目录

[第1章 变量，运算符 2](#_Toc5694318)

[1.1 知识点总结 2](#_Toc5694319)

[第2章 Scanner，Random，流程控制语句 3](#_Toc5694320)

[2.1 知识点总结 3](#_Toc5694321)

[第3章 数组 5](#_Toc5694322)

[3.1 知识点总结 5](#_Toc5694323)

[第4章 方法 6](#_Toc5694324)

[4.1 知识点总结 6](#_Toc5694325)

[第5章 自定义类，ArrayList集合 7](#_Toc5694326)

[5.1 知识点总结 7](#_Toc5694327)

[第6章 类，封装 8](#_Toc5694328)

[6.1 知识点总结 8](#_Toc5694329)

[第7章 继承，抽象 9](#_Toc5694330)

[7.1 知识点总结 9](#_Toc5694331)

[第8章 接口，多态 10](#_Toc5694332)

[8.1 知识点总结 10](#_Toc5694333)

[第9章 构造方法，this，super 12](#_Toc5694334)

[9.1 知识点总结 12](#_Toc5694335)

[第10章 static,final,匿名对象,内部类,包,修饰符,代码块 13](#_Toc5694336)

[10.1 知识点总结 13](#_Toc5694337)

[第11章 类与接口作为参数返回值 14](#_Toc5694338)

[11.1 知识点总结 14](#_Toc5694339)

[第12章 常用API(Object,String,StringBuffer) 15](#_Toc5694340)

[12.1 知识点总结 15](#_Toc5694341)

[第13章 常用API(正则表达式,Date,DateFormat,Calendar) 16](#_Toc5694342)

[13.1 知识点总结 16](#_Toc5694343)

[第14章 包装类,System,Math,Arrays,大数据运算 17](#_Toc5694344)

[14.1 知识点总结 17](#_Toc5694345)

[第15章 集合（List,Set） 18](#_Toc5694346)

[15.1 知识点总结 18](#_Toc5694347)

[第16章 集合(Map,可变参数,Collections) 19](#_Toc5694348)

[16.1 知识点总结 19](#_Toc5694349)

[第17章 异常 21](#_Toc5694350)

[17.1 知识点总结 21](#_Toc5694351)

# 变量，运算符

## 知识点总结

* 数据类型转换
  + 数据类型范围从小到大排序（byte < char < short < int < long < float < double），布尔类型Boolean不能参与类型转换；
  + 自动类型转换，范围小的数据类型向范围大的数据类型转换时使用；
  + 强制类型转换，范围大的数据类型向范围小的数据类型转换时使用。
* 算数运算符
  + 用来完成算数运算（如加减乘除计算等）
  + ++，--运算符的使用
    - ++，--前置（如++a），当参与运算时，变量a的值先自增1，然后用自增后的新值再参与运算；
    - ++，--后置（如a++），当参与运算时，变量a先使用原有值参与运算符，变量a的值再自增1。
* 赋值运算符
  + 用来完成数据的赋值（如 int a = 100;）
  + +=,-,\*=,/=这样的赋值运算符包含了一个强制转换的操作，会将左右两边运算后的结果，强制类型转换后赋值给左边

int n = 10;

byte by = 20;

by += n; // 运算完毕后，by的值为byte类型30， 相当于代码 by = (byte)(by + n);

* 比较运算符
  + 用来比较数据的大小（如 3>4），比较运算符也称为条件运算符。
  + 比较后的结果为布尔类型Boolean的值
  + “==”两个等号代表比较是否相等，“=”一个等号代表赋值。
* 逻辑运算符
  + 逻辑与& 和逻辑短路与&&：代表着并且的意思，左右两边都要条件成立，结果才为true；
  + 逻辑或| 和逻辑短路或||：代表着或者的意思，左边两边有一个条件成立，结果就为true；
  + 逻辑非!：代表着相反的意思，原先是false，结果就为true；原先是ture，结果就为false；
  + 逻辑异或^： 左右两边条件结果相同，结果就为false，左右两边条件结果不同，结果就为true；
* 三元运算符
  + 根据判断条件运算结果，执行不同的表达式值；条件为true，执行表达式1，否则，执行表达式2。

# Scanner，Random，流程控制语句

## 知识点总结

* 选择结构 if
  + 方式1:

格式：

if(条件表达式) {

语句体;

}

执行流程：

如果条件表达式值为true, 执行语句体

如果条件表达式值为false,不执行语句体

* + 方式2：

格式:

if(条件表达式) {

语句体1;

} else {

语句体2;

}

执行流程：

如果条件表达式值为true, 执行语句体1

如果条件表达式值为fales,执行语句体2

* + 方式3：

格式：

if(条件表达式1) {

语句体1;

} else if (条件表达式2) {

语句体2;

} else if (条件表达式n...) {

语句体n...

}

...

else {

语句体n+1;

}

执行流程：

哪一个条件表达式成立，执行对应的语句体，

如果所有的条件表达式都不成立，执行最后else中的语句体

* + 注意事项：

1，条件表达式无论是简单还是复杂，结果都是boolean类型的值

2, 如果语句体只有一条语句，大括号可以省略，

如果是多条语句，大括号不能省略，建议什么时候都写上大括号

* 循环语句For
  + 格式：

for (初始化表达式;条件表达式; 控制条件表达式) {

语句体;

}

* + 执行流程：

1，初始化表达式；

2，条件表达式

如果结果为true, 执行语句体

如果结果为flase, 循环结束

3, 当语句体执行后， 执行控制条件表达式

4，接下来，回到第二步, 直到条件表达式为false 循环结束

* 循环语句 while
  + 格式：

初始化表达式;

while (条件表达式) {

语句体;

控制条件表达式;

}

* + 执行流程；

1，初始化表达式；

2，条件表达式

如果结果为true, 执行语句体

如果结果为flase, 循环结束

3, 当语句体执行后， 执行控制条件表达式

4，接下来，回到第二步, 直到条件表达式为false 循环结束跳转语句（break、continue）

* 循环语句 do...while
  + 格式：

初始化表达式;

do {

语句体;

控制条件表达式;

} while (条件表达式);

* + 执行流程:

1，初始化表达式；

2, 执行语句体

3, 执行控制条件表达式

4, 执行条件表达式,

如果结果为true, 继续执行语句体

如果结果为false, 循环结束

* 跳转控制语句

break; 跳出循环

continue; 结束本次循环

* Scanner类
  + 获取键盘录入的数据，对获取数据的具体操作进行了封装，只需要调用方法，即可得到键盘录入的数据。
  + 常用的方法

public int nextInt()：获取键盘录入的数据，返回int类型值

public String next()：获取键盘录入的数据，返回String类型值

* Random类：随机数类，用来产生多种类型的随机数
  + 方法：

public int nextInt(int n) 返回一个，在 0（包括）和指定值（不包括）之间int随机数

# 数组

## 知识点总结

* 数组
  + 它是一个用来存储同一个数据类型多个元素的一个容器（数组长度是固定的，数组中存储的元素的数据类型要求一致）
  + 格式：

格式1：

数据类型[] 数组名 = new 数据类型[数组长度];

格式2:

数据类型[] 数组名 = new 数据类型[]{元素值1,元素值2,..};

格式3：

数据类型[] 数组名 = {元素值1,元素值2,..};

* + 数组操作的常见问题：

NullPointerException: 空指针异常

ArrayIndexOutOfBoundsException: 数组越界异常

* 二维数组：
  + 它是一个包含多个一维数组的数组
  + 特点：二维数组中的每个元素都是一个一维数组
  + 格式：

格式1：

数据类型[][] 数组名 = new 数据类型[m][n];

m: 代表二维数组中一维数组的个数

n: 代表每个一维数组中元素的个数

格式2：

数据类型[][] 数组名 = new 数据类型[m][];

m: 代表二维数组中一维数组的个数

每一个一维数组通过赋值来确定数组长度

格式3：

数据类型[][] 数组名 = {{元素值1，元素值2,..},{元素值1，元素值2,..},..};

# 方法

## 知识点总结

* 方法
  + 格式：

修饰符 返回值类型 方法名(参数类型 参数名1, 参数类型 参数名2, ...) {

方法体;

return 返回值;

}

* + 方法使用的注意事项：

1，方法不调用，自己不执行

2，方法中不能定义方法， 但是，方法中可以调用方法

3, 方法定义的位置在类中，其他方法的外面

4, 如果方法没有明确的返回值类型，使用'空'类型， void表示

5，void只能在方法返回值类型位置使用，不能作为 普通的数据类型使用

6, 如果方法返回值类型为void类型，可以省略 return ;

* + 方法调用方式：

有明确返回值类型的方法调用方式：

1，单独调用

2，输出调用

3，赋值调用

没有明确返回值类型的方法调用方式：

1. 单独调用
   * 方法重载：

方法重载，在同一个类中，出现了多个同名的方法，他们的参数列表不同 (参数列表的个数不同，参数列表的数据类型不同，参数列表的顺序不同)。

方法重载特点：

与方法的返回值类型无关，与方法的参数名无关，只看方法名与参数列表；

方法重载，是通过JVM来完成同名方法的调用的，通过参数列表来决定调用的是哪一个方法。

# 自定义类，ArrayList集合

## 知识点总结

* 引用数据类型（类）
  + 类的类型为两种：
    - 第一种，Java为我们提供好的类，如Scanner类，Scanner类等，这些已存在的类中包含了很多的方法与属性，可供我们使用。
    - 第二种，我们自己创建的类，按照类的定义标准，可以在类中包含多个方法与属性，来供我们使用。
  + 创建类的格式

public class 类名 {

//可以定义属性

//也可以定义方法

}

* + 使用类的格式：

类名 变量名 = new 类名();

* + 使用类中的属性与方法格式

使用属性： 变量名.属性

使用方法： 变量名.方法()

* ArrayList集合
  + 它属于引用数据类型（类）。我们可以看作一个长度可变的数组。
  + 创建集合的方式

ArrayList<要存储元素的数据类型> 变量名 = new ArrayList<要存储元素的数据类型>();

* + 集合中的常用方法
    - boolean add（Object obj）
    - Object get（int index）
    - int size（）
    - boolean add（int index, Object obj）
    - Object set（int index, Object obj）
    - Object remve（int index）
    - void clear（）

# 类，封装

## 知识点总结

* 类与对象
  + 类，用于描述多个对象的共同特征，它是对象的模板。
  + 对象，用于描述现实中的个体，它是类的实例。
  + 类的定义：使用关键字class来定义java中的类
    - 格式：

class 类名 {

//属性

数据类型 变量名;

…

//方法

修饰符 返回值类型 方法名(参数){ }

…

}

* + 创建对象：
    - 格式：

类名 对象名 = new 类名();

* 封装（private关键字）
  + 封装，把对象的属性与方法的实现细节隐藏，仅对外提供一些公共的访问方式
  + 封装的体现：
    - 变量:使用 private 修饰，这就是变量的封装
    - 方法:也是一种封装，封装了多条代码
    - 类： 也是一种封装，封装了多个方法
  + private关键字，私有的意思
    - 它可以用来修饰类中的成员(成员变量，成员方法)
    - private的特点：
      * private修饰的成员只能在当前类中访问，其他类中无法直接访问
* this关键字
  + this关键字，本类对象的引用
    - this是在方法中使用的，哪个对象调用了该方法，那么，this就代表调用该方法的对象引用
    - this什么时候存在的？当创建对象的时候，this存在的
    - this的作用：用来区别同名的成员变量与局部变量（this.成员变量）

public void setName(String name) {

this.name = name;

}

# 继承，抽象

## 知识点总结

* 继承：是指在一个现有类的基础上去构建一个新的类，构建出来的新类被称作子类，现有类被称作父类，子类会自动拥有父类所有
* 继承的好处：可继承的属性和方法。

提高了代表的可维护性

提高了代码的复用性

让类与类之间产生了继承关系

* + 继承的弊端：

类与类之间的耦合度过高

* + 继承特点：

java中类只能够单继承，不能多继承，可以多层继承

class Yy extends Object {}

class Fu extends Yy{}

class Zi extends Fu {}

所有的类都直接或者间接的继承了 Object类，Object类称为祖宗类

* + 继承的注意事项：

1，使用关键字 extends 让类与类之间 产生继承关系

2, 父类私有的成员，子类不能继承，因为根本看不到

3，不能为了继承某个功能而随意进行继承操作， 必须要符合 is a 的关系

苹果 is a 水果

男人 is a 人

狗 is a 人 ， 这种情况就不能继承了

* + 继承中的成员变量关系：

不同名的变量：

子类直接继承使用

同名的变量：

默认访问的是子类自己的成员变量, 想访问父类中的同名变量，请使用 super.成员变量;

* + 继承中的成员方法关系：

不同名的方法：

子类直接继承使用

同名的方法：

默认访问的是子类自己的成员方法，想访问父类中的同名方法，请使用 super.成员方法();

* + super:用来表示当前对象中包含的父类对象空间的引用

调用父类的成员变量：

super.成员变量;

调用方法的成员方法:

super.成员方法();

* + 方法重写(override)：指 在子父类中，出现了方法声明相同的情况，也叫做方法覆盖，方法复写
    - 方法重写的注意事项：

1. 子类的方法声明要与父类相同

2, 子类要重写方法的方法，方法的权限修饰符不能比父类的更低

* + 3, 父类私有的方法，子类不能够进行方法重写
  + 方法重载(overload)：指 在同一个类中，多个方法名称相同，它们的参数列表不同(个数不同，数据类型不同)
* 抽象
  + 抽象方法： 方法只有声明部分，没有方法体
  + 抽象类： 包含抽象方法的类，一定是抽象类

使用 abstract 修饰的类，是抽象类

* + 抽象类的特点：

1，抽象类与抽象方法都必须使用 abstract来修饰

2，抽象类不能直接创建对象

3，抽象类中可以有抽象方法，也可以没有抽象方法

4，抽象类的子类

a，实现了抽象方法的具体类

b，抽象类

* + 抽象类面试题：

1，抽象类中是否可以没有抽象方法？如果可以，那么，该类还定义成抽象类有意义吗？为什么？

可以没有抽象方法，有意义，不会让其他人直接创建该类对象

# 接口，多态

## 知识点总结

* 接口：理解为是一个特殊的抽象类，但它不是类，是一个接口
  + 接口的特点：

1，定义一个接口用interface关键字

interface Inter{}

2，一个类实现一个接口，实现implements关键字

class Demo implements Inter{}

3, 接口不能直接创建对象

通过多态的方式，由子类来创建对象，接口多态

* + 接口中的成员特点：

成员变量：

只能是final 修饰的常量

默认修饰符： public static final

构造方法：

无

成员方法：

只能是抽象方法

默认修饰符: public abstract

* + 类与类，类与接口，接口与接口之间的关系

类与类之间：继承关系，单继承，可以是多层继承

类与接口之间: 实现关系，单实现，也可以多实现

接口与接口之间：继承关系，单继承，也可以是多继承

Java中的类可以继承一个父类的同时，实现多个接口

* 多态：理解为同一种物质的多种形态
  + 多态使用的前提：

1，有继承或者实现关系

2，要方法重写

3，父类引用指向子类对象

* + 多态的成员访问特点：

方法的运行看右边，其他都看左边

* + 多态的好处：

提高了程序的扩展性

* + 多态的弊端：

不能访问子类的特有功能

* + 多态的分类
    - 类的多态

abstract class Fu {

public abstract void method();

}

class Zi extends Fu {

public void method(){

System.out.println(“重写父类抽象方法”);

}

}

//类的多态使用

Fu fu= new Zi();

* + - 接口的多态

interface Fu {

public abstract void method();

}

class Zi implements Fu {

public void method(){

System.out.println(“重写接口抽象方法”);

}

}

//接口的多态使用

Fu fu = new Zi();

* instanceof 关键字

格式： 对象名 instanceof 类名

返回值： true, false

作用： 判断指定的对象 是否为 给定类创建的对象

# 构造方法，this，super

## 知识点总结

* this关键字
  + this关键字，本类对象的引用
    - this是在方法中使用的，哪个对象调用了该方法，那么，this就代表调用该方法的对象引用
    - this什么时候存在的？当创建对象的时候，this存在的
    - this的作用：用来区别同名的成员变量与局部变量（this.成员变量）

public void setName(String name) {

this.name = name;

}

* 构造方法： 用来给类的成员进行初始化操作
  + 格式：

修饰符 类名 (参数列表) {

...

}

* + 构造方法的特点：
    - 1, 方法名与类名相同
    - 2，没有返回值，也没有返回值类型，连void也没有
  + 构造方法什么时候会被调用执行？

只有在创建对象的时候才可以被调用

* super: 指的是父类的存储空间(理解为父类的引用)

调用父类的成员变量：

super.成员变量;

调用父类的构造方法:

super(参数);

调用方法的成员方法:

super.成员方法();

* 继承中的构造方法注意事项：

1，如果我们手动给出了构造方法，编译器不会在给我们提供默认的空参数构造方法

如果我们没写任何的构造方法，编译器提供给我们一个空参数构造方法

2, 在构造方法中，默认的第一条语句为 super();

它是用来访问父类中的空参数构造方法，进行父类成员的初始化操作

3, 当父类中没有空参数构造方法的时候，怎么办？

a: 通过 super(参数) 访问父类有参数的构造方法

b: 通过 this(参数) 访问本类中其他构造方法

注意:[本类中的其他构造方法已经能够正常访问父类构造方法]

4, super(参数) 与 this(参数) 不能同时在构造方法中存在

# static,final,匿名对象,内部类,包,修饰符,代码块

## 知识点总结

* final：关键字，最终的意思

final修饰的类：最终的类，不能被继承

final修饰的变量： 相当于是一个常量, 在编译生产.class文件后，该变量变为常量值

final修饰的方法： 最终的方法，子类不能重写，可以继承过来使用

* static : 关键字， 静态的意思

可以用来修饰类中的成员(成员变量，成员方法)

注意： 也可以用来修饰成员内部类

* + 特点：

被静态所修饰的成员，会被所有的对象所共享

被静态所修饰的成员，可以通过类名直接调用，方便

Person.country = "中国";

Person.method();

* + 注意事项：

静态的成员，随着类的加载而加载，优先于对象存在

在静态方法中，没有this关键字

静态方法中，只能调用静态的成员(静态成员变量，静态成员方法

* 匿名对象：一个没有名字的对象
  + 特点：

创建匿名对象直接使用，没有变量名

匿名对象在没有指定其引用变量时，只能使用一次

匿名对象可以作为方法接收的参数、方法返回值使用

* 内部类：在一个类中，定义了一个新类，这个新的类就是内部类

class A {//外部类

class B{// 内部类

}

}

* + 特点：

内部类可以直接访问外部类的成员，包含私有的成员

* 包的声明与访问
  + 类中包的声明格式：

package 包名.包名.包名…;

* + 带有包的类，创建对象格式：包名.类名 变量名 = new包名.类名();

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

* + 导包的格式：

import 包名.类名;

* 权限修饰符

public : 公共的

protected: 受保护的

private : 私有的

public protected 默认的 private

在当前类中 Y Y Y Y

同一包中的其他类 Y Y Y

不同包中的子类 Y Y

不同包中的其他类 Y

* 代码块：

局部代码块：定义在方法中的，用来限制变量的作用范围

构造代码块：定义在类中方法外，用来给对象中的成员初始化赋值

静态代码块：定义在类中方法外，用来给类的静态成员初始化赋值

# 类与接口作为参数返回值

## 知识点总结

* 不同修饰符的使用
  + 类，最常使用public修饰
  + 成员变量，最常使用private修饰
  + 成员方法，最常使用public修饰
* 自定义数据类型的使用
  + 类作为方法参数时，说明要向方法中传入该类的对象
  + 类作为方法返回值时，说明该方法要返回一个该类的对象。
  + 抽象类作为方法参数时，说明要传入一个实现抽象类所有抽象方法的子类对象。
  + 抽象类作为方法返回值时，说明需要返回一个实现抽象类所有抽象方法的子类对象。
  + 接口作为方法参数时，说明该方法要传入一个接口实现类对象。
  + 接口作为方法返回值时，说明该方法需要返回一个接口实现类对象。

# 常用API(Object,String,StringBuffer)

## 知识点总结

* Object: 它是所有类的超类，祖宗类。java中所有的类都直接或间接的继承这个类
  + 方法

public String toString() 返回当前对象中的内容, 对于Object类默认操作来说，返回的对象的类型+@+内存地址值

public boolean equals(Object obj) 比较两个对象内容是否相同，对于Object类默认操作来说,比较的是地址值

* String: 字符串类，字符串是常量；它们的值在创建之后不能更改
  + 方法

boolean equals(Object obj) 判断两个字符串中的内容是否相同

boolean equalsIgnoreCase(String str) 判断两个字符串中的内容是否相同, 忽略大小写

boolean contains(String str) 判断该字符串中 是否包含给定的字符串

boolean startsWith(String str) 判断该字符串 是否以给定的字符串开头

boolean endsWith(String str) 判断该字符串 是否以给定的字符串结尾

boolean isEmpty() 判断该字符串的内容是否为空的字符串 ""

int length() 获取该字符串的长度

char charAt(int index) 获取该字符串中指定位置上的字符

String substring(int start) 从指定位置开始，到末尾结束，截取该字符串，返回新字符串

String substring(int start,int end) 从指定位置开始，到指定位置结束，截取该字符串，返回新字符串

int indexOf(int ch ) 获取给定的字符，在该字符串中第一次出现的位置

int indexOf(String str) 获取给定的字符串，在该字符串中第一次出现的位置

int indexOf(int ch,int fromIndex) 从指定位置开始，获取给定的字符，在该字符

byte[] getBytes() 把该字符串 转换成 字节数组

char[] toCharArray() 把该字符串 转换成 字符数组

String replace(char old,char new) 在该字符串中，将给定的旧字符，用新字符替换

String replace(String old,String new) 在该字符串中， 将给定的旧字符串，用新字符串替换

String trim() 去除字符串两端空格，中间的不会去除，返回一个新字符串

String toLowerCase() 把该字符串转换成 小写字符串

String toUpperCase() 把该字符串转换成 大写字符串

int indexOf(String str,int fromIndex) 从指定位置开始，获取给定的字符串，在该字符串中第一次出现的位置

* StringBuffer/StringBuilder:
  + 方法

public StringBuffer append(String str) 在原有字符串缓冲区内容基础上，在末尾追加新数据

public StringBuffer insert(int offset,String str) 在原有字符串缓冲区内容基础上，在指定位置插入新数据

public StringBuffer deleteCharAt(int index) 在原有字符串缓冲区内容基础上，删除指定位置上的字符

public StringBuffer delete(int start,int end) 在原有字符串缓冲区内容基础上，删除指定范围内的多个字符

public StringBuffer replace(int start,int end,String str)在原有字符串缓冲区内容基础上，将指定范围内的多个字符 用给定的字符串替换

public StringBuffer reverse() 将字符串缓冲区的内容 反转 "abc"----"cba"

public String substring(int start) 从指定位置开始，到末尾结束，截取该字符串缓冲区，返回新字符串

public String substring(int start,int end) 从指定位置开始，到指定位置结束，截取该字符串缓冲区，返回新字符串

# 常用API(正则表达式,Date,DateFormat,Calendar)

## 知识点总结

* 正则表达式：用来定义匹配规则，匹配一系列符合某个句法规则的字符串。

正则表达式的匹配规则

请参见1.2 正则表达式的匹配规则

正则表达式的常用方法：

public boolean **matches**(String regex) //判断字符串是否匹配给定的规则

public String[] **split**(String regex) //根据给定正则表达式的匹配规则，拆分此字符串

public String **replaceAll**(String regex,String replacement) //将符合规则的字符串内容，全部替换为新字符串

* Date: 日期/时间类

构造方法：

public Date()// 系统当前日期时间

public Date(long date) 得到一个1970年1月1日 0点这个时间基础上，加上参数date 毫秒值对应的日期时间

方法：

public long getTime() 获取日期所对应的毫秒值

* DateFormat:是日期/时间格式化子类的抽象类, 使用其子类SimpleDateFormat 实例化

构造方法：

public SimpleDateFormat() 默认的格式化操作

public SimpleDateFormat(String pattern) 按照指定的格式，进行日期格式化

日期和时间模式

y 年

M 年中的月份

d 月份中的天数

H 一天中的小时数（0-23）

m 小时中的分钟数

s 分钟中的秒数

S 毫秒数

方法：

public final String format(Date date) 把日期 格式化成字符串

public Date parse(String source) 把日期字符串 转换成 日期对象

* Calendar:日历类，可获取日期中指定字段的值

方法：

public static Calendar **getInstance**() //获取日期对象

public int **get**(int field) //获取时间字段值

public void **add**(int field,int amount) //指定字段增加某值

public final void **set**(int field,int value)//设置指定字段的值

public final Date **getTime**() //获取该日历对象转成的日期对象

# 包装类,System,Math,Arrays,大数据运算

## 知识点总结

* 基本类型包装类
  + 8种基本类型对应的包装类

基本类型 包装类

byte Byte

short Short

int Integer

log Long

float Float

double Double

char Character

boolean Boolean

* + 自动装箱、自动拆箱
    - 自动装箱：基本数值转成对象（int 🡪 Integer）
    - 自动拆箱：对象转成基本数值（Integer 🡪 int）
  + 常用方法

public int parseInt(String str):把字符串转成基本类型int

public static String toString(int x):把基本类型int转成字符串

public static Integer valueOf(int x):把基本类型i字符串转成Integer对象

public int intValue():以 int类型返回该包装类对象的值

* System类: 系统属性信息工具类
  + public static long currentTimeMillis()：获取当前系统时间与1970年01月01日00:00点之间的毫秒差值
  + public static void exit(int status)：用来结束正在运行的Java程序。参数传入一个数字即可。通常传入0记为正常状态，其他为异常状态
  + public static void gc()：用来运行JVM中的垃圾回收器，完成内存中垃圾的清除。
  + public static String getProperties()：用来获取指系统属性信息
* Arrays类：数组操作工具类
  + public static void sort方法，用来对指定数组中的元素进行排序（元素值从小到大进行排序）
  + public static String toString方法，用来返回指定数组元素内容的字符串形式
  + public static void binarySearch方法，在指定数组中，查找给定元素值出现的位置。若没有查询到，返回位置为-插入点-1。要求该数组必须是个有序的数组
* Math类：数学运算工具类
  + abs方法,结果都为正数
  + ceil方法，结果为比参数值大的最小整数的double值
  + floor方法，结果为比参数值小的最大整数的double值
  + max方法，返回两个参数值中较大的值
  + min方法，返回两个参数值中较小的值
  + pow方法，返回第一个参数的第二个参数次幂的值
  + round方法，返回参数值四舍五入的结果
  + random方法，产生一个大于等于0.0且小于1.0的double小数

# 集合（List,Set）

## 知识点总结

* List与Set集合的区别？

List:

它是一个有序的集合(元素存与取的顺序相同)

它可以存储重复的元素

Set:

它是一个无序的集合(元素存与取的顺序可能不同)

它不能存储重复的元素

* List集合中的特有方法
  + void add(int index, Object element) 将指定的元素，添加到该集合中的指定位置上
  + Object get(int index)返回集合中指定位置的元素。
  + Object remove(int index) 移除列表中指定位置的元素, 返回的是被移除的元素
  + Object set(int index, Object element)用指定元素替换集合中指定位置的元素,返回值的更新前的元素
* ArrayList:

底层数据结构是数组，查询快，增删慢

* LinkedList:

底层数据结构是链表，查询慢，增删快

* HashSet:

元素唯一，不能重复

底层结构是 哈希表结构

元素的存与取的顺序不能保证一致

如何保证元素的唯一的？

重写hashCode() 与 equals()方法

* LinkedHashSet:

元素唯一不能重复

底层结构是 哈希表结构 + 链表结构

元素的存与取的顺序一致

# 集合(Map,可变参数,Collections)

## 知识点总结

* Map集合:

map集合中的元素都是成对出现，成对存储的

map集合中的元素都是以一对键和值的形式组成存在的，称为键值对，理解为夫妻对

map集合中的键不能重复存储，值可以重复

map集合中的每一个键 对应着一个值

* + 方法：

V put(K key, V value) 把指定的键与指定的值添加到Map集合中

V remove(Object key) 把指定的键 所对应的键值对元素 在Map集合中删除，返回被删除元素的值

Set<Map.Entry<K,V>> entrySet() 获取到Map集合中所有的键值对对象的集合(Set集合)

V get(Object key) 根据指定的键，在Map集合中获取对应的值

Set<K> keySet() 获取Map集合中所有的键，存储到Set集合中

* Map集合遍历的两种方式
  + 方式1：根据键找值的方式

//a, 获取到Map集合中所有的键，返回对应的Set集合

//b, 遍历键的集合，获取到每一个键

//c, 通过键，找到对应的值

//获取到Map集合中所有的键，返回对应的Set集合

Set<String> keys = map.keySet();

//遍历键的集合，获取到每一个键

for (String key : keys) {

//通过键，找到对应的值

Student s = map.get(key);

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

}

* + 方式2：根据键值对对象找键和值的方式

//a, 获取Map集合中所有的键值对元素,返回对应的Set集合

//b, 遍历键值对元素集合，获取到每一个键值对元素对象

//c, 通过键值对元素对象，获取对应的键，和对应的值

//获取Map集合中所有的键值对元素,返回对应的Set集合

Set< Map.Entry<String, Student>> entrySet = map.entrySet();

//遍历键值对元素集合，获取到每一个键值对元素对象

for (Map.Entry<String, Student> entry : entrySet) {

//通过键值对元素对象，获取对应的键，和对应的值

//找键

String key = entry.getKey();

//找值

Student s = entry.getValue();

//打印

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

}

* HashMap:
  + 特点：

是Map集合的子集合

底层采用哈希表结构

HashMap集合中的key不能重复，通过重写hashCode() 与 equals()方法来保证键的唯一。

不能保证元素存与取的顺序完全一致

* LinkedHashMap:
  + 特点：

是HashMap集合的子集合

底层采用哈希表+链表结构

LinkedHashMap集合中的key不能重复，通过重写hashCode() 与 equals()方法来保证键的唯一。

* Collections中的方法：

public static <T> void sort(List<T> list) 排序

public static void shuffle(List<?> list) 集合中的元素存储位置随机打乱

# 异常

## 知识点总结

* 异常：就是程序中出现的不正常的现象(错误与异常)
  + 异常的继承体系:

Throwable: 它是所有错误与异常的超类（祖宗类）

|- Error 错误，修改java源代码

|- Exception 编译期异常, javac.exe进行编译的时候报错

|- RuntimeException 运行期异常, java出现运行过程中出现的问题

* 异常处理的两种方式：
  + 1，出现问题，自己解决 try…catch…finally

try{

可能出现异常的代码

} catch(异常类名 对象名){

异常处理代码

} finally {

异常操作中一定要执行的代码

}

* + 2，出现问题，别人解决 throws

格式：

修饰符 返回值类型 方法名(参数) throws 异常类名1,异常类名2,...{}

public void method() throws Exception{}

* 异常分类

异常的根类是Throwable，其下有两个子类：Error与Exception，平常所说的异常指Exception。

* 严重错误Error，无法通过处理的错误
* 编译时异常Exception，编译时无法编译通过。如日期格式化异常
* 运行时异常RuntimeException，是Exception的子类，运行时可能会报错，可以不处理。如空指针异常
* 异常基本操作
* 创建异常对象
* 抛出异常
* 处理异常：
  + - * 捕获处理，将异常获取，使用try/catch做分支处理

try{

需要检测的异常；

} catch(异常对象) {

通常我们只使用一个方法：**printStackTrace打印异常信息**

}

* + - * 声明抛出处理，出现异常后不处理，声明抛出给调用者处理。

方法声明上加throws 异常类名

* + 注意：异常的处理，指处理异常的一种可能性，即有了异常处理的代码，不一定会产生异常。如果没有产生异常，则代码正常执行，如果产生了异常，则中断当前执行代码，执行异常处理代码。
* 异常注意事项
* 多异常处理

捕获处理：

1多个异常可以分别处理

2多个异常一次捕获多次处理

3多个异常一次捕获，采用同一种方式处理

声明抛出异常：

声明上使用,一次声明多个异常

* 运行时异常被抛出可以不处理。即不捕获也不声明抛出
* 如果父类抛出了多个异常,子类覆盖父类方法时,只能抛出相同的异常或者是他的子集
* 父类方法没有抛出异常，子类覆盖父类该方法时也不可抛出异常。此时子类产生该异常，只能捕获处理，不能声明抛出
* 当多异常处理时，捕获处理，前边的类不能是后边类的父类
* 自定义异常

如果Java没有提供你需要的异常，则可以自定义异常类。

定义方法：编译时异常继承Exception，运行时异常继承RuntimeException。