第3章

类、对象与方法

3.1 类

3.1.1 类的定义

在 Objective-C 中，每个类的定义包括两个部分：接口（interface）和实现（implementation）。接口部分定义在 .h 文件中，包含类的声明、属性以及方法，主要作用是对外提供访问接口。实现部分定义在 .m 文件中，主要用于方法的功能实现。这种定义类的方法，好处在于将公共声明（接口）与代码实现分开，对外屏蔽了功能实现的细节，体现了面向对象的封装特性。

1. 类定义简介

类定义是一种类型的对象的原型，其声明了每个对象都具有的**属性（Property）**，同时还定义了同一类的所有对象都使用的一系列**方法（Method）**。

编译器为每个类创建一个可访问的对象，称为类的对象（也称为工厂对象），类方法负责创建属于该类的新对象。类对象是类编译后的版本，由它创建的对象叫作类的实例，创建对象的过程也常称作实例化。

在 Objective-C 中，一个类有接口文件和实现文件两部分组成，通常放在不同的文件里面。其中：

- 接口文件（.h 后缀文件）：主要完成类的具体声明。
- 实现文件（.m 后缀文件）：主要完成类的具体实现。

当使用 Xcode 创建一个类时，Xcode 会自动创建 .h 和 .m 两个文件。

2. 接口文件

接口文件完成类的声明，向外界提供如何使用该类，主要体现了以下 3 类信息。

- 继承关系：通过继承关系可以了解该类是哪个类的子类，从而可以了解可供调用的父类的方法和属性。
- 属性 Property。
- 方法 Method。

接口部分的定义以 @interface 开始，以 @end 结束。下面的示例代码，定义了一个游戏人物的 Player 类，这个类继承自 NSObject，其中又定义了两个人物属性 healthPoint 与 magicPoint，以及一个初始化类方法 player，同时又提供了两个攻击方法，一个普通攻击 normalAttack，一个魔法攻击 magicAttack。

```
#import <Foundation/Foundation.h>
@interface Player : NSObject
@property (nonatomic,assign) int healthPoint;
@property (nonatomic,assign) int magicPoint;
```

```
+(Player *)player;
-(void) normalAttack;
-(void) magicAttack;
@end
```

注意：在 iOS 开发中，属性和方法的命名一般推荐使用驼峰法命名规则。驼峰法命名规则就是当属性名或者方法名由多个单词构成时，第一个单词以小写字母开始，之后每个单词的首字母都用大写，驼峰法命名规则可以提升程序的可读性。

3. 实现文件

类的实现部分以 @implementation 开始，并以 @end 结束。在类的实现部分，需要使用 #import 命令，引用类的接口部分。

在类的实现部分（.m 文件）中，需要对接口部分定义的方法进行实现。

如下面实例，在类方法 Player 中，创建了 Player 对象，并且给 healthPoint/magicPoint 属性赋了初始值，并且定义了两个攻击方法，其中，当使用魔法攻击的时候，还会消耗自身的 magicPoint 值。

- 引用 .h 文件。

```
#import "Player.h"
```

- 实现接口文件中定义的方法。

```
@implementation Player

+(Player *) player{
    //实例化对象
    Player *player=[[Player alloc] init];
    //设置属性初始值
    player.healthPoint=100;
    player.magicPoint=100;
    //返回对象
    return player;
}

-(void)normalAttack{
    //普通攻击
    ...
}

-(void)magicAttack{
    //魔法攻击，消耗自身魔法值
    self.magicPoint -=10;
}

@end
```

3.1.2 类的继承

继承是面向对象编程的重要特性之一，类定义是累加的，每一个定义的类都是基于其父类，并且可以继承父类中定义的属性和方法。需要注意的是，NSObject 类是所有类的父类。

1. 新增子类

在 Xcode 中，依次单击 File → New → File → Cocoa Touch Class，即可新增类，如图 3-1 所示。在 Subclass of 中输入或者选择新增类的父类，如图 3-2 所示：新增的 Magician 类是 Player 类的子类，因为魔法师（Magician）也是属于游戏人物（Player）之一。

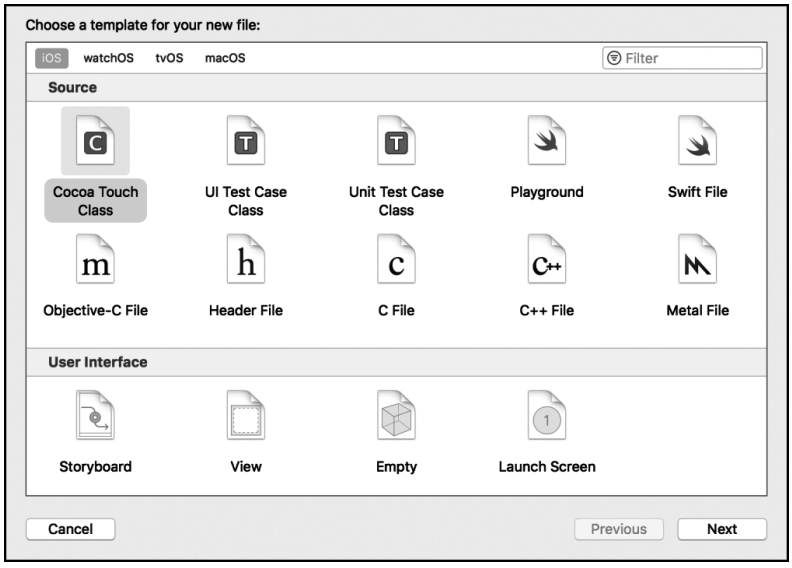

图3-1 新建类

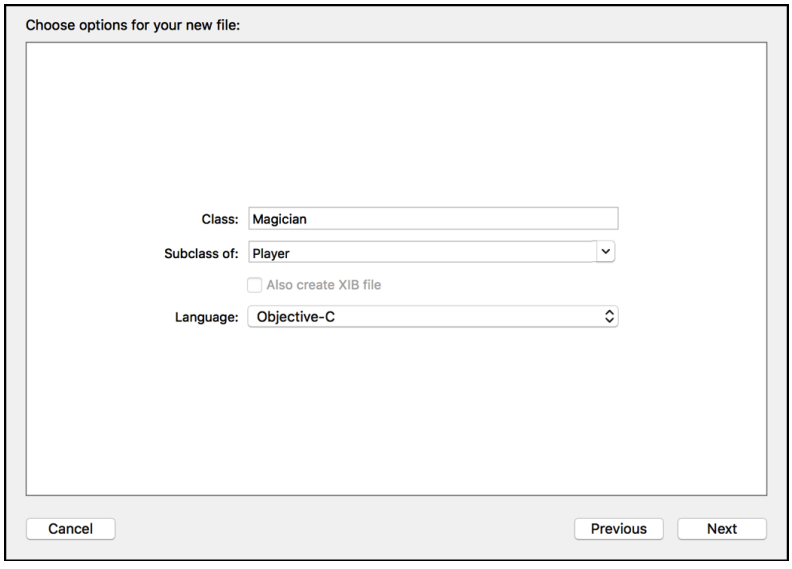

图3-2 类命名

2. 子类继承父类的属性与方法

继承是面向对象编程的重要特征，在 Objective-C 中创建的类也遵循继承的原则。主要体现在两个方面。

- 继承父类的属性：当创建一个新的类时，新的类不仅仅包含其自身定义的属性，并且还包括在其父类中定义的属性，以及其父类的父类的属性，并按照这个规则一直追溯到根类。在 iOS 开发中，**NSObject** 是所有类的根类，因此，所有类都可以使用 NSObject 中定义的属性以及方法。
- 继承父类的方法：一个对象既可以访问自身定义的方法，也可以访问所有父类的方法。当需要调用父类的方法时，需要使用到关键字 **super**。

下面的示例代码中，新增的 **Magician** 类是 Player 类的子类，它具有两个自身特有的方法 specialMagicAttack 和 mixAttack。其中，specialMagicAttack 方法中，使用到了父类的属性 magicPoint；另外，在 mixAttack 方法中，使用 **super** 关键字，调用了父类的两个方法。

- Magician.h 文件定义。

```
#import "Player.h"
@interface Magician : Player

/* 魔法师特殊魔法攻击 */
-(void) specialMagicAttack;

/* 混合攻击（普通攻击 + 魔法攻击）*/
-(void) mixAttack;
@end
```

- Magician.m 文件中实现 specialMagicAttack 方法，其中调用了父类中定义的属性 magicPoint。

```
#import "Magician.h"
@implementation Magician

-(void)specialMagicAttack{
    self.magicPoint -=50;
    NSLog(@"specialMagicAttack!");
}
```

- Magician.m 文件中实现 mixAttack 方法，其中调用了父类中的 normalAttack 以及 magicAttack 方法，注意调用时需要使用 super 关键字，说明调用的是定义在父类中的方法。

```
-(void)mixAttack{
    // 普通攻击
    [super normalAttack];
    // 魔法攻击
    [super magicAttack];
}
```

3.2 属性

3.2.1 属性的定义

属性（property）在类的定义中使用非常普遍。属性定义后，会创建一个与该属性名称同名且带下划线的实例变量，与此同时，编译器会根据属性的特性，自动合成该属性对应实例变量的存取方法（getter 和 setter 方法）。

1. 属性中的一些概念

在深入了解 Objective-C 的属性之前，首先需要了解一下与属性相关的一些基本概念。

- **属性（property）**：通常指的是由对象封装或者存储的数据，例如对于 UIView 视图类对象，可以具有多个描述该视图类的属性，例如：大小、位置、颜色、是否可以响应用户交互等；
- **存取方法（setter/getter）**：设置 / 获取该类的对象的属性值，执行设置 / 获取操作的方法称为存取方法；
- **getter 方法**：返回属性的值，名称与属性名相同。在实际开发中，常说的懒加载（lazy loading）就是 getter 方法；
- **setter 方法**：设定属性的值，setter 方法的形式为 **setPropertyName:**，其中属性名称的第一个字母大写。

2. 属性的声明

当在类中声明一个属性时，需要在类的 @interface 代码部分编写，格式如下：

```
@property (attributes) type name;
```

其中：

- **@property**　属性定义关键字；
- **attribute**　属性的特性，提供了该属性的存储方式以及属性行为的说明。常见的关键字有 weak/strong、assign、copy、atomic/nonatomic，有关属性关键字的说明后续章节会详细介绍；
- **type**　属性的类型说明，如：NSString、NSNumber、int、CGfloat 以及一些自定义类；
- **name**　属性的名称，属性的名称命名需要遵守驼峰法则。

例如，可以创建一个 MYClass 类，并且在其中添加一些属性，这些属性的类型包含了基本数据类型，Foundation 框架中定义的类型，自定义类以及 id 类型。

```
#import <Foundation/Foundation.h>
#import "Player.h"

@interface MYClass : NSObject

@property (nonatomic, copy) NSString *name;
@property (nonatomic,assign) int age;
@property (nonatomic,strong) Player *player;
@property (nonatomic,weak) id delegate;

@end
```