Android移动开发

Android Mobile Application Development

第2讲+ Java入门

吴以凡
计算机学院 一教505
yfwu@hdu.edu.cn

语言要素

语言要素

- 语言要素包括注释、标识符、分隔符以及关键字等四个部分：
  - 注释用于提高程序可读性
  - 标识符是指常量、 变量、函数、类和对象的名称，不同的语言有不同的标识符命名规则
  - 分隔符用于区分程序中的基本元素，可分为
    - 注释
    - 空白符
    - 普通分隔符
  - 关键字也被称为保留字，它是程序设计语言预先定义的、有特殊意义的标识符

注释

- 程序设计者与程序阅读者（包括程序设计者自己）之间通信的重要手段
- 注释可以改善源程序代码的可读性，使得程序条理清晰。
- 良好的注释风格和习惯是“优质”程序或者程序员必须具备的要素。
- 注释分为三种类型：
  - 单行(single-line)注释
  - 块(block)注释
  - 文档注释

注释

- 单行(single-line)注释

```
// 注释内容
/* 注释内容 */
```

- 块(block)注释
  - 一般位于一个文件或一个方法前

```
/*
 * 注释内容1
 * 注释内容2
 */
```

Java文档注释（Javadoc）

- Java文档注释（Javadoc）
  - API信息：参数个数、意义、类型等
  - Javadoc从源代码中抽取类、方法等注释
  - 输出HTML
- 使用方法
  - 以/**为开始符而以*/为结束符
  - 注释文档由描述块、块标记组成
  - 块标记以@开头、后面紧跟Javadoc标签

```
/** 该方法用于打印字符串
 * @author Ellen
 * @version 1.2
 * @param String 要打印的字符串
 * @return 没有返回值
 */
```

* http://java.sun.com/Javadoc/

标识符

- 变量、类、对象和方法等元素的名字
- 标识符应当在某种程度上反映所命名元素（变量、类、对象和方法）的实际意义
- 合适的标识符加上良好的注释风格是提高程序可读性的必备要素
- 大小写敏感
- 不能使用关键字

标识符命名

- 变量：首字母小写且使用名词，其后用大写字母分隔每个单词
  - 例：myAgeFromDB， myBirthdayFromDB
- 方法：使用动词且首字母小写，其后用大写字母分隔每个单词。
  - 例：getAgeFromDB、getBirthdayFromDB
- 常量：一般全部大写，单词之间用下划线分隔。
  - 例：DEFAULT_AGE
- 类和接口：使用名词，且每个单词首字母要大写
  - 例：Person、Car

分隔符

- 在语句、变量，类和成员、对象和成员和程序之间起着分割作用的符号。有5种分割符：
  - 圆点(“.”)：分割类和成员以及对象和成员
    - 类名.静态成员名
    - 对象名.成员名
  - 分号（“;”）：语句结束的标记或者for循环中分隔不同的成分
  - 逗号（“,”）：分割多个变量、形参以及实参
  - 空格(“ ”)：用于分隔源代码中不同的部分
  - 花括号（“{”及“}”）：用于限定某一部分的范围，一定成对使用

关键字

- 一种具有特殊意义的标识符，也被称为保留字
- 在语言里预先定义的，不能作为变量名、类名、对象名以及方法名
- 被用来做访问控制、修饰符、逻辑控制、错误处理、包处理等。例如：
  - 条件语句的if else
  - 数据类型 int float
  - 循环 for while

数据类型

基本数据类型

- 整数：byte, short, int, long
- 字符：char
  - Unicode
- 浮点：float, double
- 布尔值：bool
  - true, false

引用数据类型

- 类：class
- 接口：interface
- 数组：[], ArrayList, Vector

运算符与表达式

运算符

- 运算符分为运算符、算术运算符、关系运算符、位运算符、逻辑运算符、条件运算符及条件运算符等。
- 运算符的四个要素：
  - 操作数目
  - 优先级
  - 结合性
  - 操作类型

运算符

- 赋值运算符的符号是 “=”
  - 赋值运算是将一个表达式的值赋给一个左值
  - 赋值时必须要求左值和右值的类型一致，如果类型不匹配时需要能自动转换为对应的类型
- 算术运算符分为一元运算符和二元运算符两种
  - 一元运算符：正（“+ “）、负（”- “）、自增（”++ “）和自减（” -- “）
  - 二元运算符：加（+）、减（—）、乘（*）、除（ / ）、取余（%）
  - 精准度：不同类型操作数会按精度最高类型自动转换
- 关系运算符包括大于（“>”）、大于等于（“>=”）、小于（“<”）、小于等于（“<=”）、等于（“==”）和不等于（“!=”）
  - ==、!=对基本类型数据是比较值，对引用类型数据是比较句柄

运算符

- 位运算符是对二进制数据操作的运算符
  - 与（“&”）、或（“|”）、非（“~”）、异或（“^”）
- 逻辑运算符是逻辑量之间的运算
  - 非（“!”）、与（“&&”）以及或（“||”）
- 其他运算符
  - 移位运算符: 左移运算符（“<<”）和右移运算符（“>>”）
  - 三目运算符：<表达式1> ? <表达式2> : <表达式3>

表达式和语句

- 语句（statement）是标识符的集合，由常量、关键字、变量和表达式构成
- 表达式（expression）由常量、变量、运算符组成

控制语句

控制语句

- 控制语句用于控制程序的流程， 以实现程序的各种结构方式。
- 控制语句分为选择控制语句、循环控制语句和转移控制语句三种
  - 选择控制语句：包括if语句和switch语句。
  - 循环控制语句：包括for循环语句、while循环语句和do…while循环语句。
  - 转移控制语句：包括break语句、continue语句和return语句。

选择控制语句

- if语句

```
if (expression)
    statement1;
else
    statement2;
```

- swtich语句

```
swtich (expression) {
    case label1:
        statement1;
        break;
    case label2:
        statement2;
        break;
    ...
    default:
        statement_n;
}
```

循环控制语句

- for循环语句

```
for (expression1; expression2; expression3)
   循环体
```

- while循环语句

```
while (expression)
   循环体
```

```
do {
   循环体
} (expression)
```

- 转移控制语句
  - break
  - continue
  - return

数组

数组

- 数组是若干变量按照有序的形式组织起来的集合，并且数组中的变量具有相同的数据类型。
- 数组所包含的变量个数被称为数组长度，按照数组的长度是否可以动态变化，可将数组分为动态数组和静态数组两种类型。
  - 静态数组数组长度是固定的，不能动态变化
  - 动态数组数组长度是可以按照需要动态增加或者减少。

静态数组

- 静态数组是最常用的数组类型，这种数组不能按照需要来动态改变数组长度。有两种定义静态数组的语法格式：
  - 类型说明符 数组名 [ ]
  - 类型说明符 [ ]数组名
- 类型说明符是任一种基本数据类型或构造数据类型；而数组名是用户定义的数组标识符。例如：

```
float array1[11];
```

```
int[] myArray = new int[4];
myArray[0] = 1;
```

```
int aNums[] = { 2, 4, 6 };
```

动态数组

- ArrayList和Vector是比较常用的动态数组类。
- 程序开发人员可以通过ArrayList或者Vector对外开放的方法来动态改变数组的长度。例如：

```
ArrayList arrayList =new ArrayList();//定义动态数组 arrayList
arrayList.add("a");                  //向动态数组arrayList中添加数据
System.out.println(arrayList.size()); //输出数组长度
arrayList.add("b");                  //向动态数组arrayList中添加数据
System.out.println(arrayList.size()); //输出数组长度
String element = (String)arrayList.get(1);  //获取数组元素
arrayList.remove(2);                 //删除数组元素
```

字符串

- 字符串是程序语言中表示文本的数据类型，一般由若干个字符组成的有限序列
- 通常以字符串的整体作为操作对象
  - 在字符串中查找某个子串
  - 求取一个子串
  - 在串的某个位置上插入一个子串
  - 删除一个子串等

字符串定义

- 无论字符串常量或字符串变量，都要先创建对应的String类的实例对象才能使用。
- 有三种创建字符串实例对象的方式，下面使用这三种方式来创建字符串“Hello Android”。
  - 第一种方式：使用new创建字符串实例对象。例如：

```
String myString = new String("Hello Android");
```

  - 第二种方式：直接赋值来创建字符串实例对象。例如：

```
String myString = "Hello Android";
```

  - 第三种方式：可以串联（“+”）来创建字符串实例对象。例如：

```
String myString = "Hello " + "Android";
```

常用的字符串方法

- String类提供处理若干个字符串的方法，几种常用的String方法：
  - int length()：计算字符串的长度
  - char charAt(int location) ：获取字符串相应位置的字符
  - boolean equals(String str)：判断字符串是否相等，若相等返回true；否则返回false
  - boolean equalsIgnoreCase (String str)：该方法的功能与equals方法类似，用于判断字符串是否相等。但equalsIgnoreCase不对大小写敏感
  - String concat(String str)：将str追加到原字符串后面

类和对象

面向对象模型

- 现实世界可抽象成：
  - 描述客观实体特征的一组属性
  - 实现客观实体功能的一组方法
- Android是一种面向对象（Object Oriented）的模型。
  - 所有的操作都是以类和对象为中心
  - 程序设计人员能从现实世界的角度来分析、设计和实现一个应用程序

类

- 类是对现实世界的客观实体的抽象，描述了客观实体的共同的属性和方法。
- 类的三个特征
  - 封装性
  - 多态性
  - 继承性
- 声明一个类的格式如下所示：

```
[〈修饰符〉] class〈类名〉
{
类主体
}
```

- 修饰符
  - public：任何类都能访问
  - protected：只能被自身或子类以及同一个包下的其他类访问
  - private：只能被自身访问
  - 不加修饰符为default：只能被同一个包中的类访问

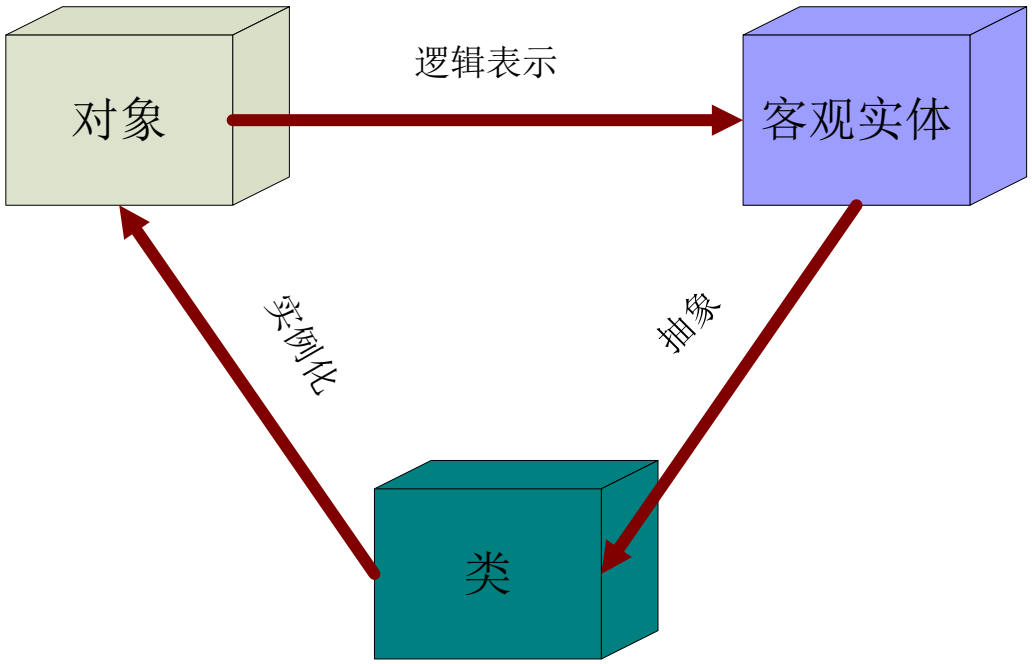
类

```
public class Computer
{
    int computerNO;
    int coputerUsage;

    int getComputerNO ()
    {
        return this.computerNO;
    }

    void setComputerNO (int computerNO)
    {
        this.computerNO = computerNO;
    }

    void setComputerUsage (int computerUsage)
    {
        this.computerUsage = computerUsage;
    }
}
```

* this关键字标识某个对象本身

对象

- 对象是对类的实例化，可以把类看成一个数据类型，对象则是该数据类型对应的变量。
- 客观实体、类以及对象之间的关系如下所示：



创建对象

- 创建类之后，就可创建该类的实例即对象
- 有两种创建对象的方式
  - 第一种方式：先声明对象，再实例化对象

```
Computer myComputer;// 声明对象
myComputer = new Computer();//使用new关键字实例化对象
```

  - 第二种方式：在声明对象的同时，实例化对象

```
Computer myComputer = new Computer();
```

构造函数

- 对象必须只能通过构造函数来创建，没有其他的创建方式。构造函数（或者构造方法）作用是在实例化对象时来初始化对象中的属性，
- 构造函数具有以下方面的特性：
  - 构造方法的方法名必须与类名一致。
  - 一个类可以包含多个构造方法
  - 如果在定义类时没有定义构造方法，则编译系统会自动在该类中创建一个无参数的构造方法，并且这个构造方法不执行任何代码。

```
public class Computer
{
   Computer(int comoputerNO)
   {
      this.computerNO = computerNO;
   }
}
```

构造函数

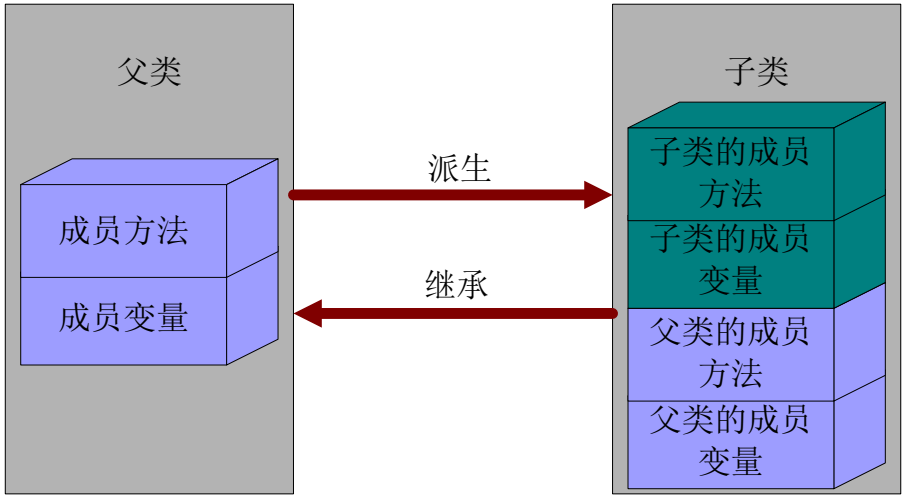
- 创建对象时调用构造函数

```
public class Person{
   Person(String  name, boolean gender, int age){...}
}
```

```
Person mike = new Person("Mike", 1, 19);
Person amanda = new Person("Amanda", 0, 18);
```

继承

- 继承是以已存在的类作为基础建立新类的机制，已有的类被称为父类，而新创建的类被称为子类。
- 继承机制使得复用以前的代码变得非常容易，因而能大大缩短开发周期，提高了程序的开发效率。



例：父类：Computer，子类：Laptop，Desktop

继承的实现

- 构父类与子类的继承关系是通过extends关键字来实现的，其语法格式为：

```
[访问权限] class 子类名 extends 父类名
{
类体定义;
}
```

  - 其中“访问权限”是指public，private，protected等
- 下面的语句创建了Computer的子类Laptop：

```
public class Laptop extends Computer {...}
```

成员变量的隐藏和方法的重写

- 子类可以定义与父类相同的成员变量和方法
- 成员变量的隐藏
  - 子类的成员变量隐藏了父类中同名的成员变量
- 成员方法的重写
  - 子类的成员方法的名字、返回类型、参数个数与父类继承的方法完全相同
  - 通过方法的重写，改变父类的行为

super关键字

- 可以通super关键字，使子类访问父类的成员。super关键字有三种用途：
  - 调用父类的构造方法

```
super(Args1 args,……Argsn args);
```

  - 调用父类的成员变量

```
super.成员变量名
```

  - 调用父类的成员方法

```
super.成员方法名([参数列表])
```

继承

```
public class Person{
    public String name;
    public void work() {
        // dump work
    }
}
```

```
public class Student extends Person{
    @Override
    public void work() {
        super.work();
        // finish homework
    }
}
```

多态

多态

- 同名的不同方法共存的情况
- 两种形式的多态机制：
  - 子类的方法与父类方法共存
  - 同一个类中同名但参数不同的方法共存，这种多态也成为重载
    - 多个同名函数，不同的参数个数/类型
    - 让类以统一的方式处理不同类型的数据

```
...
void setComputerUsage(int computerUsage)
{
   this.computerUsage = computerUsage;
}

void setComputerUsage(String computerUsage)
{
   this.computerUsage = computerUsage.toInteger(computerUsage);
}

void setComputerUsage(float computerUsage)
{
   this.computerUsage = (int)computerUsage;
}
...
```

其他

接口

- 用于组织对象的行为，定义多个不同对象通用的成员方法

```
public interface Living {
        void eat();
        void sleep();
        void work();
}
```

- 实现接口必须实现接口中定义的所有方法

```
public class Person implements Living{
        // Provide implementations
}
```

包（Package）

- 一组类和接口
- 使用命名空间（namespace）来为包命名
- 使用import关键字在自己代码中使用其他包

```
import com.ourclient.project.subproject.Person
```

包（Package）

- 一组类和接口
- 使用命名空间（namespace）来为包命名
- 使用import关键字在自己代码中使用其他包

```
import com.ourclient.project.subproject.Person
```

内部类（Inner Class）

- 目的：创建只用于外部类内部的对象
- 内部类是非静态的内嵌类
- 静态的内嵌类：
  - 该内嵌类所定义的行为不仅绑定某一个对象，而是与所有该类的对象绑定
  - 无法访问外部类的成员变量

```
public class User {
    class LoginInfo {}
    public static class ServerInfo {}
}
```

```
User.ServerInfo sInfo = new User.ServerInfo();
```

final关键字

- final变量和方法
  - 无法改变final变量的值
  - 无法重写final方法
  - 静态变量的值在编译时必须已知，而final变量不需要
  - final关键字通常和static一起用于变量，该变量为可被所有该类的对象访问的常量
- final类
  - final类无法被继承