2、具体内容

虽然在类之中的基本组成就是成员属性与方法，但是在任何的语言里面结构也是允许进行嵌套的，所以在一个类的内部可以定义其它的类，这样的类就称为内部类。

■内部类的基本定义

如果说到内部类肯定其本身是一个独立且完善的类结构，在一个类的内部除了属性和方法之外可以继续使用class定义内部类。

范例：基本类的基本定义

```
class Outer {    // 外部类
        private String msg = "www.mldn.cn" ;  // 私有成员属性
        public void fun() {   // 普通方法
                Inner in = new Inner() ;  // 实例化内部类对象
                in.print() ;        // 调用内部类方法
        }
        class Inner { // 在Outer类的内部定义了Inner类
                public void print() {
                        System.out.println(Outer.this.msg) ;       // Outer类中的属性
                }
        }
}
public class JavaDemo {
        public static void main(String args[]) {
                Outer out = new Outer() ;      // 实例化外部类对象
                out.fun() ;        // 调用外部类中的方法
        }
}
```

从整个的代码上实际上发现内部类的结构并不难理解，甚至可以说其结构与普通类一样清晰明了。那么为什么会提供有内部类这样的结构呢？

因为从整体的的代码结构上来讲内部类的结构并不合理，所以内部类本身最大的缺陷在于破坏了程序的结构，但是破坏需要有目的的破坏，那么它也一定会有其优势，如果要想更好的观察出内部类的优势，就可以将内部类拿到外面来。

范例：将以上程序分为两个类

```
class Outer {    // 外部类
    private String msg = "www.mldn.cn" ;  // 私有成员属性
    public void fun() {   // 普通方法
        // 思考五：需要将当前对象Outer传递到Inner类之中
        Inner in = new Inner(this) ;  // 实例化内部类对象
        in.print() ;      // 调用内部类方法
    }
    // 思考一：msg属性如果要被外部访问需要提供有getter方法
    public String getMsg() {
        return this.msg ;
    }
}
class Inner { // 在Outer类的内部定义了Inner类
    // 思考三：Inner这个类对象实例化的时候需要Outer类的引用
    private Outer out ;
    // 思考四：应该通过Inner类的构造方法获取Outer类对象
    public Inner(Outer out) {
        this.out = out ;
    }
    public void print() {
        // 思考二：如果要想调用外部类中的getter方法，那么一定需要有Outer类对象
        System.out.println(this.out.getMsg()) ; // Outer类中的属性
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Outer out = new Outer() ;      // 实例化外部类对象
        out.fun() ;       // 调用外部类中的方法
    }
}
```

可以发现，整体的操作之中，折腾半天主要的目的就是为了让Inner这个内部类可以访问Outer这个类的中的私有属性，但是如果不用内部类的时候整体代码非常的麻烦，所以可以得出内部类的优点：轻松的访问外部类中的私有属性。

■内部类相关说明

现在已经清楚的认识到了内部类的优势以及结构，那么随后需要对内部类进行一些相关的说明，现在所定义的内部类都属于普通的内部类的形式，普通的类内部往往会提供有属性和方法，需要注意的是，内部类虽然可以方便的访问外部类中的私有成员或私有方法，同理，外部类也可以轻松访问内部类中的私有成员或私有方法。

范例：外部类访问内部类中的私有属性

```
class Outer {   // 外部类
    private String msg = "www.mldn.cn" ;  // 私有成员属性
    public void fun() {   // 普通方法
        Inner in = new Inner() ;  // 实例化内部类对象
        in.print() ;       // 调用内部类方法
        System.out.println(in.info) ;   // 访问内部类的私有属性
    }
    class Inner { // 在Outer类的内部定义了Inner类
        private String info = "今天天气不好，收衣服啦！" ;
        public void print() {
            System.out.println(Outer.this.msg) ;      // Outer类中的属性
        }
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Outer out = new Outer() ;      // 实例化外部类对象
        out.fun() ;       // 调用外部类中的方法
    }
}
```

使用了内部类之后，内部类与外部类之间的私有操作的访问就不再需要通过setter、getter以及其它的间接方式完成了，可以直接进行处理操作。

但是需要注意的是，内部类本身也属于一个类，虽然在大部分的情况下内部类往往是被外部类包裹的，但是外部依然可以产生内部类的实例化对象，而此时内部类实例化对象的格式如下：

```
外部类.内部类 内部类对象 = new 外部类().new 内部类() ;
```

在内部类编译完成之后会自动形成一个“Outer$Inner.class”类文件，其中“$“这个符号换到程序之中就变为了”.”，所以内部类的全程：”外部类.内部类”。内部类与外部类之间可以直接进行私有成员的访问，这样一来内部类如果要是提供有实例化对象了，一定要先保证外部类已经实例化了。

```
class Outer {   // 外部类
    private String msg = "www.mldn.cn" ;  // 私有成员属性
    class Inner { // 在Outer类的内部定义了Inner类
        public void print() {
            System.out.println(Outer.this.msg) ;      // Outer类中的属性
        }
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Outer.Inner in = new Outer().new Inner() ;
        in.print() ;
    }
}
```

如果此时Inner类只允许Outer类来使用，那么在这样的情况下就可以使用private进行私有定义。

```
class Outer {    // 外部类
    private String msg = "www.mldn.cn" ;   // 私有成员属性
    private class Inner { // 在Outer类的内部定义了Inner类
        public void print() {
            System.out.println(Outer.this.msg) ;      // Outer类中的属性
        }
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        Outer.Inner in = new Outer().new Inner() ;
        in.print() ;
    }
}
```

此时Inner类无法在外部进行使用。

在Java之中类作为最基础的结构体实际上还有与类似的抽象类或者是接口，抽象类与接口中都可以定义内部结构。

范例：定义内部接口

```
interface IChannel {// 定义接口
    public void send(IMessage msg) ;  // 发送消息
    interface IMessage {       // 内部接口
        public String getContent() ; // 获取消息内容
    }
}
class ChannelImpl implements IChannel {
    public void send(IMessage msg) {
        System.out.println("发送消息：" + msg.getContent()) ;
    }
    class MessageImpl implements IMessage {
        public String getContent() {
            return "www.mldn.cn" ;
        }
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        IChannel channel = new ChannelImpl() ;
        channel.send(((ChannelImpl)channel).new MessageImpl()) ;
    }
}
```

下面继续观察一个内部的抽象类，内部抽象类可以定义在普通类、抽象类、接口内部都是可以。

范例：观察内部抽象类

```
interface IChannel {// 定义接口
    public void send() ; // 发送消息
```

```
    abstract class AbstractMessage {
        public abstract String getContent() ;
    }
}
class ChannelImpl implements IChannel {
    public void send() {
        AbstractMessage msg = new MessageImpl() ;
        System.out.println(msg.getContent()) ;
    }
    class MessageImpl extends AbstractMessage {
        public String getContent() {
            return "www.mldn.cn" ;
        }
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        IChannel channel = new ChannelImpl() ;
        channel.send() ;
    }
}
```

内部类还有一些更为有意思的结构，即：如果现在定义了一个接口那么可以在内部利用类实现该接口，在JDK1.8之后接口中追加了static方法可以不受到实例化对象的控制，现在就可以利用此特性来完成功能。

范例：接口内部进行接口实现

```
interface IChannel {// 定义接口
    public void send() ; // 发送消息
    class ChannelImpl implements IChannel {
        public void send() {
            System.out.println("www.mldn.cn") ;
        }
    }
    public static IChannel getInstance() {
        return new ChannelImpl() ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        IChannel channel = IChannel.getInstance() ;
        channel.send() ;
    }
}
```

内部类是一种非常灵活的定义结构，只要语法满足了，各种需求都可以帮你实现。

■static定义内部类

如果说现在在内部类上使用了static定义，那么这个内部类就变为了“外部类”，static定义的都是独立于类的结构，所以该类结构就相当于是一个独立的程序类了。需要注意的是，static定义的不管是类还是方法只能够访问static成员，所以static定义的内部类只能够访问外部类中static属性或方法；

范例：使用static定义内部类

```
class Outer {
    private static final String   MSG = "www.mldn.cn" ;
    private class Inner {
        public void print() {
            System.out.println(Outer.MSG) ;
        }
    }
}
```

这个时候的Inner类时一个独立的类，如果此时要想实例化Inner类对象，只需要根据“外部类.内部类”的结构实例化对象即可，格式如下：

```
外部类.内部类 内部类对象 = new 外部类().内部类() ;
```

这个时候的类名称有“.”。

范例：实例化static内部类对象

```
public class JavaDemo {
    public static void main(String args[]) {
        Outer.Inner in = new Outer().new Inner() ;
        in.print() ;
    }
}
```

以后在开发之中如果发现类名称上提供有“.”首先应该立刻想到这是一个内部类的结构，如果可以直接实例化，则应该立刻认识到这是一个static定义的内部类。、

如果以static定义内部类的形式了讲并不常用，static定义内部接口的形式最为常用。

范例：使用static定义内部接口

```
interface IMessageWarp {      // 消息包装
    static interface IMessage {
        public String getContent() ;
    }
    static interface IChannel {
        public boolean connect() ; // 消息的发送通道
    }
    public static void send(IMessage msg,IChannel channel) {
        if (channel.connect()) {
            System.out.println(msg.getContent()) ;
        } else {
            System.out.println("消息通道无法建立，消息发送失败！") ;
        }
    }
}
class DefaultMessage implements IMessageWarp.IMessage {
```

```
    public String getContent() {
        return "www.mldn.cn" ;
    }
}
class NetChannel implements IMessageWarp.IChannel {
    public boolean connect() {
        return true ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        IMessageWarp.send(new DefaultMessage() , new NetChannel() ) ;
    }
}
```

之所以使用static定义的内部接口，主要是因为这些操作是属于一组相关的定义，有了外部接口之后可以更加明确的描述出这些接口的主要功能。

■方法中定义内部类

内部类可以在任意的结构中进行定义，这就包括了：类中、方法中、代码块中，但是从实际的开发来讲在方法定义内部类的形式较多。

范例：观察在方法中定义的内部类

```
class Outer {
    private String msg = "www.mldn.cn" ;
    public void fun(long time) {
        class Inner { // 内部类
            public void print() {
                System.out.println(Outer.this.msg) ;
                System.out.println(time) ;
            }
        }
        new Inner().print() ; // 方法中直接实例化内部类对象
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        new Outer().fun(2390239023L) ;
    }
}
```

此时在fun()方法内部提供有Inner内部类的定义，并且可以发现内部类可以直接访问外部类中的私有属性也可以直接访问方法中的参数，但是对于方法中的参数直接访问是从JDK1,8开始的支持的，而在JDK1.8之前，如果方法中定义的内部类要想访问方法中的参数则参数前必须追加final。

范例：在JDK1.8以前的程序结构

```
class Outer {
```

```
    private String msg = "www.mldn.cn" ;
    public void fun(final long time) {
        final String info = "我很好" ;
        class Inner { // 内部类
            public void print() {
                System.out.println(Outer.this.msg) ;
                System.out.println(time) ;
                System.out.println(info) ;
            }
        }
        new Inner().print() ; // 方法中直接实例化内部类对象
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        new Outer().fun(2390239023L) ;
    }
}
```

之所以取消这样的限制，主要是为了其扩展的函数式编程准备的功能。

■匿名内部类

匿名内部类是一种简化的内部类的处理形式，其主要是在抽象类和接口的子类上使用的。

范例：观察一个基本的程序结构

```
interface IMessage {
    public void send(String str) ;
}
class MessageImpl implements IMessage {
    public void send(String str) {
        System.out.println(str) ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        IMessage msg = new MessageImpl() ;
        msg.send("www.mldn.cn") ;
    }
}
```

如果说现在IMessage接口中的MessageImpl子类只使用唯一的一次，那么是否还有必要将其定义为单独的类，那么在这样的要求下发现这个时候定义的子类是有些多余了，所以就可以利用匿名内部类的形式来解决此问题。

范例：使用匿名内部类

```
interface IMessage {
    public void send(String str) ;
}
```

```
public class JavaDemo {
    public static void main(String args[]) {
        IMessage msg = new IMessage() {      // 匿名内部类
            public void send(String str) {
                System.out.println(str) ;
            }
        } ;
        msg.send("www.mldn.cn") ;
    }
}
```

有些时候为了更加方便的体现出匿名内部类的使用，往往可以利用静态方法做一个内部的匿名类实现。

范例：在接口中直接定义匿名内部类

```
interface IMessage {
    public void send(String str) ;
    public static IMessage getInstance() {
        return new IMessage() {
            public void send(String str) {
                System.out.println(str) ;
            }
        } ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        IMessage.getInstance().send("www.mldn.cn") ;
    }
}
```

与内部类相比匿名内部类只是一个没有名字的只能够使用一次的，并且结构固定的一个子类。