.println(s.getName() +" "+s.getAge());

//如果学生类重写了Object类的toString()方法则可以直接输出对象 ，否则输出包名.类名.地址值

}

}

五、迭代器Iterator方法

1、判断下一个是否有元素

boolean hasNext( ) //用于遍历时判断元素是否越界

2、获取元素并且移动到下一个位置

Object next( ); //自动返回的是一个Object类的对象

六、List的功能（索引 有序 对象内容可以重复）

1、添加（根据特定索引添加元素）

void add(int index,Object element) ; // 在特定的索引位置进行添加，可以紧跟集合最后一个元素添加，但是不能越界

2、删除

boolean remove(int index) 把指定位置的元素删除

3、修改

boolean set(int index, Object element) ;//在指定的位置，用element元素替代原有元素

4、迭代器方法

ListIterator listIterator( );

5、获取

public Object get(int index); //根据具体的索引位置获取元素

六 ---------------list集合的特有遍历方法：(List集合是有序的，且可以根据具体的索引值进行元素的操作，且对象内容可以重复)

1、size() 和 get() 方法结合

//创建List集合对象

List list = new ArrayList();

//创建字符串对象

String s1 = "hello";

String s2 = "java";

//将字符串对象添加到集合中

list.add(s1);

list.add(s2);

//遍历List集合

for(int i=0;i<list.size();i++) {

//用public Object get(int index);方法实现获取元素 （返回具体索引位置的元素）

String s = (String) list.get(i);

System.out.println(s);

}

2、迭代器方法

/创建List集合对象

List list = new ArrayList();

//创建字符串对象

String s1 = "hello";

String s2 = "java";

//将字符串对象添加到集合中

list.add(s1);

list.add(s2);

//通过List集合对象创建迭代器对象

ListIterator it = list.listIterator();

while(it.hasNext()) {

String s = (String) it.next();

System.out.println(s);

}

七、ListIterator 迭代器的使用： （遍历）

1、查询下一个元素是否存在

boolean hasNext( ); //放在while循环中用于判断条件

2、获取元素，并移动到下一个位置

Object next( );

八、数组与集合的区别

A：长度区别：

数组的长度是固定的

集合的长度是可变的

B：内容区别：

数组元素的类型可以是基本类型也可以是引用类型

集合元素的类型只能是引用类型

C：形式区别

数组中元素类型必须相同

集合元素类型可以不同

九、出现不合法的并发修改时解决方法：

1、出现现象

J集合长度出现越界异常

2、原因

迭代器是依赖于集合而存在的，先有集合再有迭代器，所以当集合发生改变但是迭代器无法预知改变时，出现异常现象

3、解决方案

A：用集合进行元素的增减或其他操作 ，并且用集合进行遍历操作

/创建List集合对象

List list = new ArrayList();

//创建字符串对象

String s1 = "hello";

String s2 = "java";

String s3 = "caddo";

//将字符串对象添加到集合中

list.add(s1);

list.add(s2);

list.add(s3);

//集合修改元素

for(int i=0;i<list.size( );i++) {

String s = (String) list.get(i);

if("java".equals(s)) {

list.add("basic");

}

}

System.out.println(list); //此时添加的元素默认在集合的末尾

B：用迭代器进行元素的增减或其他操作，且用迭代器进行遍历

（注意：用迭代器进行元素的改变时，只有ListIterator可以对元素进行操作，因此用ListIterator迭代器）

public static void main(String[] args) {

//创建List集合对象

List list = new ArrayList();

//创建字符串对象

String s1 = "hello";

String s2 = "java";

String s3 = "caddo"

//将字符串对象添加到集合中

list.add(s1);

list.add(s2);

list.add(s3);

//通过集合创建迭代器对象

ListIterator it = list.listIterator();

//迭代器进行添加判断操作、并且遍历

while(it.hasNext()) {

String s = (String) list.next();

if("java".equals(s)) { //(list.contains("java")) //判断集合中是否包含某个元素

it.add("basic");

}

}

//输出

System.out.println(list); //此时通过迭代器进行添加的元素默认添加到判断元素的后面

}

十、list子类的特点

ArrayList:

底层数据结构为数组，查询快，增删慢

线程不安全，效率高

Vector:

底层数据结构为数组，查询快，增删慢

线程安全，效率低

LinkedList:

底层数据结构是链表，查询慢，增删快

线程不安全，效率高

十一、操作步骤

1、创建类

2、创建集合对象

3、创建类对象

4、迭代器遍历

A:通过集合对象创建迭代器对象（多态 向上转型）

B：迭代器对象调用方法判断是否有元素可以输出，防止越界

C：迭代器获取元素输出

向下转型输出（多态的方式，调用类的特有方法进行格式输出）

否则类重写Object类的toString() 方法以特定格式输出

**DAY15 集合框架**

Collection:是集合接口；

功能概述： (自动重写了toString方法)

1、添加功能：

boolean add(Object obj);添加一个元素

boolean addAll(Collection c) ;添加一个集合的元素(将另一个集合所有的元素)

2、删除功能：

void clear() 移除所有元素

boolean remove(Object obj)移除一个元素

Boolean removeAll(Collection c);移除一个集合的元素（移除一个就为true）

3、判断功能；

boolean contains(Object c) ;判断集合中是否包含指定的元素

boolean containsAll(Collection c) ; 判断集合中是否包含指定集合元素(只有包含所有的元素才是包含)

boolean isEmpty();判断集合是否为空

5、长度功能：

int size() ; 元素的个数

6、交集功能：

boolean retainAll(Collection c) :两个集合

（两个集合都有的元素，如果存在则为true，交集到的元素放在调用方法的集合中存储）

7、把集合转换为数组，实现数组遍历

Object[] toArray()

4、获取功能：（遍历） （迭代器依赖集合存在的）

Iterator<E> iterator()

Iterator ; 是一个接口，对象调用方法其实是一个子类的对象，采取的是多态的方式

Object next();获取元素，并移动到下一个位置

（注意，但是一旦超出范围就会抛异常）

所以每次获取时要判断是否存在元素可以取出，则用 hasNext()判断是否还有元素可以进行遍历

迭代器依赖集合对象去调用iterator方法进行迭代 ：Iterator 迭代对象 = 集合对象.iterator();

迭代对象.next(); 是一个object类对象 向下转型输出

二、集合的使用步骤

1、创建集合对象

2、创建元素对象

3、把元素添加到集合

4、遍历集合

a:通过集合对象获取迭代器对象

b:通过迭代器对象的hasNext() 方法判断是否含有元素

c:通过迭代器对象调用next() 方法获取元素并且移动到下一个位置

三、迭代器的原理及实现

原理：定义为一个接口，定义功能，但是未实现 ，由子类的内部类去实现方法

四、List类:

有序(存储和取出的元素顺序是一致的)的Collection序列 ,通过索引来操作。（与set不同其可以插入重复的）

List集合的特有功能： （索引值不能越界）

1、添加功能：

void add(int index, Object element) //在列表的指定位置插入指定元素 (能紧跟上一个元素索引，但是不能越界)

2、获取功能：

Object get(int index) //获取指定位置的元素

3、列表迭代器：

listIterator( ) ;//list集合特有迭代器

listIterator(int index) //迭代从指定位置的集合

4、删除元素

Object remove(int index) //根据索引删除元素，返回被删除的元素

5、修改元素：

Object set (int index, Object element) ;//根据索引修改元素，返回修改的元素

特有的遍历（ListIterator）：

1、get（）方法和size（）方法结合使用

2、特有遍历方法：(正向遍历)

a: hasNext() 和next()方法结合

3、Object prevoius( ) // 返回列表的前一个元素 （相当于倒着遍历）

boolean hasPrevious( ) //判断倒着遍历是否还要元素

注意：可以实现逆向遍历但是必须先正向遍历才可以逆向遍历，所以无啥意义，一般不适用

迭代器异常处理：出现不合法的并发修改时，抛异常

原因：因为迭代器是依赖于集合而存在的，当集合进行修改时迭代器不知道，所有抛异常

处理方法：

1、普通方法进行遍历get() 和size() 方法结合

for(int i=0;i<list.size();i++) {

String s = (String) list.get(i); //此时的S是一个字符串类型的对象，不单纯指代一个字符串

if("world".equals(s)) {

list.add("javaee");

}

}

System.out.println(list);

2、迭代器进行修改元素 ListIterator才有修改元素的方法

注意：添加对的元素在迭代元素的后面出现

//迭代器添加元素 // (此时添加的元素在迭代元素的后面)

ListIterator it = list.listIterator();

while(it.hasNext()) {

String s = (String) it.next();

if("world".equals(s)) {

it.add("javaee");

}

}

System.out.println(list);

五、栈 ： 先进后出

队列： 先进先出

六、List的三个子类特点：

ArrayList:

底层数据结构是数组，查询快，增删慢

线程不安全，效率高

Vector:

底层数据结构是数组，查询快，增删慢

线程安全，效率低

LinkedList:

底层数据结构是链表，查询慢，增删快

线程不安全，效率高

注意》不会分析用那种就用ArrayList

七、数组与集合的区别

A：长度区别

数组长度固定

集合长度可变

B：内容区别

数组可以是基本类型，也可以是引用类型

集合只能是引用类型

C：元素内容

数组只能存储同种类型的数据

集合可以存储不同类型的数据

**DAY16 ArrayList\_\_\_\_LinkedList**

一、Vector特有功能：

1、添加功能：

public void addElement(Object obj); 将指定的组件添加到此向量的末尾，且向量的大小加 1。

2、获取功能：

public Object elementAt(int index) ;返回指定索引处的组件

public Enumeration elements( ) //返回向量组件的枚举

// Enumeration 是一个接口，因此与迭代器差不多，其实就是一个子类实现类对象

boolean hasMoreElements( ) //判断是否还有枚举元素

Object nextElements( ) //获取枚举元素

二、LinkedList特有的功能：

1、添加功能：

public Object addFirst( Object obj) //在集合第一个位置进行添加

public Object addLast(Object obj) //在集合最后一个位置进行添加 （其实与单独的add方法无区别，因此没啥意义）

2、获取功能：(对最终的集合元素无影响)

public Object getFirst( ) //获取集合中第一个元素

public Object getLast( ) //获取集合中的最后一个元素

3、移除功能：（对最终集合元素有影响）

public Object removeFirst( ) //移除集合中第一个元素

public Object removeLast( ) //移除集合中的最后一个元素

三、删除ArrayList中重复的元素

需求：删除集合中自定义对象的重复值（对象的成员变量值都相同）

为什么自定义对象删除时进行equals判断时容易出错

为什么呢？

判断功能contains( )是集合自动生成的，因此查看源码得知

contains ( )方法其实本质上比较的是equals方法

而学生类中没有equals（）方法，因此默认使用父类Object类的equals（）方法最终比较的地址值

因为每一个对象的地址值均不同，所以未被删除掉

所有按照自我的需求，比较成员变量的值，需要在自定义类中重写equals（）方法

四、用LinkedList实现默认自定义栈集合

五、泛型：是一种特殊的类型，把明确的工作推迟到创建对象或者调用方法的时候才明确的特殊的类型

格式：<数据类型> //这里的数据类型只能是引用类型

好处：

A: 将运行时会出现的问题提前到编译期间

B; 避免了强制转换

C: 优化了程序设计，解决黄色警告线

集合也会模仿数组的数据类型明确，因此这种技术也被叫做是泛型

出现原因：早期的Object可以存储任意类型，进行向上转型时不会出现错误，但是向下转型时容易抛安全问题

六、泛型类

《一》格式：类名<T> //T表明之后要用什么类，就定义成什么

注意：定义的泛型类必须是包装类

1、具体定义的类型（对类进行操作）具体进行初始化new对象 //泛型类

2、对方法进行操作（对方法进行定义），相应的对象不需要进行泛型定义 //泛型方法

3、接口定义成泛型类 //泛型接口

注意：接口定义为泛型类时实现类不需要定义，，当接口不明确类型时，实现类也需要定义为未知类型<T>

《二》泛型的高级通配符

1、泛型如果明确了类型时，前后应该一致

例：Collection<Object> c = new ArrayList<Object>( ) ;

2、泛型如果是用 ？ 表示任意的类型都可以

例：Collection<?> c = new ArrayList<任意类型皆可>( ) ;

3、泛型用 ？ extends E 表示类型只能是 E类或者其子类 向下转型皆可

例：Collection<? extends E> c = new ArrayList<E或者子类>( ) ;

4、泛型如果用 ? super E 表示类型只能是 E或者父类 向上转型

例：Collection<? super E> c = new ArrayList<E或者E的父类>( ) ;

七、JDK新特性

1、自动装箱，自动拆箱

2、增强for : 是for循环的一种

格式：

for(元素数据类型 变量 ： 数组或者Collection集合 ) {

使用变量即可，该变量就是元素

}

优缺点： 好处：优化了数组以及集合的遍历

缺点：容易出现当对象为空时，抛异常

解决方案 ： 使用前先对对象是否为空进行判断

八、静态导入

格式：import 包名.类名.方法名

(一般不使用)

注意：

1、导入的方法必须是静态方法

2、当出现多个同名的方法时候，必须在使用导入方法前加上前缀（java.lang.math.ads(-10)）

九、可变参数

---在定义方法的时候不知道参数有多少个

格式

修饰符 返回值类型 方法名 （数据类型... 变量名）{ }

注意：

1、此处的变量名其实是一个数组

2、如果这个方法有多个参数，并且有可变参数，那么这个可变参数应该定义在最后

3、因为从本质上此处的可变参数是一个数组，数组的长度是固定的，因此当可变参数位于前面的时候表明

数组已经是固定的且不可改变的

十、Arrays工具类中的一个方法 ：将数组转换成集合

public static <T> List<T> asList(T... a）

//经过这个方法将数组转换为集合时，本质上是一个数组，长度是固定的，因此对集合不能进行长度修改有关的操作

//如add、remove

**DAY16 ArrayList----LinkedList总结**

一、vector：底层数据结构是数组，查询快，增删慢

线程安全，效率低

特有方法：

public void addElements(Object obj) ;增加元素

public Element elements( ) ; 枚举 （相当于迭代器）

public boolean hasMoreElement( ) ; 集合中是否还有元素

public void nextElement( ) ; 从集合中取出元素

二、Linkedliset： 底层数据结构是链表，查询慢，增删快

线程不安全，效率高

1、添加元素

public void addFirst(Object obj) ; 把元素添加到第一个位置

public void addLast( object obj) ; 把元素添加到最后(几乎无意义)

2、获取元素

public void getFirst( ) ; 取出集合中的第一个元素

public void getLast( ) ; 取出集合中的最后一个元素

3、移除元素

public void removeFirst( ) ; 移除集合中的第一个元素

public void removeLast( ) ; 移除集合中的最后一个元素

四、静态导包

格式： import 包名. 类名. 方法名

(一般不使用)

注意：导入的方法必须是静态的

如果在一个类中具有相同的方法名，必须在直接在导入的方法中添加前缀（java.lang.Math.abs(-10)）

十、练习题

1、产生10个1-20之间的随机数，要求随机数不可以重复

package dh16;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.Random;

//产生10个1-20之间的随机数，要求随机数不可以重复，并且输出这是个随机数

public class RandomText2 {

public static void main(String[] args) {

//创建集合对象用于存储随机数

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

//定义一个统计变量用于统计是否存储了十个数

int count = 0;

//创建随机数对象

Random random = new Random();

//循环判断是否进行了十次添加操作

while(count<10) {

//生成随机数nextInt(n) 表示0-n范围内int类型的随机数

int number = random.nextInt(20)+1;

//判断生成的随机数在集合中有无重复的数

if(!list.contains(number)) {

list.add(number);

count++;

}

}

//最终遍历集合输出随机数

//迭代器迭代

Iterator<Integer> it =list.iterator();

while(it.hasNext()) {

System.out.println(it.next());

}

System.out.println("===============");

//List集合特有方法遍历

for(int i=0;i<list.size();i++) {

System.out.println(list.get(i));

}

System.out.println("================");

//增强for遍历

for(Integer i : list) {

System.out.println(i);

}

}

}

2、集合的循环嵌套，用泛型实现 ,并且用增强for进行遍历

package dh16;

import java.util.ArrayList;

//集合的循环嵌套，用泛型实现 ,并且用增强for进行遍历

public class QianTaoListText3 {

public static void main(String[] args) {

// 创建大集合

ArrayList<ArrayList<Student>> arrayList = new ArrayList<ArrayList<Student>>();

// 创建单个小集合并且存储相应的对象类型数据

ArrayList<Student> list1 = new ArrayList<Student>();

// 创建第一个集合中要进行添加的学生类对象

Student s1 = new Student("caddo", 21);

Student s2 = new Student("dada", 20);

// 添加对象到集合中

list1.add(s1);

list1.add(s2);

// 将第一个小集合添加到大集合中

arrayList.add(list1);

// 第二个小集合

ArrayList<Student> list2 = new ArrayList<Student>();

// 创建第二个集合中要进行添加的学生类对象

Student s3 = new Student("mike", 21);

Student s4 = new Student("lili", 20);

// 添加对象到集合中

list1.add(s3);

list1.add(s4);

// 将第二个小集合添加到大集合中

arrayList.add(list2);

// 第三个小集合

ArrayList<Student> list3 = new ArrayList<Student>();

// 创建第二个集合中要进行添加的学生类对象

Student s5 = new Student("jack", 21);

Student s6 = new Student("liu", 20);

// 添加对象到集合中

list1.add(s5);

list1.add(s6);

// 将第二个小集合添加到大集合中

arrayList.add(list3);

//遍历

//外层循环对大集合尽心遍历，

for(ArrayList<Student> x : arrayList) {

for(Student y : x) {

System.out.println(y.getName()+" "+ y.getAge());

}

}

}

}

3、键盘录入多个数据，以0结束，要求在控制台输出这些数据的最大值，并且将所有数据进行输出

package dh16;

import java.util.LinkedList;

//创建自定义类实现默认自定义栈

public class MyLinkedListText6 {

//定义集合对象为私有，在其他类中不能通过创建对象来使用

private LinkedList link;

public MyLinkedListText6() {

//通过创建自定义对象实际上创建的是LinkedList的对象

link = new LinkedList();

}

public void add(Object obj) {

link.addFirst(obj);

}

public Object get() {

return link.removeFirst();

//每次移除第一个，并且返回第一个对象

}

public boolean isEmpty() {

return link.isEmpty();

}

}

4、删除ArrayList集合中重复的元素

package dh16;

//删除List集合中重复出现的元素

//方法一、定义新的集合，进行元素添加判断

//注意，判断是否重复使用contains时，必须注意，其实使用的是equals()比较是否相同

//自定义类中没有equals（）方法，则默认使用object类的，若要使用本身的，应该对方法进行重写

import java.util.ArrayList;

public class RemoveSameText5 {

public static void main(String[] args) {

// 创建集合对象

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

// 创建学生对象

Student s1 = new Student("林", 21);

Student s2 = new Student("刘", 20);

Student s3 = new Student("游", 21);

Student s4 = new Student("林", 21);

Student s5 = new Student("吴", 22);

Student s6 = new Student("刘", 20);

// 将学生对象添加到集合中

list.add(s1);list.add(s2);list.add(s3);list.add(s4);list.add(s5);list.add(s6);

//方法一、定义新集合

//创建新集合

ArrayList<Student> l = new ArrayList<Student>();

//遍历原集合进行判断是否在新集合中存在相同的元素

for(int i=0;i<list.size();i++) {

if(!l.contains(list.get(i))) {

l.add(list.get(i));

}

}

//遍历新集合

for(Student s : l) {

System.out.println(s.getName()+"\t"+s.getAge());

}

//方法二、选择排序

for(int i=0;i<list.size()-1;i++) {

for(int j=i+1;j<list.size();j++) {

if(list.get(i).equals(list.get(j))) {

list.remove(j);

}

}

}

System.out.println("===================");

for(Student ss : list) {

System.out.println(ss.getName()+"\t"+ss.getAge());

} }

5、创建一个类，用于实现自定义栈

package dh16;

import java.util.LinkedList;

//创建自定义类实现默认自定义栈

public class MyLinkedListText6 {

//定义集合对象为私有，在其他类中不能通过创建对象来使用

//此时的link对象是一个成员变量

private LinkedList link;

public MyLinkedListText6() {

//通过创建自定义对象实际上创建的是LinkedList的对象

//如果写成以下形式表示的是一个新的局部对象，在其他方法中无法使用

//LinkedList link = new LinkedList();

link = new LinkedList();

}

public void add(Object obj) {

link.addFirst(obj);

}

public Object get() {

return link.removeFirst();

//每次移除第一个，并且返回第一个对象

}

public boolean isEmpty() {

return link.isEmpty();

}

}

测试类

package dh16;

//自定义栈测试类

public class Tool {

public static void main(String[] args) {

MyLinkedListText6 text = new MyLinkedListText6();

text.add("hello");

text.add("java");

text.add("world");

while(!text.isEmpty()) {

System.out.println(text.get());

}

}

}

**DAY17 SET-----TreeSet---HashSet**

一、Set集合

要求：一个不包含重复元素的 collection(不允许重复出现相同的元素)

无序（元素进行存储和取出的顺序不一样）

元素唯一性

二、HashSet ：存储数据是无序并且唯一的

问题：为什么存储字符串内容时只存储了一个呢？

通过查看add方法的源码，我们知道这个方法实际上是利用了hashCode( ) 和 equals（）方法

步骤;

首先比较哈希值

如果相同，比较地址值或者用equals（）方法

如果不同，就添加到集合中

按照方法步骤来说：

A：先看hashCode( )值是否相同

相同; 用equals方法比较

返回true : 代表元素重复，不添加

返回false: 添加到集合

不同：直接添加到集合

/-------如果自定义的类没有重写这两个方法，默认使用Object类的方法（比较不出是否相同）

为什么String可以进行方法的使用？

因为String重写了方法

三、LinkedHashSet : 底层数据结构由哈希表和链表组成

哈希表保证元素的唯一性

链表保证元素有序（存储和取出顺序相同 ）

四、哈希表：一种有算法计算出来的哈希值 （相当于新华字典查找元素）

五、TreeSet : 能够对元素按照某种规则进行排序

特点： 元素排序且唯一

排序两种方式：

A：自然排序 （实现Comparable接口就是自然排序） 构造方法默认使用自然排序

// 通过源码可知，add( )方法里最终排序调用的是compareTo( )方法

//真正的比较依赖的于元素的compareTo（）方法，而这个方法是定义在Comparable接口里的

//所以如果要重写该方法，就必须实现Comparable接口，这个接口表示的就是自然排序

B：比较器排序

有参构造：

TreeSet(Comparator<? super E> c);

构造一个新的空 set，该 set 根据指定的比较器进行排序。

1、@@ (Comparator<? super E> c代表的是接口实现类的父类或者本身

其实就是定义了一个实现Comparable接口的类，对这个类进行创建对象

2、格式为

TreeSet<Student> set = new TreeSet<Student>(new Comparable接口的实现类)；

此时应该在接口实现类中重写接口里的compare（）方法

@@@@@@ 注意：如果一个方法的参数是接口，其实要的是接口的实现类对象

匿名内部类可以实现该操作

六、TreeSet ：底层是二叉树结构（红黑树是一种自平衡的二叉树）

元素如何存储进去？

1、第一个元素作为根节点进行存储

2、从第二个元素开始，每个元素都和根节点进行比较

大 就作为右孩子

小 就作为左孩子

相等 就不添加

元素如何取出？（前序遍历，中序遍历，后序遍历）

前序遍历：

1.访问根节点

2.前序遍历左子树

3.前序遍历右子树

中序遍历：

1.中序遍历左子树

2.访问根节点

3.中序遍历右子树

后序遍历：

1.后序遍历左子树

2.后序遍历右子树

3.访问根节点

七、TreeSet集合保证元素排序唯一的原理

唯一性：是根据比较返回的0来决定的

排序：

A：自然排序;(元素具备比较性)

让元素所属的类去实现自然排序接口Comparable

B：比较器排序;(集合具备比较性)

让集合构造方法接收一个比较器接口的子类对象Comparator

**DAY17 Set总结**

一、登录注册案例

二、Set集合（理解）

1、元素是唯一的（不允许重复）

2、集合是无序的

三、HashSet ：存储无序并且唯一

-----存储数据时是依靠什么保证数据的唯一性的？

通过查看add( )方法的源码可以知道实际依据的是hashCode( ) 和 equals ( ) 方法

-----步骤：

1、先比较哈希值是否一样？

是：接着比较地址值或者用equals( )方法；

如果返回的是true；表示元素重复不添加

如果返回的是false；添加元素到集合

否：添加元素到集合

-------------------------

为什么String类进行添加时可以直接判断是否重复？

因为String类自动重写了hashCode( ) 和equals( ) 方法；

如果要实现自定义对象类的话，必须重写这两个方法；

四、TreeSet ：底层数据结构是一种红黑树（自平衡的二叉树）

元素无序且唯一；

----------------元素是如何存储进去的？

1、第一个元素作为根结点

2、从第二个元素开始每个元素和根节点进行比较

大了 就作为右孩子

小了 就作为左孩子

相等 就不添加

---------------

五、LinkedHashSet : 底层数据结构是哈希表和链表

1、哈希表确认元素的唯一性

2、链表确认元素的有序性

六、TreeSet排序:

1、自然排序

A：是依据集合无参构造的，定义无参构造，默认实现的是自然排序

自定义的类就要实现接口Comparable( ) ，并且重写compareTo( )方法

注意：public int compareTo(自定义类 类对象 )

返回的是一个int类型的值，如果返回0表示元素相同，不添加，需要自己重写方法里的代码实现功能

2、比较器排序

A：是依据集合有参构造形式产生的

格式TreeSet<T> set = new TreeSet<T>(comparator <? super E> c); //接口实现类的父类或者本身

其实就是定义一个类，用于实现Comparable( )接口，实现类再进行重写compare( )方法

TreeSet<T> set = new TreeSet<T>(new 接口实现类对象);

如果一个方法需要实现了接口的参数方法，其实要的是接口的 实现类对象

可以用匿名对象进行操作

七、针对Collection集合我们到底如何使用呢？

唯一吗？

是：Set

排序吗?

是：TreeSet

否; HashSet

如果你知道是Set,但是不知道是哪个Set,使用 HashSet

否：List

要安全吗？

是：Vector

否; ArrayList 或LinkedList

查询多：ArrayList

增删多：LinkedList

如果知道要用List , 但是不知道用哪个，就用ArrayList

如果知道要用集合，不知道用什么就用ArrayList

八、TreeSet排序

是根据方法返回的int类型值进行判断的，0表示相同

1、自然排序（作用于元素上的）

用于元素定义类上实现接口，重写方法的；

2、比较器排序（作用于集合对象上的有参构造）

需要的是一个接口实现类，并且集合通过有参构造的（匿名内部类）

**DAY18 Map集合**

一、Map<K , value> 接口

特点：存储键值，获取元素, 无序

将键映射到值的对象.一个映射不能包含重复的键；每个键最多只能映射一个值。

注意：键值对应是一个Set集合(元素唯一性)，value对应的是一个List集合（元素可以重复）

二、 Map和Collection 集合的区别？

1、Map集合存储元素是成对存在的，键值不能重复，只能对应一个元素，元素可以重复

Collection集合：存储元素是单独出现的，元素可以重复（List）也可以不重复(Set )

2、Map集合数据结构是针对键值，跟值无关

Collection集合数据结构是针对元素有效

三、Map的功能概述

1、添加功能：

V put(K key, V value) ; 添加元素

注意事项：当键是第一次存储就返回null ， 当键不是第一次存储就把第一次的元素进行替换(如果输出时是返回第一次的元素)

2、删除功能

void clear( ); 移除所有的键值对元素

V remove(Object key) ; 根据键值移除掉键值元素对，并返回

3、获取功能：(V代表泛型)

Set<Map.Entry<K,V>> entrySet() ; 获取所有的键值和元素

V get(Object key) ; 根据键值获取元素

Set<K> keySet() ; 获取集合元素中所有的键值 ,并且存储到集合中

Collection<V> values() : 获取集合中的所有元素

4、判断功能：（包含返回true）

boolean containsKey(Object key) ; 判断集合是否包含特定的键值

boolean containsValue(Object value) ; 判断集合是否包含特定的元素值

boolean isEmpty() ; 判断集合是否为空

5、长度功能

int size( ) ; 返回集合中的键值对长度

四、Map集合的遍历：

方式1：根据键找值

1、获取所有键的集合

2、根据键的集合获取每一个键值

3、根据键值输出元素值

方式2：根据键值对对象找值

1、获取所有键值对象的集合

2、遍历键值对对象的集合，获取每一个键值对对象

3、根据键值对象找键和值

五、HashMap: 是基于哈希表的Map接口实现

特性：哈希表用来保证元素的唯一性

注意事项：HashMap对键进行的操作，具有唯一性，如果要进行键的添加put( ) 方法操作时则要依靠hashCode( ) 和equals( ) 方法

而如果键的数据类型是字符串类型，或其他包装类，可以直接操作，因为自动重写了那两个方法

但是如果是自定义类型的话，需要重写hashCode( ) 和equals( )方法

六、LinkedHashMap : 是Map接口的哈希表和链表实现

哈希表：保证元素是唯一的

链表：保证键值是有序的(有序不代表排序);

八、TreeMap: 键是红黑树，保证键是唯一的;

1、自然排序：无参构造实现自然排序,键值所在类实现comparable接口 ,并且在类中重写compareTo() 方法

2、比较器排序：创建集合有参构造默认实现比较器排序,需要创建接口的实现类对象,,接口实现类重写compare()方法，

因为实际上需要的实现类对象，而操作方法的参数是接口的方法，

可以通过创建匿名对象类创建对象，并重写方法

九、集合嵌套集合

1、HashMap嵌套HashMap集合

2、HashMap嵌套ArrayList集合

3、HashMap 嵌套 HashMap 再嵌套ArrayList集合

十、面试题

1、HashMap 和Hashtable 的区别：

HashMap ：线程不安全，效率高，允许null键和null值

Hashtable ：线性安全，效率低，不允许null键和null值

2、List , Set , Map 等接口是否都继承Map接口?

List , Set : 继承的是Collection 接口

Map : 其本身就是一个顶层接口

十一、Collections； 是针对集合进行操作的工具类，都是静态方法

十一、Collection和Collections 的区别？

Collection : 是单列的顶层集合接口，有子接口List和Set

Collections : 是对集合进行操作的工具类，有对集合进行排序和二分查找的方法

十一、Collections

public static <T> void sort(List<T> list) ; 排序默认使用自然排序

public static <T> int binarySearch(List<?> list ,T key ) : 二分查找

public static <T> T max (Collection<?>coll) ; 最大值

public static void reverse(List<?> list) ;反转

public static void shuffle(List<?> list) ; 随机置换

十二、ArrayList存储自定义对象并且进行排序

注意：ArrayListu元素是有序，指代的并不是元素的大小或其他格式的排序，而是存储和取出的顺序是相同的

因此称为是有序的，，要对其进行排序可以使用Collections集合操作类对集合进行排序，使用其静态方法sort( )

但是此操作类对集合进行排序操作时，是依靠Comparable接口的，所以自定义的类实现接口，还要重写接口的方法

重写compareTo( ) 方法

十三、集合的特点，数据结构的总结

1、集合分为两个体系：Collection （单列集合） 和Map（双列集合)

Collection : 单列集合

List : 有序，可以重复

ArrayList : 底层数据结构是数组，查询快，增删慢

线程不安全，效率高

Vector ： 底层数据结构是数据，查询快，增删慢

线程安全，效率低

LinkedList ： 底层数据结构，查询慢，增删快

线程不安全， 效率高

Set : 无序，元素唯一

HashSet : 底层数据结构是哈希表

比较元素是否重复，保证元素唯一性

哈希表依赖于两个方法进行判断hashCode( ) 和equals ( ) 方法

1、先比较哈希值hashCode( )

· 相同，再接着比较地址值或equals( ) 方法

返回true； 代表元素重复，不添加

返回false : 代表元素可以进行添加操作

不同 ： 直接添加元素到集合中

注意：如果是自定义的类，需要重写以上两个方法，可以直接系统生成

TreeSet : 底层数据结构是一种二叉树（红黑树是一种自平衡的二叉树）

二叉树，进行排序操作 根据返回值是否为0 ，进行排序

自然排序：(对元素所在的类进行实现接口Comparable)

依赖于无参构造，实现接口Comparable，重写compareTo（）方法

比较器排序 (集合具备比较性，让集合接收一个comparator实现类对象)

依赖于有参构造，产生接口的实现类，重写接口的compare( ) 方法

注意，最重要的是接口的实现类对象，所以直接使用匿名内部类进行操作，重写compare( ) 方法

linkedHashSet : 底层数据结构是链表和哈希表

特点： 由链表确定元素的有序性，有哈希表确认元素的唯一性

Map： 双列集合 是对键值进行的操作，产生的是键值对 , 数据结构仅仅针对键有效，值无效

一种键和值进行映射的关系，键值不允许重复，元素值可以是重复的，每个键值最多对应一个元素值

HashMap : 线程不安全，效率高，允许键为null，值为null

键的数据结构是一种哈希表，保证键值唯一

对键进行添加操作是，使用put(T key , T value) ；方法中，依赖的是两个方法hashCode( ) 和equals（)

最终系统进行自动生成即可

linkedHashMap : 线程安全，效率低，不允许键值和元素为null

键数据结构是一种，链表和哈希表

链表确认键值存取和取出顺序相同，哈希表确认键值唯一性

和上述HashMap : 差不多用法

TreeMap : 键的数据结构是红黑树

自然排序：(对元素所在的类进行实现接口Comparable)

依赖于无参构造，实现接口Comparable，重写compareTo（）方法

比较器排序 (集合具备比较性，让集合接收一个comparator实现类对象)

依赖于有参构造，产生接口的实现类，重写接口的compare( ) 方法

注意，最重要的是接口的实现类对象，所以直接使用匿名内部类进行操作，重写compare( ) 方法

**DAY19 IO流**

一、异常 Throwable 接口

严重问题：Error 不处理，一般是很严重的问题,与写程序基本无关

问题：Exception

编译期问题 : 不是RuntimeException的异常 必须进行处理的，如果不进行处理，编译就不能通过

运行期问题 : 是RuntimeException 不处理，这个问题的出现是代码不够严谨，需要修改代码

把异常的信息显示在控制台，属于哪种异常问题，异常的名称以及异常出现的原因和名称

但是会把会终止其他部分的操作

二、处理异常的方案

1、try ..... catch..... finally

2、throws 抛出异常

三、try .... catch ... finally 的处理方式

1、格式：

try{

可能出现问题的代码块;

} catch (异常处理的变量){

针对问题的处理;

} finally {

释放资源;

}

2、变形的格式：

try{

可能出现问题的代码；

} catch （异常处理的变量）{

针对问题的处理;

}

3、注意：try 里面的代码越少越好，但是catch里面一定要有内容, 给出提示语句

一个try可以对应多个catch，一旦try里面出现问题，就把问题抛出去，就和catch里面的问题进行比较

一旦有匹配的就执行catch里的处理，然后结束try......catch

继续运行其他的语句

也就是说当一个catch里的内容匹配到了，其他catch的内容就不会进行比较

A ： 当不知道异常是哪种问题是可以用所有异常的父类Exception 进行匹配判断，但是尽量能明确就明确

因为用父类异常时，并不能知道具体出了什么问题

B；平级关系的异常，谁前谁后没有关系，如果有子父关系，一定把子问题放在前面在找大的问题

C ： 新特性

try {

可能出现问题的代码

} catch（异常类1 | 异常类2 .... 变量 ） {

针对问题的处理

}

注意：这种方式，虽然简洁，但是也不是很好

a : 处理方式的方法是一样的，也就是说这是针对同种类似问题的处理

b: 只能是同级关系的异常类

三、throwable 异常处理

1、在try里面出现问题后，jvm会帮我们生成一个异常对象，然后将这个对象抛出，和catch里面的内容进行比较

如果该对象是某这个类型的，就会执行catch里面的语句

2、异常中要了解的方法：

public String getMessage() 返回异常的详细消息字符串

public String toString ( ) ； 返回异常的简单信息描述

此对象的类的name (全路径名)

": " (冒号和一个空格)

调用此对象getLocalizedMessage( ) 方法的结果（默认返回的是getMessage( ) 方法的内容

（重要并且常用的方法）printStackTrace( ) 获取异常名和异常信息，以及异常出现的位置，返回值为void

四、throws处理异常方案

抛出：运用场景为，程序有问题但是是我们解决不了的问题，针对这类问题就进行另一种方案： 抛出

格式：

throws 异常类名

注意： 这个格式必须在方法的后面

在方法上申明，是为了告诉调用者，方法存在问题

注意：尽量不要在main方法上进行异常抛出

小结：

编译时期的问题调用时需要处理

运行期异常要进行处理

五、throw :如果出现异常情况，我们可以把异常抛出，但此时抛出的应该是异常类的对象

作用在方法内部抛出异常对象

六、throws 和 throw 的区别

1、throws ；

用在方法申明上，跟的是异常类名

可以跟多个异常类名，用逗号隔开

表示抛出异常，由方法的调用者来处理

throws表示出现异常的一种可能性，但是不一定会出现这个问题

2、throw ：

使用在方法体内部的，此时抛出的异常类对象

只能抛出一个异常对象名

表示抛出异常，由方法体内的语句处理

七、到底使用哪种处理方式？

区别：

如果后续程序需要继续运行就try ，一般在main方法中使用try

如果后续程序不需要运行就throws

八、finally 的作用及用法

特点：

被finally控制的语句一定会被执行

注意：如果执行在finally之前jvm退出了，就不能执行finally里的语句

作用 : 用于释放资源

九、final 和 finally 和 finalize 的区别 (面试题)

1、final ； 表示最终的意思

修饰类：表示类不能被继承

修饰方法 : 表示方法不能被重写

修饰变量：变量是常量

2、finally ：

是异常处理的一部分，一般被其控制的语句会被执行，除非在这之前虚拟机退出了

3、finalize：是Object类的一个方法，用于进行垃圾回收，和System类的 gc( ) 方法（运行垃圾回收器）一起使用

4、如果在catch里面有return语句，那么请问finally里面的的代码还会执行吗？

如果会，请问是在return前还是return后

答 ： 会执行，并且在return前

十、自定义的异常

自定义的类要实现异常处理，必须继承Exception 或者RuntimeException

注意事项：

A; 如果子类重写父类的方法时，子类方法必须抛出相同的异常或父类异常类的子类

B; 如果父类抛出了多个异常，子类重写父类时，只能抛出相同的异常或者父类异常类的子类，子类不能抛出父类没有的异常

C; 如果被重写的方法在父类中没有抛出异常，那么子类的方法绝对不可以抛出异常，如果子类方法中

有异常出现只能try ... catch ， 不能throws

十一、IO 类的File

File：文件和目录（文件夹）路径名的抽象表示

构造方法：

File(String pathname) ; 根据一个路径的到File对象

File(String parent , String child) 根据一个目录和子文件目录得到File对象

File(File parent ,String child ) 根据一个父File对象，和一个子文件/目录得到File对象

成员方法：

1、创建功能 (注意在文件夹中创建文件，文件夹必须存在，否则出错) (A 和 B 中的上一级目录不存在，下一级就无法创建 )

A: public boolean creatsNewFile( ) ; 创建文件 , 如果存在就不创建了，返回false （方法抛异常）

B：public boolean mkdir( ) ; 创建文件夹 ; 如果存在就不创建了，返回false (方法不抛异常)

C：public boolean mkdirs( ) ; 创建多级文件夹 ， （之前的文件夹不存在也可以创建）

2、删除功能：（如果创建文件或者文件夹忘了写盘符路径，那么，默认在项目路径下）

A：public boolean delete ( )

注意：Java中的文件删除不走回收站

如果要删掉文件夹，必须把其子文件夹或者文件删除

3、重命名功能;

A: public boolean renameTo(File dest) ; 要创建新的File对象

注意：路径以盘符开始: 决定路径 c:\\a.txt

路径不以盘符开始： 相对路径 (可以进行剪切和改名)

4、判断功能：

A：public boolean isDirectory( ) : 判断是否是目录

B：public boolean isFile( ) : 判断是否是文件

C：public boolean exists( ) ; 判断是否存在

D：public boolean canRead( ) ; 判断是否可读

E：public boolean canWrite( ) ; 判断是否可写

F：public boolean isHidden( ) : 判断是否隐藏

5、获取功能：

A：public String getAbsolutePath( ) ；获取绝对路径

B: public String getPath( ) :获取相对路径

C：public String getName( ) :获取名称

D：public long length( ) ; 获取长度，字节数

E：public long lastModified( ) 获取最后一次修改时间的毫秒值

6、高级获取功能：

A：public String[ ] list（）；获取指定目录下的文件或文件夹的名称数组

B：public Flie[ ] listFiles( ) ; 获取指定目录下所有文件或文件夹的File对象数组

文件名称过滤器：是一个接口

public String[ ] list(FilenameFilter filter)

public Flie[ ] listFiles( FilenameFilter filter) ;

需要的是接口实现的对象，并且接口实现类要进行方法的重写，所以在创建接口实现类对象时，可以采取匿名内部类

**DAY20 递归----IO流------字节输出流，字节输入流**

一、递归：方法定义中调用方法本身的现象

注意事项：A：递归一定要有出口，否则就是死递归

B：递归的次数不能太多，否则内存溢出

C：构造方法中不能递归使用

二、递归的思想以及解决问题的思路

分解法

合并法

三、解决条件

1、递归必须要有方法

2、要找到递归的出口

3、找到递归的循环条件

四、例题

1、不死神兔

2、递归获取指定目录下文件后缀为 " . Java" 的文件 , 输出满足条件的所有文件的绝对路径

3、递归删除带内容的目录(和取文件差不多)

五、IO流：用于进行设备间的数据传输问题(上传或者下载)

1、按流向分类

输入流：读取数据

输出流：写出数据

2、数据类型

字节流

字节输入流 读取数据 InputStream 基类 是一个抽象类

字节输出流 写出数据 OutputStream

字符流：

字符输入流 读取数据 Reader

字符输出流 写出数据 Writer

为了方便操作文本数据，Java提供给了字节流

如何判断使用哪种类型的流？

-----如果要进行操作的数据是文本类型，就使用字符流

把你要进行操作的文件用记事本打开，如果是可以读懂的数据就用字符流，否则用字节流

-----如果你什么都不知道就用字节流

六、字节输出流：OutputStream是一个抽象类，要找到其实现类FileOutputStream

FileOutputStream:

构造方法：

FileOutputStream（File file）;

//第二种方式相当于调用系统功能自动创建了文件，创建指向于这个文件的对象

FileOutputStream(String name);

//追加写入内容

FileOutputStream(File file , boolean append) ; 第一个参数可以是上面构造方法的任意参数

第二个参数代表是否在第一次的末尾进行追加写入

成员方法

//将字符串变成字节数组 String类的功能

public byte[ ] getBytes()

//写入数据

public void write(byte[ ] bytes) ; 将字节数组写入到文件中

public void write(int b) ; 将字节写入到文件中 （写入的是字节）

public void write(byte[ ] b , int off ,int len ) ; 将字节数组的一部分写入到文件

//关闭功能

close ( ) ; 关闭流操作 , 释放资源

作用：让流对象变成垃圾，这样就可以被垃圾回收器回收

通知系统 释放该文件资源

字节输出流步骤：1、创建字节输出流对象 2、写入数据 3、释放资源

如何实现数据换行呢？

写入换行符号即可

不同的系统换行的识别方式不同

Windows：\r\n

Linux : \n

Mac : \r

如何实现数据的追加写入？

构造方法

FileOutputStream(File file , boolean append) ; 第二个参数为true代表允许追加，否则不允许追加

七、字节输入流：读取数据并输出在控制台

FileInputStream:

构造方法： 创建对象指向文件进行读取 ， 一般使用第一种形式 , 前提条件此时文件必须先存在，否则抛异常

FileInputStream(String name );

FileInputStream(File file);

成员方法：

//读取数据

int read() ； 读取出的是int类型 (一次读取一个字节)

如果此时返回的是-1 代表读到了末尾没有数据可读了

int read(byte[] b) 一次读取一个字节数组

int read(byte[] b, int off, int len) 一次读取字节数组的一部分

字节输入流步骤：1、创建字节输入流对象 2、读取数据 3、释放资源

一次读取一个字节 int read( );

int by = 0 ;

//表示有数据可以读

while((by=输入流对象. read()) !=-1） {

System.out.print(by);

}

一次读取一个字节数组 int read(byte[ ] b) ; 返回的实际读取字节的长度（最后还剩几个打出来可能就是几个）

//数组的长度一般是1024或者是1024的整数倍，，，1024代表1k个字节

byte[ ] bytes = new byte[1024];

int len = 0;

//代表有数据可以读取, 返回的是字节的实际长度

while((len = 输入流对象. read(bytes))!= -1) {

//使用String类的构造方法，将byte类型的数组的一部分，或者byte数组转化为 String 类型输出

System.out.print(new String(bytes, 0 , len));

}

八、字节缓冲流 ; 出现原因，读取字节数组比读取字节更快，所有java提供了高效缓冲区

1、字节缓冲输出流 ； 写入数据

BufferedOutputStream （其构造方法可以指定大小，但是一般用不上）

构造方法 ：BufferedOutputStream(OutputStream out) //这里可以使用FileOutputStream对象, 相当于OutputStream的匿名对象

2、字节缓冲输入流：读取数据

BufferedInputStream

构造方法：BufferedInputStream（OutputStream out）//创建匿名的对象(FileInputStream)

**DAY 21 IO字符流**

一、字节流读取数据可能出现的小问题

1、字节流进行一次读取一个字节时，如果不进行强制类型转换，那么读取的返回值是int类型不会出错

2、一次读取一个字节时，当进行强制类型转换为（char）类型时，当遇到中文时会出现乱码

3、因为中文字符是降阶两个字符的，一次取一个字符是取不到中文字符的

二、字符流=字节流+编码表

三、编码表：由现实世界的字符和对应的数据组成的一张表

1、ASCII码表：最高位为符号位，其余为数值位

2、ISO-8859-1 ：拉丁码表 8位表示一个数据

3、GB2312 简体中文表

4、GBK 中文编码表升级，融入更多中文字符 (简体中文)

5、GBI18030 ： GBK的取代版本

6、BIG-5码 ： 通行于台湾，香港的一个繁体字节编码方案，俗称五大码

7、Unicode :国际标准码，融入多种字符 , 所有文字两个字节表示

8、UTF-8 ：最多用三个字节表示一个字符(能一个，也可以两个，最多三个)

四、String类的编码问题

String (byte[ ] bytes , String charsetName) ; 通过指定的字符解码成字节数组

byte[ ] getBytes(String charsetName) ; 使用指定的字符集合，将字符串转换成字符数

编码：把看的懂的变成看不懂的

String ...... byte[ ]

解码：把看不懂的变成看的懂的

byte[ ] .... String

五、转换流：

OutputStreamWriter 写数据

构造方法：

OutputStreamWriter(OutputStream) 根据默认编码把字节流的数据转换成字符流

OutputStreamWriter(OutputStream out,String characsetName) , 根据指定编码，将字节流转换为字符流

成员方法:

close( ) ;关闭

flush( ) ; 刷新 (字符流写数据记得flush)

public void write(int c) ; 一次写入一个字符

public void write(char[ ] ch) ; 一次写入一个字符数组

public void write(String str ,0, len) ; 一次写入一个字符串的一部分

public void write(String str ) ；一次写入一个字符串

public void write(char[ ] ch ,0, len) ; 一次写入字符数组的一部分

InputStreamReader 读数据 ， 是字节流通向字符流的桥梁

构造方法：

InputStreamReader(InputStream input ) ; //按照默认编码读取

InputStreamReader(InputStream input , String characsetName) ; //指定编码进行解码操作

成员方法：

close( ) ;释放资源

int read( ) ; 一次读取一个字符

int read(char[ ] chs , int offset , int length) ; 一次读取字符数组的一部分，也可以代表一次读一个字符数组

六、转换流

FileWriter 对应上述OutputStreamWriter一样的功能 ， 并且和 FileOutputStream的构造方法一样

FileReader 对应上述InputStreamReader一样的功能 ，并且和FileInputStream的构造方法一样

七、高速缓冲字符输入输出流

1、BufferedWrite ; 高速缓冲输出流

构造方法：

BufferedWrite buffer = new BufferedWriter(OutputStream out) ; //这里直接用FileWriter

2、BufferedReader : 高速缓冲输入流

构造方法：

BufferedReader bufferRead = new BufferedReader(InputStream in) ; 这里建议使用FileReader

八、字符缓冲流的特殊功能

1、BufferedWrite void newLine( ) ;根据系统来决定分隔符

2、BufferedReader String readLine( ) ; 一次读取一行数据

注意 ：根据换行或者回车符，表示终止，如果到达字符串末尾就返回null

之前的操作都不要println 换行，，，但是此处必须使用println

readLine( ) 本身没有提供换行

代码：

String line = null ;

while((line=字符输入流对象 . readLine( ) )!=null) {

System.out.println(line) ; 注意此处一定要是println

}

**DAY 22 数据输入输出流**

一、数据输入输出流

可以进行基本数据类型的书写

1、数据输出流 DataOutputStream :

父类是FileOutputStream 爷爷是OutputStream

2、数据输入流 DataInputStream：

父类是FileInputStream 爷爷是InputStream

特有的读取数据和写入数据的方法

例如 writeInt(int v) 一次写入一个int类型的数据

二、打印流 ; 只能进行写，不能进行读（只能进行输出，不能进行输入）

字节打印流 PrintStream

字符打印流 PrintWriter

通过构造方法进行自动刷新

1、可以操作任意类型的数据

print（）；打印任意数据类型的参数

println( ) 方法

2、启动自动刷新

//通过构造方法实现自动刷新, 是要使用println( ) ,或者printf( ) 或者format( )方法

PrintWriter(Writer out, boolean autoFlush)

//不仅实现了数据的自动的刷新还实现了数据的换行

三、合并流 （只能进行读取） 读取字节或者字节数组

1、SequenceInputStream(InputStream s1, InputStream s2)

SequenceInputStream（将按顺序读取这两个参数，先读取 s1，然后读取 s2）

2、构造方法摘要

SequenceInputStream(Enumeration<? extends InputStream> e)

该参数必须是生成运行时类型为 InputStream 对象的 Enumeration 型参数。

由此可知，在集合Vector中用到了枚举，因此使用集合将 InputStream 对象存入到集合

四、序列化和反序列化

1、序列化流：将对象按照一定的方式存入到文本文件中或者在网络中传输，对象------流数据

ObjectOutputStream()

2、反序列化：将文本文件中的流对象数据或者网络中的流对象数据还原成对象 ； 流数据-----对象

ObjectInputStream()

五、Properties : 属性集合类 ，是一个可以和IO流结合使用的集合类

Properties ： 可保存在流中或者从流中加载，属性列表中及其对应的值都是一个字符串

//

注意：是Hashtable的子类，也就是一个Map集合

构造方法

Properties （） 构造一个无参的属性集合

成员方法

Object setProperty(String key, String value) // 调用 Hashtable 的方法 put。//添加元素

public String getProperty(String key ） ；获取元素

public Set<String> StringPropertyNames( ) ; 获取所有键的集合

//和IO流的结合使用

public void load(InputStream input) ; // 从输入流中读取属性列表（键和元素对 )。

public void load(Reader reader)

public void store(OutputStream out, String comments) 以适合使用 load(InputStream) 方法加载到 Properties 表中的格式，将此 Properties 表中的属性列表（键和元素对）写入输出流。

**DAY 23 线程**

一、多线程（方式一） jvm虚拟机是抢占式执行

1、自定义类继承Thread类

2、自定义类重写run方法

3、创建对象

4、启动进程

注意事项：run () 方法是封装被线程执行的代码，直接调用是普通方法

start()首先启动了线程，然后用jvm去调用该线程的run( ) 方法

二、线程的基本获取和设置线程姓名

public final String getName( );

public final void setName(String name);

//相当于构造方法

获取main方法的所在的线程名

public static Thread currentThread( ) //获取当前正在执行的线程对象

//Thread.currentThread( ) .getName( ) ; //获取当前线程的名称

三、获取和设置线程优先级：

public final int getPriority( ) //获取线程的优先级

public final int setPriority(int newPriority) //设置优先级

注意：

线程默认优先级是5

线程优先级的范围是：1-10

线程优先级仅仅表示线程获取CPU时间片的机滤高，但是要进行多次运行才能得到结果

四、线程休眠:

public static void sleep(long millis); 在指定毫秒值内让线程休眠 写在run方法里面

五、线程控制加入进程

public static void join( ) ;等待此进程结束 ，

必须在下一个进程的start( ) 之前

六、进程礼让

public static void yeild( ) ；暂停当前正在执行的线程对象，并执行其他线程

可以使线程运行更和谐，但是不能保证一人一次

写在run方法里

七、守护线程

public final void setDaemon(boolean on) ;将该线程标记为守护线程或者是用户线程

当正在运行的线程都是守护线程时，Java虚拟机退出，该方法必须在启动线程前使用

八、多线程 （方式二）

自定义的类实现接口Runable接口

步骤；

1、自定义类实现接口Runable接口

2、重写run( ) 方法

3、创建自定义类对象

4、创建Thread 类的对象，并且把步骤3的对象 加入到Thread中作为参数进行传递

5、启动线程

九、线程出现安全问题的原因？

1、是否有多个线程环境

2、是否有共享数据

3、是否有多条语句操作共享数据

解决方法：同步机制；同步代码块 (锁)

格式：synchronized(对象) {

需要同步的代码；

}

A：对象是什么？ private Object obj = new Object( ); //定义为成员对象 随便那个对象

B：需要同步的代码？多条语句操作共享数据部分

注意：同步可以解决安全问题就在于那个对象上

多个线程必须是同一把锁

同步方法：将关键字定义在方法上

public synchronized 返回值类型 方法名（ ）{

多条语句操作共享数据代码；

} 同步方法上没有对象，但是在类中默认有一个对象，，，用this 表示本类中的对象

静态方法锁对象？

类的字节码文件对象

十 、将线程不安全的集合变成线程安全的集合

使用Collections 的静态方法

public static List<T> synchronized(List<T> list> ；返回值List

同里可以对Set集合和Map集合操作

十一、Lock锁

public void lock( ) ;获取锁

public void unlock( ) ；释放锁

创建对象: Lock lock = new ReentrantLock() ; 接口的实现类

十二、同步的弊端：

A：效率低

B：容易产生死锁

死锁：

两个或两个以上的线程在争夺资源的过程中，发生的一种进程等待的现象

**DAY 26 网络编程**

一、网络编程

就是用来实现网络互连的不同计算机上运行的程序之间可以进行数据的交换

二、InetAddress

此类表示互联网协议 (IP) 地址

注意：

如果一个类没有构造方法：

1、成员全部都是静态的 (Math,Arrays,Clooections)

2、单列设计模式(Runtime)

3、类中有静态方法返回该类的对象

成员方法：

public static InetAddress getByName(String host ) : 根据主机或这IP地址的字符串获取ip地址的对象

public String getHostName( ) ; //获取主机的主机名

public String getHostAddress( ) //获取主机的IP地址的字符串表示形式

三、Socket编程：网络编程 ，套接字编程

scoket包含了：IP地址和端口

scoket原理机制：

通信的两端都要Scoket

网络通信起始就是Scoket之间的通信

数据在两个Scoket间通过IO传输

四、UDP用户协议发送数据：

A; 创建发送端Scoket对象

B：创建数据，并将数据进行打包

C：调用Scoket对象的发送方法发送数据包

D：释放资源

创建Socket对象 ：

DatagramScoket

创建数据包对象

DatagramPacket(byte[] buf , int length , InetAddress address , int port) ;

buf：代表发送的字节数组（也就是要发送的内容） //包数据

length：数据包的长度 // 包长度

address ：表示接收端的IP地址 //目的地址

port：目的端口号

//从此套接字发送数据报包

public void send(DatagramPacket p)

五、UDP协议接收数据 （DatagramSocket）

A：创建结束端Scoket对象

B：创建一个数据包（接收容器）

C；调用Scoket对象的接收方法接收数据

D：解析数据包，并显示在控制台

E：释放资源

//创建Scoket对象，并且绑定在指定的端口

DategramScoket(int port) ;

//创建一个数据包（接收容器）

DategramPackage(byte[] buf , int length)

//调用Scoket对象接收方法接收数据

public void receive( DatagramPacket p )

//解析数据包，并且显示在控制台

//获取对方的IP (数据包的方法)

public InetAddress getAddress( )

//解析数据包，并且显示在控制台

public byte[ ] getData( ) 获取数据缓冲区

public int getLength( ) ；获取数据的实际长度

注意：一定是要先执行发送端的程序'

1、多次启动接收端，会抛异常，显示端口号被占用

2、如果接收端是键盘录入的话，套一个while(true) { } 循环

3、此时接收端因为要一直开着，所以不需要释放资源

注意二、发送方代码改进

1、封装键盘录入数据

BufferedReader(new InputStream(System.in)) ； 键盘录入数据，进行读取

String line = null;

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

byte[] bytes = line.getBytes( );

//创建数据包对象

//发送数据

}

六、TCP协议传输数据并且读取 (发送端Socket)

TCP协议发送数据 :

A：创建Socket对象

Socket(InetAddress address, int port) 创建一个流套接字并将其连接到指定 IP 地址的指定端口号。

Socket(String address , int port) ; //通过连接套接字指向指定的IP地址的端口号

这一步如果成功，就说明连接已经成功了

B：获取输出流，写数据

public OutputStream getOutputStream() // 返回此套接字的输出流。对象

输出流对象.write( )

C：释放资源

注意事项 ；TCP协议发送数据是一定要先开服务器 ，

Tcp协议接收数据

A；创建接收端的Socket对象

//服务器端口

ServerSocket(int port) ; //创建绑定端口的服务器套接字

B ：监听客户端连接，返回一个对应的Socket对象

public Socket accpet( ) ; // 创建监听并接收此套接字，返返回一个套接字的对象

C：获取输入流 ， 读数据

//用返回的套接字的对象，来调用输入流。读数据

public InputStream getInputStream( ) ；

输入流对象 . read( ) ；

D：释放资源

七、多个客户端上传到一个服务器

1、如果在接收端的监听客户机外加上while（true） 可行，但是受网络带宽（每秒内传输的数据多少）的影响, 因此用多线程进行改进

2、在服务器端加上线程，通过构造方法创建Socke对象

3、核心代码

ServerSocket server = new ServerSocket(8080）；

while（ture） {

Socket s = new Socket( ) ;

new Thread(new 服务器端多线程对象(s) ) . start( ) ；

}

**DAY 27 反射**

一、反射

在运行状态内，对于一个任意一个类，能知道类的所有属性和方法，

对于任意一个对象，能够调用任意一个方法和属性

( 这种动态获取的信息以及调用动态调用对象的方法的功能称为Java语言的反射机制 )

A、类加载器 ：负责将class文件加载在内存中，并且为之生成对应的Class对象

B、拓展类加载器

C、系统内加载器

反射：就是通过class文件对象，去使用该文件中的成员变量，构造方法，成员方法

1、要想这样使用，必须得到class文件 对象 ，也就是都得到class类的对象

class类：

成员变量：Filed

构造方法：Constructor

成员方法：Method

二、获取class文件对象

A: object类中 getClass( ) 方法

例如： person p = new person( );

class c1 = p.getClass( ）；

B：数据类型的静态属性class

例如 : class c = person . class;

C：class类中的静态方法

public static Class forName(String className);

className：必须是全路径 （包括包名，就算同一个包名下也要写包名）

注意：开发中经常用到的是第三种

因为写入的是一个字符串，而不是一个具体明确的类名，这样可以将一个字符串写入到配置文件

三、通过反射获取构造方法并使用

public Constructor[ ] getConstructors( ) ; 获取的是公共（public）的构造方法

public Constructor[ ] getDeclaredConstructors( ) ; 所有的构造方法

//获取单个公共的构造方法

public Constructor<> getConstructor(class<>... parameterTypes ) ;

参数表示的是：你要获取的构造方法的参数的个数以及数据类型的class字节码文件对象 (String . class , int . class)

//返回一个Constructor对象 , 实际上是一个构造方法的对象

//构造方法对象的实例化s

newInstance(Object... initargs ) ;

//Object... initargs代表带参构造中的实例初始化内容

Object obj = Constructor对象 . newInstance( ) ;

四、获取有参构造

getDeclaredConstruct(class<>... parameterTypes ) ; 获取指定参数，及类型的所有构造方法

void setAccessible(boolean flag) 将此对象的 accessible 标志设置为指示的布尔值。

//值为 true 则指示反射的对象在使用时应该取消 Java 语言访问检查 （也就是不管是公有还是私有都能访问）

五、通过反射获取成员变量

Field[] getDeclaredFields()

返回 Field 对象的一个数组，这些对象反映此 Class 对象所表示的类或接口所声明的所有字段。

//配套使用 只能是公共的

Field getField(String name)

返回一个 Field 对象，它反映此 Class 对象所表示的类或接口的指定公共成员字段。

void set(Object obj , Object value)

将指定对象变量上此 Field 对象表示的字段设置为指定的新值。

//获取所有的成员变量 (配套使用) (可以取出私有的方法)

Field getDeclaredFiled( String name)

void SetAccessible(boolean flag ) ; 取消权限

void set(Object obj , Object value)

六、通过反射获取成员方法

1、获取所有的方法

getMethods

getDeclaredMethods

2、获取单个方法

getMethod(String name , Class<?>... parameteTypes) //第一个参数代表方法名称，第二个代表方法中参数的类型

getDeclaredMethod

// 用方法的对象对 方法的对象和参数进行显示

public object invoke(Object obj , Object... args )

//返回值是Object接收，第一个参数表示对象是谁，第二个参数表示该方法的实际参数

3、暴力访问

method . setAccessible(true )

七、通过反射运行配置文件内容

Properties集合

八、面试题

//给定ArrayList<Integer>的一个对象，在集合中想添加字符串元素如何实现

//创建一个集合对象

ArrayList<Integer> array = new ArrayList<Integer>();

//获取array对象的字节码对象

Class c = array.getClass();

//通过字节码对象获取方法对象

Method m = c.getMethod("add", Object.class);

//设置方法中对象的参数值

m.invoke(array, "hello");

System.out.println(array);

**接口**

接口的特点：

1、用关键字interface表示：

interface 接口名 { }

2、类实现接口用implements 表示：

class 类名 implements 接口名 { }

3、接口是抽象的，无法实例化；按照多态的方式进行实例化

4、具体类多态，抽象类多态，接口多态

5、接口的子类:

要重写抽象类的抽象方法，或者定义为抽象类（但是此方法意义不大）

接口的成员特点：

A：成员变量：接口中的变量默认为常量（值不可以改）：

默认为静态的（不写也默认）可以用接口调用 接口名.变量名

默认修饰符： public static final 变量名

B：构造方法：无构造方法 （子类中的super( ) 其实调用的是Object类）

C：成员方法：接口的方法不能有主体，默认为抽象类（只能是抽象方法 public abstract void show ( ) ;）

所有的类都默认继承一个类 Object

类与类的关系：

继承关系：只能单继承，可以多层继承；

类与接口：

实现关系：一个类可以实现多个接口（ 多个接口之间用逗号分割）

并且可以在继承一个类的同时，实现多个接口；

注意事项：对应的接口实现对应的方法

接口与接口：

继承关系：可以单继承，也可以多继承

抽象类和接口的区别：

A：成员区别

抽象类：

成员变量：可以常量，也可以是变量

构造方法：有

成员方法：可以抽象，也可以非抽象

接口：

成员变量：只能是常量

构造方法：无

成员方法：只能是抽象的

设计理念区别：

抽象类：被继承体现是 “ is a”的关系。抽象类所定义的是该继承体系的共性功能

接口类；被实现是“ like a " 的关系。 接口中定义的是继承体系中的扩展功能