

Java程序设计

(基础)

2017.10.24

isszym sysu.edu.cn

目录

- 概述
- C++编译器和Java虚拟机
- Java的特点
- JRE和JDK
- 第一个Java程序
- 数据类型
- 与C++的不同点
- 字符串
- 对象和类
- 包
- 访问权限
- 容器类和映射类
- 数据库操作
- 附录1、捕捉错误
- 附录2、数值和字符串之间的转换
- 附录3、日期和字符串之间的转换
- 附录4、Math
- 附录5、控制台输入输出
- 附录6、泛型
- 附录7、与C++的部分不同点总结
- 附录8、参考资料

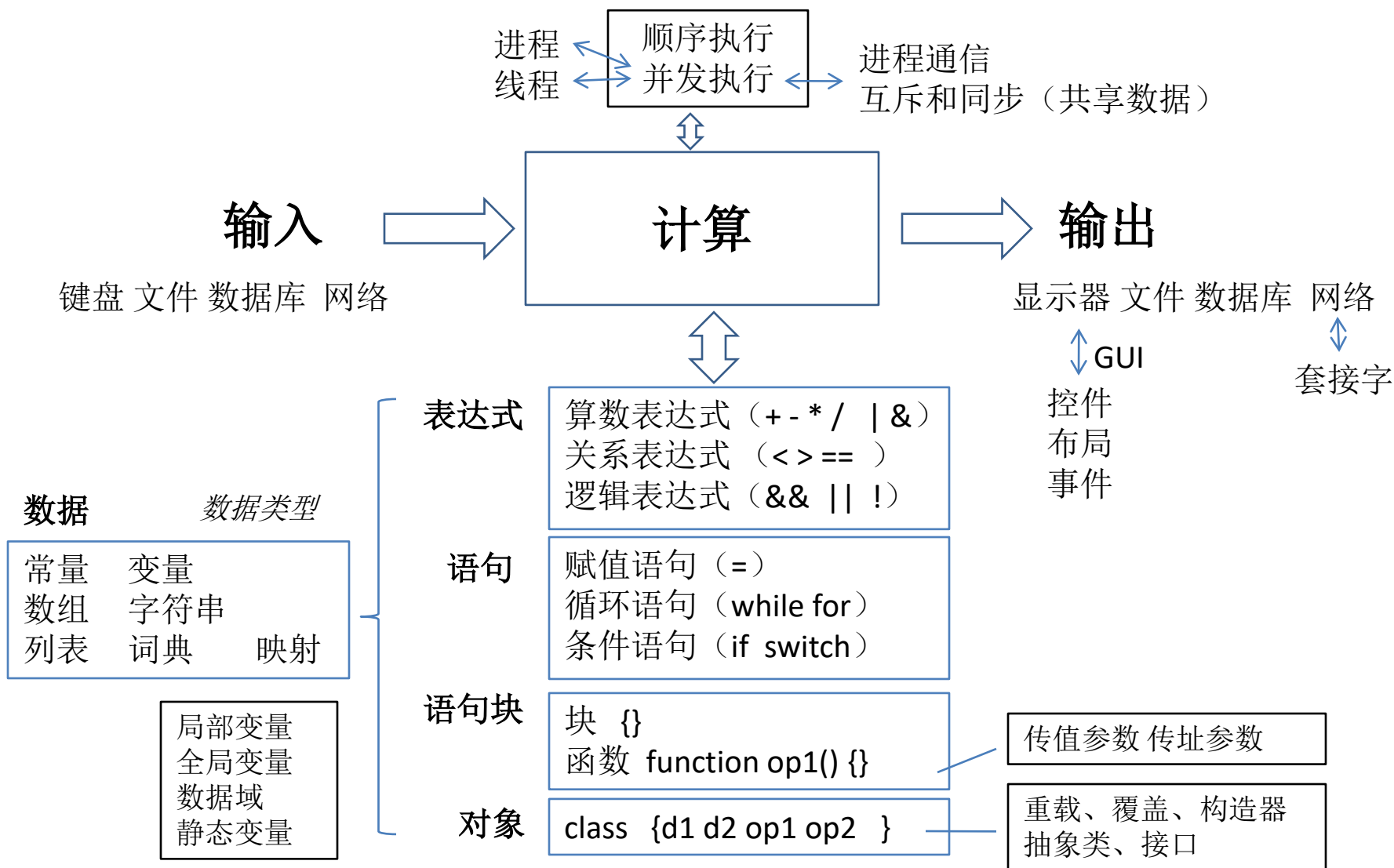
概述

- 1991 年，Sun Microsystems 的James Gosling 、 Patrick Naughton 、 Chris Warth 、 Ed Frank和Mike Sheridan共同构想了Java语言。
- 令人惊讶的是设计Java 的最初动力是想构建一种独立于平台的语言，使该语言生成的代码可以在不同环境下的不同CPU上运行，以满足消费类电子设备(洗衣机、空调和微波炉等)的软件的需要。
- 就在快要设计出Java 的细节的时候，另一个在Java 未来中扮演关键角色的更重要的因素出现了。这就是World Wide Web。
- 1993 年Web出现后，Java 的重点立即从消费类电子产品转移到了Internet 程序设计，并最终促成了Java 的燎原之势。

<http://docs.oracle.com/javase/8/docs/>

<http://api.apkbus.com/reference/java/io/package-summary.html>

程序设计语言



GitHub中程序设计语言的活跃度统计

JavaScript

Java

Python

Ruby

PHP

C++

CSS

C#

C

Go

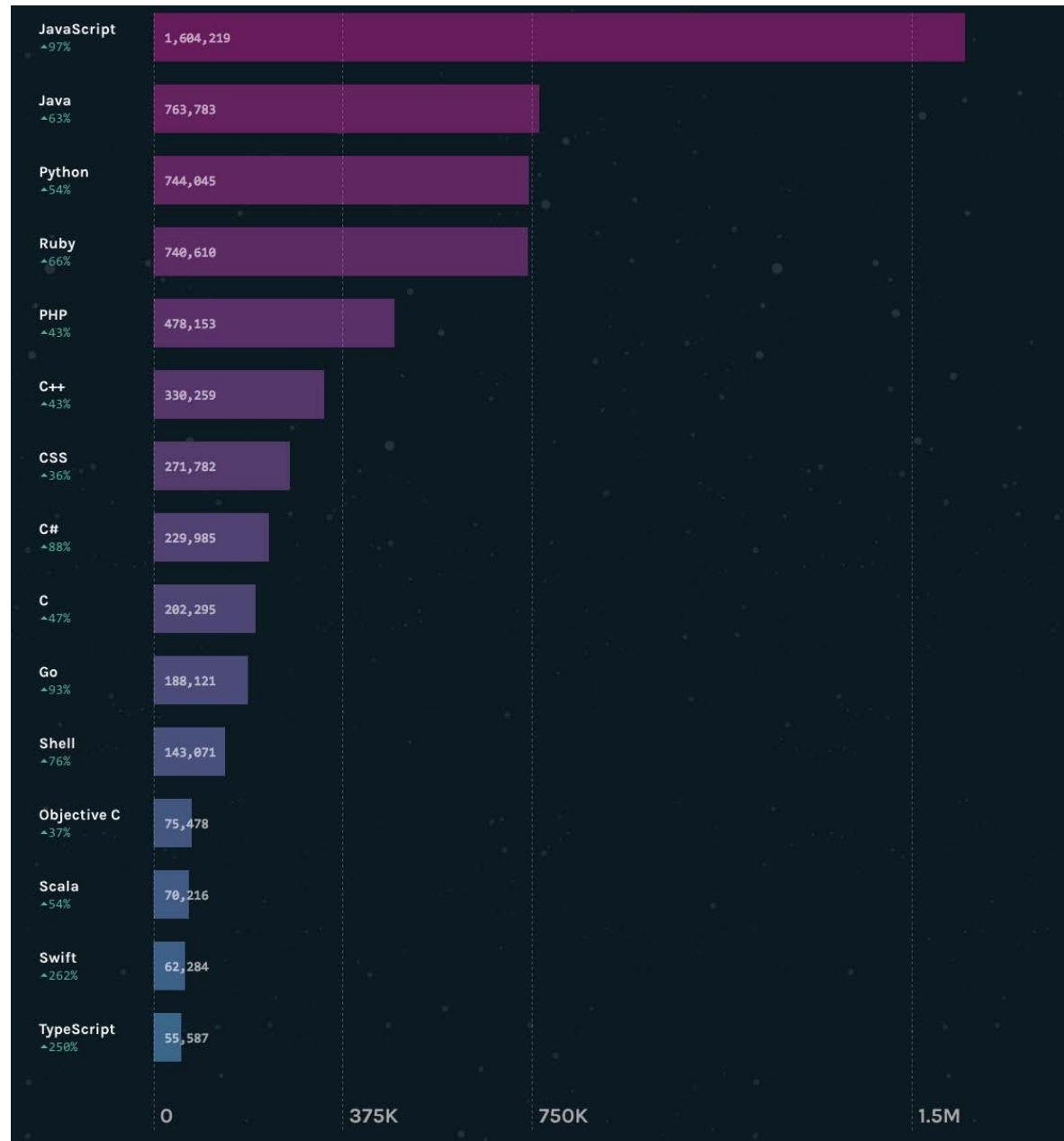
Shell

Objective C

Scala

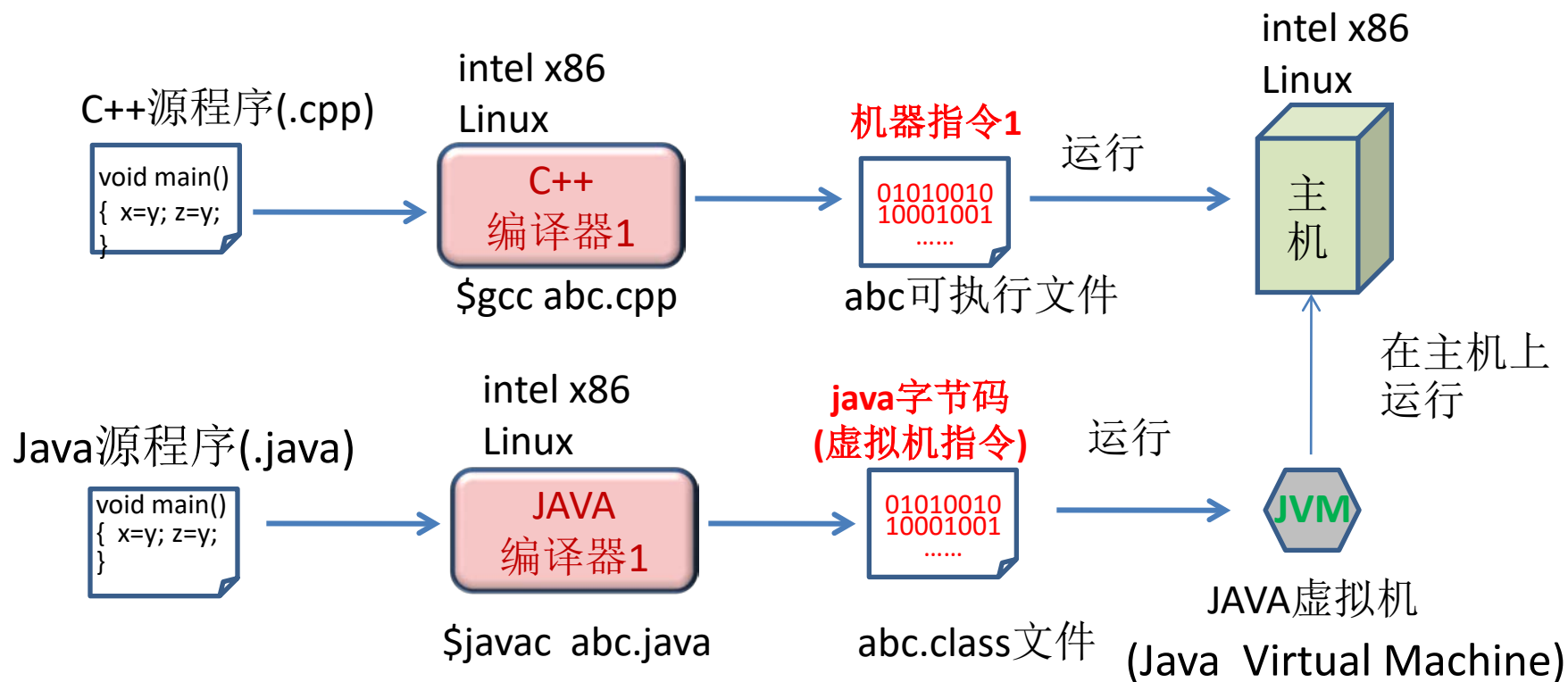
Swift

TypeScript



C++编译器和Java虚拟机

- C++源程序(.cpp)采用intel x86 Linux的C++编译器gcc编译成机器指令文件。该文件只能在intel x86 Linux系统上执行。
- JAVA源程序(.java)采用intel x86 Linux的Java编译器javac编译成JAVA字节码文件(.class)。该文件可以在所有安装了Java虚拟机(JVM)的系统上执行。



Java的特点

□ 简单(Simple)

设计Java的目的之一就是简化C++的功能，使其易于学习和使用。它**没有指针类型(pointer)**，避免了内存溢出等安全性问题。采用**垃圾收集器(garage collection)**自动收集存储空间的垃圾，使程序员摆脱了时常忘记释放存储空间的苦恼。

□ 易于移植(Portable)

通过使用Java字节码，Java支持交叉平台代码，Java程序可以在任何可以运行Java虚拟机的环境中执行，其执行速度被高度优化，有时甚至超过了C++的程序。

□ 面向对象(Object-oriented)

Java语言中一切都是对象，并且Java程序带有大量在运行时用于检查和解决对象访问的运行时(run-time)类型信息。

JRE和JDK

- Java平台有两个主要产品：Java Runtime Environment (JRE) 和Java Development Kit (JDK) 。
- JRE提供Java库、Java虚拟机以及运行Java应用程序所需的其它组件。字节码文件就是在JRE下运行的。
- JDK是JRE的超集，用于开发Java小程序和Java应用程序。它包括了JRE所有的内容, 并且加入了编译器和调试器等工具。JDK共有三个版本，分别用于不同的目的：

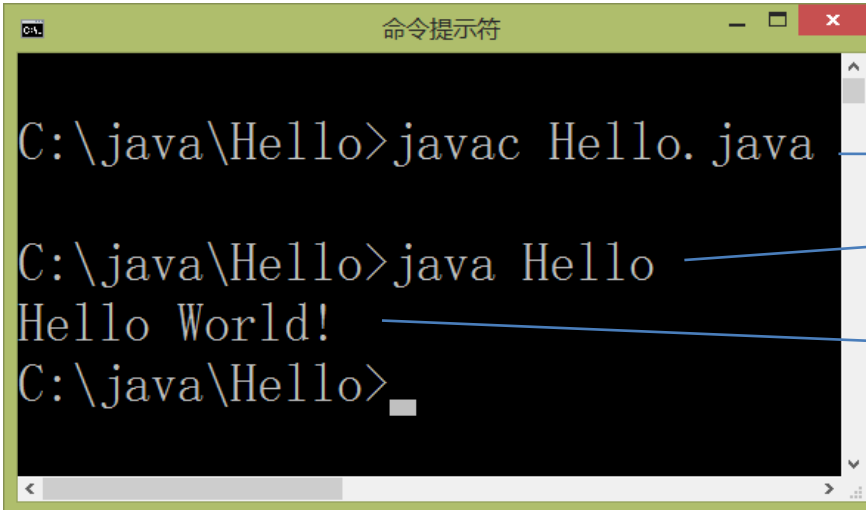
- ✓ **Standard Edition(Java SE):** 标准版，是最常用的一个版本。
- ✓ **Enterprise Edition(Java EE):** 企业版，用于开发大型Java应用程序。
- ✓ **Micro Edition(Java ME):** 微型版，用于移动设备上java应用程序。

安装JDK的方法见附录2

第一个Java程序

文件名: Hello.java

```
public class Hello                // 类名，要与文件名一致!!!
{
    public static void main(String args[]) // 主程序入口
    {
        System.out.print("Hello World!"); // 显示Hello World!
    }
} // print()为系统对象System.out的方法
```



```
C:\java\Hello>javac Hello.java
C:\java\Hello>java Hello
Hello World!
C:\java\Hello>
```

编译并生成Hello.class

运行Hello.class

显示结果

安装Java见附录

Javac -encoding UTF-8 Hello.java (如果源程序采用了UTF-8编码而不是ansi编码)

数据类型

- 程序的每个数据都需要以一定的格式存放，并可以参与运算，这就需要定义数据类型，比如，整数类型、字符类型等。
- Java中共有8种基本数据类型：

类型	字节	数据容器(类)	数的范围	默认值
byte	1	Byte	-128~127	0
short	2	Short	-32768~32767	0
int	4	Integer	$-2^{31} \sim 2^{31}-1$	0
long	8	Long	$-2^{63} \sim 2^{63}-1$	0
float	4	Float	$3.4e^{-038} \sim 3.4e^{+038}$	0.0
double	8	Double	$1.7e^{-308} \sim 1.7e^{+308}$	0.0
char	2	Character	0~65535	0
boolean	1	Boolean		false

*可以用两种方法定义整数变量：(1)int x=0; (2)Integer y=0; 它们的使用方法相同，但是x是基本数据，y是对象。

* 默认值为基本数据类型作为数据域的默认值。对象作为数据域的默认值为null。

常量

常量是指程序中不变的量。通常需要为常量命名后使用。

布尔常量:	<code>true false</code>	
整型常量:	<code>100</code> (10进制整数) <code>016</code> (8进制整数) <code>0x2EF</code> (16进制整数) <code>386L</code> (长整数)	
浮点常量:	<code>19.6f</code> <code>3.14E3F</code> (3.14×10^3)	-- float
	<code>3.14</code> <code>2.536D</code> <code>3.1415926E-3D</code>	-- double
字符常量:	<code>'a'</code> 、 <code>'8'</code> 、 <code>'#'</code> 、 <code>'\n'</code> (换行)、 <code>'\r'</code> (回车) <code>'\\'</code> 、 <code>'\"'</code> 、 <code>'\''</code> 、 <code>'\u0027'</code> (单引号, unicode码)。	

*其中所有字母不区分大小写

```
final int LEVEL_NUM=0x1000; //为常量命名是个好习惯
```

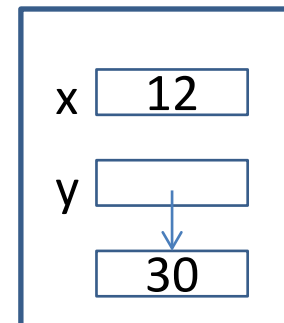
变量

- 如果有一个值在计算过程中会发生变化，比如，测量的气温值，就需要把它定义为**变量**。
- 变量是通过赋值语句改变其值。赋值语句用等于号连接变量和表达式，变量位于左边，表达式位于右边。表达式计算出来的值用于修改变量的值。
- 首先要为变量取一个名字，然后再给它赋值。**Java的变量命名规则**：
\$、字母、下划线开头的数字字母下划线串。为了分配内存和进行计算，还需要为变量定义数据类型。

```
float temperature;    // 定义一个浮点类型的变量
temperature = 26;     // 把右边的值赋给左边的变量
```

- 系统为每个变量都会分配一个存放位置，用来存放该变量的值，也可以存放一个指针，指向存放变量值的地方。

```
int x = 12;
Integer y = 30; //y为对象
```



运算符与表达式

➤ 算术表达式：用算术运算符形成的表达式，计算结果为整值

```
int x = 5, y = 6, z = 10;    // 一次定义多个变量，并赋初值
int exp = (y + 3) * z + x;    // 右边为算术表达式，exp得值95
```

➤ 关系表达式：用关系运算符形成的表达式，计算结果为真假值

```
int x = 300;
boolean r1 = x > 10;    // x是否大于10。r1得值true
boolean r2 = x <= 100;   // x是否小于等于100。r2得值false
```

➤ 逻辑表达式：用逻辑运算符形成的表达式，计算结果为真假值。与C语言不同，**在Java的逻辑表达式中，非0不会自动当成true。**

```
int y = 99;
boolean r3 = (y > 10) && (y < 100); // y是否大于10并且小于100. true
```

➤ 位表达式：用位运算符和移位运算符形成的表达式，计算结果为整数(int)。

```
int z = 16;
int r4 = z | 0x1E0F; // 按位或。0x1E1F (7711)
```

算术运算符: $a+b$ $a-b$ $a*b$ a/b (商) $a\%b$ (余数) i/j (整除,i和j为整数)
 关系运算符: $a>b$ $a<b$ $a>=b$ $a<=b$ $a==b$ (等于) $a!=b$ (不等于)
 逻辑运算符: $a\&\&b$ (短路与, a为false时不计算b) $a||b$ (短路或) $!a$ (非)
 位运算: $\sim a$ (按位非) $a\&b$ (按位与) $a|b$ (按位或) a^b (按位异或)
 移位运算: $b<<1$ (左移1位) $a>>2$ (带符号右移2位) $b>>>3$ (无符号右移3位)
 三目运算: $x<3?10:7$ (如果x小于3, 则取值10, 否则, 取值7)
 单目运算: $++x$ (x先加1, 再参与运算) $--x$ $x++$ $x--$ $-x$ (变符号)
 赋值运算: $x+=a$ ($x=x+a$) $x-=a$ $x*=a$ $x/=a$ $x\%=a$ $x\&=a$
 $x|=a$ $x\&=a$ $x|=a$ $x^a=a$ $x>>=a$ $x>>>=a$ $x<<=a$

运算优先级:

高 $[]()$ \rightarrow 单目 $\rightarrow * / \% \rightarrow + - \rightarrow << >> >>>$
 $\rightarrow < > <= >= \rightarrow == != \rightarrow \& \rightarrow ^$
 $\rightarrow | \rightarrow \&\& \rightarrow || \rightarrow$ 三目运算 \rightarrow 复杂赋值 低

类型转换

- 如果赋值语句的变量与值的数据类型不一致，系统会进行**隐式类型转换**。

```
int x = 100;  
long y = x;
```

- 如果要把取值范围小的数据类型转化为取值更大的数据类型，则要用**强制类型转换**。

```
long x = 100;  
int y = (int)x;
```

- 强制类型转换要注意出现截断错误。

```
Long i = 65536;  
short j = (short)i; //j赋值为0  
double x = 10.876;
```

- ```
int y = (int)x;
```

 //y赋值为10（取整）。Math.round()四舍五入
- 对于移位运算，char、byte和short类型会先变为int类型再进行移位运算，long和int类型直接进行移位。

# Java基本语句

- 赋值语句

|                              |                     |
|------------------------------|---------------------|
| <code>int x;</code>          | //变量定义              |
| <code>x = 20;</code>         | //赋值语句，左边为变量，右边为表达式 |
| <code>float y = 1000;</code> |                     |
| <code>x = (int)y;</code>     | //强制类型转换            |

- 注释语句

|                           |                                                                                  |
|---------------------------|----------------------------------------------------------------------------------|
| <code>//</code>           | 注释一行                                                                             |
| <code>/* ..... */</code>  | 注释若干行                                                                            |
| <code>/** ..... */</code> | 文档注释， 可以用javadoc提取，产生html文件。Java会自己加入标题，自动加入public和protected成员。每行开头的星号和空格不会包含进去。 |



## ● 分支控制语句

```
if (逻辑表达式){
 语句1;
 语句2;
 ...
}

if (逻辑表达式){
 语句1;
 语句2;
 ...
}
else{
 语句1;
 语句2;
 ...
}
```

//逻辑表达式取值true或false。

```
if (逻辑表达式1) {
 语句1;
 语句2;
 ...
}
else
if (逻辑表达式2){
 语句1;
 语句2;
 ...
}
else{
 语句1;
 语句2;
 ...
}
```

\*条件嵌套: else属于最靠近它的if

```
//Statements.java
String bkcolor;
String item= "table";
if(item.equals("table")){
 bkcolor="white";
}
else{
 bkcolor="black";
}
System.out.println("The back color is" + bkcolor);

int age = 20;
if(age<1)
 System.out.println("婴儿");
else if(age>=1 && age<10)
 System.out.println("儿童");
else if(age>=10 && age<18)
 System.out.println("少年");
else if(age>=18 && age<45)
 System.out.println("青年");
else
 System.out.println("中年或老年");
```

**switch**(整数表达式)

```
{
 case 数值1: 语句1;语句2;...;break;
 case 数值2: 语句1;语句2;...;break;
 ...
 case 数值n: 语句1;语句2;...;break;
 default:语句1;语句2;...
}
```

// 整数表达式的类型为byte, char, short, int。

//Statements.java

```
int cnt = 10;
double x;
switch(cnt){
 case 1: x=5.0;break;
 case 12: x=30.0;break;
 default: x=100.0;
}
System.out.println("x="+x);
```

## ● 循环控制语句

```
for(表达式1;布尔表达式2;表达式3)
```

循环体

```
for(type variable: collection) //foreach语句
```

循环体

```
while(逻辑表达式)
```

循环体

```
do
```

循环体

```
while(逻辑表达式)
```

```
break [标号];
```

```
continue [标号];
```

```
return 表达式;
```

- ✓ 循环体可以是单条语句，也可以是用括号{}括起的语句序列。
- ✓ 表达式1和表达式3为逗号隔开的多条语句，分别用于进入循环执行一次和每次循环都执行一次。
- ✓ break和continue加标号表示跳出加标号的循环语句或跳到加标号的循环语句执行。

```
//Statements.java
```

```
int sum=0;
```

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

```
 sum=sum+i;
```

```
}
```

```
System.out.println("sum1(1~100)="+sum);
```

```
double sum=0;
```

```
cnt = 0;
```

```
double scores[]={100.0, 90.2, 80.0, 78.0,93.5};
```

```
for(double score:scores){
```

```
 sum=sum+score;
```

```
 cnt++;
```

```
}
```

```
System.out.println("avg score="+sum/cnt);
```

```
sum=0;
```

```
int k=0;
```

```
while(k<=100){
```

```
 sum=sum+k;
```

```
 k++;
```

```
}
```

```
System.out.println("sum2(1~100)="+sum);
```

foreach语句

```

sum=0;
k=0;
do{
 sum=sum+k;
 k++;
}while(k<=100);
System.out.println("sum3(1~100)="+sum);

```

/\* 求距阵之和

```

* 1 2 3 ... 10
* 1 2 3 ... 10
*
* 1 2 3 ... 10
*/

```

```

sum=0;
for(int i=1;i<=10;i++){
 for(int j=1;j<=10;j++){
 sum=sum+j;
 }
}
System.out.println("triangle1="+sum);

```

//块定义域

```

void f(){
 int k=0;

 ...

 if(){
 int n =5;
 int k =5; //错误！！
 }
}

```

```

sum=0;
Label1:
for(int i=1;i<=10;i++){
 for(int j=1;j<=10;j++){
 if(j==i){
 continue; //跳到for结束处继续执行
 }
 sum=sum+j; //除去对角线的矩阵之和
 } ← continue跳到这里
}
System.out.println("triangle2="+sum);

```

```

sum=0;
Label2:
for(int i=1;i<=10;i++){
 for(int j=1;j<=10;j++){
 if(j==i){
 break; //跳出for循环继续执行
 }
 sum=sum+j; //下三角加对角线矩阵之和
 }
} ← break跳到这里
System.out.println("triangle3="+sum);

```

# 数组

- 数组也是一个对象(Arrays)，用于存储一系列相同类型的数据。数组的优点是存储和访问效率高，缺点是不能改变元素个数。初始化数组后，如果数组元素为基本数据类型，则自动取默认值，否则取值null。

```
// ArrayDef.java
import java.util.Arrays;
int sample[]; // 定义数组对象（未初始化，不能使用）
sample = new int[8]; // 初始化数组，分配8个元素的存储空间。
sample[7]=100; // 数组引用方法
System.out.println(sample[7]); // 显示第7个元素：100。下标从0开始
System.out.println(sample[0]); // 显示：0（默认值）。
int rnds[];
rnds = new int[]{1,3,4,5,6};
System.out.println(Arrays.toString(rnds)); //显示： [1,3,4,5,6]
char[] chars = {'我', '是', '中', '大', '人'}; //初始化一维字符数组
System.out.println(chars[3]); // 显示第4个的字符：和。
String[] s1= {"John", "Wade", "James"}; // 初始化一维字符串数组
System.out.println(s1[1]); // 显示从第1个字符串：Wade
```



```
int nums[] = {9, -10, 18, -978, 9, 287, 49, 7};
for(int num:nums){ // 枚举循环法
 System.out.println(num); // 显示数组nums的全部元素
}
double map[][] = new double[3][10]; // 定义二维数组：3行10列
map[0][9] = 20;
System.out.println(map[0][9]);
```

// **Java只有一维数组**，二维数组为数组的数组，所以数组的每行的列数是可变的。

```
int table[][] = {{1},{2,3,4},{5,6,7,8}}; // 二维数组（可变长）
for(int i=0; i<table.length; i++){ // table.length为行数
 for(int j=0; j<table[i].length; j++){ // 处理每行的元素
 System.out.println(table[i][j]); // 显示第i行第j列的元素
 }
}
```

//下面程序做了什么？

```
int table1[][] = new int[10][];
for(int j=0; j<table1.length; j++){
 table1[j]=new int[j+1];
}
```

```

// ArrayOp.java
import java.util.Arrays; //导入数组类
char s1[]={ 'H', 'e', 'l', 'l', 'o' };
s1=Arrays.copyOf(s1,8); // 复制出一个8元素数组:Hello*** *为null字符
System.out.println(s1); // Hello***
char s2[];
s2=Arrays.copyOf(s1,3); // 复制: s2得到一个3元素数组:Hel
char s3[]=Arrays.copyOfRange(s1, 1, 3); // 复制: s3得到el

Arrays.fill(s2, 'a'); // 把s2的全部元素填充为a
System.out.println(s2); // 结果:aaa
Arrays.fill(s3,2,5, 'o'); // 把s1的第2~4个元素填充为o
System.out.println(s3); // 结果:Heoo
boolean r = Arrays.equals(s1,s2); //比较元素个数和值是否都相等:false
System.out.println(r);

```

```
int pos=Arrays.binarySearch(s1,'1'); // （二分）查找值为1的元素
System.out.println(pos);
Arrays.sort(s1); // 排序s1:***Hello *为null
System.out.println(s1);
int a[]={3,5,4,26,19,2,9};
Arrays.sort(a,1,5); // 排序第1~4个元素:3,4,5,19,26,2,9
for(int x:a){
 System.out.println(x);
}
```

\* `binarySearch()`:使用二分搜索算法来搜索指定的 `int` 型数组，以获得指定的值。必须在进行此调用之前对数组进行排序（通过上面的 `sort` 方法）。如果没有对数组进行排序，则结果是不明确的。如果数组包含多个带有指定值的元素，则无法保证找到的是哪一个。

# 字符串

字符串类型(String)为一个用于文字操作的类，内部采用Unicode编码（一个字符用两个字节表示）。

```
// StringDef.java
import java.util.Arrays;
char c1[] = {'a','b','c','d','e'};
String s1 = "Hello";
String s2 = new String("World");
String s3 = new String(c1);
String s4 = new String(c1,1,3);
String s5[]={ "This", "is", "a", "test."}; // 字符串数组
String s6 = s1.concat(s2);
String s7 = s1 + s2;
boolean b1 = s1.equals(s2);
boolean b2=s1.equalsIgnoreCase("hello"); // 相等比较，忽略大小写.true
boolean b3 = (s1==s2); // 是否为同一个对象(变量值)
boolean b4 = s1.isEmpty(); // 是否为空串。与equals("")相同
int len = s1.length(); //字符串长度。 5
String s8 = Arrays.toString(s5); //把数组元素变为逗号隔开的字符串
```

```

// StringOp.java
import java.util.Arrays;
import java.util.regex.*;

String s1 = "Hello";
char c1[]={ 'W', 'o', 'r', 'l', 'd' };
String s2 = String.copyValueOf(c1);

String s3 = s1.toUpperCase();
String s4 = s1.toLowerCase();
String s5 = s1.substring(0,4);
String s5a= s1.substring(2);
char c2 = s1.charAt(4);
String[] s6 = s1.split("e");
String s7 = s1.replace("l", "L");
String s8 = s1.replaceAll("l", "L");
String s9 = s1.replaceFirst("l", "L");
String s8a = s1.replaceAll("[Hl]", "L");
String s9a = s1.replaceFirst("[Hl]", "L");
String s10 = " We learn Java";
boolean b1=s10.endsWith("Java");
boolean b2=s10.startsWith("We");
String s11=s10.trim();

```

//把字符数组变为字符串:World

//变大写字母:HELLO  
//变小写字母:hello  
//第0到3个字符的子串:Hell  
//第2个字符开始的子串:llo  
//取第4个字符:o.  
//分割字符串: "H","llo"  
//替换字符串(所有):heLLo。  
//替换字符串(所有):heLLo  
//替换第一次出现:heLlo  
//替换所有字符H或l:LeLLo  
//替换第一次出现字符H或l:Lello

// 以什么子串结尾:true  
// 以什么子串开始:false  
// 删除头尾空格:We learn Java

```

int i=25;
String s12=String.valueOf(i); // 把整数转换为字符串
boolean b3=s10.contains("learn"); // 是否包含子串learn: true
int pos1 = s11.indexOf("e"); // 匹配子串的索引:1.未找到返回-1.
int pos2 = s11.indexOf("e",3); // 匹配子串的索引:4。从位置3开始
int pos3 = s11.lastIndexOf("e"); // 从尾部往前一个匹配子串的索引:4
s12=String.format("%05d,%s",501,"op"); //格式化字符串: 00501,loop
String regex = "(;|,)"; //正则表达式 :;或,
String[] s14= "a;b,c;d".split(regex); //以;或,拆分字:"a","b","c","d"

boolean b4=s11.matches("^We.*") ; //匹配正则表达式(以We开头):true.
int n1 = "abcd".compareTo("abcD"); //词典序:32 0等于<0小于>0大于
int n2 = "abcd".compareToIgnoreCase("abcD"); //忽略大小写

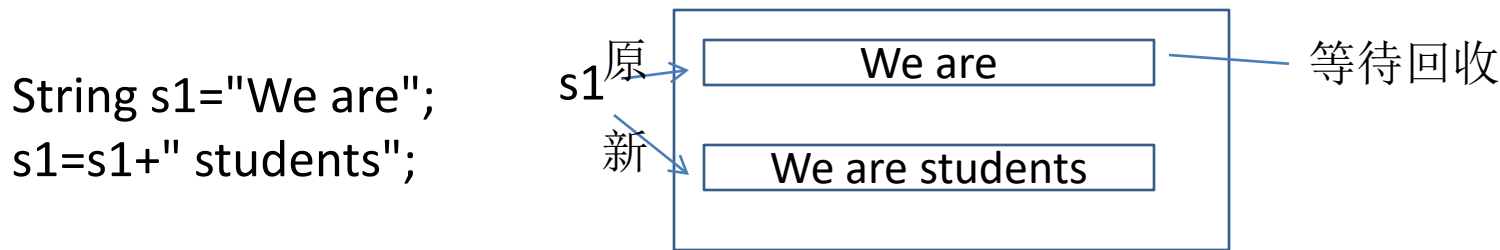
bytes[] s14 = s11.getBytes("ISO-8859-1"); //转换成字节数组
String s15 = new String(s14,"GB2312"); //s14内需要保存GB2312编码的字节数组
int n1=s1.codePointAt(i); //取到s1的第i字符的编码(UNICODE)

```

Java的源程序默认是采用操作系统默认的编码，例如，简体中文Windows采用ANSI编码(实际为GBK编码)，如果采用其它种类的编码，编译时就要使用encoding选项：javac -encoding utf-8 ex5.java，否则，编译时会出错。

\*ISO-8859-1编码包含所有欧洲字符，常用于转换为单字节数组。

**String 是不可变的对象**，每次修改其内容都会生成了一个新的 String 对象，原来的String对象将不再使用，并等待垃圾回收器自动清理它们，因此，经常改变字符串内容是十分低效的。



如果需要对字符串频繁修改，可以采用StringBuilder和StringBuffer提高效率。StringBuffer和StringBuilder (java.lang.\*)都是通过缓冲区操作字符串的对象，主要操作有 append 和 insert 方法：

```
StringBuffer s20=new StringBuilder("uv");
s20.append("xyz"); // 并入末尾。uvxyz
s20.insert(3,"w"); //插入到中间。uvwxyz
String s21=s20.toString(); //取出s20的内容
```

StringBulider用于单线程环境。StringBuffer对方法有同步机制，可以用于多线程环境。对于单线程编程，StringBulider比StringBuffer更有效率。

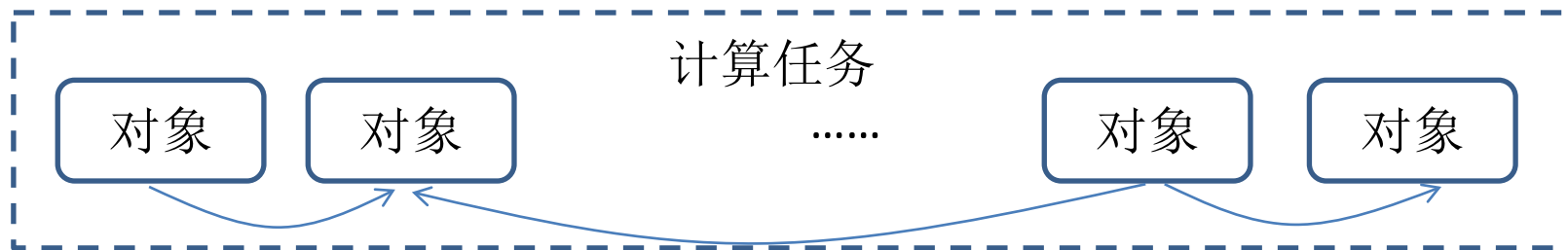
## StringBuffer和StringBuilder对象的常用方法

```
append(String str) // 数值变量等类型也可以作为参数
insert(int offset, String str) // 第二个参数和append一样
delete(int start, int end) // 删除一个子串
indexOf(String str, int fromIndex) // 从fromIndex开始查找一个子串
replace(int start, int end, String str) // 从start到end替换一个子串
substring(int start, int end)
reverse() // 字符反转。"abc"=>"cba"
length()
toString() // 取出缓存的字符串
```



# 对象和类

- 要完成一个计算任务必然要涉及很多事物，在面向对象程序设计中把这些事物称为**对象(object)**，并把具有相同属性和操作的对象划分为一类，定义为类(class)。因此，对象也称为类的实例。
- 与**面向过程程序设计**采用函数调用完成计算功能不同，**面向对象程序设计**是通过对象之间的相互作用来完成计算任务。



- 在Java中，对象的属性和操作被称为**数据域(data field)**和**方法(method)**，也被称为**成员变量(数据成员)**和**成员函数**，统称为对象的成员。
- 面向对象程序设计方法有哪三个主要特征？

(1) 封装性

(2) 继承性

(3) 多态性

如何通过需求设计类？

**封装性(Encapsulation)**是指把属性和操作封装为一个整体，使用者不必知道操作实现的细节，只是通过对象提供的操作来使用该对象提供的服务。

下面的例子是计算面积的例子，调用者不需要知道面积的计算方法。

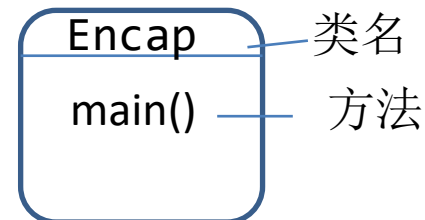
//Encap.java

```
class Rectangle {
 double height;
 double width;
 double getArea(){
 double area=height*width; //局部变量
 return area; //返回值
 };
}
```

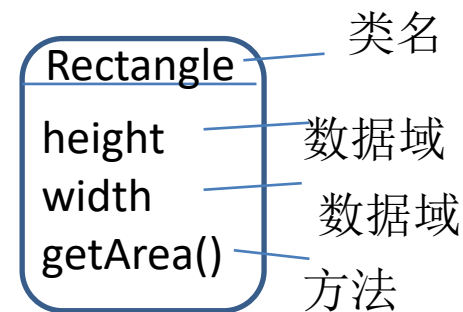
```
public class Encap {
 public static void main(String args[]){
 Rectangle rect = new Rectangle();
 rect.height=10;
 rect.width=20;
 System.out.println(rect.getArea());
 }
}
```

//运行结果: 200.0

//类名  
//数据域(成员变量)  
//数据域(成员变量)  
//方法(成员函数)  
//局部变量  
//返回值



//类名  
//方法  
//建一个新对象  
// 数据域赋值  
// 数据域赋值



```
void setHeight(val){ height=val;}
int getHeight(){return height;}
```

一个java文件可以包含很多class，但是只能有一个public的class。类的信息保留在其class文件的头部，在装载类时会装入内存中作为Class类的实例

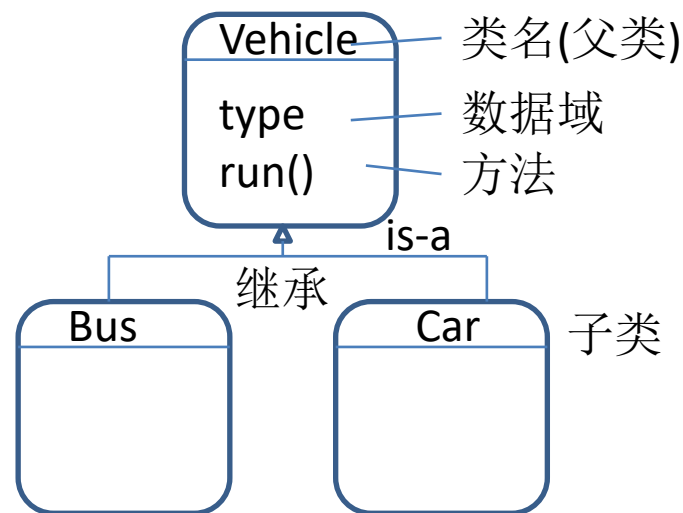
通过**继承性(Inheritance)**，子类（导出类）可以自动共享父类（积累）的属性和操作。子类与父类之间是**is-a**（完全继承）或**is-like-a**（加入了自己的属性或方法）的关系。Java的所有基类自动继承内部的Object类。

下面是继承性的例子：

```
class Vehicle{ //父类(基类)名
 int type; //数据域
 void drive(){ //方法
 System.out.println("run!" + this.getClass());
 };
}
class Bus extends Vehicle{ //子类(导出类)名 is-a
}
class Car extends Vehicle{ //子类名 is-a
}
```

```
public class Inherit{
 public static void main(String args[]){
 Bus bus = new Bus();
 bus.drive();
 Car car = new Car();
 car.drive();
 }
}
```

```
//运行结果: run! class Bus
// run! class Car
```



**多态性(Polymorphism)**是指对属于同一类的不同对象采用相同的操作时可以产生完全不同的行为。

下面是多态性的例子：

```
class Shape{ // 父类(基类)
 int color; // 数据域
 void draw(){ // 方法
 System.out.println("draw!" + this.getClass());
 }
}

class Circle extends Shape{ // 子类(导出类)
 void draw(){ System.out.println("draw!" + this.getClass());};
 // super.draw()可以用于访问父类中的方法
}

class Square extends Shape{
 void draw(){
 System.out.println("draw! " + this.getClass());
 }; //覆盖(override)才是多态
}

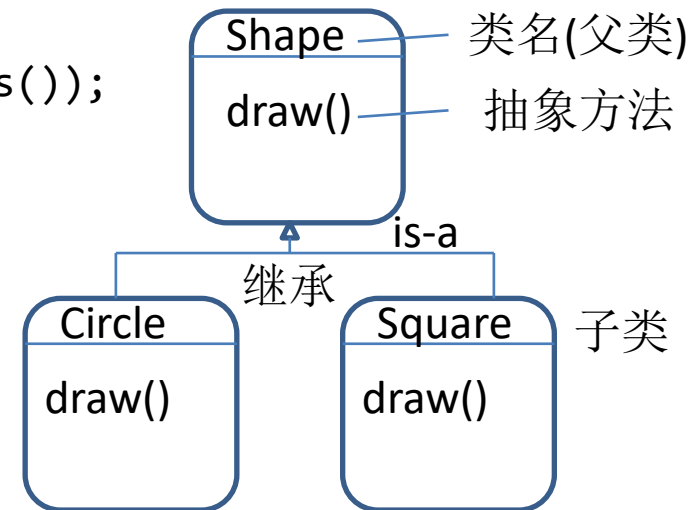
public class Poly{
 public static void main(String args[]){
 Shape shape1 = new Shape(); shape1.draw();
 Shape shape2 = new Square(); shape2.draw();
 Shape shape3 = new Circle(); shape3.draw();
 }
}
```

—— 向上转型，后期绑定(运行时绑定)

运行结果：

```
draw! class Shape
draw! class Square
draw! class Circle
```

Shape类的三个对象调用draw()得到的结果不同。



\* 编译时绑定为前期绑定。

# 包

- 概念

Java中的一切都是对象(类)。Java有很多类，为了有效地访问它们，必须对它们进行组织。包(package)是一个由相关类和接口组成的集合的命名空间，是对类的一种分类和组织方式。你可以把包看成计算机中不同的目录。实际上，Java就是通过子目录来管理class文件的。同一个子目录下的所有.class文件被认为在同一个包中。还可以把一个或者多个这样的子目录打包成一个Java的类库文件（.jar文件）。

- 定义和引用包

Java源文件在第一行定义包名，例如，`package com.group.food`。一般使用网站名(group.com)来保证包名的唯一性。没有定义包的文件属于默认包。

要引用包com.group.food中定义的类Cookie需要在源文件前加入import com.group.food.Cookie。用\*代替Cookie可以引入该包的所有类。如果不使用import，则需要使用包名引用类：`com.group.food.Cookie v = new com.group.food.Cookie()`。在出现引用出现在多个包的重名类时，也必须采用这种做法。

[参考](#)

## 使用包的例子

- Cookie.class和Bread.class放在目录c:\java\PackEx\com\group\food下

```
package com.group.food;
public class Cookie {
 public void eat(){
 System.out.println("Eat cookie");
 }
}
// Cookie.java
```

```
package com.group.food;
public class Bread {
 public void eat(){
 System.out.print("Eat bread");
 }
}
// Bread.java
```

- PackEx.java放在目录c:\java\PackEx下,编译时会自动编译包的类

```
import com.group.food.*; // PackEx.java
public class PackEx{
 public static void main(String[] args){
 Cookie cookie = new Cookie();
 cookie.eat();
 Bread bread = new Bread();
 bread.eat();
 }
}
// 运行结果: Eat cookie
// Eat bread
```

如果Cookie.java和Bread.java都放入包com.group.food中,编译PackEx.java时会自动编译它们。

采用环境变量CLASSPATH指明查找包的路径,例如, set CLASSPATH =c:\java\PackEx;c:\classes;. 定义了三个查找路径(.为当前路径)。

# 使用jar文件的例子

第一步、把.class文件放在子目录(com\group\food)中。

```
package com.group.food;
public class Corn {
 public void eat(){
 System.out.print("Eat corn");
 }
} // Corn.java
```

```
package com.group.food;
public class Rice {
 public void eat(){
 System.out.println("Eat rice");
 }
} // Rice.java
```

第二步、执行命令"jar cvf food.jar \*" 得到文件food.jar （内部包含了子目录信息）

第三步、把food.jar放在c:\java\PackJar\中，加入c:\java\PackJar\food.jar到CLASSPATH。

```
// PackJar.java
import com.group.food.*;
public class PackJar{
 public static void main(String[] args){
 Rice rice = new Rice();
 rice.eat();
 Corn corn = new Corn();
 corn.eat();
 }
}
// 运行结果: Eat cookie
// Eat bread
```

# • Java程序如何找到并引用所需要的类？

- 通过设置CLASSPATH告诉Java执行环境在哪些目录下可以找到要执行的Java程序所需要的类或者包。
- 如果设置了CLASSPATH，则只在其设置的目录中查找所需的类和包，否则，只在当前目录中查找。
- 没定义包的class文件使用默认包，默认包只能放在CLASSPATH设置的目录下。
- Jar包必须直接在CLASSPATH中指明路径，Java还会自动查找JDK或JRE的lib目录的jar包。
- 编译和运行时可以用参数-CLASSPATH或-cp指出CLASSPATH：  
    javac -cp c:\java\PackJar\food.jar;c:\java;. PackJar.java  
    java -cp c:\java\PackJar\food.jar;c:\java;. PackJar
- CLASSPATH或cp的路径中不要使用环境变量和~，但是在环境变量CLASSPATH中可以用其它环境变量。
- Linux下的CLASSPATH使用冒号“:”间隔目录，不是用分号“;”



# 访问权限

- 类的访问权限

|         |               |
|---------|---------------|
| 没有修饰词   | 只能被同一个包的类所访问  |
| public: | 能被任何类在任何地点所访问 |

\* 只有内部类有private和protected权限。内部类还可以定义为static的。

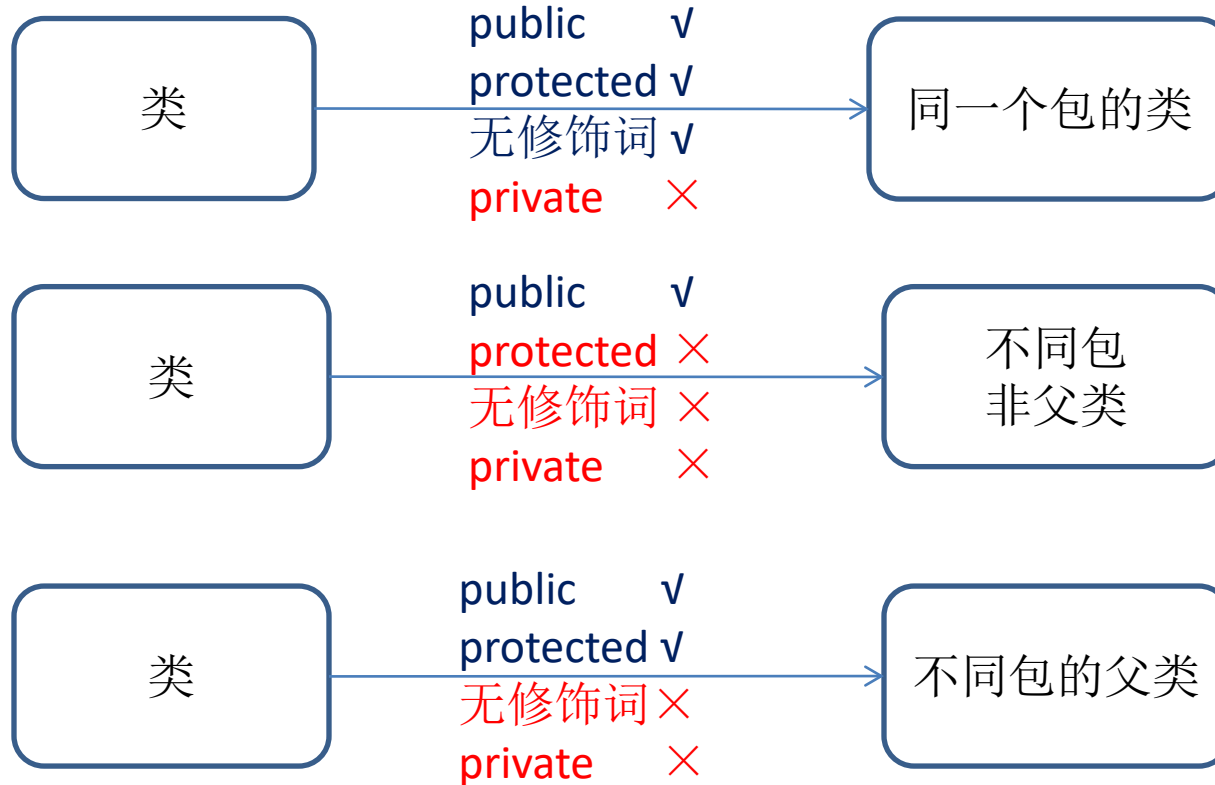
- 数据域和方法的访问权限

|           |                              |
|-----------|------------------------------|
| public    | 能被任何类在任何地点所访问（通过类的方法）        |
| protected | 只能被导出类和同一个包的类所访问             |
| 没有修饰词     | 只能被同一个包的类所访问(类似C++的friendly) |
| private   | 只能被同一个类的方法所访问。               |

总结：一个方法可以访问同一个包的除了private的所有数据和方法，只能访问不同包的所有public数据和方法以及父类的protected数据和方法。

\* 为了更好地控制对成员变量的访问，经常把成员变量(例如，Variable)设置为private，采用getVariable()和setVariable()的方法取值和设置。

## 方法和属性的 访问权限



- 一个.java文件中可以放入多个类，它们都属于同一个包的类，会被分别编译为.class文件。这些类中只能有一个是public的。

# 容器类和映射类

- 数组对最大的元素个数有限制，不能进行插入和删除，查找元素速度也很慢，使用容器类可以解决这些问题。
- **容器类Collection**可以存放需要经常增加元素或删除元素的一组元素，其主要子类为**Set**(无重复无序)和**List**(有序可重复):
  - (1) **HashSet**、**TreeSet**、**LinkedSet**为三种**Set**子类。它们分别用Hash表、红黑树和链表作为集合的数据结构，性能不同但是操作类似。
  - (2) **LinkedList**和**ArrayList**是两类**List**子类，分别采用链表和数组实现。
- **映射类Map**可以将键(Key)映射到值(Value)。其中的键值不能重复，每个键值只映射到一个值。**HashMap**和**TreeMap**是其子类，其中，**TreeMap**为有序类。

容器类还有**Vector**类、**Stack**类、**Queue**类。**LinkedList**可以作为队列使用，**Vector**类可以用于多线程环境，**Stack**类是**Vector**类的子类。还有其他一些集合类。**Dictionary**也是一种映射类(已过时)，**Hashtable**为其子类(已过时)。

## • ArrayList和LinkedList

ArrayList是一个用顺序存储结构实现线性表的类，随机访问速度快，插入删除操作比较慢，而LinkedList用链表实现，插入删除快，随机访问速度慢。

```
import java.util.*;
public class ArrayListTest {
 public static void main(String[] args) {
 ArrayList<Student> hs = new ArrayList<Student>(); // 泛型
 Student stu1 = new Student(101, "Wang");
 Student stu2 = new Student(102, "Li");
 Student stu3 = new Student(103, "He");
 Student stu4 = new Student(112, "李四");
 hs.add(stu1); // 尾部追加
 hs.add(stu2);
 hs.add(stu3);
 hs.add(1, stu4); // 中间插入
 Iterator<Student> it = hs.iterator(); // 顺序取出
 while (it.hasNext()) {
 System.out.println("***" + it.next());
 }
 System.out.println("+++" + hs.get(2)); // 随机取出
 }
}
```

<Student>为泛型(generic type)，指明列表中元素的类型。泛型要求使用类，不能使用基本数据类型。

```

class Student {
 int num; // 学号
 String name; // 姓名
 Student(int num, String name) { this.num = num; this.name = name; }
 public String toString() { return num + " " + name; }
}

```

ArrayList和LinkedList的主要方法:

```

boolean add(E e) // 将指定的元素添加到列表尾部。E为泛型
void add(int index, E e) // 将指定的元素插入列表指定位置。
void clear() // 移除列表的所有元素。
boolean contains(Object o) // 如果列表包含指定的元素，则返回 true。
E get(int index) // 返回列表指定位置上的元素。
int indexOf(Object o) // 返回列表首次出现的指定元素的索引或 -1。
boolean isEmpty() // 如果列表为空，则返回 true
int lastIndexOf(Object o) // 返回列表最后一次出现指定元素的索引或 -1。
E remove(int index) // 移除此列表中指定位置上的元素。
boolean remove(Object o) // 移除列表中首次出现的指定元素（如果存在）。
E set(int index, E e) // 用指定的元素替代此列表中指定位置上的元素。
int size() // 返回此列表中的元素数。
void trimToSize() // 将此 ArrayList 实例的容量调整为列表的当前大小
protected void removeRange(int fromIndex, int toIndex)
 // 移除 fromIndex和 toIndex(不包括)之间的所有元素。

```

\* 泛型E可以使用用户自定义类或标准类(Integer, String等)

## • HashMap和TreeMap

HashMap和TreeMap是分别用哈希和红黑树结构的方法实现键值映射的类，随机访问和插入删除操作很快，HashMap要更快一些，但是内存占用量更大。TreeMap的键值可以有序遍历，而HashMap不行。

```
import java.util.*; HashMapTest.java
public class HashMapTest {
 public static void main(String[] args) {
 HashMap<Integer, Student> map = new HashMap<Integer, Student>();
 Student stu1 = new Student(103, "He");
 Student stu2 = new Student(112, "李四");
 Student stu3 = new Student(101, "Wang");
 map.put(103, stu1); // 加入新的键值对
 map.put(112, stu2);
 map.put(101, stu3);
 Iterator<Integer> it = map.keySet().iterator(); // 无序遍历
 while (it.hasNext()) {
 Integer key = (Integer) it.next();
 Student value = map.get(key); // 根据键值取出值
 System.out.println("***" + value.toString());
 }
 System.out.println("+++" + map.get(112));
 }
}
```

## HashMap的主要方法:

|                                            |                                      |
|--------------------------------------------|--------------------------------------|
| <b>void</b> clear()                        | // 从此映射中移除所有映射关系。                    |
| Object clone()                             | // 返回此 <b>HashMap</b> 实例的副本(不复制键和值)  |
| <b>boolean</b> containsKey(Object key)     | // 是否包含指定键                           |
| <b>boolean</b> containsValue(Object value) | // 是否包含指定值                           |
| V get(Object key)                          | // 返回指定键所映射的值或null。                  |
| <b>boolean</b> isEmpty()                   | // 映射集是否为空。                          |
| Set<K> keySet()                            | // 返回此映射中所包含的键的 <b>Set</b> 视图。       |
| V put(K key, V value)                      | // 在此映射中关联指定值与指定键。                   |
| V remove(Object key)                       | // 从此映射中移除指定键的映射关系(如果存在)             |
| <b>int</b> size()                          | // 返回此映射中的键-值映射关系数。                  |
| Collection<V> values()                     | // 返回此映射所包含的值的 <b>Collection</b> 视图。 |

\* 泛型E可以使用用户自定义类或标准类(Integer, String等)

- \* 采用Comparator对象控制TreeMap排序: [http://blog.sina.com.cn/s/blog\\_530fe9870100l5oy.html](http://blog.sina.com.cn/s/blog_530fe9870100l5oy.html)
- \* 用对象引用作为key可以用使用IdentityHashMap: [http://blog.csdn.net/wx\\_962464/article/details/7701141](http://blog.csdn.net/wx_962464/article/details/7701141)

# 数据库操作

## ● 关系数据库

数据库的类型有层次数据库、网状数据库、关系数据库、面向对象数据库（对象持久化）、时态数据库等等，关系数据库是目前最普遍使用的数据库。

关系数据库是面向关系的。以行(row)和列(column)来存储数据，行和列组成二维表（**table**），很多二维表又组成一个数据库。在数据库里，表列被称为字段(**field**)，表行被称为记录(**record**)。

为了减少冗余和避免引起数据不一致性，设计关系数据库需要符合基本的规范：

- (1) 要求每个表的一列只能包含一个数据，不能包含多个数据，例如，学生名册包含学号、姓名、班级四列，把学号和姓名放在同一列就不符合规范。
- (2) 每个表列都必须依赖整个主键，而不是主键的一部分。成绩册可以用课程号和学号做主键，然后记录成绩，如果包含课程名就不符合规范。一般的做法是每个表增加一个自动加1的id字段用作主键。
- (3) 非主键列不依赖其它非主键列。例如，成绩册采用学号做主键，如果逐渐中再包含姓名就不符合规范，因为通过学号可以得到姓名。



# ● 数据库语言

- 数据库语言包括**数据定义语言DDL(Data Definition Language)**和**数据操纵语言DML(Data Manipulation Language)**。
- 数据库系统中要定义很多对象，包括数据库(Database)、表(table)、域(field)、视图(view)、存储过程(stored procedure)、触发器(trigger)、索引(index)等。
- **数据定义语言**是SQL语言集中负责建立(create)、修改(alter)和删除(drop)数据库对象的语言。
- **数据操纵语言**实现对数据库的基本操作，例如，在表(table)中插入(insert)数据(row)、删除(delete)和修改(update)表中的数据(fields)、查询和显示表中的数据(fields)等。

[mysql参考](#)

# ● 建立数据库(数据库定义语言)

进入mysql

```
C:>mysql -u user -p
```

输入用户user的密码123

创建数据库

```
mysql>CREATE DATABASE teaching;
```

删除数据库: *DROP DATABASE teaching;*

或者

```
mysql>CREATE DATABASE IF NOT EXISTS teaching DEFAULT CHARSET utf8 COLLATE
utf8_general_ci;
```

COLLATE-排序

创建表

```
mysql>USE teaching;
```

```
mysql>CREATE TABLE stu(
```

```
 id int not null auto_increment primary key,
```

```
 num varchar(20),
```

```
 name varchar(20)
```

```
);
```

学号

姓名

删除表: *DROP TABLE stu;*

## 创建字段

```
mysql>ALTER TABLE stu ADD age int not null default 0;
```

## 创建唯一索引（学号唯一）

*删除列: ALTER TABLE stu DROP age;*

```
mysql>ALTER TABLE stu ADD UNIQUE INDEX(num);
```

## 显示信息

```
mysql>SHOW DATABASES;
```

显示所有数据库

```
mysql>USE teaching;
```

```
mysql>SHOW TABLES;
```

显示数据库teaching的所有表

```
mysql>SHOW COLUMNS FROM stu;
```

显示表stu的所有列的信息

- 数据库和表的创建和修改、权限的设置现在一般都是采用一个具有图形界面的数据库管理软件来做，例如，MySQL的WorkBench或者Navicat。

## ● 插入语句(数据库操纵语言)

```
INSERT INTO tableName(fieldName1, fieldName2,...)
VALUES(value1,value2,...);
```

更多格式见附录

例: `INSERT IGNORE INTO stu(`num`,`name`)  
VALUES("140001", "Zhang");` !IGNORE忽略出错, 插入无效

例: `INSERT INTO stu(`num`,`name`)  
VALUES("140001", "Zhang"), ("140002", "Wang");` !同时插入多个纪录

例: `INSERT INTO stu(num,name) (SELECT num,name FROM stuNew);` !插入SELECT结果, 要求  
! 字段个数和类型相同

例: `INSERT INTO table (a,b,c) VALUES (1,2,3) ON DUPLICATE  
KEY UPDATE c=VALUES(c);` ! 重复时修改字段c为值c

```
INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6) ON
DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

- 修改语句

```
UPDATE tableName
 SET fieldName1=value1, fieldName2=value2,...
 WHERE condition;
```

例: UPDATE stu  
 SET name = 'Wang'  
 WHERE id=5;

- 删除语句

```
DELETE FROM stu
 WHERE num="140005";
```

- 查询语句

```
SELECT `fieldname1`, `fieldname2`, ...
 FROM tablename
 WHERE condition
```

例: SELECT num,name FROM stu  
 WHERE num="140003";

```
SELECT * FROM stu
WHERE num="140003";
```

```
SELECT * FROM stu;
```

下列语句可以取到本次会话(或者本次连接)最近插入记录自动生成的id:

```
SELECT LAST_INSERT_ID(); // 字段名为LAST_INSERT_ID
```

创建课程表course和成绩表transcript

```
mysql>CREATE TABLE course(
 id int not null auto_increment primary key, 课程id
 num varchar(20), 课程号
 name int 课程名
);
```

```
mysql>CREATE TABLE transcript(
 id int not null auto_increment primary key, 课程表的id
 courseId int, 学生表的id
 stuId int, 分数
 score int
);
```

例：查出同学Wang课程所有课程成绩。

```
SELECT stu.num, stu.name, course.num, course.name, transcript.score
FROM stu, course, transcript
WHERE stu.num="140002"
 AND stu.id= transcript.stuid
 AND course.id= transcript.courseld;
```

字段别名



```
SELECT stu.num, stu.name, cou.num AS courseNum, cou.name AS courseName, tr.score
FROM stu, transcript tr, course cou
WHERE stu.num="140002"
 AND stu.id= tr.stuid
 AND cou.id= tr.courseld;
```

表别名

可以为表和字段取别名

假设总共有1000个学生和30门课，每个学生都选了所有的课并且都有成绩（ $1000 \times 30 = 30000$ 个成绩），这个查询的实现可以这样做：

方法1、先组合所有表的记录再筛选，组合记录数为 $1000 \times 30 \times 30000 = 900000000$ 。组合加筛选要遍历的记录数约为1800000000。

方法2、边筛选边组合表记录。这种方法需要遍历记录的(最大)次数为 $1000(\text{stu}) + \text{筛出的学生记录数} \times 30000(\text{transcript}) + \text{筛出的stu和transcript组合记录数} \times 30(\text{course})$ 。

\* 这里都是用遍历的方法查找，如果采用索引或其他方法，可以更快找出所有满足条件的记录并入。

例：查出选了编号为“50001”的课程所有同学的成绩。

```
WHERE cou.num="50001"
 AND cou.id= tr.courseld
 AND stu.id = tr.stuld;
```

查出所有选了名字包含“Web”的课程的同学的成绩。

```
WHERE cou.name like "%Web%"
```

...

查出所有选了名字包含“Web”的课程的同学的成绩中第100个开始的20个。

```
WHERE ...
LIMIT 100,20;
```

查出所有同学是否选择了名字包含为“Web”的课程的成绩，由于有些同学一门都没选，所以，要求有则显示成绩，没有则成绩均为空，并且按照学号升序排序。

```
SELECT stu.num, stu.name, tr.score
 FROM stu LEFT JOIN transcript tr
 ON stu.id= tr.stuld
 AND tr.courseld IN
 (SELECT id from course Where name LIKE "%Web%")
ORDER BY stu.num;
```

*SELECT出来的单列结果可以  
看成一个集合*

升序降序：ASC|DESC



查出所有同学的平均成绩：

```
SELECT stu.num, AVG(tr.score) as score
FROM stu, transcript tr
WHERE stu.id= tr.stuld
GROUP BY stu.num;
```

*GROUP BY*可以根据多个字段进行分组（字段值相同的记录一组），然后每个组单独进行统计。

另一种方法：

```
SELECT stu.num, stu.name, sco.score
FROM stu,
(SELECT stuld, AVG(score) as score
FROM transcript
GROUP BY stuld) sco
WHERE stu.id = sco.stuld;
```

*SELECT*出来的结果当成一个表

GROUP BY的另一个例子：

```
SELECT dept, MAX(salary) AS max1, COUNT(*) AS cnt FROM staff
GROUP BY dept
HAVING AVG(salary) >3000
ORDER BY dept
```

函数：SUM(), COUNT(), MAX(), MIN(), AVG()

## ➤ 查询语句格式:

|                                                                 |                |
|-----------------------------------------------------------------|----------------|
| <b>SELECT</b> [tablename1 .] `fieldname1` <b>AS</b> aliasname1, | ! 字段列表         |
| [tablename2 .] `fieldname2` <b>AS</b> aliasname2,               |                |
| ...                                                             |                |
| <b>FROM</b> tablename1 taliasname1,                             | ! 要查询的表(可以有多个) |
| tablename2 taliasname2,                                         |                |
| ...                                                             |                |
| <b>WHERE</b> condition                                          | ! 查询条件         |
| <b>ORDER BY</b> fieldnamei [asc/desc], ...                      | ! 排序字段         |
| <b>GROUP BY</b> fieldnamej,...                                  | ! 分组查询统计       |
| <b>LIMIT</b> [offset,] rows                                     | ! 选择一页         |

- 如果字段名中包含汉字或者空格，必须加上反引号。
- **SELECT** \* **FROM** tablename 可以获取表的所有内容，\* 表示所有字段。
- 在字段名前面加上表名或者表的别名可以用来区分同名字段。
- 选择一页表示从查询结果中选择第offset个记录开始的rows个记录。只有一项时，表示选取从第1页开始的rows个记录。
- 与表(table)在查询语句里是一样的，可以使用表(table)的地方也可以使用视图(view)。

## WHERE 表达式中的运算符:

= != > < >= <=

等于, 不等于, ...

[NOT ] BETWEEN ... AND ...

WHERE age BETWEEN 20 AND 30

[NOT] IN (项1,项2,...)

WHERE city IN('beijing','shanghai')

WHERE id IN(SELECT id FROM ItemA)

[NOT] LIKE

WHERE username LIKE '%user'

% 0或多个任意字符, \_ 任意单个字符, 正则表达式

IS NULL

空值判断符

IS NOT NULL

非空判断符

NOT、AND、OR

否、与、或

NOT EXISTS

WHERE NOT EXISTS (SELECT \* WHERE id="1234")

## 组合查询语句

! UNION可以把多个表的查询结果合并到一起

```
SELECT type,name FROM ItemA
UNION SELECT stype,sname FROM ItemB
```

! 把第一个SELECT查询结果作为一个表:

```
SELECT A.name,B.sname
FROM (SELECT * FROM ItemA WHERE name LIKE "%A") A, ItemB B
WHERE A.id = B.id
```

| ItemA |      |      |
|-------|------|------|
| id    | type | name |
| 1     | X    | AAA  |
| 2     | Y    | BBB  |
| 3     | Z    | CCC  |
| 4     | X    | DAA  |

| ItemB |       |       |
|-------|-------|-------|
| id    | stype | sname |
| 1     | X     | 111   |
| 2     | U     | 222   |
| 3     | X     | 333   |
| 4     | Z     | 444   |

| UNION |      |
|-------|------|
| type  | name |
| X     | AAA  |
| Y     | BBB  |
| Z     | CCC  |
| X     | DAA  |
| X     | 111  |
| U     | 222  |
| X     | 333  |
| Z     | 444  |

| FROM (SELECT ...) |       |
|-------------------|-------|
| name              | sname |
| AAA               | 111   |
| DAA               | 444   |

表ItemA 和ItemB在test数据库中定义:

\* id字段为自动增长的int类型, 其它字段都为varchar类型, 长度为24。id是key。

! JOIN语句: 只要分别来自两个表的记录满足条件, 就组成一行:

LEFT JOIN            左边表所有行都要加入, 即使右边表没有匹配的。

RIGHT JOIN          右边表所有行都要加入, 即使左边表没有匹配的。

INNER JOIN          两边都要存在

LEFT OUTER JOIN    同LEFT JOIN。 RIGHT OUTER JOIN   同RIGHT JOIN

例: **SELECT** A.name,B.sname

**FROM** ItemA A **LEFT JOIN** ItemB B **ON** A.type = B.stype

**SELECT** A.name,B.sname

**FROM** ItemA A **RIGHT JOIN** ItemB B **ON** A.type = B.stype

**SELECT** A.name,B.sname

**FROM** ItemA A **INNER JOIN** ItemB B **ON** A.type = B.stype

ItemA

| id | type | name |
|----|------|------|
| 1  | X    | AAA  |
| 2  | Y    | BBB  |
| 3  | Z    | CCC  |
| 4  | X    | DAA  |

ItemB

| id | stype | sname |
|----|-------|-------|
| 1  | X     | 111   |
| 2  | U     | 222   |
| 3  | X     | 333   |
| 4  | Z     | 444   |

LEFT JOIN

| name | sname  |
|------|--------|
| AAA  | 111    |
| DAA  | 111    |
| AAA  | 333    |
| DAA  | 333    |
| CCC  | 444    |
| BBB  | (Null) |

RIGHT JOIN

| name   | sname |
|--------|-------|
| AAA    | 111   |
| AAA    | 333   |
| CCC    | 444   |
| DAA    | 111   |
| DAA    | 333   |
| (Null) | 222   |

INNER JOIN

| name | sname |
|------|-------|
| AAA  | 111   |
| DAA  | 111   |
| AAA  | 333   |
| DAA  | 333   |
| CCC  | 444   |

## ➤建立外键约束

如果要求表transcript中的stuId必须是表stu的某个记录的id值，则需要为表transcript建立外键fkStuId。

```
ALTER TABLE transcript ADD CONSTRAINT fkStuId FOREIGN KEY(stuId)
REFERENCES stu(id) on DELETE CASCADE;
```

其中，on DELETE CASCADE是一个选项，表示如果删除了父表stu的一个记录，该记录的id在子表transcript中所关联的记录将自动被删除。

on DELETE的其它选项：

Set Null（子表对应字段为null）

No Action（不让删父表）

Set Default（子表对应字段设置为默认值）

Restrict（不让删父表）。

on UPDATE 也有这5个选项。具体见“[参考](#)”

再另外建立外键fkCourseId把表transcript中的字段courseId与表course的字段id关联上。

```
ALTER TABLE transcript ADD CONSTRAINT fkCourseId FOREIGN KEY(courseId)
REFERENCES course(id) on DELETE CASCADE;
```

在保存transcript的记录前，要检查外键，如果条件不满足，则会出现保存失败。

**删除外键：** `ALTER TABLE transcript DROP CONSTRAINT fkCourseId;`

在创建表时创建主键和外键：

```
CREATE TABLE product(pid int unsigned auto_increment primary key, cid int
references stu(id), name varchar(16));
```

```
CREATE TABLE product(pid int unsigned auto_increment, cid int, name varchar(16),
primary key(pid), constraint fkCid foreign key(cid) references stu(id));
```

第二种方法的主键和外键可以使用多个字段（用逗号隔开）。

## • 权限设置语句

采用GRANT语句可以设置用户对数据库操作的权限。

例： 给予从本地登录mysql系统的用户zym对数据库teaching的表stu查询、插入、更新和删除数据的权利：

```
GRANT SELECT, INSERT, UPDATE, DELETE on teaching.stu to zym@'localhost'
```

允许从172.18.187.\*登录mysql系统的用户slave对所有数据库和所有表进行任何操作(%表示任意0或多个字符)：

```
GRANT ALL on *.* to slave@'172.18.187.%';
```

允许从任何地点登录mysql系统的用户slave访问任何数据库的任何表：

```
GRANT ALL on *.* to slave@'%';
```

[参考](#)



## • SCHEMA表

MySQL系统中的数据库和表的信息实际上保存系统表中，可以作为SCHEMA表取出。

例：从数据库teaching中找出所有名字包含cou的表。

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'teaching' AND TABLE_NAME like '%cou%'
```

从数据库teaching的表stu中找出所有名字包含n的字段信息，包括字段名、字段类型、是否可空、缺省值。

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'stu' AND TABLE_SCHEMA = 'teaching'
AND COLUMN_NAME LIKE '%n%'
```

[参考](#)

- Java的数据查询实现

```
// ShowStus.java
import java.sql.*;
public class ShowStuInfo {
 static private Connection conn;
 static int cnt = 0;
 public static void main(String args[]){
 if(!connect()){
 System.out.println("Connect Error!");
 return;
 }
 ResultSet rs = executeQuery("select * from stu;");
 showStuInfo(rs);
 }
 //建立连接
 private static boolean connect() { ...}
 //执行SQL查询语句，返回结果集
 static private ResultSet executeQuery(String sqlSentence){...}
 //显示查询结果
 private static void showStuInfo(ResultSet rs){...}
}
```

Mysql系统所在主机的主机名或IP地址

```

//建立连接
private static boolean connect() {
 String connectionString= "jdbc:mysql://localhost:3306/teaching"
 +"?autoReconnect=true&useUnicode=true"
 +"&characterEncoding=UTF-8&useSSL=false";

 try {
 Class.forName("com.mysql.jdbc.Driver");
 conn = DriverManager.getConnection(connectionString, "user", "123");
 return true;
 }
 catch (Exception e) {
 System.out.println(e.getMessage());
 }
 return false;
}

```

端口号

数据库名

用户名 密码

- \* 应用程序采用TCP连接访问mysql系统。connectString为连接字串。
- \* Class.forName用于查找连接包com.mysql.jdbc的类Driver，该包在mysql-connector-java-5.1.39-bin.jar中。
- \* user和123分别为登录数据库的用户名和密码。
- \* 把localhost改为172.18.187.230可以用教学服务器的数据库进行测试。

//执行SQL查询语句，返回结果集

```
static private ResultSet executeQuery(String sqlSentence) {
 Statement stat;
 ResultSet rs = null;

 try {
 stat = conn.createStatement(); //获取执行sql语句的对象
 rs = stat.executeQuery(sqlSentence); //执行sql查询，返回结果集
 } catch (Exception e) {
 System.out.println(e.getMessage());
 }
 return rs;
}
```

createStatement(int type, int concurrency);

参数 int type:

ResultSet.TYPE\_FORWARD\_ONLY  
ResultSet.TYPE\_SCROLL\_INSENSITIVE  
ResultSet.TYPE\_SCROLL\_SENSITIVE

结果集的游标只能向下滚动。（默认）  
结果集的游标可以上下移动，当数据库变化时，当前结果集不变。  
返回可滚动的结果集，当数据库变化时，当前结果集同步改变。

参数 int concurrency

ResultSet.CONCUR\_READ\_ONLY  
ResultSet.CONCUR\_UPDATETABLE

不能用结果集更新数据库中的表。（默认）  
能用结果集更新数据库中的表。

```
//显示查询结果
private static void showStuInfo(ResultSet rs){
 try {
 while(rs.next()){
 System.out.println(rs.getString("name"));
 }
 }
 catch (Exception e) {
 System.out.println(e.getMessage());
 }
}
```

- 游标(cursor)是指向当前行(row)的指针。
- rs.next()将游标移动到下一行。第一次执行该方法时游标会移动到第一行。如果游标已经处于最后一行记录时执行该方法会返回false。
- rs.getString("type")返回域type的值。

## ResultSet其它方法:

|                                               |                                                                                                                                            |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <code>public void rs.previous();</code>       | 将游标移动到前一行(row)                                                                                                                             |
| <code>public void rs.first();</code>          | 将游标移动到第一行                                                                                                                                  |
| <code>public void rs.next();</code>           | 将游标移动到下一行, 如果到了末尾返回FALSE                                                                                                                   |
| <code>public void rs.last();</code>           | 将游标移动到最后一行                                                                                                                                 |
| <code>public void rs.afterLast();</code>      | 将游标移动到最后一行之后                                                                                                                               |
| <code>public void rs.beforeFirst();</code>    | 将游标移动到第一行之前                                                                                                                                |
| <code>public boolean isAfterLast();</code>    | 判断游标是否在最后一行之后。                                                                                                                             |
| <code>public boolean isBeforeFirst();</code>  | 判断游标是否在第一行之前。                                                                                                                              |
| <code>public boolean isFirst();</code>        | 判断游标是否指向结果集的第一行。                                                                                                                           |
| <code>public boolean isLast();</code>         | 判断游标是否指向结果集的最后一行。                                                                                                                          |
|                                               |                                                                                                                                            |
| <code>rs.getRow();</code>                     | 得到当前行的行号(从1开始), 如果结果集没有行, 返回0。                                                                                                             |
| <code>rs.getInt(fieldName);</code>            | 取出当前行指定字段的整数值。                                                                                                                             |
| <code>rs.getDate(fieldName);</code>           | 取出当前行指定字段的日期值。                                                                                                                             |
| <code>rs.getBoolean(fieldName);</code>        | 取出当前行指定字段的布尔值。                                                                                                                             |
| <code>rs.getFloat(fieldName);</code>          | 取出当前行指定字段的浮点值。                                                                                                                             |
| <code>public boolean absolute(int row)</code> | 将游标移到参数row指定的行号。如果row取负值, 就是倒数的行数, <code>absolute(-1)</code> 表示移到最后一行, <code>absolute(-2)</code> 表示移到倒数第2行。当移动到第一行的前面或最后一行的后面时, 该方法返回false |

## • Java的数据修改实现

```
// UpdateStus.java
import java.sql.*;
public class UpdateStuInfo {
 static private Connection conn;
 static int cnt = -1;
 public static void main(String args[]){
 if(!connect()){
 System.out.println("Connect Error!");
 return;
 }
 updateStuInfo("update stu set name ='Li' where id=3;");
 }
 //建立连接
 private static boolean connect() {...} // 与数据查询的connect相同

 //执行SQL修改语句，返回结果集
 private static boolean executeUpdate(String sqlSentence) {...}

 //进行修改
 private static void updateStuInfo(String sqlSentence){...}
}
```

//执行SQL修改语句,失败返回false,成功则返回true并把影响的记录数保存在cnt中。

```
private static boolean executeUpdate(String sqlSentence) {
 Statement stat;

 cnt=-1;
 try {
 stat = conn.createStatement(); // 根据连接获取一个执行sql语句的对象
 cnt = stat.executeUpdate(sqlSentence); //执行sql语句,返回所影响行记录的个数
 }
 catch (Exception e) {
 System.out.println(e.getMessage());
 }
 return (cnt>=0);
}
```

//进行修改

```
private static void updateStuInfo(String sqlSentence){
 if(executeUpdate(sqlSentence)){
 System.out.println(""+cnt + " records are updated.");
 }
}
```

//执行stat.executeUpdate()可以进行插入、修改和删除行的操作。

- 使用execute见 [Java 使用execute方法执行Sql语句](#)
- 一次执行多条语句: <http://www.cnblogs.com/kxdblog/p/4115326.html>
- 返回列名: <http://outofmemory.cn/code-snippet/4528/java-get-jdbc-resultset-columns-name>



## • 调用存储过程的方法

存储过程sumtwo(int, int,int )把前两个参数相加，第三个参数为返回值(传址参数)。调用该存储过程的方法如下：

```
//
```

```
CallableStatement cs
```

```
=conn.prepareCall("{call sumtwo(?,?,?)}"); //指出存储过程名和参数
```

```
cs.setInt(1,50);
```

```
// 将第一个参数的值设置成50
```

```
cs.setInt(1,150);
```

```
// 将第二个参数的值设置成150
```

```
cs.execute();
```

```
// 执行存储过程
```

```
System.out.println (cs.getInt(3));
```

```
// 显示返回结果
```

# 附录1、捕捉错误

```
public class CatchError {
 public static void main(String args[]){
 int x;
 try { // 打开例外处理语句
 for (int i = 5; i >= -2; i--) {
 x = 12 / i; // 出现例外(x==0)后将不执行后面的语句
 System.out.println("x=" + x); // 直接跳到catch内执行
 }
 }
 catch (Exception e) { // 捕捉例外信息。可以并列用多个catch
 System.out.println("Error:" + e.getMessage()); // 显示当前错误信息
 // e.printStackTrace(); // 显示系统错误信息
 }
 finally{ // 出现例外必须执行这里的语句
 x=0;
 }
 System.out.println(x);
 }
}
```

执行结果:

x=2  
x=3  
x=4  
x=6  
x=12  
Error:/ by zero  
0

如果一个方法不愿意处理一些例外，可以采用throws定义方法的例外，把这些例外交给调用者去处理。

```
public class DivideClass {
 void divide() throws Exception { // 出错后交给调用程序处理
 for (int i = 5; i >= -2; i--) {
 int x = 12 / i; // 出现例外(x==0)后将不执行后面的语句
 System.out.println("x=" + x);
 }
 }
}

public class ThrowError {
 public static void main(String args[]){ // main为主程序入口
 try { // 打开例外处理语句
 DivideClass div = new DivideClass();
 div.divide();
 }
 catch (Exception e) { // 捕捉例外信息。可以并列用多个catch
 System.out.println("Error:"+e.getMessage()); // 显示当前错误信息
 }
 }
}
```

执行结果：  
x=2  
x=3  
x=4  
x=6  
x=12  
Error:/ by zero

# 附录2、数值与字符串之间转换

```
// DataConversion.java
```

```
int i = 123456;
println(""+i);
println(Integer.toString(i));
println(Integer.toBinaryString(i));
println(Integer.toHexString(i));
println(Integer.toString(i, 16));
println(Integer.MAX_VALUE);
int j=Integer.reverse(i);
println(j);
println(String.format("%08d", i));
String s1="567890";
i = Integer.parseInt(s1);
i = Integer.parseInt(s1,16);
i = Integer.valueOf(s1);
i = Integer.valueOf(s1,16);
float f = 1234.0f;
println(""+f);
println(Float.toString(f));
println(String.format("%010.3f",f));
f = Float.valueOf("4567.789");
```

```
// 把i转化为字符串
// 把i转化为字符串
// 转化为二进制字符串:11110001001000000
// 转化为十六进制字符串: 1e240
// 转化为十六进制: 1e240
// 最大整数: 2147483647
// i的二进制(32位)的逆转:38240256
// 逆转: 1001000111100000000000000000
// 按格式转换: 00123456(不够8位前面添0)
```

```
//把字符串s1转换为整数
// 转换为16进制整数
//把字符串s1转换为整数
//转换为16进制整数
```

```
//把浮点数s1转换为字符串
//把浮点数s1转换为字符串
//按格式转换: 001234.568(共10位)
//把字符串转换为浮点数
```

# 附录3、日期和字符串之间的转换

```
// DataConversion.java
import java.util.*; //格式化时间: Date Calendar
import java.text.*; //格式化时间: DateFormat SimpleDateFormat
// 把时间转换为字符串
Date now = new Date();
System.out.println(""+now); //显示: Thu Aug 21 11:36:59 CST 2014
DateFormat sdf1 // 默认为本地语言 (省略第二个参数)
 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss aa E",Locale.ENGLISH); //M hh
System.out.println(sdf1.format(now)); //显示: 2014-08-21 11:42:37 AM Thu

// 把字符串转化为时间
Date date1 = sdf1.parse("2008-07-10 19:20:00 PM FRI");
System.out.println(sdf1.format(date1));

//取出年月日: Calendar.YEAR .MONTH .DAY_OF_MONTH .HOUR .MINUTE
// .SECOND .MILLISECOND .DATE .AM_PM .DAY_OF_YEAR
// .DAY_OF_WEEK .HOUR_OF_DAY .ZONE_OFFSET
Calendar cal=Calendar.getInstance();
cal.setTime(new Date()); //cal设置为当前时间
System.out.println("cal.DAY_OF_MONTH: "+cal.get(Calendar.DAY_OF_MONTH));
System.out.println("cal.DAY_OF_WEEK: "+cal.get(Calendar.DAY_OF_WEEK));
System.out.println("cal.HOUR: "+cal.get(Calendar.HOUR));
System.out.println("cal.HOUR_OF_DAY: "+cal.get(Calendar.HOUR_OF_DAY));
```

```
// 增加日期
Calendar cal1=Calendar.getInstance();
cal1.setTime(now); //cal1 设置为当前时间. cal.getTime().
cal1.add(Calendar.DAY_OF_YEAR, 20); //增加20天。-20就是减20天
System.out.println("cal.add: "+cal1.get(Calendar.DAY_OF_MONTH));

// 比较日期
boolean b1=cal.before(cal1);
System.out.println("cal.before: "+b1);

// 求两个日期之间的天数
DateFormat sdf2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"); //M hh
Calendar cal2 = Calendar.getInstance();
cal2.setTime(sdf2.parse("2012-09-08 10:10:10"));
Calendar cal3 = Calendar.getInstance();
cal3.setTime(sdf2.parse("2012-09-15 00:00:00"));
System.out.println(""+(cal3.getTimeInMillis()-cal2.getTimeInMillis())/(1000*3600*24));
```

Long time= System.currentTimeMillis();

或

Long time= new Date().getTime();

两种方法都取到距离新纪元时间(epoch time)1970年1月1日0点0分0秒的毫秒数。

# 附录4、Math

|                              |              |                                  |                           |
|------------------------------|--------------|----------------------------------|---------------------------|
| <code>Math.E</code>          | 常量 2.71828   | <code>Math.pow(n1,n2)</code>     | $n1$ 的 $n2$ 次幂            |
| <code>Math.PI</code>         | 常量3.14159    | <code>Math.random()</code>       | 随机数: $\geq 0.0$ 且 $< 1.0$ |
| <code>Math.abs(n)</code>     |              | <code>Math.round(n)</code>       | $n$ 的四舍五入                 |
| <code>Math.acos(n)</code>    | 反余弦          | <code>Math.sin(n)</code>         | 三角正弦                      |
| <code>Math.asin(n)</code>    |              | <code>Math.sinh(n)</code>        | 双曲线正弦                     |
| <code>Math.atan(n)</code>    |              | <code>Math.sqrt(n)</code>        | 正平方根                      |
| <code>Math.cbrt(n)</code>    | 立方根          | <code>Math.tan(n)</code>         | 三角正切                      |
| <code>Math.ceil(n)</code>    | 往上到达的第一个整数   | <code>Math.tanh(n)</code>        | 双曲线余弦                     |
| <code>Math.cos(n)</code>     |              | <code>Math.toDegrees(rad)</code> | 把弧度转换为角度                  |
| <code>Math.cosh(n)</code>    | 双曲线余弦        | <code>Math.toRadians(deg)</code> | 把角度转换为弧度                  |
| <code>Math.exp(n)</code>     | $e$ 的 $n$ 次幂 |                                  |                           |
| <code>Math.floor(n)</code>   | 往下到达的第一个整数   |                                  |                           |
| <code>Math.log(n)</code>     | 自然对数 底数是 $e$ | <code>MathFunc.java</code>       |                           |
| <code>Math.log10(n)</code>   | 底数为 10 的对数   |                                  |                           |
| <code>Math.max(n1,n2)</code> | 取较大的一个       |                                  |                           |
| <code>Math.min(n1,n2)</code> | 取较小的一个       |                                  |                           |

`Integer`和`Math`都是类，为什么可以不通过对象定义而直接使用其方法？

# 附录5、控制台输入输出

```
//ConsoleIO.java
import java.util.*;
Scanner in = new Scanner(System.in);
System.out.println("What is your name?(line)");
String name = in.nextLine(); // 输入一行
System.out.println("How old are you?(int)");
int age = in.nextInt(); // 输入一个整数
System.out.println("How much do you weigh?(float)");
float weight = in.nextFloat(); // 输入一个浮点数
System.out.println("name:" + name + " age:" + age + " weight:" + weight);
System.out.println("Input three words:\r\n");
int cnt = 0;
while (in.hasNext() && cnt < 3) { // 是否还有单词
 String word = in.next(); // 读取下一个单词
 System.out.println("" + cnt + ": " + word);
 cnt++;
}
in = new Scanner("11.0 22.0 33.0 44.0 55.0"); // 直接输入字符串
while (in.hasNextDouble()) { // 是否还有双精度数
 double x = in.nextDouble(); // 读出下一个双精度数
 System.out.println(x);
}
```



# 附录6、泛型

泛型(Generic Type)为参数化的类型，也就是在定义时先不给出类型，在使用时才给出类型。泛型可以用于类、接口、方法和类型限定四种情况。

```
import java.util.*;
// 泛型接口
interface Show<T,U>{
 void show(T t,U u);
}

class GenericIntTest implements Show<String,Date>{
 @Override
 public void show(String str,Date date) {
 System.out.println(str);
 System.out.println(date);
 }
 // 泛型方法
 public static <T, U> T get(T t, U u) {
 if (u!=null)
 return t;
 else
 return null;
 }
}
```

```

// 限定T为接口Comparable的子类（这里的extends也用于接口）；
// 可以用&并列两个接口，例如： <T extends Comparable&Serialize>
public static <T extends Comparable> T get2(T t1,T t2) { //添加类型限定
 if(t1.compareTo(t2)>=0);
 return t1;
}
public static void main(String[] args) throws ClassNotFoundException {
 GenericIntTest showTest=new GenericIntTest();
 showTest.show("Hello",new Date());
 String str=get("Hello", "World");
 System.out.println(str);
 int i=get(100,200);
 System.out.println(i);
}
}

```

执行结果：

```

C:\java\GenericType>java GenericIntTest
Hello
Sat Apr 01 00:32:24 CST 2017
Hello
100

```

用?可以表示未知类型，一般用于带入参数，也用于限定类型，例如：“? **super** Apple”和“? **extends** Apple”限定只能使用祖先类或子类（包括本类）。

```
class Food{}
class Fruit extends Food{}
class Meat extends Food{}
class Banana extends Fruit{}
class Pork extends Meat{}
class Beef extends Meat{}
class Apple extends Fruit{}
class GaliApple extends Apple{}
class SnakeApple extends Apple{}
class Plate<T>{ // T为泛型，只要是个标识符就行，例如，用SSS也可以。
 private T item;
 public Plate(T t){item=t;}
 public void set(T t){item=t;}
 public T get(){return item;}
}
```

```

class GenericTypeTest{
 public static void func1(Plate<? extends Fruit> p){ }
 public static void main(String[] args){
 Plate<?> p=new Plate<Apple>(new Apple());

 // Plate<Fruit> q1=new Plate<Apple>(new Apple()); /*error*/

 // <? super Apple>使用Fruit及其子孙类作为泛型（上界）
 Plate<? extends Fruit> r2=new Plate<Fruit>(new Fruit());
 Plate<? extends Fruit> r3=new Plate<Apple>(new Apple());
 Plate<? extends Fruit> r4=new Plate<GaliApple>(new GaliApple());
 // Plate<? extends Fruit> s1=new Plate<Food>(new Food()); /*error*/
 // Plate<? extends Fruit> s2=new Plate<Pork>(new Pork()); /*error*/

 // <? super Apple>使用Apple及其祖先类作为泛型（下界）
 Plate<? super Apple> v1=new Plate<Apple>(new Apple());
 Plate<? super Apple> v2=new Plate<Fruit>(new Fruit());
 Plate<? super Apple> v3=new Plate<Food>(new Food());
 // Plate<? super Apple> v3=new Plate<GaliApple>(new GaliApple()); /*error*/
 //r2.set(new Apple()); /*error*/
 Fruit x1=r2.get();
 Apple x2=(Apple)r3.get();
 v1.set(new Apple());
 Object z=v1.get();
 // Apple y=v1.get(); /*error*/

 func1(r2);
 }
}

```

# 附录7、与C++的部分不同点总结

- Java语言是C++的简化版，与C++十分相似。下面把他们的不同点列出来。
- Java的变量定义与C++相同，常量定义不同：  
`final int LEVEL_NUM=0x1000;` //为常量命名是个好习惯
- Java的算术表达式、关系表达式、逻辑表达式和位表达式与C++相同，只是用逻辑运算符形成的表达式，计算结果为真假值。在Java的逻辑表达式中，非0不会自动当成true。
- Java的类型转换可以采用强制类型转换：  
`double x = 10.876;`  
`int y = (int)x;`
- 不同种类之间的类型转换要用对应的方法：  
整数转为字符串：`Integer.toString(i)` 或 `""+I`  
字符串转整数：`int i = Integer.parseInt("567890");`
- `/** ..... */` 文档注释，可以用javadoc提取其内容。
- Java只有一维数组，二维数组为数组的数组，因此Java数组的每行可以具有不同的列数：  
`int table[][] = {{1},{2,3,4},{5,6,7,8}};` // 二维数组

- Java的赋值语句、条件语句、 循环语句与C++相同。增加了一个foreach语句：

```
double sum=0;
cnt = 0;
double scores[]={100.0, 90.2, 80.0, 78.0,93.5};
for(double score:scores){
 sum=sum+score;
 cnt++;
}
```

- Java的局部变量的作用域都是所在方法的整个范围。

//块定义域

```
void f(){
 int k=0;
 ...
 if(){
 int n =5;
 int k =5; //错误！！
 }
}
```

# 附录8、参考资料

- Bruce Eckel, Java编程思想（第4版），机械工业出版社，2012
- Cay S.Horstmann, G.Cornell, Java核心技术，机械工业出版社，2012
- Y. Daniel Liang , Java语言程序设计（第8版），机械工业出版社，2011
- <http://docs.oracle.com/javase/8/docs/>
- <http://api.apkbus.com/reference/java/io/package-summary.html>
- <http://docs.oracle.com/javase/tutorial/>
- <http://blog.csdn.net/dfdsggdgg/article/details/51290764>(内存分配)
- <http://www.jb51.net/article/105077.htm> (Java加载一个类的过程)
- <https://my.oschina.net/stephenzhang/blog/380106?p={{currentPage+1}}>  
(Java运行时数据区域)
- [http://blog.sina.com.cn/s/blog\\_618298140102vv7s.html](http://blog.sina.com.cn/s/blog_618298140102vv7s.html) (JAVA中的域，静态域，实例域)
- <http://blog.csdn.net/yang3wei/article/details/7381578> (编码问题)
- <http://blog.csdn.net/likika2012/article/details/17055021> (编码问题)