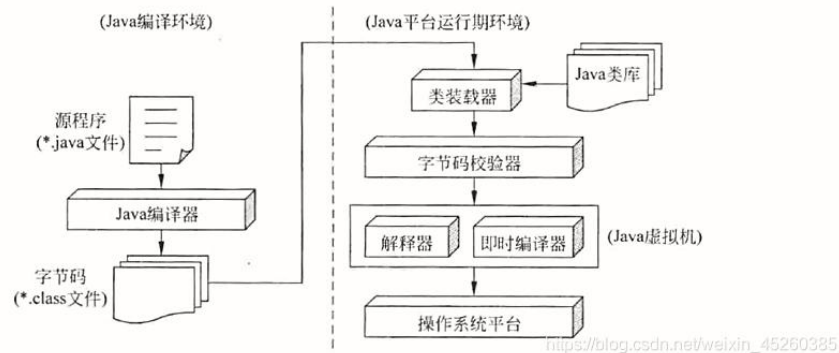


## 第一章 Java 语言概述:

1.Java 虚拟机的作用: 负责执行指令, 还要管理数据, 内存, 存储器

2.Java 运行机制: 将 java 源程序编译成字节码文件, 然后由 java 虚拟机来执行这个字节码文件

如果再深入一点探讨 Java 技术, 它是由 Java 源程序、Java 字节码文件、Java 虚拟机和 Java 类库 (Java API) 等 4 个方面组成。Java 又可分为编译环境和平台运行期环境, 它们的关系如图 1-2 所示。



3.JDK 的配置: Path: 指定 Java 的类路径、Classpath: JDK 命令搜索路径

4.Java API 各种包作用:

Java.util 是 JAVA 的 utility 工具包

java.lang 是 JAVA 的 language 核心语言包

java.awt 是 JAVA 的 abstract window toolkit, 抽象窗口工具包

java.applet 是创建 APPLET 的必须包

java.NET 是 JAVA 有关网络操作的包

java.io 是 JAVA 的输入输出流的包

java.sql 是 JAVA 的数据库操作包

javax.swing 是新的界面包

javax 开头的是扩展包

5.Java 程序的开发过程: 编辑源码、编译为 class 文件、执行全过程, 会用到的编译器、解释器程序是什么?

用到的 JDK 编译器: javac; 解释器程序是: jdk 或 jre 目录下 bin 目录中的 java.exe 文件

6.Java 源文件的命名规则

源文件的名称与文件中的类名有关系。

(1) 如果源文件中只有一个类, 那么源文件的名称必须与这个类的名称完全相同, 扩展名为 java。

(2) 如果源文件中有多个类, 那么这些类中只能有一个类在其类名前加上 public, 这时, 源文件的名称与这个类的名称完全相同, 扩展名为 java。

(3) 如果源文件中没有 public 类, 则源文件的名称可以与其中的任意一个类名相同, 扩展名为 java。

(4) 类名一般以大写英文字母开头, 后面可以是字母、数字等符号。类名的第一个字符不能是数字。

[https://blog.csdn.net/weixin\\_45260385](https://blog.csdn.net/weixin_45260385)

7. Java 环境配置的三个变量名:

(1) JAVA\_HOME: 指向 Java 的 JDK 安装目录的根目录

(2) CLASSPATH: lib 目录下两个 jar 包的安装目录, 提供 java 的运行环境, 存放 .class 文件

(3) Path: 两个 bin 文件夹的目录, 提供 java 的编译环境。JDK 的 bin 目录下, 有很

多 java 命令，如：javac(编译命令)，java（运行命令）等等。当操作系统需要运行 java 命令时，会在当前目录下寻找程序，如果找不到，就去 path 目录下去找 java 程序命令。将 JDK 配置在 path 路径之后，系统可以在任何地方运行 java 程序命令

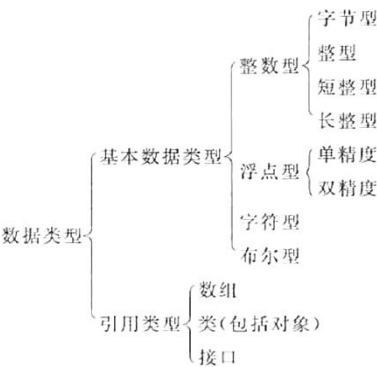
8. 在安装 JDK 的过程中，有三个功能模块可供选择，它们分别是**开发工具**、**源代码**、**公共 JRE**

9. 将 Hello.java 文件编译为 class 文件，需要键入 **javac Hello.java** 命令

第二章 Java 语言基础：

1.数据类型的分类以及各种数据类型占用长度：

数据类型的分类：（Java 不支持指针类型、结构体 struct 和联合类型 union）



八大基本数据类型：byte、short、int、long、double、float、boolean、char 对应着各自的封装类型，且不进行初始化时值默认为空（基本数据类型必须初始化），无法输出。而引用类型（如：Integer、String）变量中存储的是地址，对应着地址存储数据。

各种数据类型占用长度：

表 2-1 Java 的基本数据类型

类 型	数据类型关键字	适 用 于	类 型 长 度	值 域 范 围
字节	byte	非常小的整数	1	-128~127
短整	short	较小的整数	2	$-2^{15} \sim 2^{15} - 1$
整数	int	一般整数	4	$-2^{31} \sim 2^{31} - 1$
长整	long	非常大的整数	8	$-2^{63} \sim 2^{63} - 1$
单精度	float	一般实数	4	$-3.402\,823 \times 10^{38} \sim 3.402\,823 \times 10^{38}$
双精度	double	非常大的实数	8	$-1.7977 \times 10^{308} \sim 1.7977 \times 10^{308}$
字符	char	单个字符	2	
布尔	boolean	判断	1	true 和 false

数据类型按是否可修改可分为常量 final 和变量，变量分为类成员变量和局部变量（一个作用域中多个同名变量可以访问时按照“临近”原则）

2.Java 标识符的命名规则

只能用 4 种：字符、美元符（\$）、下划线（\_）、数字，但是数字不能放在第一位

3.字符类型与整数之间的关系

char 本质上是 int 存储的变量，但级别比 int 低。比如：char c1='\123'表示 Unicode 码八进制为 123 的字符，char c2='\u0061'表示 Unicode 码十六进制为 61 的字符。

字符型数据在做数字运算时实际上是对字符本身对应的 ASCII 码进行相应的数值运算。并且 char 做数值运算时可以自动转换成 int（低级向高级），但要做完运算的 int 转换成 char 的话就要强制转换。比如：char c=(char)(2000+'a'), int j='a'

#### 4.浮点类型

浮点数一般表示为标准计数法和科学计数法,有 float 和 double 型。默认浮点数是 double 型,若声明 float 型需在数字末尾加 F 或 f (整数可以不加, float f='a')

#### 5.不同编码的区别:

ASCII: ASCII 只有 128 个字符,表示英文字母的大小写、数字和一些符号

Unicode: Unicode 就是将一些语言(汉字等)统一到一套编码格式中,通常两个字节表示一个字符,而 ASCII 是一个字节表示一个字符

#### 6.数据类型的转换:

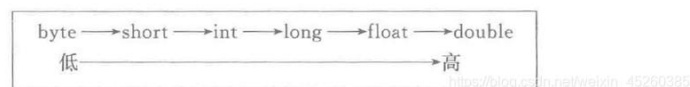
自动类型转换(低级向高级):

在程序中已经对变量定义了一种数据类型,若想以另外一种数据类型表示时,要符合以下两个条件:

(1) 转换前的数据类型与转换后的数据类型兼容。

(2) 转换后的数据类型比转换前的数据类型表示的范围大。

基本数据类型按精度从“低”到“高”的顺序为:



强制类型转换(高级向低级):

(类型名) 要转换的值或变量;

例如,设有

```
int a;  
double b = 3.14;
```

则

```
a = (int)b; //将 b 强制类型转换为 int 类型后,再赋值给 a
```

结果 a = 3, b 仍然是 double 类型, b 的值仍然是 3.14。

赋值时若变量类型比表达式类型长,则系统自动转换(即赋值相容),否则编译错误

#### 7.运算符

- (1) 二元运算符结果一般为数据类型大的
- (2) Java 中 % 运算符可以对浮点数进行运算
- (3) byte, char, short 等类型混合运算时自动转换成 int 运算,结果也是 int
- (4) 移位运算时会先对第二个操作数模 32
- (5) Java 中 true 和 false 不能用 1 和 0 表示

#### 8.equals 和 ==

== 是比较内存地址是否相同, equals 是比较值是否相同。注意:基本数据类型无 equals 方法。

#### 9.控制语句

- (1) case 语句可以并列
- (2) for 语句可由多个变量控制,用逗号隔开
- (3) Java 独有的 for(int e:a) 语句
- (4) break label 语句指定跳出某一层语句
- (5) switch 语句中 case 结束不加 break 的话会执行后面的 case 而不跳出 switch

10. 常量 final: final 全局变量必须要手动初始化,要么在声明时就赋初值,要么在类的构造方法里面赋初值,或者在构造代码块里赋初值(因为如果不手动赋初值的话,就是虚拟机给它赋值 0 不能再更改); final 局部变量只需要在使用前初始化就可以了。

11. 分配内存就相当于初始化了默认值,可以打印;没分配内存则不可

```

package G12;
public class T1_9 {
    public static void main(String[] args){
        int[] ar=new int[5];
        System.out.println(ar[0]);
    }
}

package G12;
public class T1_9 {
    public static void main(String[] args){
        int[] ar;
        System.out.println(ar[0]);
    }
}

```

System.out.println(ar[0]);

Variable 'ar' might not have been initialized

### 第三章 数组

#### 1.构造方法

- (1) 定义数组时，赋值变量的中括号内不可以出现数字
- (2) 数组构造时可以整体赋值，如：int s[]={75,86,92,35,21}，但初始化时不可以整体赋值，如：char s[]=new char[5]; s[]={‘a’,'s','f','g'}
- (3) 定义数组（int a[]）后必须要先用 new 分配空间后才能访问

#### 2.数组长度：a.length，没有括号

#### 3.Arrays 类的成员函数

```

Arrays.sort(arr);
Arrays.fill(arr,0,arr.length,3);
int index=Arrays.binarySearch(arr,6);

```

#### 4.Arrays.sort()函数注意事项

- (1) Arrays.sort(arr)时 arr 的数据类型可以是基本数据类型，但 Arrays.sort(arr,comparator)时必须使用引用类型
- (2) 自定义排序方式（Comparator 需重载函数 compare，返回值为正的在后）

```

Arrays.sort(a,new Comparator<Integer>() {
    public int compare(Integer x,Integer y) {
        if(x<y) return 1;
        else return -1;
    }
});

```

#### 5.数组在函数定义中应用

```

public static int[] Bubblesort(int[] arr) {
    int res[]=new int[arr.length];
    //...
    return res;
}

```

### 第四章 类和对象设计

#### 1.super: 代表了父类对象，因此可以实现对父类成员变量、方法的访问

#### 2.this: 表示当前对象本身

- (1) 引用成员变量 this.a=a
- (2) 引用构造方法 this()/this(8)，此处 8 为实参
- (3) 代表自身对象 A instance=this
- (4) 引用成员方法 this.method()，此处 this.可以省略

#### 3.方法调用分为传值调用和引用

传值调用：实参单向传递给形参，方法内对形参的改变不会影响实参

引用传值：双向传递地址，改变会同步

- 4.类名/对象名调用其静态变量与方法作用等同
- 5.同一个包中的类不可重名，不同包中的可以
- 6.访问权限：public > protected > default > private（Java 中 protected 在同一个包中可以被访问，而 C++中 protected 在全局函数中不可见）
- 7.final 修饰的**最终类**不可以拥有子类、被重写、被更改
- 8.Java 中解决成员变量与局部变量名称冲突时，可以使用 **this** 关键字
- 9.在垃圾回收机制中，可以通过调用 **System.gc()**方法来通知 Java 虚拟机立即进行垃圾回收

## 第五章 继承和多态

### 1. 方法重载与重写：

重载是指一个类中有多个同名不同参的方法，重写是指子类继承父类时重写覆盖原来父类里面的方法（此方法和父类的方法同名同参）或变量（常用@Override 做注释）

### 2.面向对象的三特性：封装性、继承性、多态性

（1）Java 的类不支持多继承，多继承应使用接口

（2）Java 中多态体现在方法重载实现的静态多态性（编译多态）和方法重写实现的动态多态性（运行多态）

### 3.在子类中访问父类被覆盖的成员，可以使用 super 关键字：

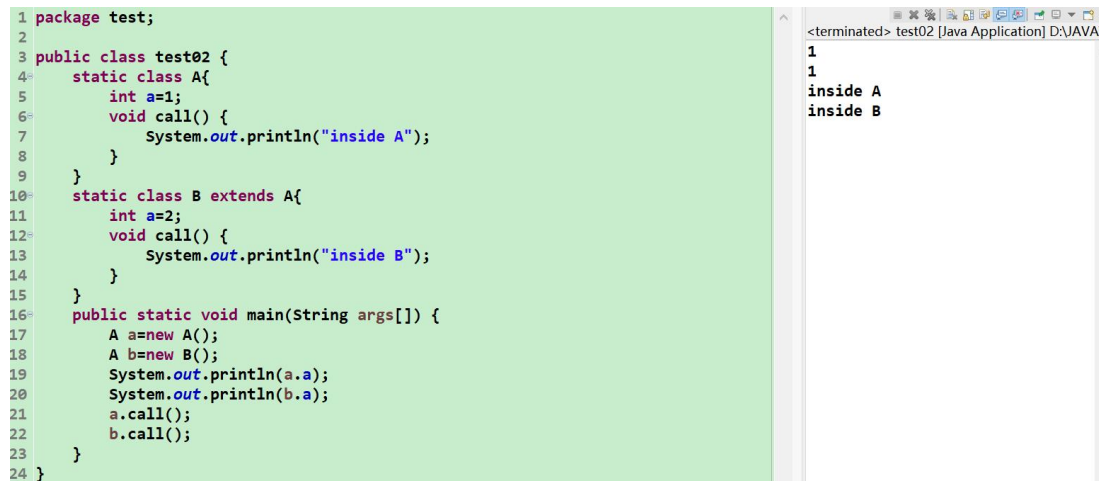
super.father / super.getfather() / super() / super(a,b)

系统在调用子类构造方法前，会自动调用父类构造方法，因此子类的构造方法中可以省略父类的构造方法

### 4.类对象的转换方式：子类-->父类（显示转换或隐式转换）

父类-->子类（显示转换且转换可行才行）

5.Java 调用重写的方法时会根据实例的类型选择方法。如果将父类变量实例化为子类，虽然发生了隐式转换，但调用的方法还是子类的方法，而访问其成员就是父类的成员



```
1 package test;
2
3 public class test02 {
4     static class A{
5         int a=1;
6         void call() {
7             System.out.println("inside A");
8         }
9     }
10    static class B extends A{
11        int a=2;
12        void call() {
13            System.out.println("inside B");
14        }
15    }
16    public static void main(String args[]) {
17        A a=new A();
18        A b=new B();
19        System.out.println(a.a);
20        System.out.println(b.a);
21        a.call();
22        b.call();
23    }
24 }
```

Output: <terminated> test02 [Java Application] D:\JAVA\1  
1  
1  
inside A  
inside B

上述操作可以实现将不同子类对象存入同一个父类数组。

### 6.抽象类 abstract

（1）不能实例化，其子类必须重写其所有抽象方法，也可以正常增加变量

（2）包含抽象方法的类必须是抽象类，但抽象类不一定要包含抽象方法（保护自身不被 new 实例化创建对象）

（3）抽象方法不可以有方法体（不能有大括号）

### 7.父类与接口的区别：



- (1) 继承是 is a 关系，接口是 has a 关系
- (2) Java 中类只可以单继承（继承一个父类），但接口可以继承多个父接口
- (3) 接口不可以继承类，类也不可以继承接口，并且接口继承父接口后还是抽象的，必须要类实现（implements）后才可以实例化
- (4) 接口也可以实现继承（extends）和多态
- (5) 接口中的方法必须都是抽象的

## 8.泛型

- (1) Java 中泛型类类比 C++中模板类，泛型方法类比模板函数
- (2) 使用泛型类时必须指明参数类型（Point<Integer,String> p1 = new Point<Integer,String>），而使用泛型方法时不必指明（public<T1,T2>void print(T1 X,T2 Y){}...p.print(x,y)）
- (3) 泛型方法可以定义在普通类中

```
public class test02 {
    public static <T1,T2> void print(T1 x,T2 y){
        System.out.println("(" +x+", "+y+"");
    }
    public static void main(String args[]) {
        print(3,5);
    }
}
```

9.类型通配符：函数传参为泛型类时用？代替类型（public static void getData(Box<?>data)）

## 10.反射机制

```
class OverrideTest {
    void show() {
        System.out.println("super show");
        System.out.println(this.getClass().getName());
        this.getName();
    }
    void getName() {
        System.out.println("OverrideTest");
    }
}
public class SubOverride extends OverrideTest {
    void show() {
        System.out.println("sub show");
        super.show();
        this.getName();
    }
    void getName() {
        System.out.println(this.getClass().getSuperclass().getName());
    }
}

public static void main(String args[]) {
    OverrideTest s = new SubOverride();
    s.show();
}
```

sub show  
super show  
SubOverride  
OverrideTest  
OverrideTest

## 11 instanceof 和 getclass 的区别：

- (1) S instanceof T 判断对象 S 是否为类 T 或其子类的实例
- (2) getClass 返回运行时该对象的类
- (3) 父类进行向下转型以后（father obj=new son()），运行时调用 obj.getClass()方法得到的就是子类，相当于父类在以子类的身份运行

## 第六章 Java 标准类库

- 1. 库单元是包，描述包的字符都是小写
- 2. package 语句放在 Java 源程序文件的第一行，指明该文件中定义的类存放哪个包中。并且 Java 源程序中 package 语句最多只能有一条
- 3. 使用通配符\*（import java.util.\*）只能导入某个包中的所有类，但不能导入该包内的子包

4. java.lang 包由解释程序自动加载，不需在程序中显式地使用语句 import java.lang.\*

<b>Boolean</b>	<b>Long</b>	<b>StrictMath</b>
<b>Byte</b>	<b>Math</b>	<b>String</b>
<b>Character</b>	<b>Number</b>	<b>StringBuffer</b>
<b>Class</b>	<b>Object</b>	<b>StringBuilder</b>
<b>ClassLoader</b>	<b>Package</b>	<b>System</b>
<b>Compiler</b>	<b>Process</b>	<b>Thread</b>
<b>Double</b>	<b>Runtime</b>	<b>ThreadGroup</b>
<b>Float</b>	<b>RuntimePermission</b>	<b>ThreadLocal</b>
<b>InheritableThreadLocal</b>	<b>SecurityManager</b>	<b>Throwable</b>
<b>Integer</b>	<b>Short</b>	<b>Void</b>

5.java.util 包提供一些实用工具：Arrays、Comparator、Date、DateFormat、Calendar、Random、Scanner、Stack、ArrayList、Bitset、HashTable

6.String 与数值转换

```
int x=Integer.parseInt(str)           String s=Integer.toString(-152)
double y=Double.parseDouble(str)      String s=Double.toString(3.14)
char c=str.charAt(0)（无 parseCharacter 函数） String s=Character.toString('f')
char car[]=str.toCharArray()
Byte b[]=str.getBytes()（b 中元素为 str 每一位的 ASCII 码）
```

7.String 调用 split 函数时若分隔符为 “.” 和 “|” 时前面要加 “\”

8.包装器类数值与字符串转换

```
Integer i=Integer.valueOf("1234",radix);
Double d=Double.valueOf("356.866");
String s=Integer.toString(-152,2) / String s=String.valueOf(3000)
String s=Double.toHexString(5.86);（浮点数也可以转成十六进制表示的字符串）
```

9.System.currentTimeMillis()调用系统时间可以获得程序运行时间

10.arraycopy 是 System 类下的，不是 ArrayList 下的

11.Runtime 类获得的是 Java 虚拟机内存使用情况，而不是运行时间

12.Math 类：

```
Math.PI/Math.E
Math.random()（生成数范围 0.0~0.9）
Math.pow(a,b)
Math.ceil(d)/Math.floor(d)/Math.round(d)
Math.exp(n)/Math.log(p)
Math.sin(s)/Math.asin(a)
Math.toDegrees(d)
```

13.Random 类

```
int ran=new Random().nextInt(b)+a 生成范围 a~a+b
```

无参构造时 Random r=new Random()：每次运行程序时 seedValue 不一样，得到的随机数序列不一样；含参构造 Random r=new Random(seedValue)每次运行程序得到长度为 seedValue 的随机数序列都是一样的。例如第一次运行程序得到的随机数是 2, 4, 1, 5, 7。那么重启程序，再次得到的随机数还是 2, 4, 1, 5, 7。因为 Random 产生的随机数实际上属于伪随机数，是按照一定算法计算出来的。构造方法如果没有传入随机种子的话，系统会默认采用系统时间作为随机种子

```

package test;
import java.util.Random;

public class test02 {
    public static void main(String[] args) {
        Random random1 = new Random(9);
        Random random2 = new Random(8);
        for(int i=0;i<5;i++){
            System.out.print(random1.nextInt(5));
        }
        System.out.println();
        for(int i=0;i<5;i++){
            System.out.print(random2.nextInt(5));
        }
    }
}

```

```

<terminal>
41304 package test;
41012 import java.util.Random;

public class test02 {
    public static void main(String[] args) {
        Random random1 = new Random(10);
        Random random2 = new Random(10);
        for(int i=0;i<5;i++){
            System.out.print(random1.nextInt(5));
        }
        System.out.println();
        for(int i=0;i<5;i++){
            System.out.print(random2.nextInt(5));
        }
    }
}
<terminal>
30301
30301

```

## 14.Java 的集合类

(1) 集合类由几大接口及其子接口实现，同一集合类可以实现不同接口（LinkedList 实现了 List 和 Deque 两个接口）

(2) 可以用实现了的集合类实例化接口对象（Set<String> set = new HashSet<>()）

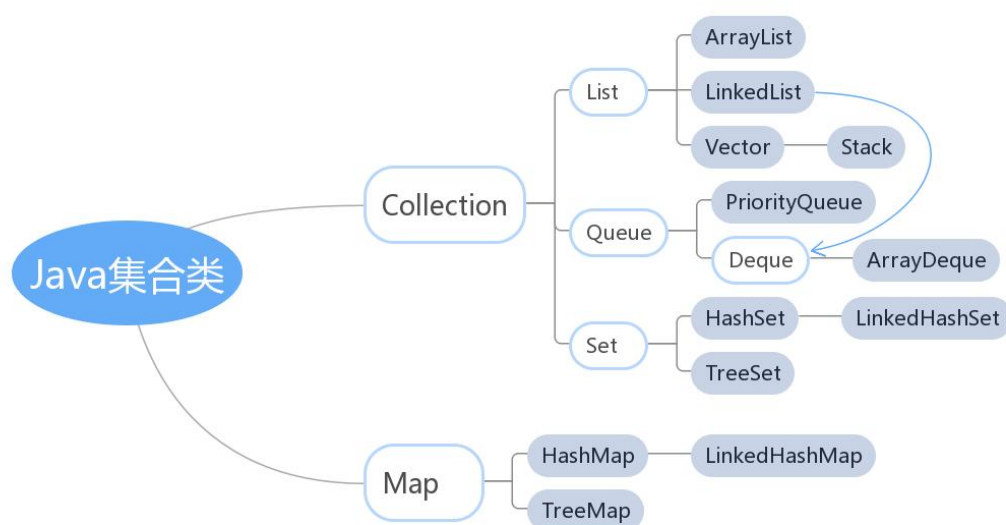
(3) Vector 用法与 ArrayList 类似，底层都是由数组实现，但 Vector 线程安全

(4) 集合中只能存储对象，基本数据类型必须转换成对象

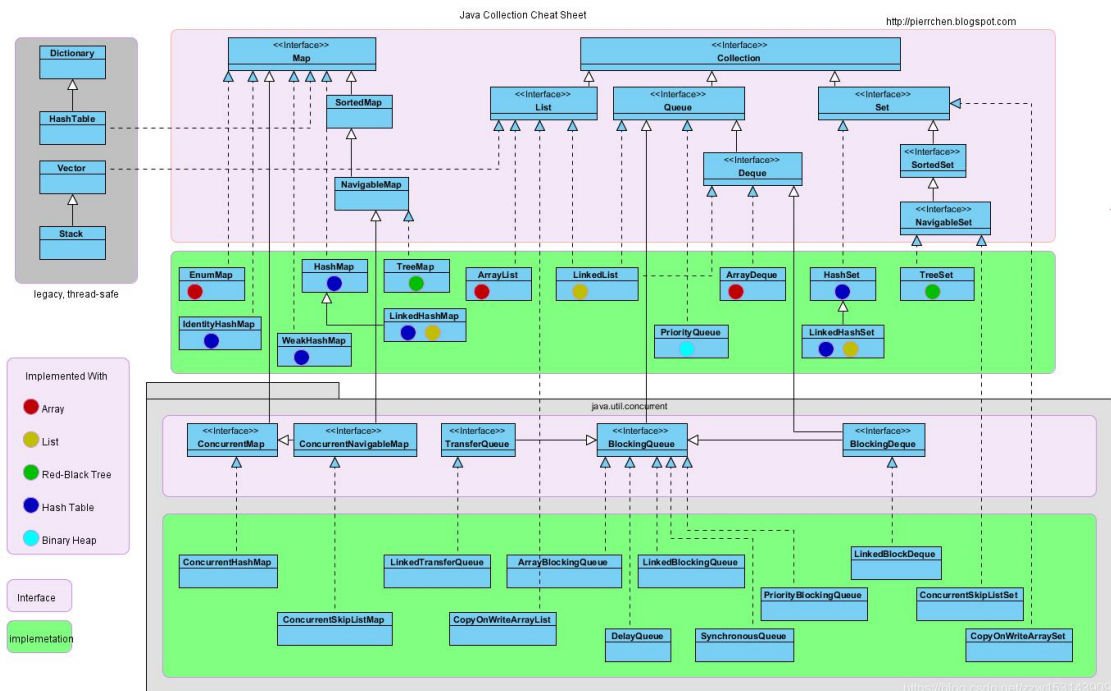
(5) Hashtable（属于 Dictionary 类）用法近似 HashMap，前者多线程性能差，后者线程不安全

(6) HashMap 中的 key-value 都是存储在 Entry 类中

(7) toArray 函数可以将集合转变为数组（Object[] element=arrlist.toArray()）







## 15. 遍历集合

`toArray()` 函数遍历集合是将集合转变为数组，此时数组与集合已然分离，对数组的改变不会影响集合。而 `Iterator` 遍历集合时是对原集合的操作，任何增减改变都会在集合上呈现。

## 16. 集合自定义排序

```
// 自定义排序1
Collections.sort(list, new Comparator<Student>() {
    @Override
    public int compare(Student o1, Student o2) {
        return o1.getId() - o2.getId();
    }
});

// 自定义排序2
list.sort(new Comparator<Student>() {
    @Override
    public int compare(Student o1, Student o2) {
        return o1.getId() - o2.getId();
    }
});
```

## 第七章 异常处理

1. 定义方法时可以使用 `throws` 关键字抛出异常（声明将要抛出何种类型的异常），方法体内使用 `throw` 抛出异常（将产生的具体异常抛出）

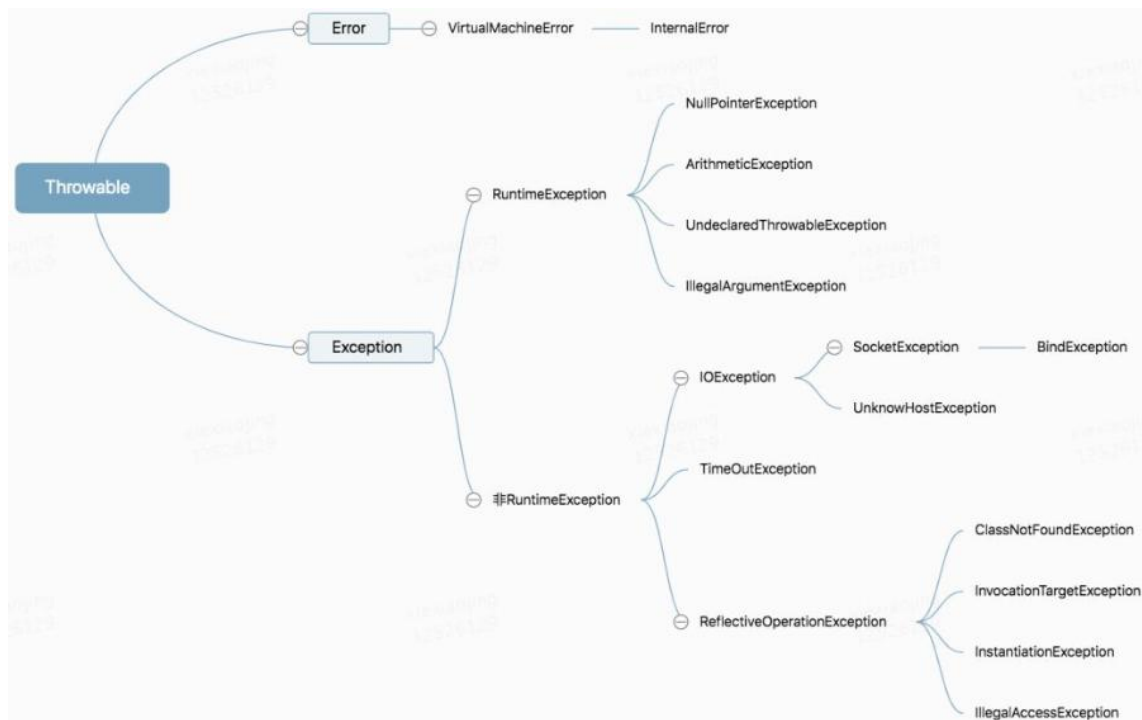
2. 抛弃多个异常时用 “,” 隔开

3. `catch` 可以连续使用捕捉多种异常

4. `Throwable` 是所有异常的父类，分为 `Error` 和 `Exception`。

(1) `Error` 是外部环境出现问题引起的错误，捕获没有意义，一般直接抛出

(2) `Exception` 外部环境没有问题，请开发人员自己搞定的异常



## 5.自定义异常类:

- (1) 检查性异常类继承 `Exception` 类
- (2) 运行时异常类继承 `RuntimeException` 类

## 第八章 输入输出处理

1. 以当前程序为中心，要读入到程序内存中的称为输入流；要从程序内存中写出去的，称为输出流

### 2. 标准输入输出流:

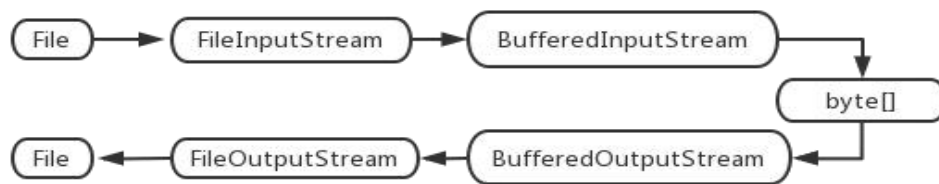
- (1) `System.in.read()`函数读取字节或字节数组，返回值为 ASCII 码
- (2) `Scanner sc=new Scanner(System.in)`读取工作区数据（`import java.util.Scanner;`）
- (3) `System.out.print()/println()/printf()`

### 3. File 类:

- (1) `File` 构造函数只是生成了一个对象，要在电脑中创建该文件的话还需要 `f1.createNewFile()`或者 `fdir.mkdir()`
- (2) `fdir.list()`返回字符串数组，用于打印文件名，`fdir.listFiles()`返回文件数组，用于获取文件对象并对其操作
- (3) 实现 `FilenameFilter` 接口（重写 `accept` 函数）可以过滤文件

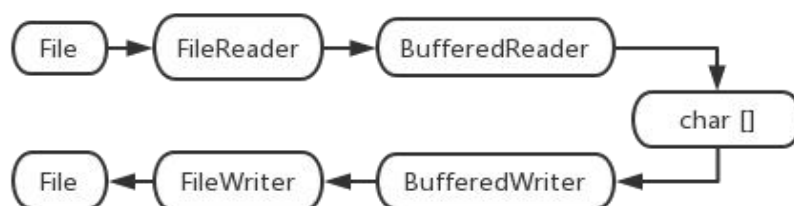
### 4. 字节流

- (1) 文件字节输入流 `FileInputStream`
- (2) 文件字节输出流 `FileOutputStream` 默认以覆盖方式打开
- (3) 字节输入过滤流 `FilterInputStream`
- (4) 字节输出过滤流 `FilterOutputStream`
- (5) 字节输入缓冲流 `BufferedInputStream`
- (6) 字节输出缓冲流 `BufferedOutputStream`

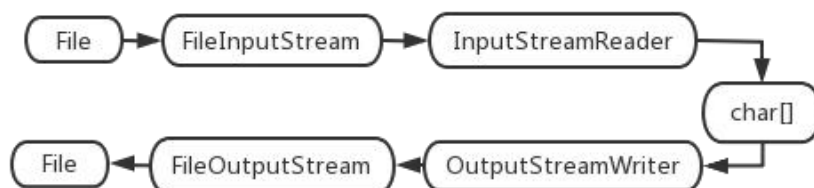


## 5. 字符流

- (1) 文件字符输入流 `FileReader`
- (2) 文件字符输出流 `FileWriter`
- (3) 字符输入缓冲流 `BufferedReader`
- (4) 字符输出缓冲流 `BufferedWriter`
- (5) 字符缓冲流的特有功能：读取整行/写入时换行  
`br.readLine()/bw.newLine()`



## 6. 字节流转字符流



## 7. 序列化

- (1) 实现 `Serializable` 接口的对象才可以序列化，该接口无抽象方法，只是标记作用
  - (2) 一个可以序列化的类的子类都可以序列化
  - (3) 可以将对象序列化为文件的对象流要求类实现哪个接口？ --**`Serializable` 接口**
8. `read()`后光标将自动跳过读过内容，输入流关闭前再输入时直接从没读过的地方继续
9. `FileOutputStream.write(int b)`时截取低 8 位（一字节），忽略高三位

```

import java.io.FileInputStream;
import java.io.FileOutputStream;
public class FileTest {
    public static void main(String[] args) throws Exception {
        int data1 = -2;
        FileOutputStream fout = new FileOutputStream("f1.txt");
        fout.write(data1);
        fout.write(new byte[] { -1, -2 });
        fout.close();
        FileInputStream fin = new FileInputStream("f1.txt");
        int data2 = fin.read();
        byte[] b = new byte[4];
        int count = fin.read(b);
        fin.close();
        for (int i : b) {
            System.out.println(i);
        }
        System.out.println(count);
        System.out.printf("data1=%d,data2=%d",data1,data2);
    }
}
  
```

-1  
 -2  
 0  
 0  
 2  
 data1=-2, data2=254

10.System.out 和 System.err 是 **PrintStream** 类，System.in 是 **InputStream** 类

## 第九章 Java 多线程

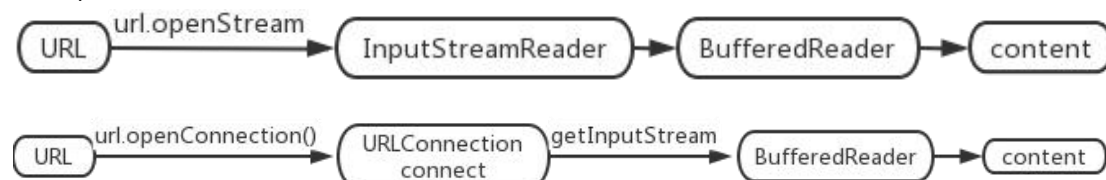
1. 自定义线程重写 **run()** 函数，但执行的时候调用的是 **start()**
2. 不同进程完全独立，不受影响；同一进程内的不同线程共用空间与资源，相互影响
3. 线程的生命周期，5 种状态：创建、就绪、运行、阻塞、死亡
4. 线程调度优先级高只能说明获得 CPU 概率大，并不一定先获得 CPU 执行权
5. **synchronized** 实现线程互斥
6. **wait()** 函数和 **notify()** 函数位于 **synchronized** 代码块中，配对使用实现线程同步
- 7.**ReentrantLock** 显示锁可以代替 **synchronized** 实现线程互斥

## 第十章 GUI 程序设计

- 1.swing 是 awt 的子类，包括了 awt 的全部组件，前者有更多的组件并且不依赖运行平台得本地组件，后者使用时完成图形化界面的是本地操作系统
- 2.JFrame、JDialog、JWindow 是顶层容器，JPanel、JScrollPane 是中间容器，JButton、JTextfield 等为组件
- 3.JFrame、JPanel、JButton 三者属于 swing 包中；Frame、Panel、Button 属于 awt 包
- 4.frame.setVisible(true)才可以显示窗体
- 5.(swing 包下)JFrame 类关闭窗口只需要 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)，(awt 包下)Frame 类窗口关闭需要 addWindowListener()并重写 windowClosing 函数
- 6.布局管理时默认 BorderLayout，若想设置无布局管理，需要 setLayout(null)
- 7.Button 组件的大小不会随框架大小变化而变化的布局方式是 **FlowLayout** 布局
- 8.事件适配器只用重写事件处理所关心的方法，如监听键盘输入的话只用 addKeyListener(键盘适配器类 KeyAdapter)重写 keyPressed()方法，即可，而不是重写 KeyListener 的所有方法
- 9.DefaultMutableTreeNode 类构造传入 JTree 可绘出树
- 10.模态框和非模态框：
  - (1) 模态框：此对话框激活时，其他窗体都不能操作。
  - (2) 非模态框：可在此对话框窗体与其他窗体之间，随意切换。

## 第十一章 网络通信

- 1.URL 是网址，URLConnection 是网址与程序之间的通信链接，用来读写网页上的资源
- 2.URL 对象一旦生成，不可更改(protocol: //hostname:port/resourcename#anchor)
- 3.URL/URLConnection 获取网页资源



- 4.IP 地址是传输层协议 TCP，UDP 的基础，用于唯一标识一台计算机，它有 IPv4（32 位地址）和 Ipv6（128 位无符号整数）两个版本。InetAddress 是 Java 对 IP 地址的封装
- 5.端口号 0~1023 为系统保留，因此选择端口需要>1023
- 6.关闭 Socket 前需先关闭与 Socket 相关的输入输出流
- 7.**流式通信协议 TCP** && **用户数据报通信协议 UDP**

- (1) TCP 基于**连接**以流的形式**顺序**传输数据，可靠；UDP **无连接**，不对数据到达的顺

序进行检查，可能丢数据包

(2) TCP 精准但速度慢且容易被攻击；UDP 速度快但网络通信质量不高

(3) TCP 无边界（客户端分多次发送数据给服务器，若服务器的缓冲区够大，那么服务器端会在客户端发送完之后一次性接收过来）；UDP 有边界（客户端每发送一次，服务器端就会接收一次）

#### TCP客户端

Java提供了java.net.Socket类ServerSocket类对计算机操作系统的Socket进行调用。客户端使用Socket(InetAddress, port)构造方法传入IP地址和端口号打开Socket，与远程服务区指定端口进行连接，然后调用socket的getInputStream和getOutputStream方法获取输入和输出流就可以读写TCP的字节流：

#### TCP服务端

服务器端通过ServerSocket(port)构造方法传入端口号来监听指定的端口，然后通过accept()方法得到一个Socket对象与远程客户端建立连接，同样调用Socket对象的getInputStream和getOutputStream方法就可以读写字节流，服务器端完成传输后可以通过close()方法关闭远程连接和监听端口：

#### TCP多线程

服务端的一个ServerSocket可以同时和多个客户端建立连接进行双向通信，实现起来也很简单，在设置好监听端口后，在一个无限for循环中调用ServerSocket的accept()方法，返回与客户端新建的连接，再启动线程或者使用线程池来处理客户端的请求，这样就可以同时处理多个客户端的连接

#### UDP编程

UDP协议不需要建立连接，可以直接发送和接收数据，UDP协议不保证可靠传输，使用Java提供的DatagramSocket的send()和receive()方法就可以发送和接收数据，UDP协议接收和发送数据没有IO流接口

8.基于 TCP/IP 的参考模型将协议分成四个层次，分别是链路层、网络层（核心）、传输层、应用层

9.127.0.0.1 指本机地址，该地址一般用来测试使用

10.在 JDK 中，提供了一个与 IP 地址相关的 **InetAddress** 类，该类用于封装一个 IP 地址

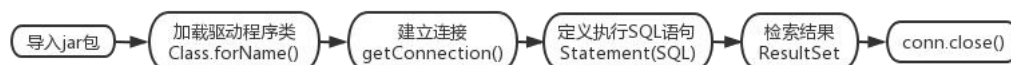
## 第十二章 JSP 和 Servlet 技术

1. JSP 是服务器端动态页面技术的组件规范，文件中主要有 html 和少量 java 代码；Servlet 是扩展 Web 服务器功能的组件规范（JSP 是在 html 里写 java，Servlet 是在 java 里写 Servlet）

## 第十四章 JDBC 技术

1.JDBC: java 数据库连接，用 Java 语言操纵数据库，可以访问多种关系型数据库提供统一的标准，是连接数据库与 Java 程序的桥梁。本质是 sun 公司定义的一套操作关系型数据库的接口，各数据库厂商去实现这个接口，提供数据库的驱动 jar 包，我们使用的时候真正执行的代码是这些 jar 包中的实现类。

2.JDBC 访问流程：



一般步骤：

(a) 加载和注册驱动程序

加载驱动程序—

需要将MySQL的mysql-connector-java-5.1.47.jar 加入项目

```
1 Class.forName("com.mysql.jdbc.Driver");
```

(b) 连接数据库

// 连接数据库

```
1 Connection conn = DriverManager.getConnection(
2 "jdbc:mysql://127.0.0.1:3306/javadb", "root", "123456");
```

(c) 向数据库发送SQL语句并处理结果

```
1 // statement用来执行SQL语句
2 Statement statement = conn.createStatement();
3 // 执行语句
4 statement.executeUpdate(sql);
5 // 关闭数据库的连接
6 conn.close();
```



简单的SQL语句及应用。(增删改查)

(执行时,把sql语句放入上面步骤c的//执行语句中的sql位置即可)

查看表中所有信息 (按表内列的顺序操作,下同)

```
1 | String sql = "select * from teacher";
```

插入数据

```
1 | String sql = "insert into teacher values('20051102001', '张老师', '13802287655', 35)";
```

更新 (修改) 数据

```
1 | String sql = "update teacher set age=38 where id='20051102001' ";
```

删除数据

```
1 | String sql = "delete from teacher where id='20051102001' ";
```

### 3.各个对象:

(1) DriverManager 驱动管理对象:

- 1) 加载驱动: 告诉程序是哪个数据库驱动 jar 包
- 2) 获取数据库连接

(2) Connection 数据库连接对象:

- 1) 获取执行 sql 的对象

Statement createStatement()

PreparedStatement prepareStatement()

2) 管理事务 (启动事务 setAutocommit(boolean autocommit)当参数为 false 时为开启事务,提交事务,回滚事务)

(3) statement 执行 sql 语句的对象:

boolean execute(string sql): 执行任何 sql 语句

int executeUpdate(string sql): 执行 SQL 中的 insert、update 和 delete 语句

ResultSet executeQuery()执行 DQL 语句,返回一个结果集

(4) ResultSet 结果集对象:

next(): 游标向下移动一行

Last(): 将游标移动到最后一行

getxxx(参数): 获取数据

(5) PreparedStatement(): 执行预编译 sql 语句