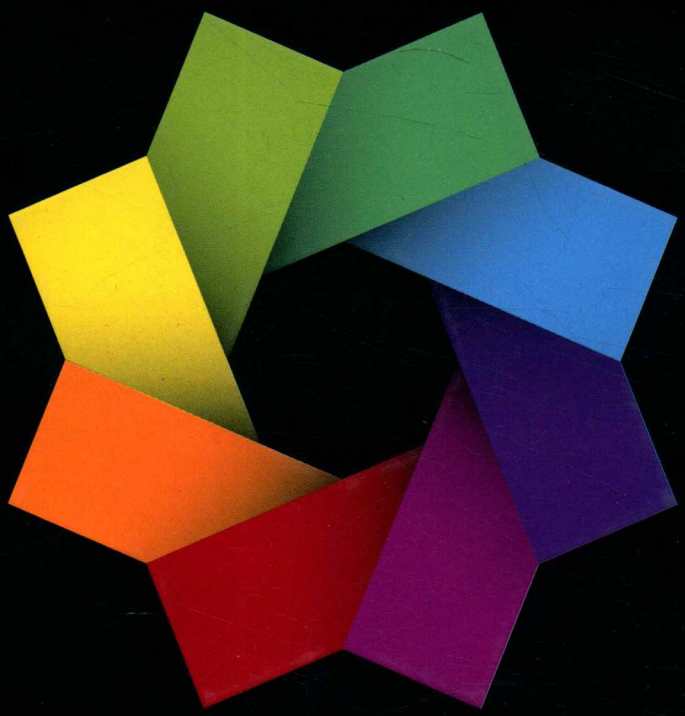
☆ 十年教学经验的积累　☆ 百个经典的项目案例

☆ 深入浅出、循序渐进　☆ 轻松突破编程之难点

配套资源（http://www.tangcco.com/）

Android 编程实战学习手册

唐城教育　编



中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Android 编程实战学习手册

唐城教育　编

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书基于最新的 Android 4.2 编写，Android SDK、ADT 都基于 4.2 版本进行设计。本书全面介绍了 Android 应用开发的相关知识，内容涵盖 Java 基础知识、Android 用户界面开发、Android 四大组件、Android 资源访问、图形/图像处理、事件处理机制、Android 输入/输出处理、音频/视频多媒体开发、网络通信编程、Android 平台的 WebService、传感器应用开发、GPS 应用开发、地图开发等。

本书并不局限于介绍 Android 编程的各种理论，而是从项目案例的角度讲授，全书包括近百个实例，可帮助读者更好地理解各知识点在实际开发中的应用，供开发时参考。如果读者在阅读本书时遇到技术问题，可在网上发帖（www.tangcco.com），作者会及时予以解答。

本书适合对面向对象编程思想有简单了解的读者、计算机相关专业大学生、有志于从事 Android 应用开发的 Java 工程师，以及对编程感兴趣的人员。

图书在版编目（CIP）数据

Android 编程实战学习手册 / 唐城教育编. —北京：电子工业出版社，2016.1
ISBN 978-7-121-27666-8

Ⅰ. ①A… Ⅱ. ①唐… Ⅲ. ①移动终端—应用程序—程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2015）第 285524 号

责任编辑：许存权　　特约编辑：谢忠玉　冯彩茹
印　　刷：涿州市京南印刷厂
装　　订：涿州市京南印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1 092　1/16　印张：32.75　字数：838 千字
版　　次：2016 年 1 月第 1 版
印　　次：2016 年 1 月第 1 次印刷
定　　价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前 言

当我们团队的老师们萌生写书的念头，一切显得那么自然，四本书一气呵成，涵盖技术类、营销类、专业英语类，本书是其中之一。十年磨一剑，今年是唐城的第十个年头，在这个时候出版，既是献礼，也是十年技术、教学经验扎扎实实的沉淀。我们深知，教育是一种理念、一份责任！我们秉承“让无业者有业，让有业者乐业”的职业教育理念，用专业、职业、敬业的施教精神，帮助超 8000 名学员成功跨入 IT 行业、网络营销行业的大门！

因教学经验的积累，我们更知道如何让读者通过阅读，从“不会”到“学会”，从“学会”到“精通”。深入浅出、循序渐进地带领读者走进安卓开发，揭开安卓神秘的面纱，经典的案例，详细的讲解，汇聚了我们多年的教学研讨与企业调研，我们形成了一套成熟的课程体系，完善的组织架构。

移动互联网已经成为当今世界发展最快、市场潜力最大、前景最诱人的业务，而 Android 则是移动互联网上市场占有率最高的平台，也搭载着 Android 系统的设备层出不穷。与此同时，Android 应用选择了 Java 作为其开发语言，这对于 Java 程序员来说是一个极好的机会。

本书特色

本书主要由 4 名资深安卓老师编写，精雕细琢，一切从读者角度出发，案例设置合理紧凑，知识点讲解细致到位。从如何搭建环境开始，到后期 NDK 开发，从前端的 UI 设计，到后台的服务器端建设，面面俱到、以点盖面，提升读者的研发能力，拓展产品思路。由于各位老师长期处于授课第一线，更了解学员学习过程中可能遇到的困难，了解如何通过生活中的案例能使学员快速掌握重点难点内容。以教育为出发点，以学员学会为目标，以实用技能为核心，把这个整体的教育理念灌输于本书。减轻了读者的学习压力，提升了学习效率，通过丰富的项目案例提升读者的个人成就感。本书内容由基础知识到实际开发，结构清晰、语言简洁，非常合适 Android 初学者和 Android 进阶程序开发者阅读参考。

关于读者

本书适合具有一定 Java 编程基础的读者学习，如果读者已熟练掌握 Java 编程语法，并

具有一定的界面编程经验，将更适合阅读本书。否则，阅读本书之前，建议学习一些 Java 编程的基础知识，相关内容可以访问 www.tangcco.com。

同类书比较

本书案例简单易懂，知识架构清晰，由浅入深，由前端到后台，整体诠释了安卓 app 开发的每个要点，所有案例均采用图文混编的形式，并附有通过测试的所有关键代码。总结开发过程中的经验，使读者达到举一反三的效果。真正意义上地实现从零基础快速入门，开始 Android 开发之路，指引新手入门捷径。

感谢

特别感谢鲍健婷、钟华两位校长对本书编写的大力支持，以及对本书内容的指导。

感谢霍炜、唐碧、孙沛林三位老师的整理与撰写，他们兢兢业业、精益求精，将自己最宝贵的知识积累呈现给读者。

感谢聂众老师在编写本书过程中，提供 Java 与 Android 之间的差异化及如何实现在两者间过渡编程的思想。

唐城教育

目 录

第1章

Java OOP基础串讲

欢迎进入唐城 Android 的课程学习，在之前的学习中，已经学习了 Java 的知识，经过一段时间后可能会有些淡忘，那么，在进入正式的 Android 学习前，先来简单复习一下 Java。通过本章内容的学习，应该着重掌握以下内容。

（1）类和对象。

（2）封装。

（3）继承。

（4）多态。

（5）抽象类及接口。

（6）集合框架及泛型。

1.1 对象

Java 是面向对象的编程语言，首先，就来回忆一下什么是对象。

对象是人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看作对象，它不仅能表示具体的事物，还能表示抽象的规则、计划或事件（万物皆对象）。

对象的状态（属性）和行为（方法）如下：

（1）对象具有状态（属性）。使用一组数据值来描述对象的状态（属性）。状态是关于个体的具体事实。比如你的名字，就是你的具体事实。简单做法就是通过名词找属性。

（2）对象具有行为（方法）。用以实现某个功能或修改对象状态的操作就是对象的行为（方法）。简单做法就是通过动词找行为（方法）。

（3）对象实现了数据和操作的结合，用以更准确地描述对象。

（4）通过对象的状态（属性）能够简单地区分不同的对象。比如，你的身份证号和其他人的身份证号不同，用以区分你和其他人。

1.2 类

具有相同特性（数据元素）和行为（功能）的对象的抽象就是类。因此，对象的抽象是类，类实际上就是一种数据类型。

（1）类具有属性，它是对象的状态的抽象，用数据结构来描述类的属性。

（2）类具有操作，它是对象的行为的抽象，用操作名和实现该操作的方法来描述。

从对象中抽取类就是从一系列相似的对象中抽取出它们的共通点。比如，人类。要抽取出人类，那么就必须找到所有隶属于人类的对象的共有特征。如姓名、年龄、身高、体重等。这些都是人类所拥有的共同特征，都是使用名词进行的描述，那么，这些特征就可以定义为类的属性。还可以抽取出一些动作。如吃、跑、跳、说等，这些都是动词也就是类的方法。那么，如何确切地定义有用的属性和方法呢？简单地说，就是抽取出对你的程序运行有帮助的属性和方法，而那些没有帮助的属性和方法，就可以忽略掉。

为了便于回忆，我们先定义一个简单的类。

```
    class student{
        int stuNo;
        String stuName;
        int stuAge;
        int stuGender;

        void show(){
              System.out.print("我叫" + stuName + ", 今年" + stuAge + "岁了。
")
    }
    }
```

其中，stuNo、stuName、stuAge、stuGender 为类的属性，show()方法为类的操作。通过赋予类的属性以不同的属性值，可以区分不同的对象。而 show()方法则是所有隶属此类对象的共通方法。

1.3 类和对象的关系

在面向对象的编程中，会频繁地定义类和使用对象。那么在程序中，类和对象的关系就非常重要，理清类和对象的关系，有助于在实际编程中快速地定义类和使用对象协作。那么，类和对象之间到底是什么样的关系呢？

以前的学习中，了解到类和对象的关系就是抽象和具体的关系。简单来说，就是包含的关系。类包含对象，对象属于类。类能够代表对象，而对象无法代表类。

1.4　封装

什么是封装？封装就是在程序上，隐藏对象的属性和实现细节，仅对外公开接口，控制在程序中属性的读和修改的访问级别；将抽象得到的数据和行为（或功能）相结合，形成一个有机的整体，也就是将数据与操作数据的源代码进行有机的结合，形成“类”，其中数据和函数都是类的成员。

封装的目的是增强安全性和简化编程，使用者不必了解具体的实现细节，而只是通过外部接口，以特定的访问权限来使用类的成员。

封装在原则上要求把尽可能多的东西藏起来，对外提供简捷的接口。

```
class student{
    private int stuNo;
    public int getStuNo(){
        return stuNo;
}
public int setStuNo(int StuNo){
    this.stuNo = StuNo;
}
}
```

以上就是一个最简单的封装，能够看到将属性使用 private 修饰，外部类无法直接访问属性。增加了两个方法，get()和 set()，用来使外部类能够通过方法访问内部属性，那么，get()和 set()就是属性 stuNo 对外的接口。

1.5　继承

继承一词来自于生活，指按照法律或遵照遗嘱接受死者的财产、职务、头衔、地位等。

程序中的继承是软件技术当中的一个概念。如果一个类 A 继承自另一个类 B，就把这个 A 称为 B 的子类，而把 B 称为“A 的父类”。继承可以使得子类具有父类的各种属性和方法，而不需要再次编写相同的代码。在令子类继承父类的同时，可以重新定义某些属性，并重写某些方法，即覆盖父类的原有属性和方法，使其获得与父类不同的功能。另外，为子类追加新的属性和方法也是常见的做法。

Java 继承是使用已存在类的定义作为基础建立新类的技术，新类的定义可以增加新的数据或新的功能，也可以用父类的功能，但不能选择性地继承父类。

Java 不支持多继承，单继承使 Java 的继承关系很简单，一个类只能有一个父类，易于管理程序，同时一个类可以实现多个接口，从而克服单继承的缺点。

继承避免了对一般类和特殊类之间共同特征进行的重复描述。同时，通过继承可以清晰地表达每一项共同特征所适应的概念范围——在一般类中定义的属性和操作，适应于这个类本身以及它以下的每一层特殊类的全部对象。运用继承原则使得系统模型比较简练也比较清晰。

1.6 多态

多态（Polymorphism）按字面的意思就是“多种状态”。在面向对象语言中，接口的多种不同实现方式即为多态。引用 Charlie Calverts 对多态的描述，即多态性是允许你将父对象设置成为和一个或更多的它的子对象相等的技术，赋值之后，父对象就可以根据当前赋值，给它的子对象的特性以不同的方式运作。如果一个语言只支持类而不支持多态，只能说明它是基于对象的，而不是面向对象的。

从程序设计的角度而言，多态可以如下来实现。

```
public interface Parent {
 public void show();
}

public class Child_ONE implements Parent {
 public void show(){
     //子类 1 的具体实现
}
}

public class Child_TWO implements Parent {
 public void show(){
     //子类 2 的具体实现
}
}

public class Test {
 Public static void main(String[] args) {
     //以父类对象名称指向子类对象实例的方式调用子类 1 的方法
Parent p1 = new Child_ONE();
     p1.show();
     //以父类对象名称指向子类对象实例的方式调用子类 2 的方法
Parent p2 = new Child_TWO();
     P2.show();
}
}
```

p1.show()调用子类 1 的方法，p2.show()调用子类 2 的方法。

所以，对于抽象的父类或者接口给出了具体的实现后，p1 和 p2 可以完全不管实现的细节，而只访问定义的方法，就可以。事实上，这就是多态所起的作用，可以实现控制反转，这在大量的 J2EE 轻量级框架中被用到，比如，Spring 的依赖注射机制。

1.7　抽象类

关键字 abstract 声明的方法称之为抽象方法。抽象方法是一个不完整的方法，只包含方法的声明，不包含方法的具体实现。

如果一个类中至少包含一个或一个以上的抽象方法，则这个方法所在的类必须声明为：抽象类。

抽象类以及抽象方法的示例如下。

```
//抽象类
public abstract class Human {
 //抽象方法
 public abstract  void eat();
}
```

但是，即使不包括任何 abstract 方法，亦可将一个类声明成抽象类。如果一个类没必要拥有任何抽象方法，而且又想禁止那个类的所有实例，这种能力就会显得非常有用。

在面向对象的概念中，所有的对象都是通过类来描绘的，但是反过来却不是这样。并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类。抽象类往往用来表征对问题领域进行分析、设计中得出的抽象概念，是对一系列看上去不同，但是本质上相同的具体概念的抽象。比如，如果进行一个图形编辑软件的开发，就会发现问题领域存在着圆、三角形这样一些具体概念，它们是不同的，但是它们又都属于形状这样一个概念，形状这个概念在问题领域是不存在的，它就是一个抽象概念。正是因为抽象的概念在问题领域没有对应的具体概念，所以，用以表征抽象概念的抽象类是不能够实例化的。

现在来说说抽象类在继承中的体现。

如果从一个抽象类继承，而且想生成新类型的一个对象，就必须为基础抽象类中的所有抽象方法提供方法具体实现。如果不这样做，则衍生类也会是抽象的，而且编译器会强迫用 abstract 关键字标志那个类的抽象本质。

综上所述，在面向对象领域，抽象类主要用来进行类型隐藏。可以构造出一个固定的一组行为的抽象描述，但是这组行为却能够有任意个可能的具体实现方式。这个抽象描述就是抽象类，而这一组任意个可能的具体实现，表现为所有可能的派生类。模块可以操作一个抽象体，由于模块依赖于一个固定的抽象体，因此，它可以是不允许修改的；同时，通过从这个抽象体派生，也可扩展此模块的行为功能。

1.8　接口

接口实现和类继承的规则不同，为了数据的安全，继承时一个类只有一个直接父类，也就是单继承，但是一个类可以实现多个接口，接口弥补了类不能多继承的缺点，继承和接口的双重设计既保持了类的数据安全，也变相实现了多继承。

Java 接口本身没有任何实现，因为 Java 接口不涉及表象，而只描述 public 行为，所以 Java 接口比 Java 抽象类更抽象化。但是接口不是类，不能使用 new 运算符实例化一个接口。

```
//接口
public interface Parent{
}

public class Test {
 public static void main(String[] args){
 Parent p =  new Parent();//这是错误的!!
}
}

public class Test1 {
 public static void main(String[] args){
 Parent p;//这是正确的
}
}
```

Java 接口的方法只能是抽象的和公开的，Java 接口不能有构造器，Java 接口可以有 public、静态的和 final 属性。即接口中的属性可以定义为 public static final int value = 3。

接口把方法的特征和方法的实现分割开来。这种分割体现在接口常常代表一个角色，它包装与该角色相关的操作和属性，而实现这个接口的类便是扮演这个角色的演员。一个角色由不同的演员来演，而不同的演员之间除了扮演一个共同的角色外，并不要求其他的共同之处。

1.9 集合框架

集合框架是为表示和操作集合而规定的一种统一标准的体系结构。任何集合框架都包含三大块内容：对外的接口、接口的实现和对集合运算的算法。

Java2 的集合框架，就其核心，主要有三类：List、Set 和 Map。List 和 Set 继承了 Collection，而 Map 则独成一体。初看上去，可能会对 Map 独成一体感到不解，它为什么不继承 Collection 呢？但是仔细想想，这种设计是合理的。一个 Map 提供了通过 Key 对 Map 中存储的 Value 进行访问，也就是说，它操作的都是成对的对象元素，比如 put()和 get()方法，而这是一个 Set 和 List 所不具备的。当然在需要时，可以由 keySet()方法或 values()方法从一个 Map 中得到键的 Set 集或值的 Collection 集

1．Collection 接口提供了一组操作成批对象的方法

它提供了基本操作，如添加、删除。它也支持查询操作，如是否为空 isEmpty()方法等。为了支持对 Collection 进行独立操作，Java 的集合框架给出了一个 Iterator，它可以泛型操作一个 Collection，而不需知道这个 Collection 的具体实现类型是什么。

2. List 接口对 Collection 进行了简单的扩充

它常用的具体实现类有 ArrayList 和 LinkedList。可以将任何东西放到一个 List 容器中，并在需要时从中取出。从 ArrayList 命名中可以看出，它以一种类似数组的形式进行存储，因此，它的随机访问速度极快，而 LinkedList 的内部实现是链表，它适合用于在链表中间需要频繁进行插入和删除操作。在具体应用时，可以根据需要自由选择。List 最重要的特征就是有序；它能确保以一定顺序保存元素。List 在 Collection 基础上添加了大量方法，使之能在序列中间插入和删除元素（只对 LinkedList 推荐使用）。List 可以制造 ListIterator 对象，除能用它在 List 中间插入和删除元素外，还能用它沿两个方法遍历 List。

ArrayList：一个用数组实现的 List。能进行快速的随机访问，但是往列表中间插入和删除元素时比较慢。ListIterator 只能用在反向遍历 ArrayList 的场合，不能用它插入和删除元素，因为相比 LinkedList，在 ArrayList 里用 ListIterator 的系统开销比较高。

LinkedList：对顺序访问进行了优化。在 List 中间插入和删除元素的代价也不高，随机访问的速度相对较慢。此外，它还有 addFirst()、addLast()、getFirst()、getLast()、removeFirst()和 removeLast()等方法，能把它当成栈（stack）、队列（queue）、双向队列（deque）来用。

3. Map 是一种把键对象和值对象进行关联的容器

一个值对象又可以是一个 Map，因此，这样就可形成一个多级映射。对于键对象来说，像 Set 一样，一个 Map 容器中的键对象不允许重复，这是为了保持查找结果的一致性；如果有两个键对象一样，想得到那个键对象所对应的值对象时就有问题，可能得到的并不是想要的那个值对象，结果会造成混乱，所以键的唯一性很重要，也符合集合的性质。当然，在使用过程中，某个键所对应的值对象可能会发生变化，这时会按照最后一次修改的值对象与键对应。对于值对象没有唯一性的要求，可以将任意多个键都映射到一个值对象上，这不会发生任何问题（不过对使用可能会造成不便，不知道得到的到底是那一个键所对应的值对象）。Map 有两种比较常用的实现方法：HashMap 和 TreeMap。HashMap 也用到了哈希码的算法，以便快速查找一个键，TreeMap 则是对键按序存放，因此，它有一些扩展的方法，比如 firstKey()、lastKey()等，还可以从 TreeMap 中指定一个范围以取得其子 Map。键和值的关联很简单，用 put(Object key,Object value)方法即可将一个键与一个值对象相关联。用 get(Object key)可得到与此 key 对象所对应的值对象。

HashMap：基于 hash 表的实现（用它来代替 Hashtable）。提供时间恒定的插入与查询，在构造函数中可以设置 hash 表的 capacity 和 load factor，可以通过构造函数来调节其性能。

TreeMap：基于红黑树数据结构的实现。当查看键或 pair 时，会发现它们是按顺序（根据 Comparable 或 Comparator，全面介绍）排列的。TreeMap 的特点，所得到的是一个有序的 Map。TreeMap 是 Map 中唯一有 subMap()方法的实现。这个方法能获取这个树中的一部分。

综上所述：

①Collection 持有单个元素，而 Map 持有相关联的 pair。

②和数组一样，List 也把数字下标与对象联系起来，可以把数组和 List 想象成有序的容器，List 会随元素的增加自动调整容量。

③如果要做很多随机访问，那么用 ArrayList，但是，如果要在 List 的中间做很多插入和删

除的话，就应该用 LinkedList。

④LinkedList 能提供队列、双向队列和栈的功能。

⑤Map 提供的不是对象与数组的关联，而是对象和对象的关联。

⑥HashMap 看重的是访问速度，而 TreeMap 看重键的顺序，因而 TreeMap 不如 HashMap 快。

⑦没必要在新代码里再使用旧类库留下来的 Vector、Hashtable 和 Stack。

1.10 泛型

泛型（Generic type 或 generics）是对 Java 语言的类型系统的一种扩展，以支持创建可以按类型进行参数化的类。可以把类型参数看作使用参数化类型时指定类型的一个占位符，就像方法的形式参数是运行时传递值的占位符一样，Java 泛型是 JDK1.5 的新特性。

示例如下：

```
//不使用泛型
 List list = new ArrayList();
 list.put(new Integer(3));
 Integer I = (Integer)list(0);

//使用泛型
 List<Integer> list = new ArrayList<Integer>();
 list.put(new Integer(3));
 Integer I = list(0);
```

在 Java 语言中，引入泛型的一个重要目标，就是维护向后兼容。尽管 JDK 5.0 的标准类库中的许多类，比如集合框架，都已经泛型化，但是，使用集合类（比如 HashMap 和 ArrayList）的现有代码，将继续不加修改地在 JDK 5.0 中工作。当然，没有利用泛型的现有代码，将不会赢得泛型的类型安全好处。

这里，需要注意一个问题。关于泛型的混淆， List<Object> 不是 List<String> 的父类型。他们之间没有继承关系，即使 String 继承了 Object。

下面的代码是非法的：

```
//使用泛型
 List<String> list = new ArrayList<String>();
 List<Object> olist = list;
```

这样设计的原因，在于根据 olist 的声明，编译器允许向 olist 中添加任意对象（例如 Integer），但是，此对象是 List<String>，破坏了数据类型的完整性

泛型同样可以用于方法和类，即泛型方法和泛型类。

1. 泛型方法

在引入泛型之前，类中的方法要支持多个数据类型，就需要对方法进行重载，在引入泛型后，可以解决此问题（多态），进一步可以定义多个参数以及返回值之间的关系。