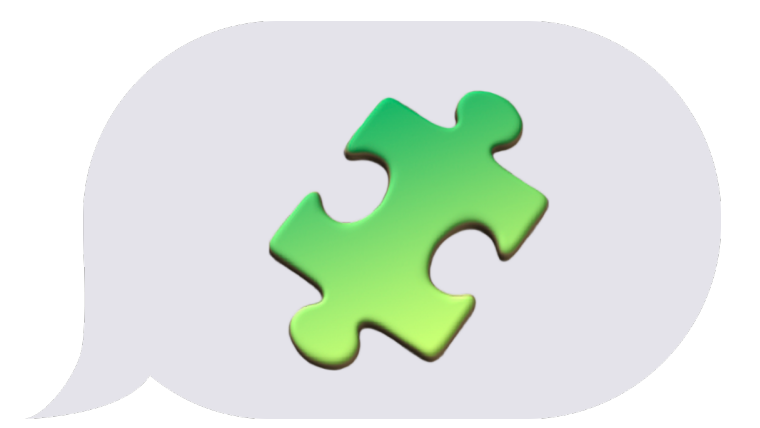
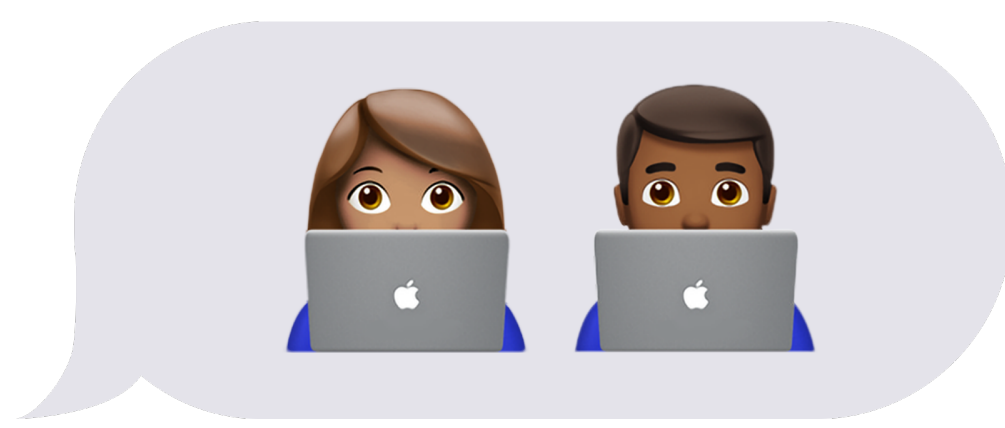
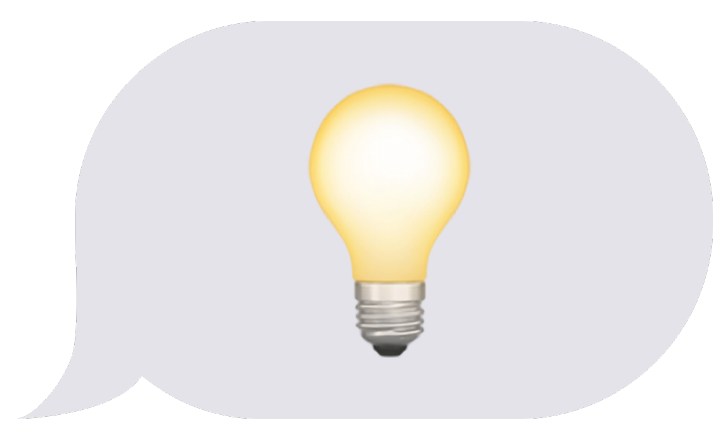
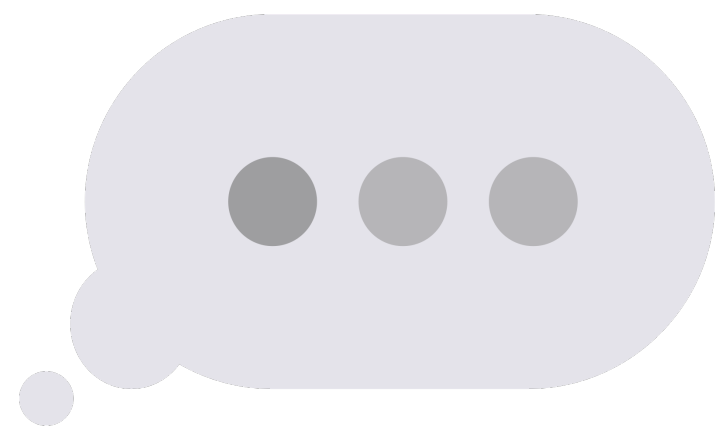


移动平台开发

实验课五：属性、动画与数据存储实操



2025春夏学期

课程目录



1

iOS基于SwiftUI的动画设计与交互

- 复习关于属性的相关知识
- 属性观察器和属性包装器
- 基于SwiftUI的动画

2

CoreData基础

iOS基于SwiftUI的动画设计与交互

在学习**动画**之前...

存储属性

- 是存储在**class**或**struct**中的属性，可以是常量或者变量

- 初始化：在定义时进行初始化，或者

在**init()**函数中进行初始化

思考题🤔：右边的代码块存在什么问题？

```
struct ValueType{  
    var value1: Int?  
    let value2 = 2  
}
```

```
class ReferenceType{  
    var value1: Int?  
    let value2 = 2  
}
```

```
let vt = ValueType()  
let rt = ReferenceType()
```

```
vt.value1 = rt.value1  
rt.value1 = vt.value1
```

iOS基于SwiftUI的动画设计与交互

在学习动画之前...

存储属性

- 是存储在class或struct中的属性，可以是常量或者变量

- 初始化：在定义时进行初始化，或者

在Init()函数中进行初始化

对于值类型（结构体），如果声明为常量let，那所有的属性都是不可以改变的
而对于引用类型（类），即使声明为常量let，其中的变量属性仍可以修改

```
struct ValueType{  
    var value1: Int?  
    let value2 = 2  
}
```

```
class ReferenceType{  
    var value1: Int?  
    let value2 = 2  
}
```

```
let vt = ValueType()  
let rt = ReferenceType()
```

```
vt.value1 = rt.value1  
rt.value1 = vt.value1
```

iOS基于SwiftUI的动画设计与交互



在学习**动画**之前...

存储属性

延时存储属性

计算属性

get & set

属性观察器

willSet & didSet

属性包装器

@wrapperName

在学习动画之前...

延时存储属性

- 其初始值到第一次被使用的时候才会计算的属性，使用关键字`lazy`
- 因为常量必须在实例化完成之前初始化，所以一般延时存储属性都定义为变量（`var`）
- 使用场景：
 - 初始值由外部环境决定，使用前无法确定初始值
 - 需要大量的、复杂的计算才能完成初始化的值，应该在调用前再初始化

```
class DataImporter{
    var fileName: String?

    func read(from fileName: String)->String?{
        self.fileName = fileName
        // 读取文件很耗时
        return nil
    }
}
```

```
class DataManager{
    lazy var importer = DataImporter()
    var data = [String]()
}
```

```
let manager = DataManager()
manager.importer.read(from: "here")
```

iOS基于SwiftUI的动画设计与交互



在学习动画之前...

计算属性

- 计算属性没有存储值，而是提供了一个Getter和一个可选的Setter，在引用这个属性时再进行计算
- 对于没有设置Setter的计算属性一般称为只读计算属性

```
struct Temperature {  
    var celsius: Double // 存储属性 (摄氏度)  
  
    // 计算属性 (华氏度)  
    var fahrenheit: Double {  
        get {return (celsius * 9/5) + 32}  
        set {celsius = (newValue - 32) * 5/9}  
    }  
}  
  
var temp = Temperature(celsius: 25)  
print("摄氏度: \(temp.celsius)")  
print("华氏度: \(temp.fahrenheit)")  
// 修改华氏度，会自动更新摄氏度  
temp.fahrenheit = 98.6  
print("新的摄氏度: \(temp.celsius)")
```

iOS基于SwiftUI的动画设计与交互



在学习**动画**之前...

属性观察器

- 用于监控和响应属性值的变化，每当属性值被设定的时候都会调用相关的观察器，
 - 可以在以下地方设置属性观察器
 - 自定义的存储属性
 - 继承的存储属性 (添加属性观察器，父类的属性观察器也会被执行)
 - 继承的计算属性 (很少这样做，一般重写getter和setter更常见)
 - 可被定义的属性观察器包括：willSet 和 didSet (可以单独只定义一个)

iOS基于SwiftUI的动画设计与交互



浙江大学
ZHEJIANG UNIVERSITY

在学习动画之前...

思考题🤔：输出结果是什么？

属性观察器

在自定义的存储属性后加上属性观察器

```
class Vehicle{  
    var currentSpeed = 0.0{  
        didSet(newSpeed){  
            self.currentSpeed = newSpeed  
            print("Now at \(currentSpeed) miles per hour")  
        }  
        didSet{  
            if currentSpeed > 120.0 {  
                print("Over Speeding!")  
            }  
        }  
    }  
  
    var description: String{  
        return "traveling at \(currentSpeed) miles per hour"  
    }  
}
```

```
class Car: Vehicle{  
    var gear = 1  
    override var currentSpeed: Double{  
        didSet{  
            gear = Int(currentSpeed/10.0)+1  
            print("Switch to \(gear)")  
        }  
    }  
}
```

在继承的存储属性后加上属性观察器

```
let myCar = Car()  
myCar.currentSpeed = 10  
myCar.currentSpeed = 160
```

iOS基于SwiftUI的动画设计与交互

在学习动画之前...

基类 Now at 10.0 miles per hour
子类 Switch to 2
基类 Now at 160.0 miles per hour
基类 Over Speeding!
子类 Switch to 17

属性观察器

在自定义的存储属性后加上属性观察器

```
class Vehicle{  
    var currentSpeed = 0.0{  
        didSet(newSpeed){  
            self.currentSpeed = newSpeed  
            print("Now at \(currentSpeed) miles per hour")  
        }  
        didSet{  
            if currentSpeed > 120.0 {  
                print("Over Speeding!")  
            }  
        }  
    }  
  
    var description: String{  
        return "traveling at \(currentSpeed) miles per hour"  
    }  
}
```

```
class Car: Vehicle{  
    var gear = 1  
    override var currentSpeed: Double{  
        didSet{  
            gear = Int(currentSpeed/10.0)+1  
            print("Switch to \(gear)")  
        }  
    }  
}
```

在继承的存储属性后加上属性观察器

```
let myCar = Car()  
myCar.currentSpeed = 10  
myCar.currentSpeed = 160
```

iOS基于SwiftUI的动画设计与交互



在学习动画之前...

属性包装器

- 属性包装器可以对属性的操作进行统一过滤，即：
 - 定义好某种包装器
 - 包装到任意属性上
 - 对该属性的操作会经过包装器过滤
- 定义方法：@wrapperName，并创建一个包含wrappedValue计算属性的结构体、枚举类型或者类
- 使用方法，需要访问包装了的属性的值，直接.propertyName，返回的是wrappedValue的getter

iOS基于SwiftUI的动画设计与交互



浙江大学
ZHEJIANG UNIVERSITY

在学习动画之前...

属性包装器名称

属性包装器

`@propertyWrapper`

```
struct TwelveOrLess{  
    private var number: Int  
    init(){  
        number = 0  
    }  
}
```

用于过滤属性的计算属性

```
var wrappedValue: Int{  
    get{  
        return number  
    }  
    set{  
        number = min(newValue, 12)  
    }  
}
```

使用相关的包装器

`struct SmallBox{`

```
    @TwelveOrLess var height: Int  
    @TwelveOrLess var width: Int  
}
```

```
var myBox = SmallBox()  
myBox.height = 15  
myBox.width = 10
```

思考题🤔: `myBox.height * myBox.width`的结果是什么?

iOS基于SwiftUI的动画设计与交互

在学习动画之前...

属性包装器名称

属性包装器

`@propertyWrapper`

```
struct TwelveOrLess{  
    private var number: Int  
    init(){  
        number = 0  
    }  
}
```

用于过滤属性的计算属性

```
var wrappedValue: Int{  
    get{  
        return number  
    }  
    set{  
        number = min(newValue, 12)  
    }  
}
```

使用相关的包装器

`struct SmallBox{`

```
    @TwelveOrLess var height: Int  
    @TwelveOrLess var width: Int  
}
```

```
var myBox = SmallBox()  
myBox.height = 15  
myBox.width = 10
```

答案: 120

iOS基于SwiftUI的动画设计与交互

在学习**动画**之前...

属性包装器

@State

- SwiftUI用于管理存储的属性的包装器，当被**@State**包装的属性发生更改时，当前的视图直接废止，并重新计算**body**的值
- 规定：只在当前视图内部访问被**@State**包装的属性，防止外部干扰，所以被**@State**包装的属性一般声明为**private**

```
struct PlayView: View{
    @State private var isPlaying: Bool = false

    var body: some View{
        HStack{
            Text("Start Play")
                .font(.title)
                .padding()
            Button(action:{
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ?
                    "pause.circle":"play.circle")
            }
        }
    }
}
```

bool值取非



iOS基于SwiftUI的动画设计与交互

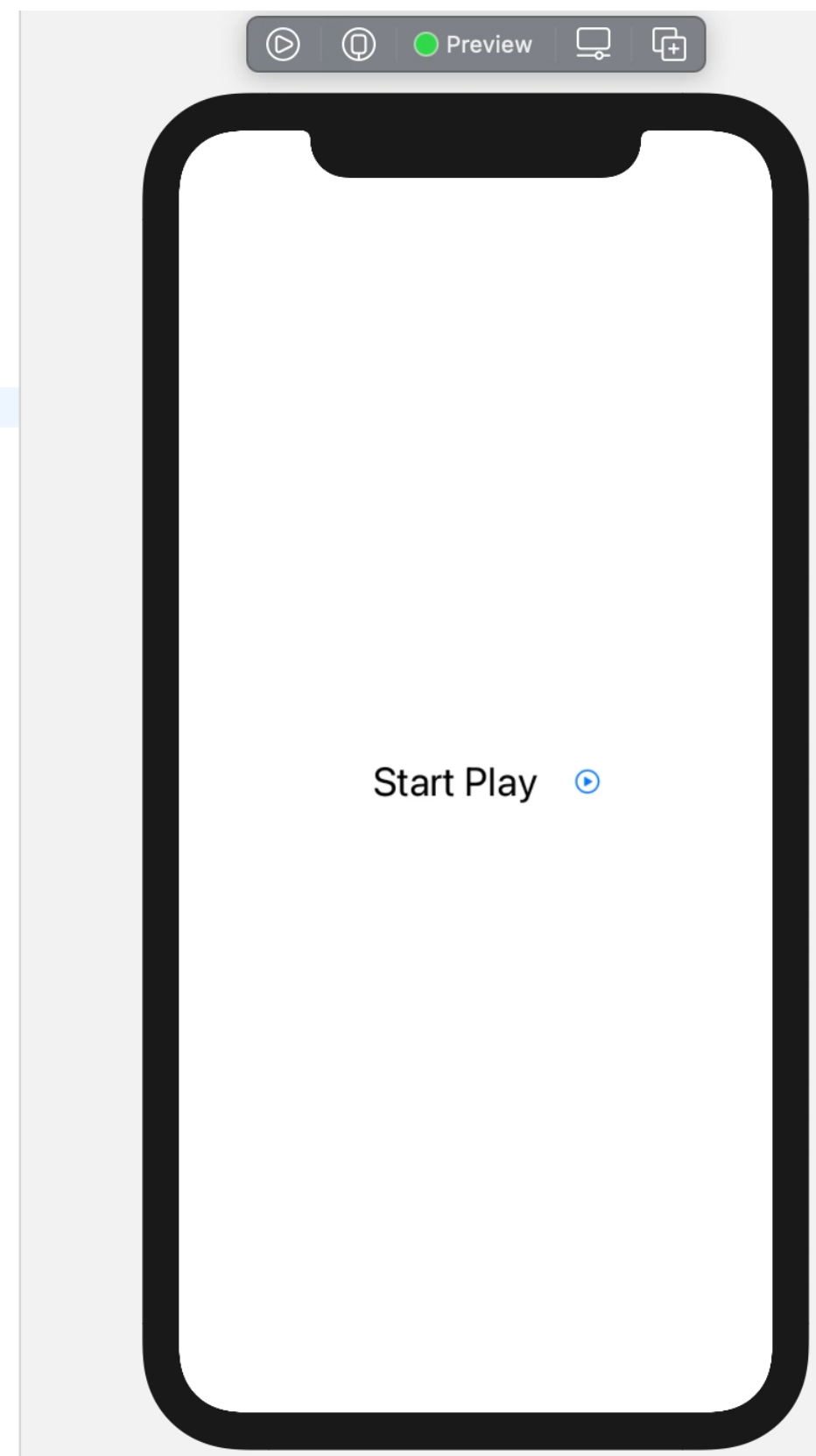
在学习动画之前...

```
// Created by Gibson Yang on 2021/9/22.
//
import SwiftUI

struct PlayView: View {
    @State private var isPlaying: Bool = false

    var body: some View {
        HStack {
            Text("Start Play")
                .font(.title)
                .padding()
            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName:
                    isPlaying ?
                    "pause.circle": "play
                    .circle")
            }
        }
    }
}

struct Start_Previews: PreviewProvider {
    static var previews: some View {
        PlayView()
    }
}
```



```
struct PlayView: View {
    @State private var isPlaying: Bool = false

    var body: some View {
        HStack {
            Text("Start Play")
                .font(.title)
                .padding()
            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ?
                    "pause.circle": "play.circle")
            }
        }
    }
}
```

bool值取非

iOS基于SwiftUI的动画设计与交互

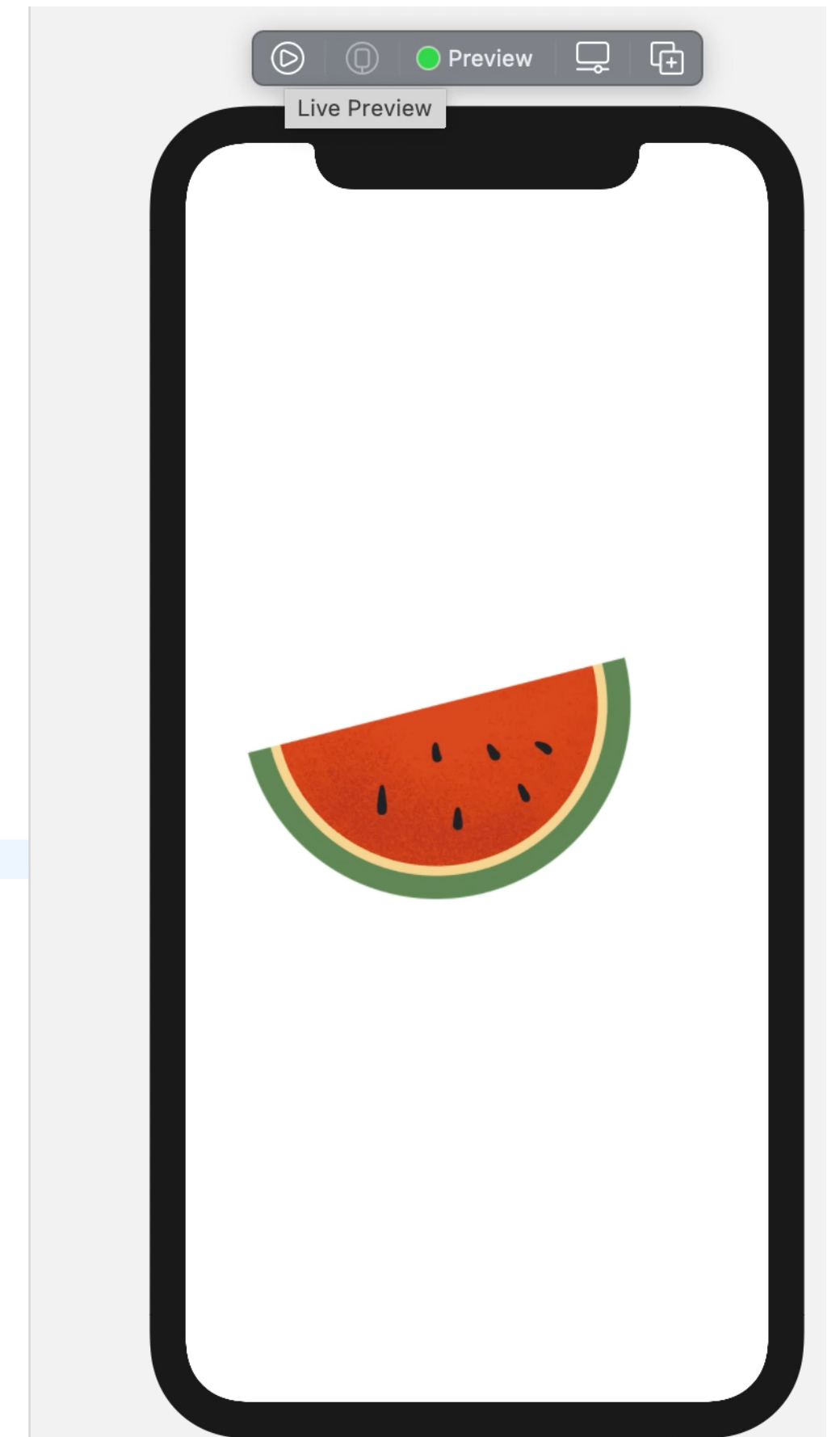


基于SwiftUI的动画

SwiftUI 里有两种动画——显式的和隐式的。

- 隐式动画通过 `.animation()` 这个modifier 指定。每当视图的某些参数改变时，SwiftUI 会动画化呈现旧值到新值的变化。这些参数包括尺寸，偏移量，颜色，缩放值，等等。
- 显式动画通过 `withAnimation { ... }` 闭包指定。只有那些依赖 `withAnimation` 闭包中的值变化的参数才会被动画化。

```
8 import SwiftUI
9
0 struct Animation: View {
1     @State private var half = false
2     @State private var dim = false
3
4     var body: some View {
5         VStack {
6             Image("watermelon")
7                 .scaledToFit()
8                 .scaleEffect(half ? 0.5 : 1.5)
9                 .opacity(dim ? 0.2 : 1.0)
0                 .animation(
1                     .easeInOut(duration: 1.0))
2                 .onTapGesture {
3                     self.dim.toggle()
4                     self.half.toggle()
5                 }
6         }
7     }
8
9 struct Animation_Previews:
0 PreviewProvider {
1     static var previews: some View {
2         Animation()
3     }
4 }
```



iOS基于SwiftUI的动画设计与交互

基于SwiftUI的动画

```
struct Animation: View {  
    @State private var half = false  
    @State private var dim = false
```

← @State装饰的属性改变时, body计算属性重新计算

```
    var body: some View {  
        VStack {
```

```
            Image("watermelon")
```

动画效果.easeIn(), .easeOut(), .linear()等等

缩放操作

```
                .scaledToFit()
```

```
                .scaleEffect(half ? 0.5 : 1.5)
```

变化时动画的选择

```
                .opacity(dim ? 0.2 : 1.0)
```

```
                .animation(.easeInOut(duration: 1.0))
```

持续1s

```
                .onTapGesture {
```

```
                    self.dim.toggle()
```

```
                    self.half.toggle()
```

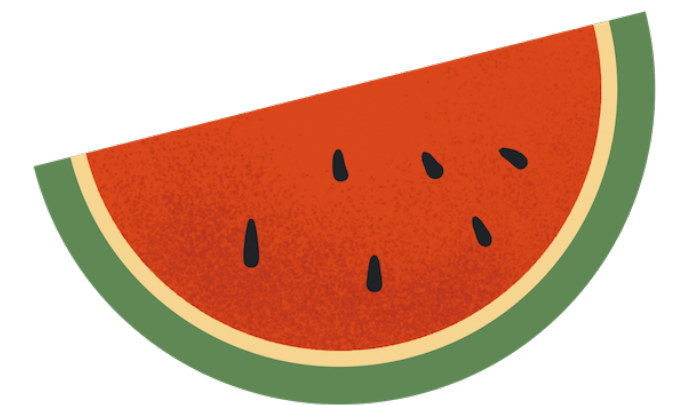
```
                }
```

当检测到点击动作时, 对属性的bool值取非

透明度选择

```
        }
```

```
    }
```



iOS基于SwiftUI的动画设计与交互

基于SwiftUI的动画

思考题🤔：如何达到之前隐式动画一样的效果？

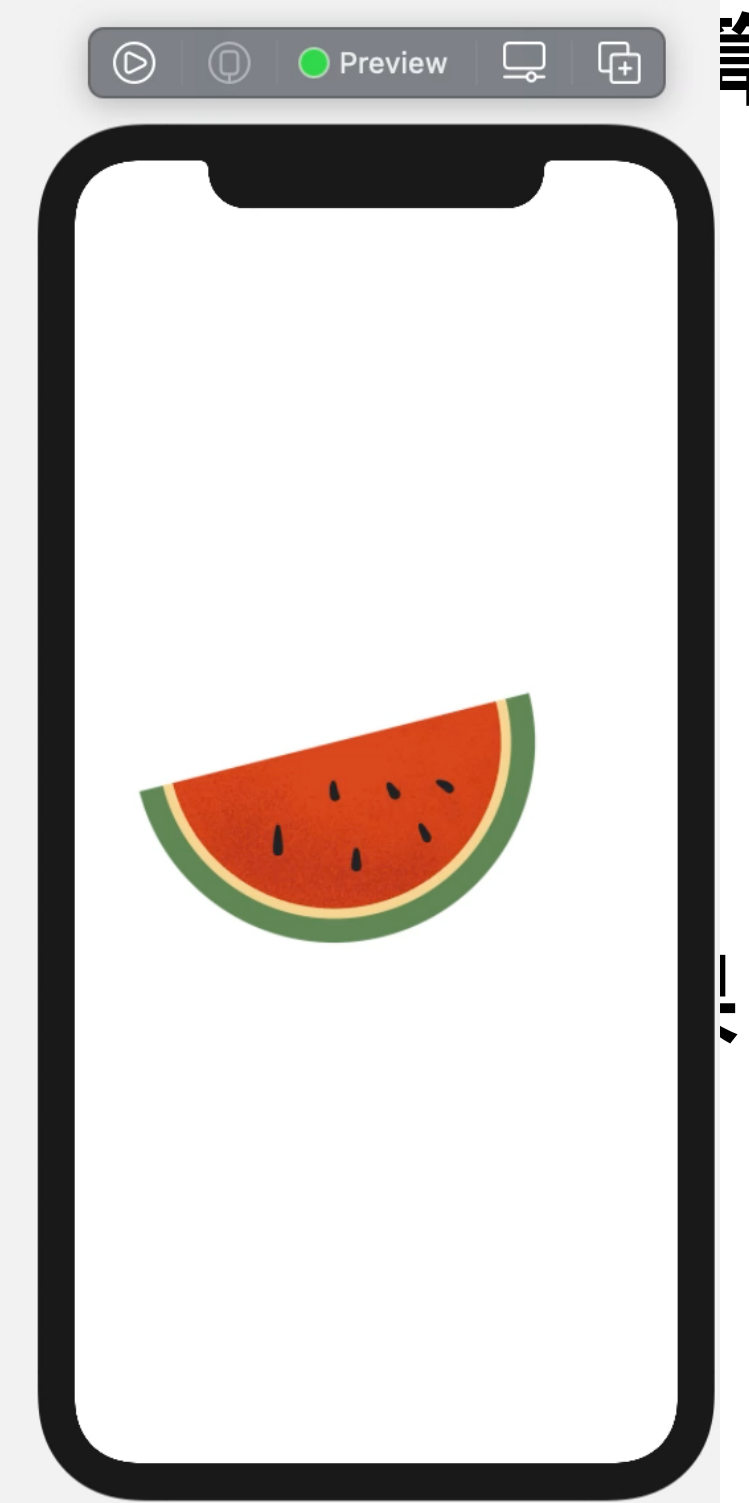
```
struct Animation: View {  
    @State private var half = false  
    @State private var dim = false  
  
    var body: some View {  
        VStack {  
            Image("watermelon")  
                .scaledToFit()  
                .scaleEffect(half ? 0.5 : 1.5)  
                .opacity(dim ? 0.2 : 1.0)  
                .onTapGesture {  
                    self.half.toggle()  
                    withAnimation(.easeInOut(duration: 1.0)) {  
                        self.dim.toggle()  
                    }  
                }  
        }  
    }  
}
```

缩放操作

透明度选择

动画与CoreData基础

```
8 import SwiftUI  
9  
10 struct Animation: View {  
11     @State private var half = false  
12     @State private var dim = false  
13  
14     var body: some View {  
15         VStack {  
16             Image("watermelon")  
17                 .scaledToFit()  
18                 .scaleEffect(half ? 0.5 : 1.5)  
19                 .opacity(dim ? 0.2 : 1.0)  
20                 .onTapGesture {  
21                     self.half.toggle()  
22                     withAnimation(  
23                         .easeInOut  
24                         (duration: 1.0))  
25                     {  
26                         self.dim.toggle()  
27                     }  
28                 }  
29         }  
30     }  
31  
32 struct Animation_Previews:  
33     PreviewProvider {  
34         static var previews: some View {  
35             Animation()  
36         }  
37     }  
38 }
```



章

iOS基于SwiftUI的动画设计与交互

基于SwiftUI的动画

思考题🤔：如何达到之前隐式动画一样的效果？

```
struct Animation: View {
    @State private var half = false
    @State private var dim = false

    var body: some View {
        VStack {
            Image("watermelon")
                .scaledToFit()
                .scaleEffect(half ? 0.5 : 1.5)
                .opacity(dim ? 0.2 : 1.0)
                .onTapGesture {
                    self.half.toggle()
                    withAnimation(.easeInOut(duration: 1.0)) {
                        self.dim.toggle()
                    }
                }
        }
    }
}
```

提示

iOS基于SwiftUI的动画设计与交互

基于SwiftUI的动画

思考题🤔：如何达到之前隐式动画一样的效果？

```
struct Animation: View {
    @State private var half = false
    @State private var dim = false

    var body: some View {
        VStack {
            Image("watermelon")
                .scaledToFit()
                .scaleEffect(half ? 0.5 : 1.5)
                .opacity(dim ? 0.2 : 1.0)
                .onTapGesture {
                    withAnimation(.easeInOut(duration: 1.0)) {
                        self.dim.toggle()
                        self.half.toggle()
                    }
                }
        }
    }
}
```

答案如左所示



课程目录



1

iOS基于SwiftUI的动画设计与交互

2

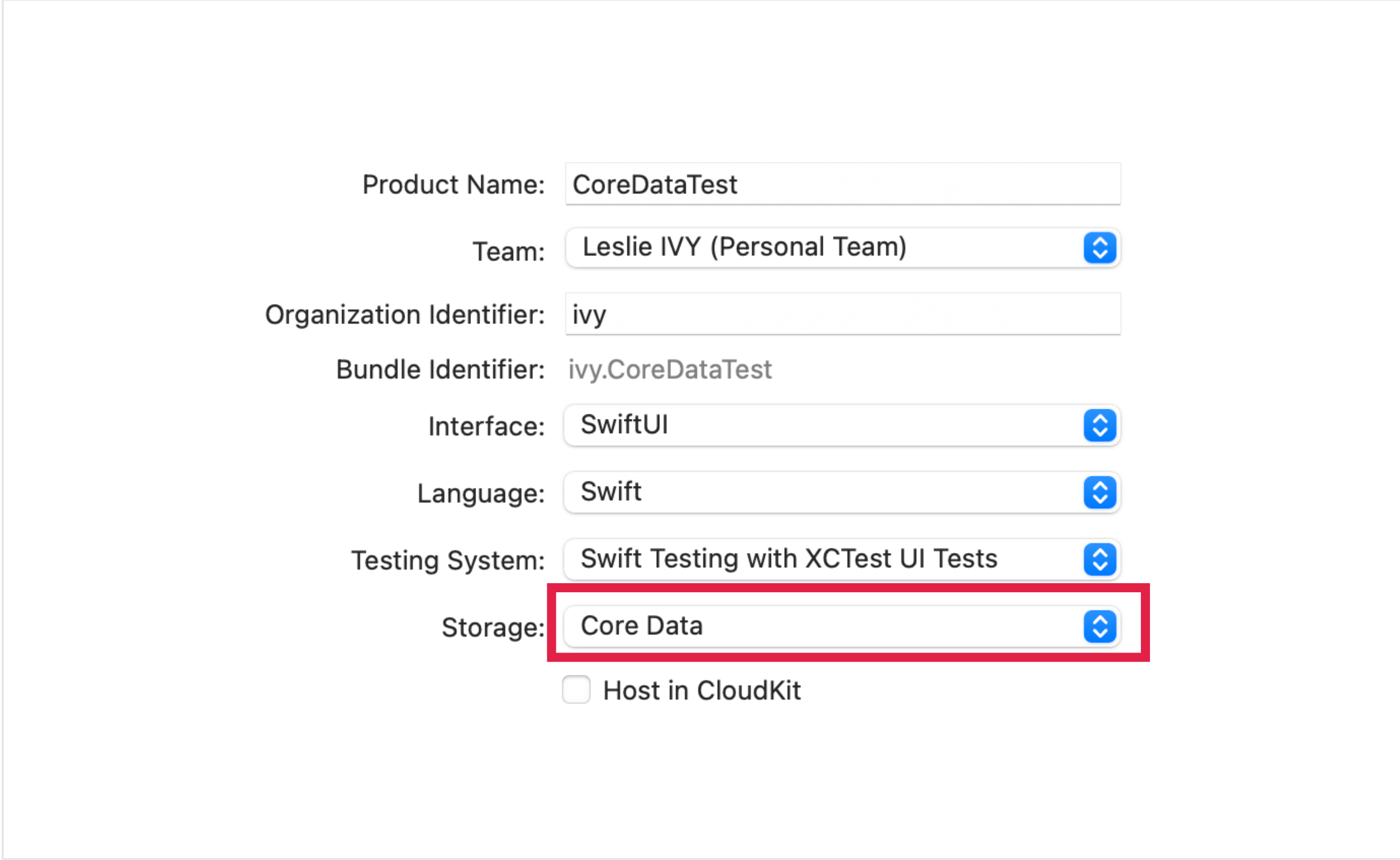
CoreData基础

CoreData基础

一个简单的例子

- 在使用Xcode新建项目时，点击 **Use CoreData**即可在代码中使用 **Core Data**相关的API，也可以后期手动添加

Choose options for your new project:



The screenshot shows the 'Choose options for your new project' dialog in Xcode. The options are as follows:

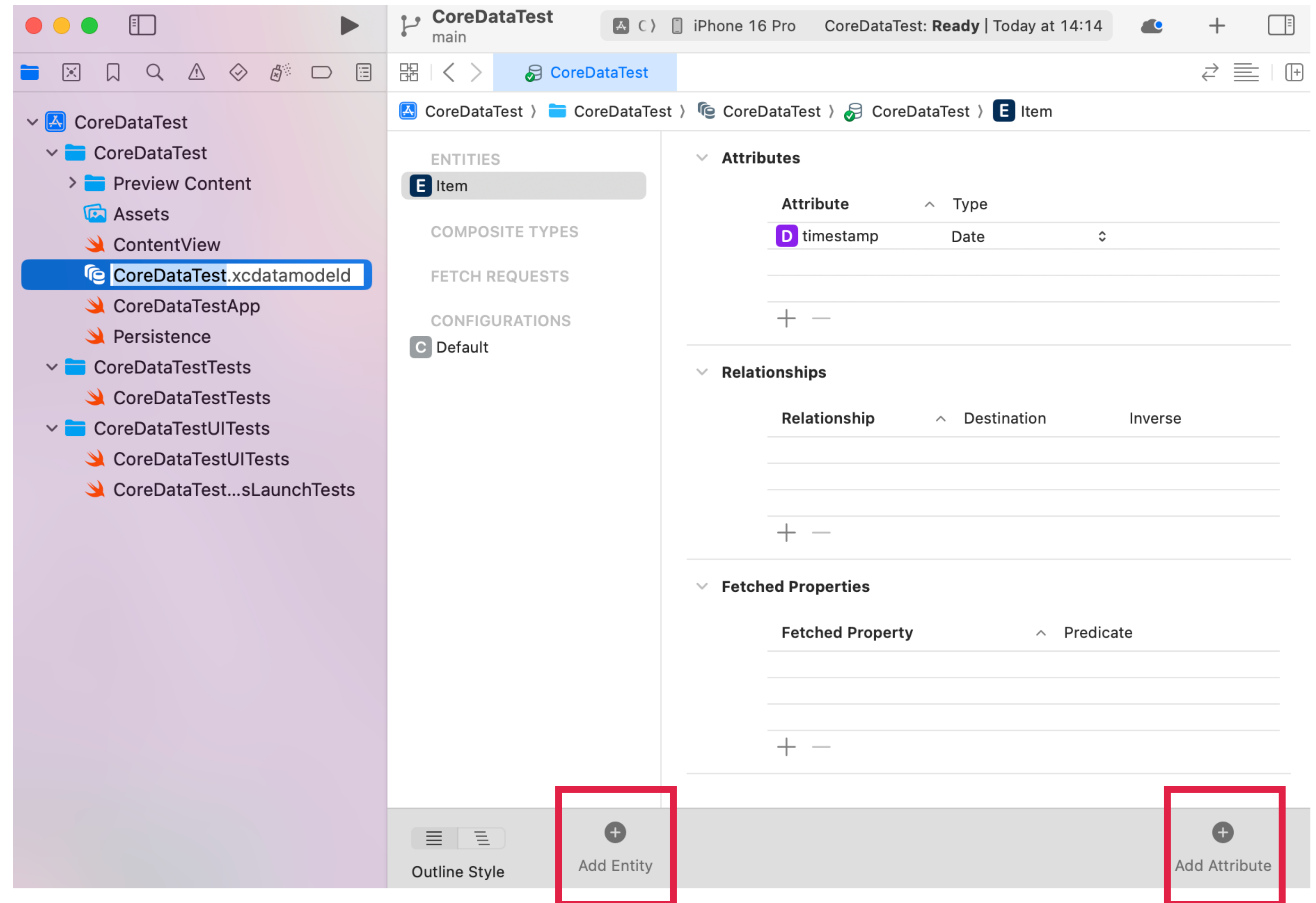
Option	Value
Product Name:	CoreDataTest
Team:	Leslie IVY (Personal Team)
Organization Identifier:	ivy
Bundle Identifier:	ivy.CoreDataTest
Interface:	SwiftUI
Language:	Swift
Testing System:	Swift Testing with XCTest UI Tests
Storage:	Core Data
Host in CloudKit:	☐

At the bottom, there are three buttons: 'Cancel', 'Previous', and 'Next'.

CoreData基础

一个简单的例子

- 在导航栏中会多出一个文件名为*.xcdatamodeld，即使用Core Data 的数据库模型文件
- 点击xcdatamodeld文件，我们可以看到这个界面，这个界面是我们操作Core Data的可视化界面
- 点击Add Attribute或者在Attributes项目下点击+按钮可以添加该实体的属性



CoreData基础

一个简单的例子

- 添加属性之后需要指定该属性的名字和类型
- 属性类型将会与Swift中的类型一一对应。实际上，我们可以先定义CoreData数据，再由XCode自动生成Swift下对应的数据类型和成员变量

▼ Attributes

Attribute ^	Type
 text	String

CoreData基础

一个简单的例子

Entity-codgen:

- Manual

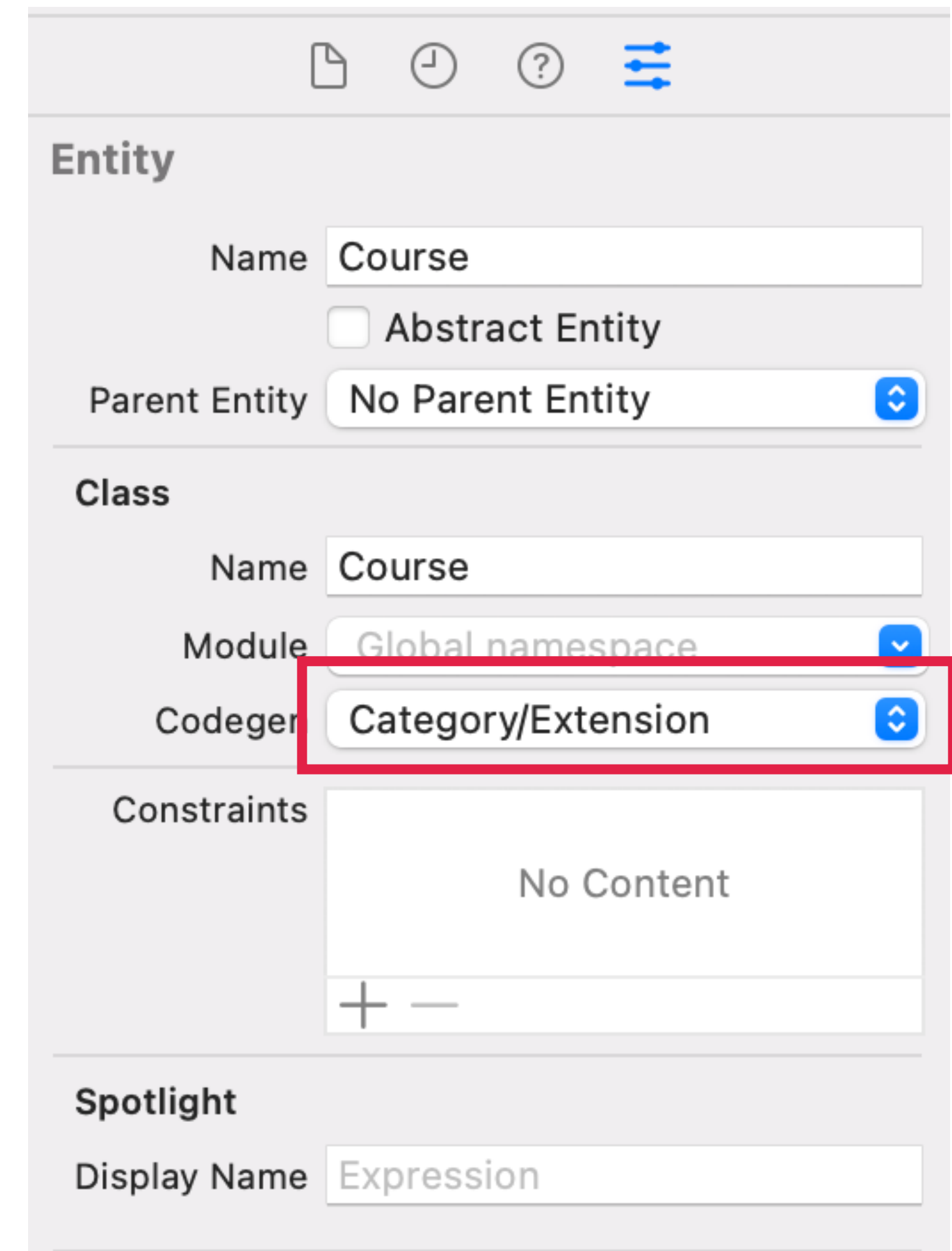
- 每次对数据库的操作都必须使用 `value(forKey:)` 这一套API

- Class Definition

- XCode将为你的Entity生成Class，但是该类不会在导航栏中出现，即你无法修改该类的代码

- Category

- 生成该类的扩展，你可以自己写该扩展的内容
- 一般情况下选择Category作为代码生成工具

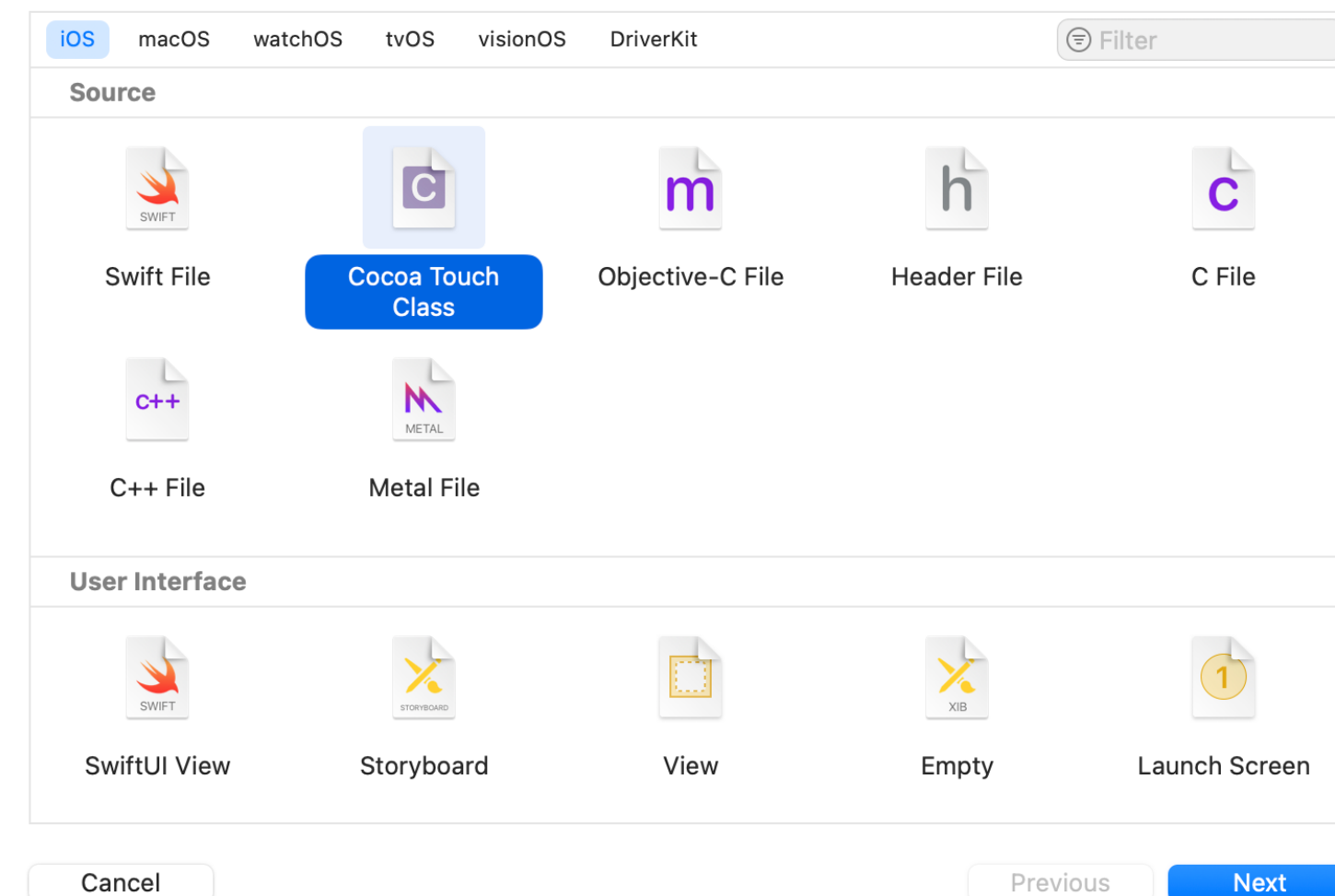


CoreData基础

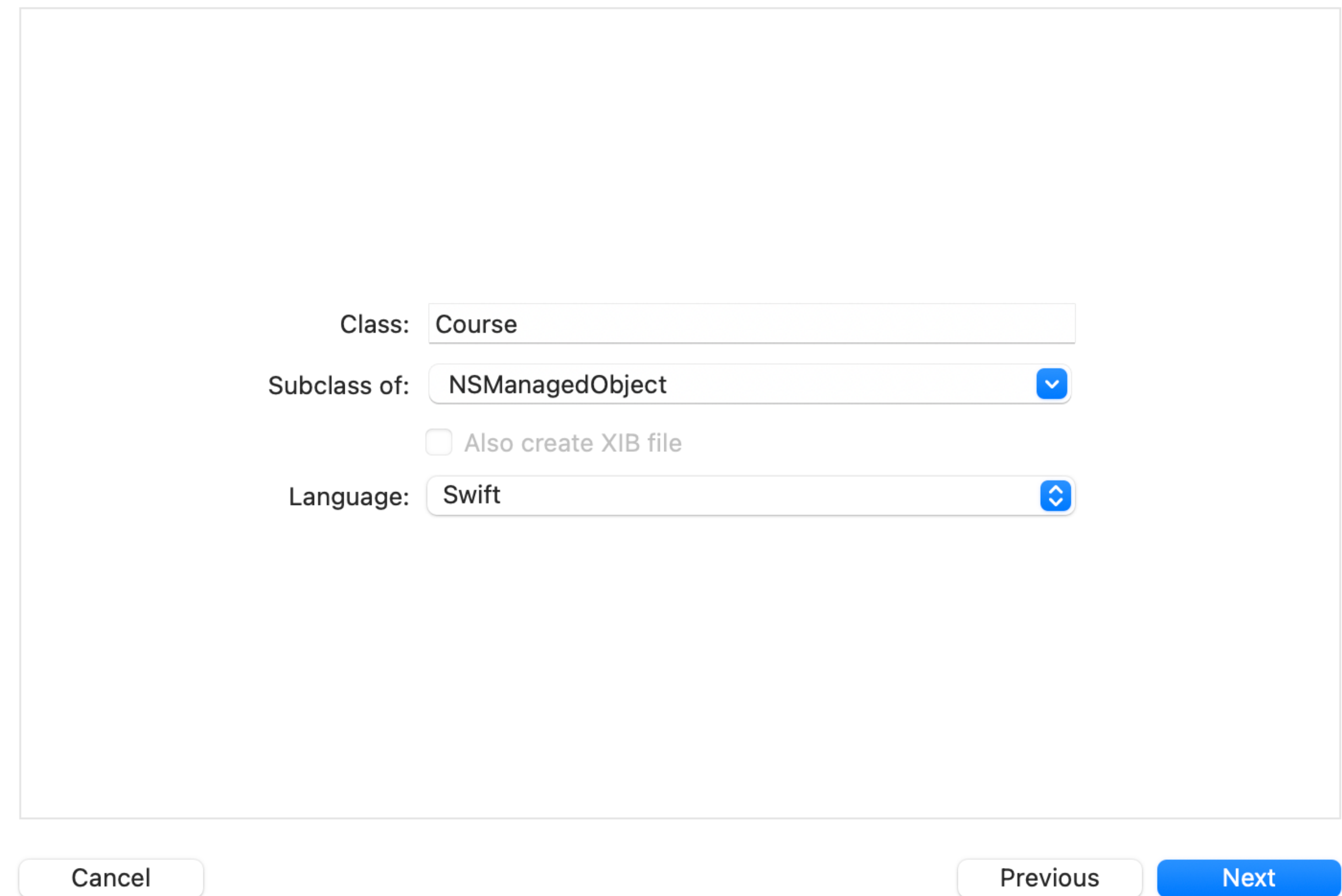
一个简单的例子

- 选择Category之后，新建一个Cocoa Touch Class，选择 NSObject作为它的父类，命名需要与实体的名字一样。

Choose a template for your new file:



Choose options for your new file:



CoreData基础

一个简单的例子

- 在生成的文件中加入Core Data库，我们就可以自己编写代码对实体进行CRUD操作了。
- 我们需要在Swift中定义一个空的实体。这是因为Core Data是自动实现了实体的Extension来完成对属性的处理。

```
import UIKit
import CoreData

@objc(Course)
class Course: NSObject {

}
```

一个简单的例子

- 现在我们可以直接获取实体，并直接对实体的属性进行操作。直接使用类的语法对属性进行赋值/取值。

```
let newCourse = Course(context: viewContext)
    newCourse.id = UUID()
    newCourse.teacher = "IVY"
    newCourse.time = Date()
```

一个简单的例子

- 在数据库中进行删除和插入都十分简单，获取 `NSManagedObjectContext`后，在 `context`中使用 `insert`或者 `delete`即可（`insert`不需要手动调用）
- 如果在relationship中设置好了删除时的方法是 `Nullify`，那么我们就不需要做其他的额外操作，同时应避免强指针指向删除的对象

```
open func insert(_ object: NSManagedObject)
open func delete(_ object: NSManagedObject)
```

CoreData基础



一个简单的例子

- 进行数据库查询时，我们只需要新创一个request，request中有三个比较重要的属性需要设置：
 - Entity(定义想要查询的对象实体)
 - Predicate
 - sortDescriptors

```
let fetchRequest: NSFetchedRequest<Course> =  
    Course.fetchRequest()  
    fetchRequest.predicate =  
    NSPredicate(format: "teacher == %@", t)  
    fetchRequest.fetchLimit = 5  
    let sortDescriptor =  
    NSSortDescriptor(key: "time", ascending:  
true) // 按time升序排序  
    fetchRequest.sortDescriptors =  
    [sortDescriptor]  
  
do {  
    let result = try  
viewContext.fetch(fetchRequest)  
    searchResult = result.first  
} catch {  
    print("查询失败: \  
(error.localizedDescription)")  
}
```

一个简单的例子

predicate

- 查询要求在这里提供，如右边代码对应的查询要求是：查询tweeter的screenName属性为"Guofeng Zhang"的实体以及查询text包含字符串"Haha"的实体(大小写敏感)
- 可以使用NSCompoundPredicate来连接多个predicate，如右边代码对应的查询使用and操作符要求同时满足两个条件

```
let predicate1 =  
    NSPredicate(format: "teacher = %@",  
                  "Guofeng Zhang")  
  
let predicate2 =  
    NSPredicate(format: "text contains[c] %@",  
                  "Haha")  
  
let predicate = NSCompoundPredicate(  
    andPredicateWithSubpredicates: [predicate1,  
                                     predicate2])
```

一个简单的例子

sortDescriptor

- 即查询结果返回时应当排列成什么样的顺序，右图展示的 **sortDescriptor** 作用为:将该实体的**screenName**属性值使用**selector**获取的方法进行比较，最终结果按照升序值返回
- 传入的多个**sortDescriptor**即使用这些排序方法进行多关键字排序

```
let sortDescriptor = NSSortDescriptor(  
    key: "screenName",  
    ascending: true,  
    selector: #selector(  
        NSString.localizedStandardCompare(_:))  
    )
```

关于作业



AR作业（下周之前发布细则）

Andriod作业（由陶老师进行布置）

大作业（下周之前发布细则）