

移动平台开发技术

iOS开发中的数据存储

章国锋 2025春夏学期

基础概念

CoreData使用

LeanCloud使用

基础概念

CoreData使用

LeanCloud使用

数据持久化存储

Persistent Data Storage



浙江大学
ZHEJIANG UNIVERSITY

如何保存数据？

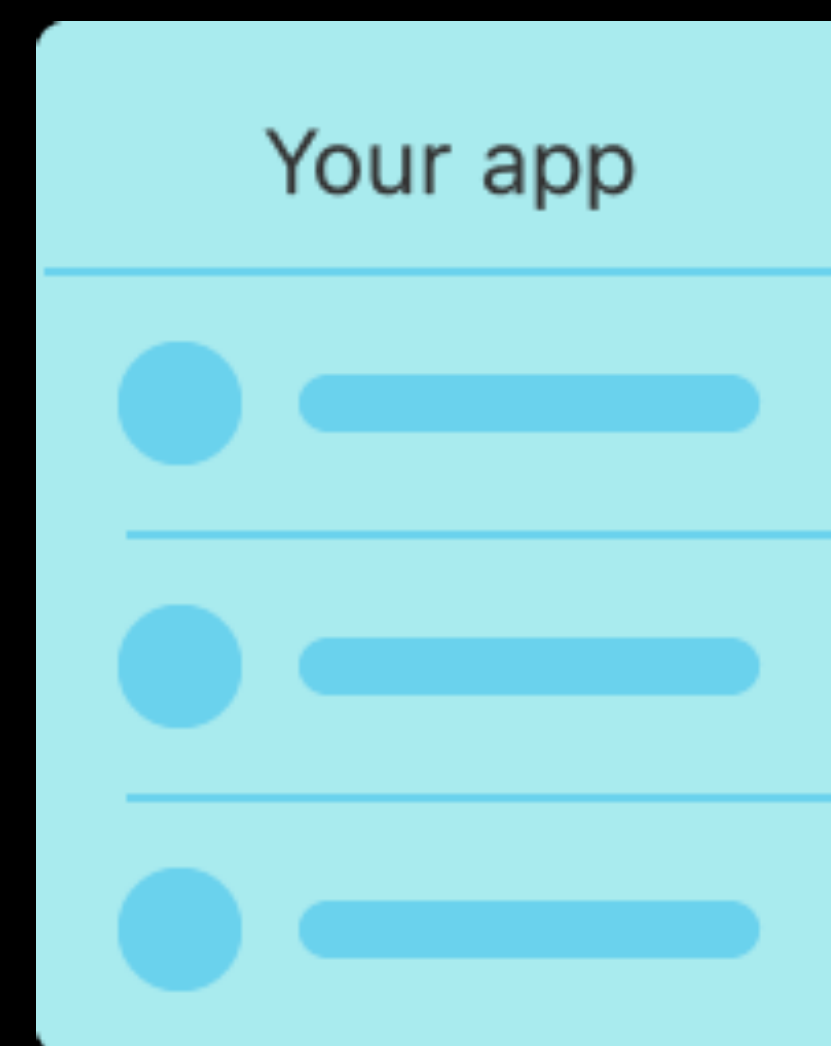
思考题

数据持久化存储

Persistent Data Storage



如何保存数据?



使用关系型数据库，如SQLite

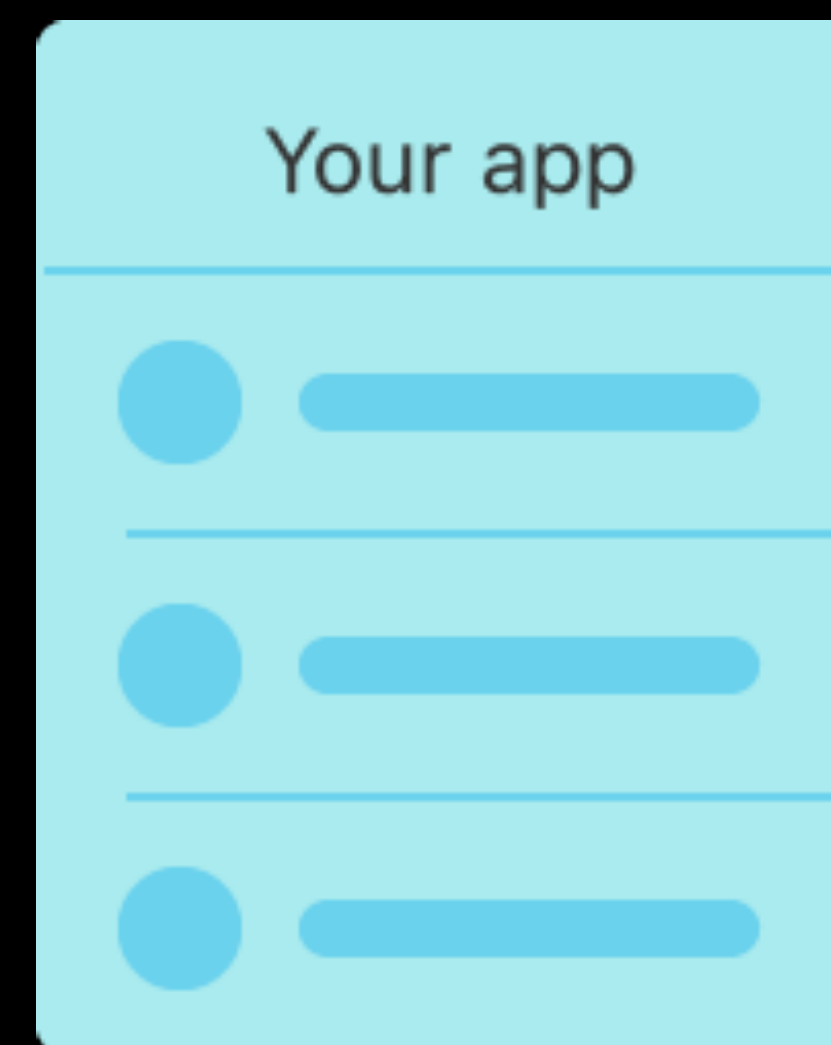
数据持久化存储

Persistent Data Storage



如何保存数据?

如何同步数据?



使用关系型数据库，如SQLite

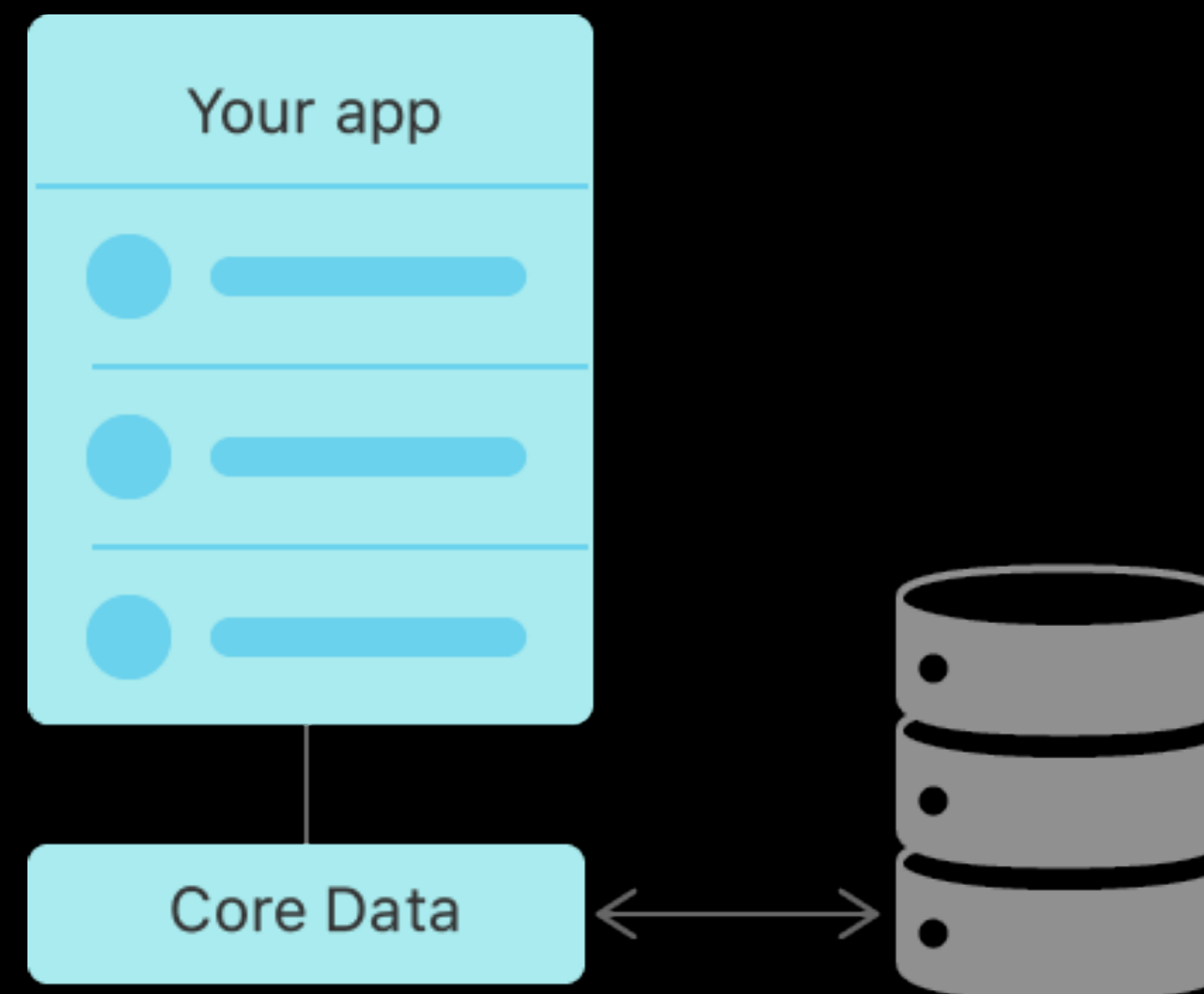
需要手写SQL语句；处理复杂逻辑；调试麻烦

数据持久化存储

Persistent Data Storage

如何保存数据?

如何同步数据?



使用Core Data管理

数据持久化存储

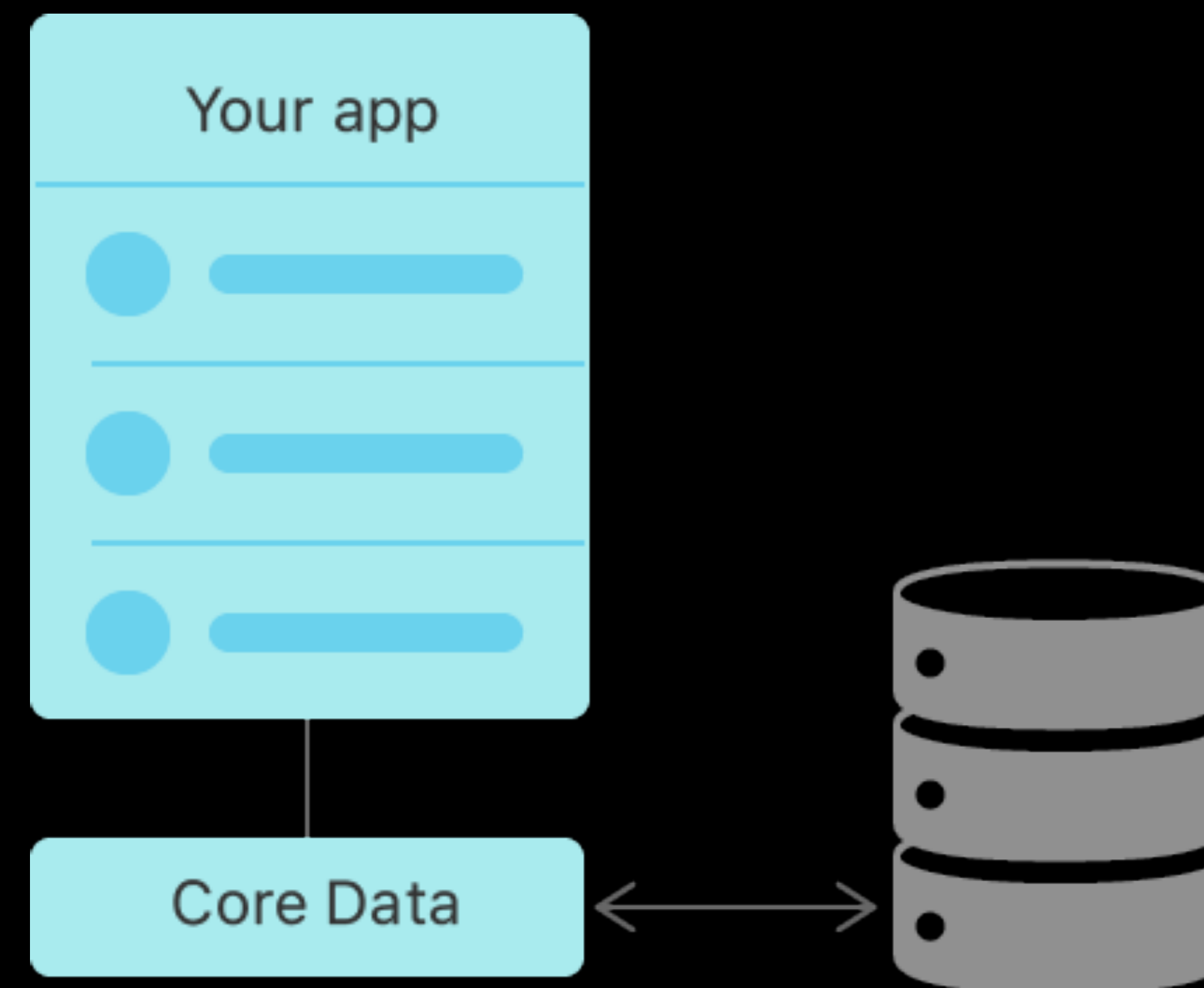
Persistent Data Storage

Object-relational mapping

对象关系映射，ORM

面向对象是从软件工程基本原则（如耦合、聚合、封装）的基础上发展起来的，而**关系数据库**则是从数学理论发展而来的，两套理论存在显著的区别。为了解决这个不匹配的现象，**对象关系映射**技术应运而生。

在软件工程中，**序列化**操作也是一种持久化存储、对象关系映射。



使用Core Data管理

对象数据库与关系数据库

ODBMS and RDBMS



浙江大学
ZHEJIANG UNIVERSITY



对象数据库与关系数据库

ODBMS and RDBMS



浙江大学
ZHEJIANG UNIVERSITY

相比常见的关系型数据库，面向对象数据库具有以下**优点**：

- 当一个工程规模较大，涉及到的数据之间的关系会变得十分复杂，这个时候关系数据库在管理复杂数据时显得笨重，而通过存储对象数据可以**大大简化关系的表达**。
- 应用软件操作的数据一般是用**面向对象的编程语言**如 C++, Java, Swift 等，这个时候使用关系型数据库需要使用大量用来转化数据表示和关系数据库元组的代码，这些代码十分冗繁且效率不高，而面向对象型数据库则十分**适合于这种场景**。



对象数据库与关系数据库

ODBMS and RDBMS



浙江大学
ZHEJIANG UNIVERSITY

面向对象数据库还具有以下特性：

- 在纯对象式数据库中，资料以对象的形式存储，这些对象只能由其所属的类中定义的方法来操作。
- 对象中可以有到其他对象的引用，于是应用程序可以以一种导航式的编程风格访问数据。
- 多数对象式数据库也提供了一些查询语言，允许用声明式编程访问对象。但是在对象查询语言以及查询和导航接口的集成领域，不同产品间区别很大。
- 访问数据可以更快，相较于关系型数据库可以不用进行表的联合操作，并且无需查询只需通过指针(Pointer)就可以直接获得对象。



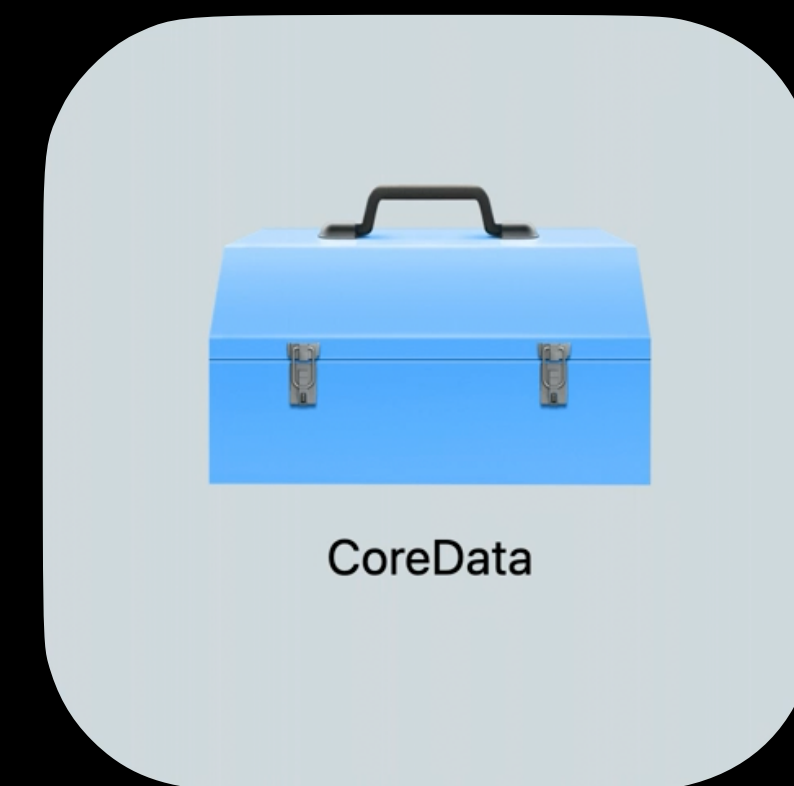
Core Data

Introduction to Core Data: Overview



Core Data: **Persist** or cache data on a single device, or sync data to multiple devices with CloudKit.

- Apple Developer Documentation



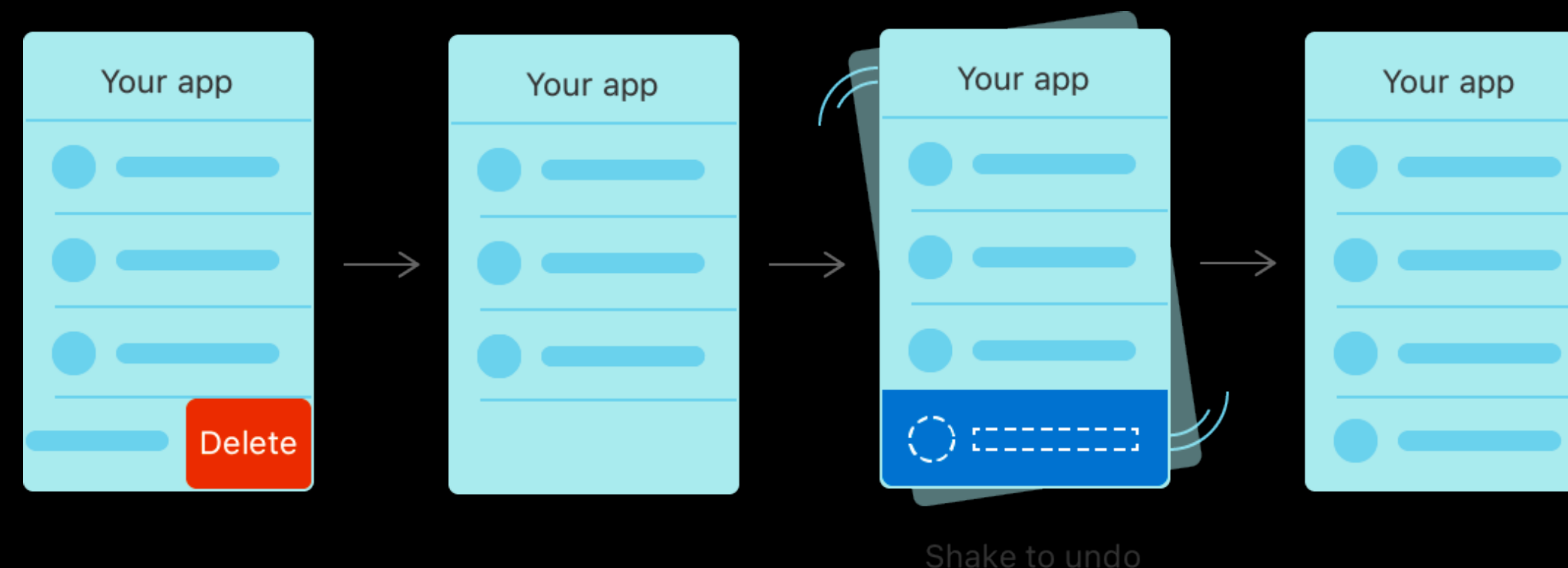
Core Data 其他高级特性

Other Advanced Core Data Features



便捷回滚操作

可以单独、分组或一次性地回滚这些更改，方便在APP中添加撤销和重做

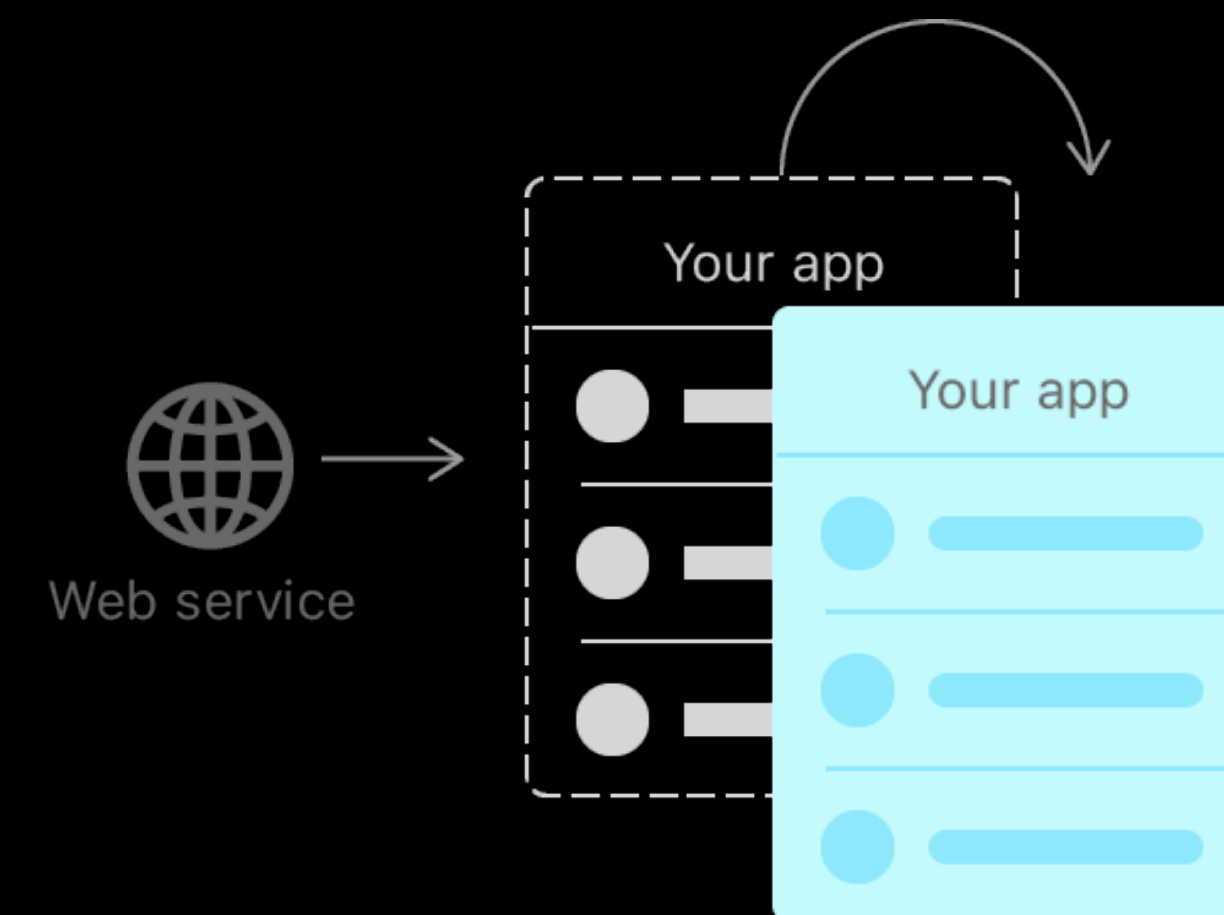


后台数据处理

在后台执行数据任务，可以避免阻塞UI、减少服务器访问

iCloud同步集成

可以结合CloudKit实现苹果生态圈的云端同步，而不需要耗费软件运营商的云空间



数据迁移工具

数据模型进行版本管理和便捷的迁移用户数据工具

Core Data 框架

Structure of Core Data



浙江大学
ZHEJIANG UNIVERSITY

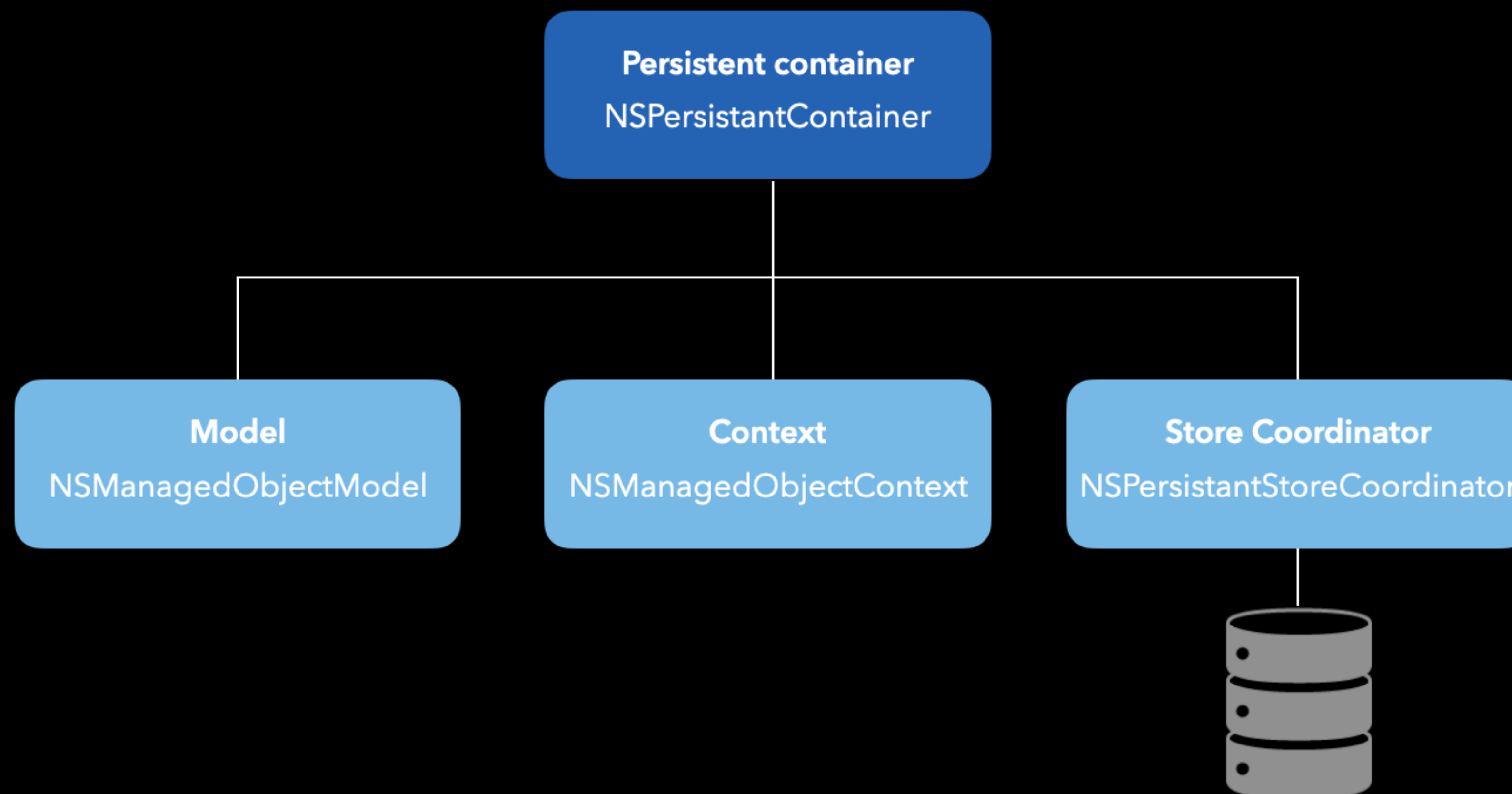
Core Data 的核心由以下几个
核心对象组成：

NSPersistentContainer

NSManagedObjectModel

NSManagedObjectContext

NSPersistentStoreCoordinator



Core Data 框架

Structure of Core Data



浙江大学
ZHEJIANG UNIVERSITY

NSPersistentContainer

NSManagedObjectModel

管理数据模型：当我们用**Core Data**时，我们需要一个用来存放数据模型的地方，数据模型文件就是我们要创建的文件类型。它的后缀是**.xcdatamodeld**。只要在项目中选 新建文件→Data Model 即可创建。

NSManagedObjectContext

NSPersistentStoreCoordinator



Core Data 框架

Structure of Core Data

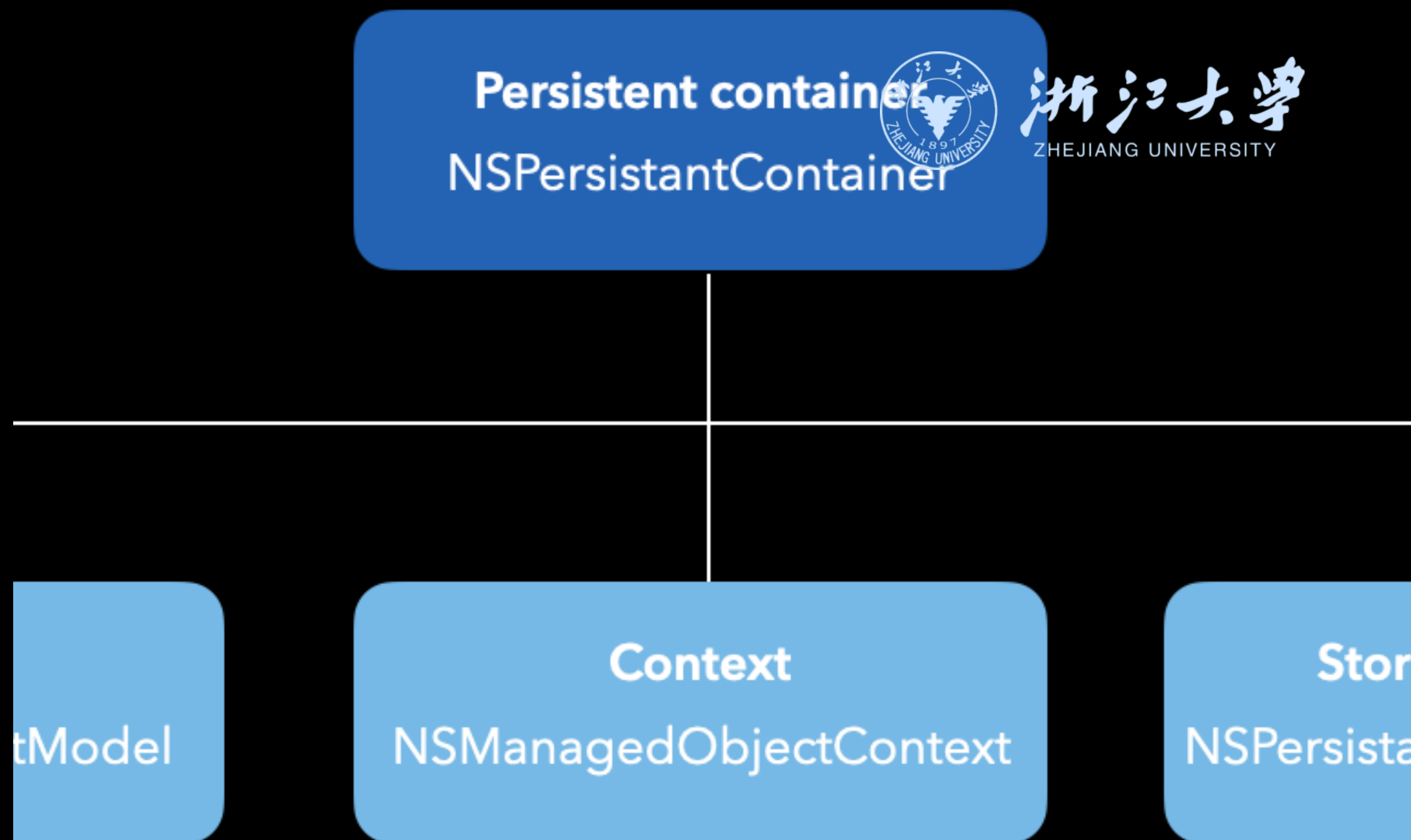
NSPersistentContainer

NSManagedObjectModel

NSManagedObjectContext

对象管理上下文: 负责管理模型的对象的集合这一部分, 涉及到数据的**CRUD操作API**

NSPersistentStoreCoordinator



Core Data 框架

Structure of Core Data



浙江大学
ZHEJIANG UNIVERSITY

NSPersistentContainer

NSManagedObjectModel

NSManagedObjectContext

NSPersistentStoreCoordinator

存储调度器: 负责将数据保存到磁盘，设置数据存储的名字，位置，存储方式等

Persistent container
NSPersistentContainer

Context
ManagedObjectContext

Store Coordinator
NSPersistentStoreCoordinator



Core Data 框架

Structure of Core Data



浙江大学
ZHEJIANG UNIVERSITY

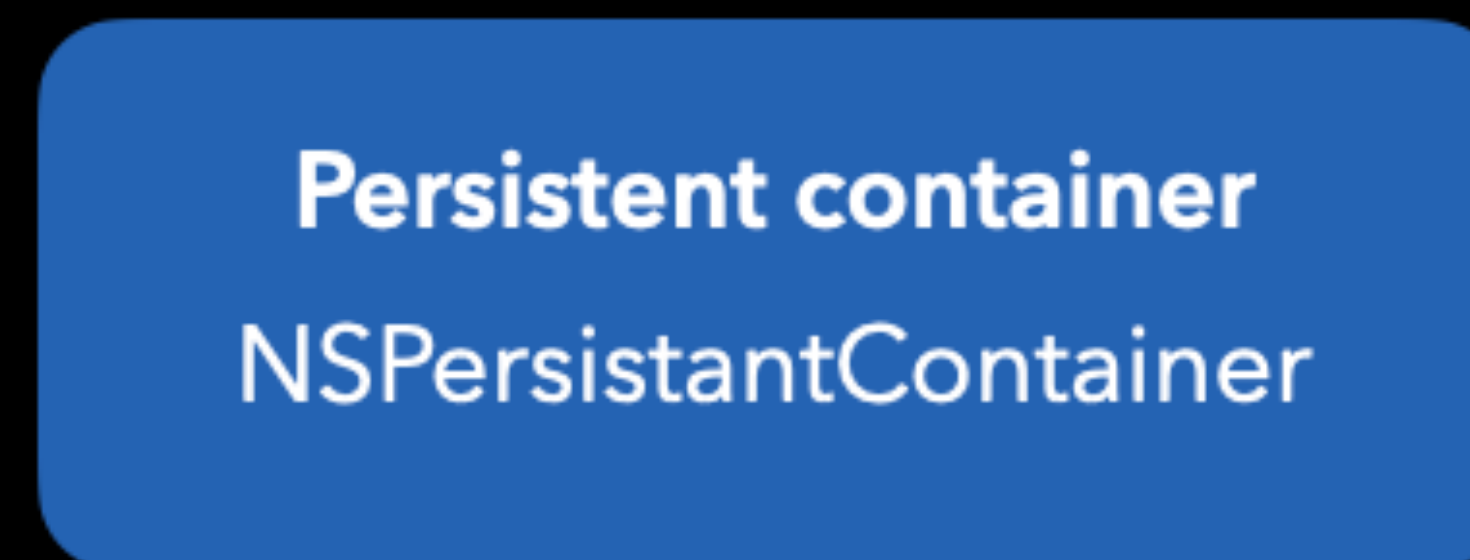
NSPersistentContainer

Core Data的仓库类，包含了下方的三个工具类

NSManagedObjectModel

NSManagedObjectContext

NSPersistentStoreCoordinator



数据模型中的几个概念

Introduction to Core Data: Some Concepts



Entity 实体

如果把数据模型文件 Data Model 比作数据库中的“Database 库”，那么 Entity 就相当于数据库的“Table 表格”

Attribute 属性

属性就相当于一个类中的成员。每个属性都有名字和数据类型，一个 Attribute 相当于数据库中的一个 Column

Relationship 关系

实体之间是具有一定联系的，这很类似关系数据库中的外键。通常根据使用场景我们有一对一、一对多、多对多关系。

Entity

属性名	类型
name	String
isbn	String
page	Integer32

基础概念

CoreData使用

LeanCloud使用

基础概念

CoreData使用

LeanCloud使用

Core Data 使用

Introduction to Core Data



浙江大学
ZHEJIANG UNIVERSITY

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:

Product Name:

Team:

Organization Identifier:

Bundle Identifier:

Interface:

Language:

☒ Use Core Data
☐ Host in CloudKit
☐ Include Tests

Cancel Previous Next

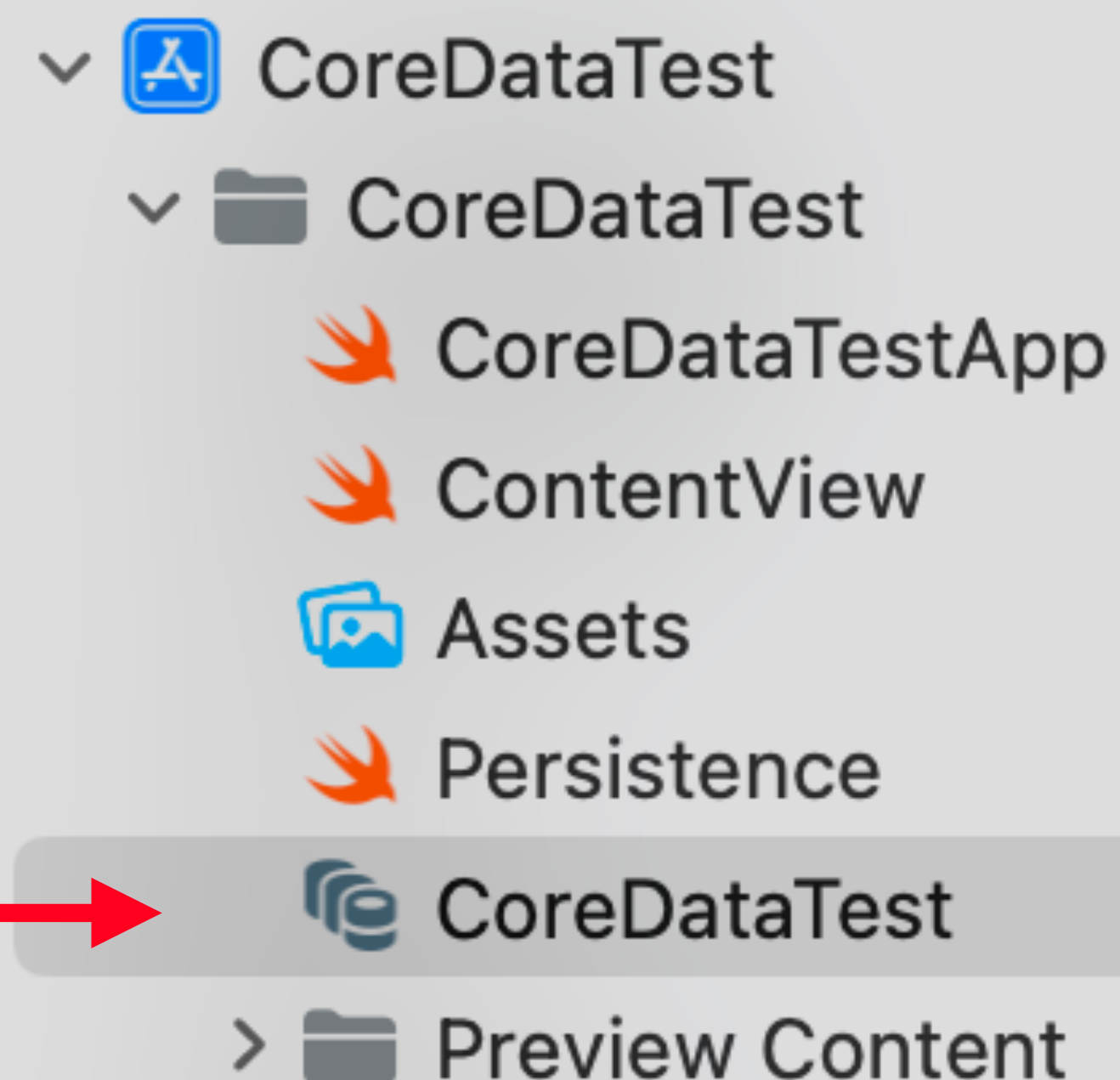
Core Data 使用

Introduction to Core Data



浙江大学
ZHEJIANG UNIVERSITY

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



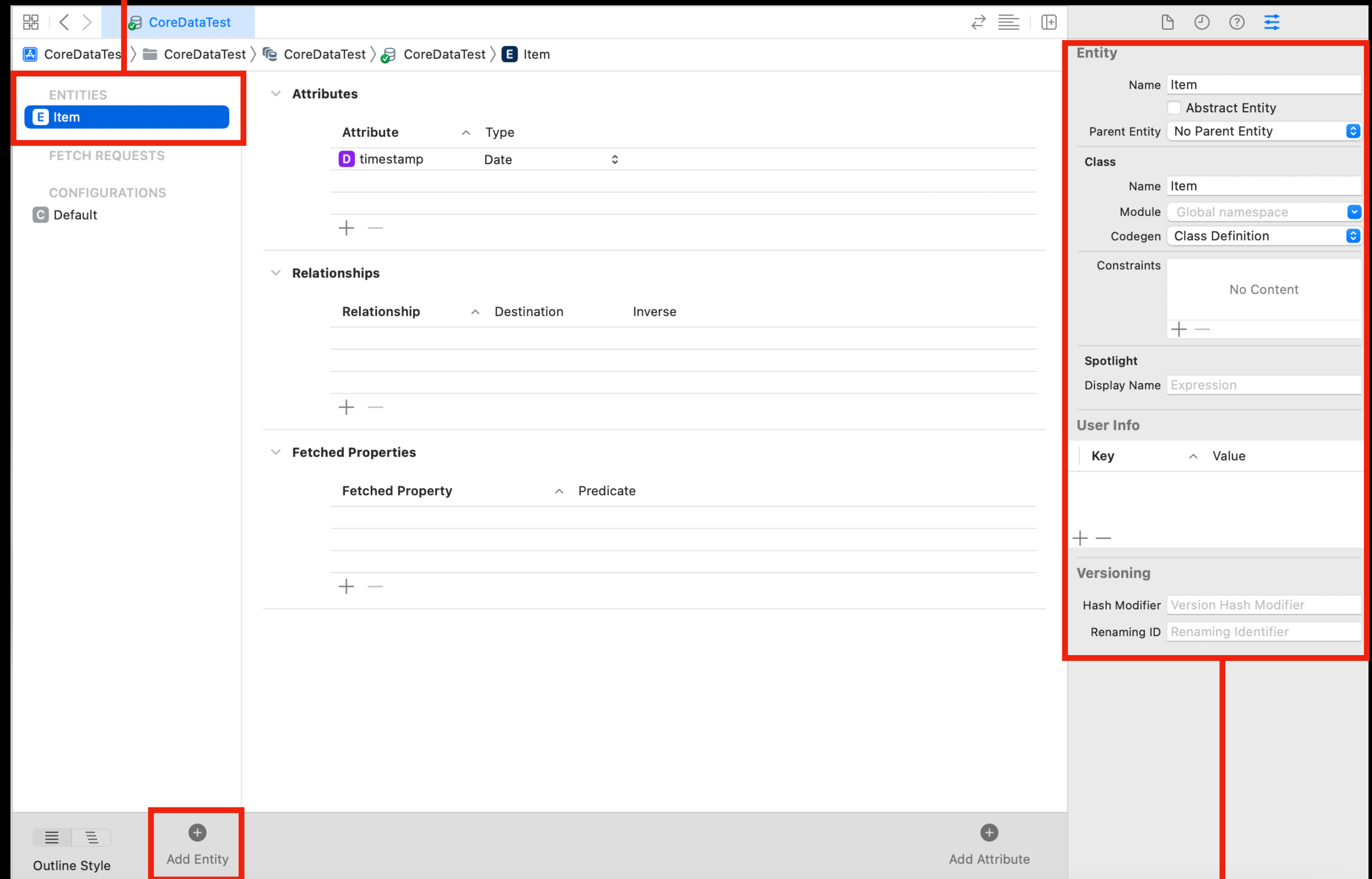
Core Data 使用

Introduction to Core Data



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

Entities list



Add Entity button

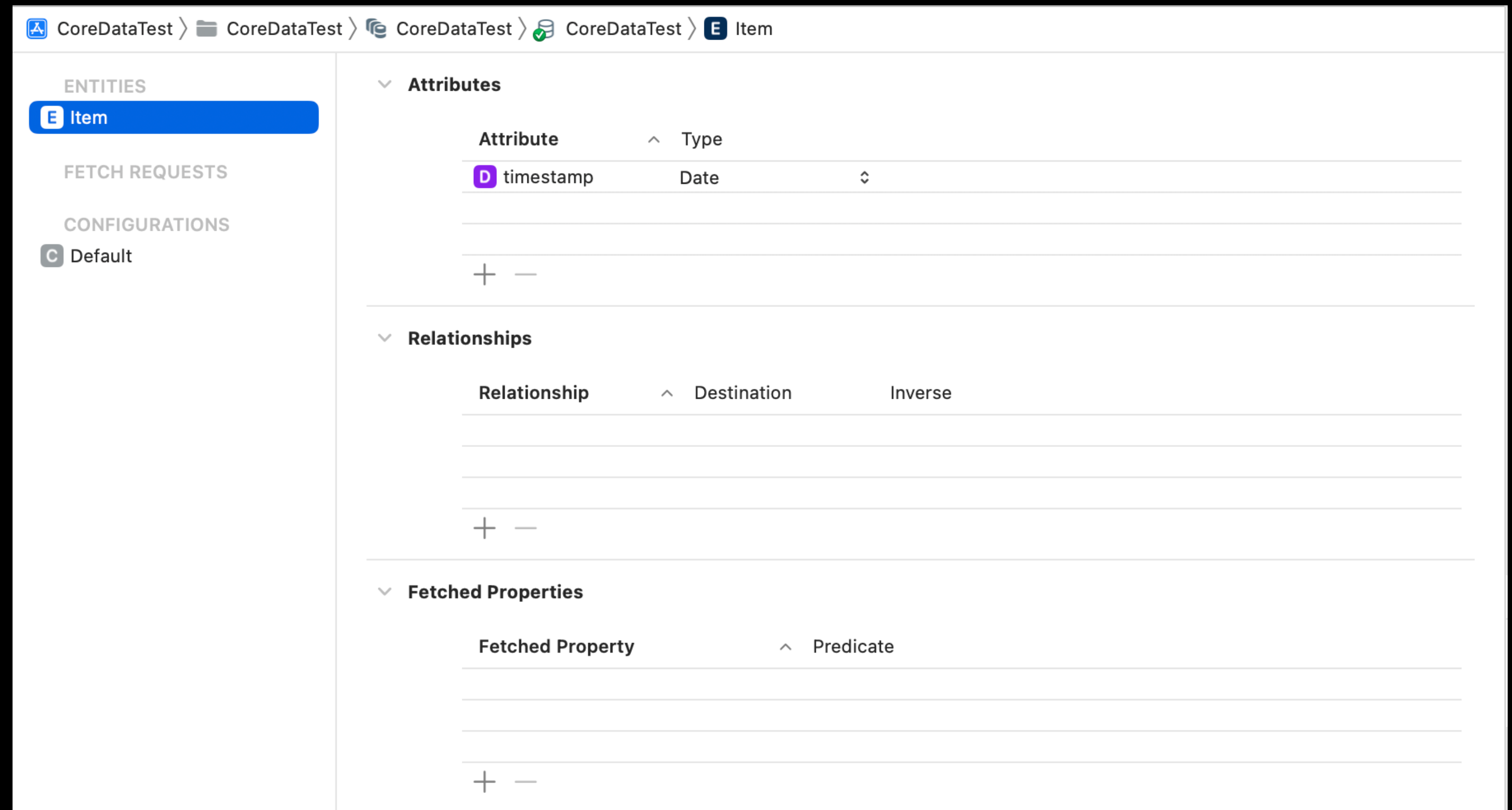
Data Model inspector

Core Data 使用

Introduction to Core Data



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

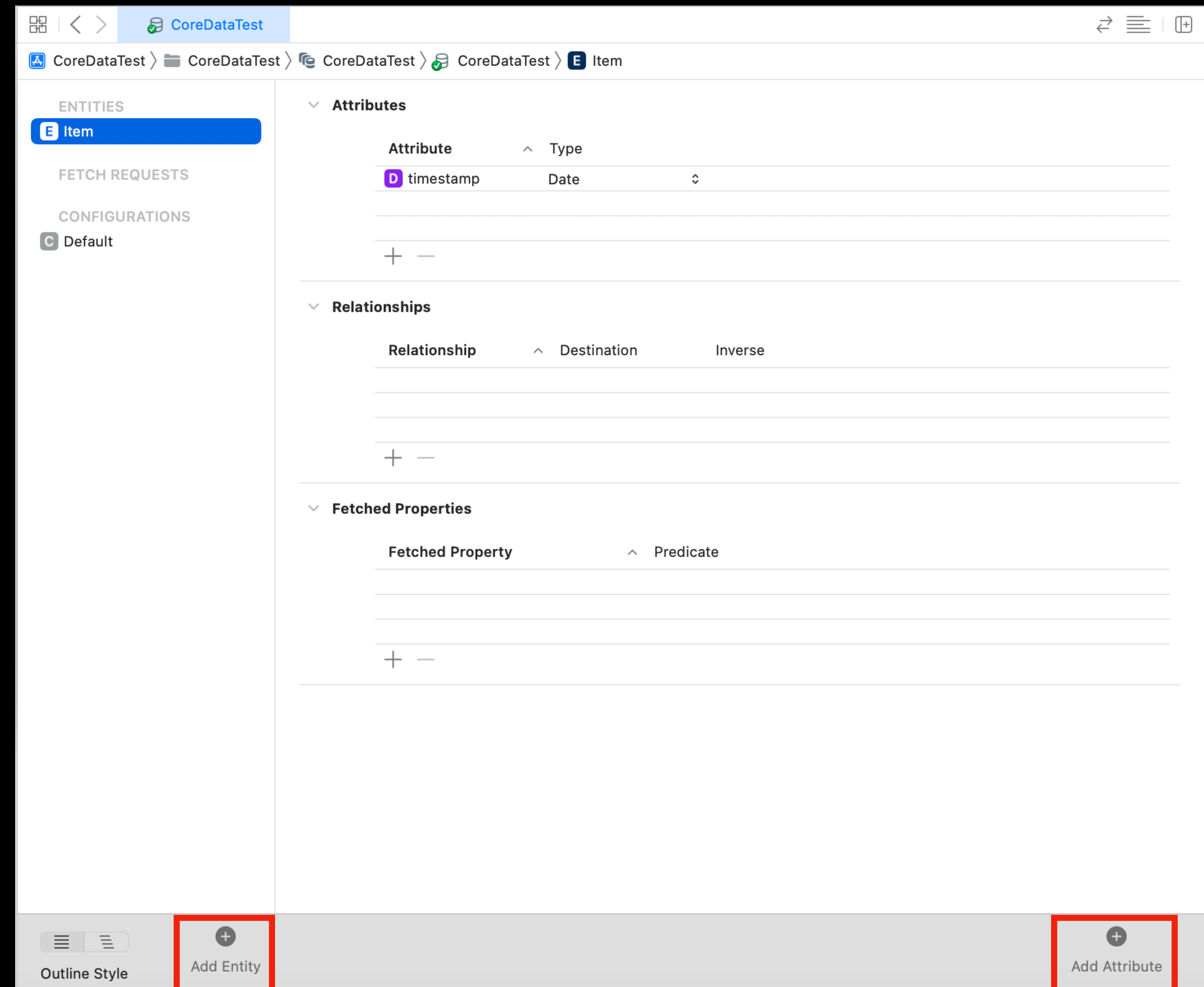


Core Data 使用：实体

Introduction to Core Data: Entity

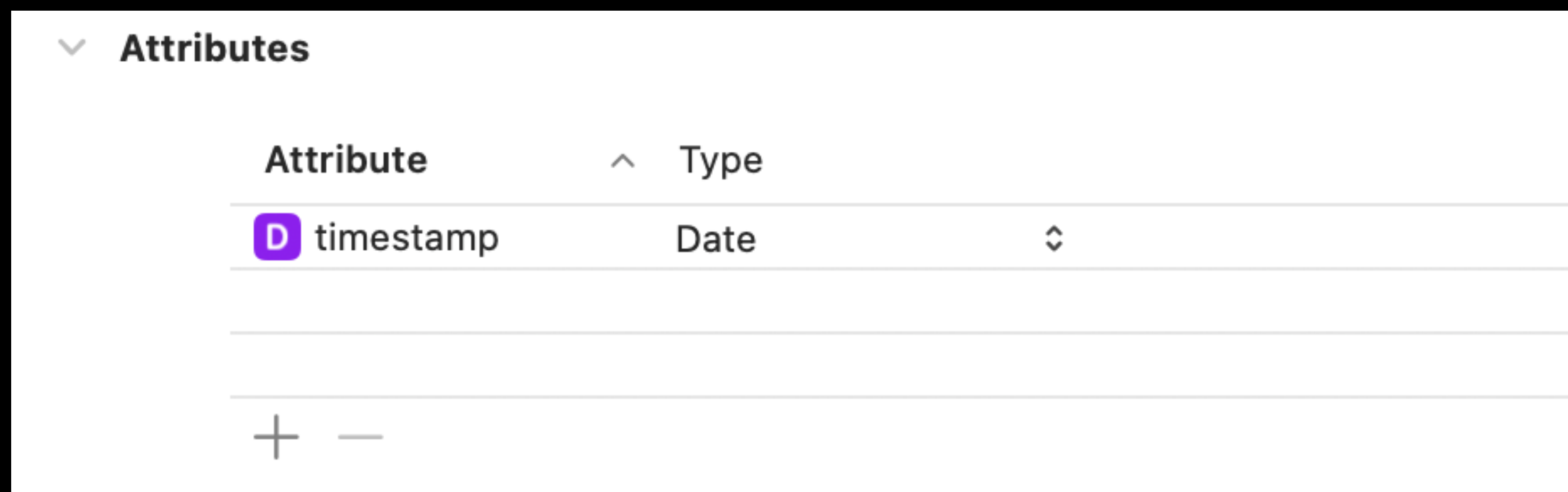


点击Add Entity可以添加数据库的实体
点击Add Attribute或者在Attributes
项目下点击+按钮可以添加该实体
的属性



Core Data 使用：属性

Introduction to Core Data: Attribute



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

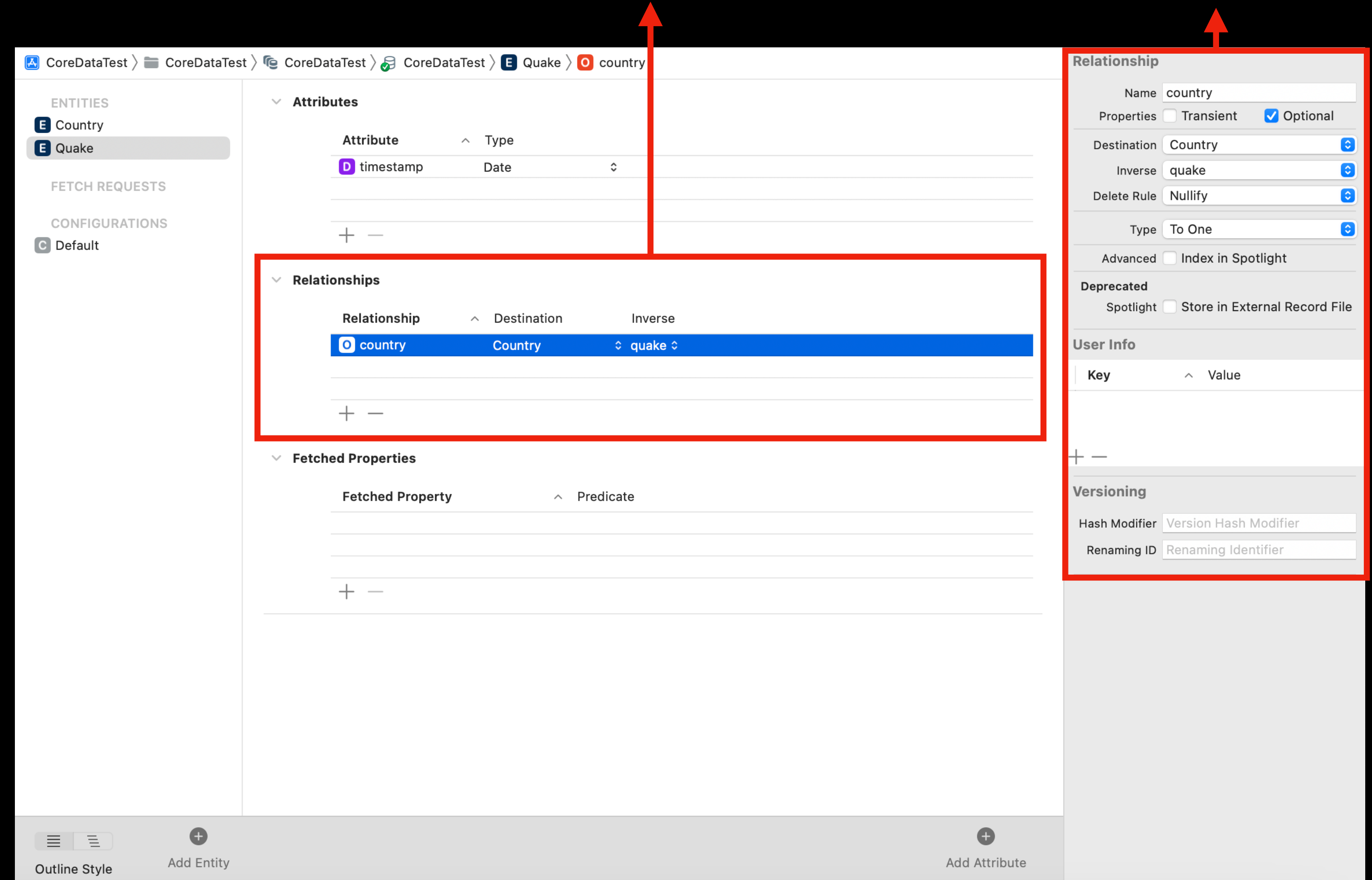
Core Data 使用：关系

Introduction to Core Data: Relationship



Relationships list

Data Model inspector



在默认的编辑器视图，可以在中间第二部分找到对应的关系列表。选中后右边会出现对应关系的检查器

Core Data 使用：关系

Introduction to Core Data: Relationship



点击新建的relationship，可以查看并编辑该relationship的属性。
右图展示了属性的两个重要部分：**Delete Rule** 和 **Type**

The screenshot shows the 'Relationship' inspector in Xcode. The relationship is named 'country'. It is optional (checked) and transient (unchecked). The destination entity is 'Country' and the inverse relationship is 'quake'. The delete rule is set to 'Nullify'. The type is 'To Many'. The arrangement is 'Ordered'. The count is 'Unbounded' for both minimum and maximum. The advanced section shows 'Index in Spotlight' is unchecked. The deprecated section shows 'Store in External Record File' is unchecked. The user info section is empty. The versioning section shows 'Version Hash Modifier' and 'Renaming Identifier' are both set to their default values.

当关联的实体被删除时，该指针指向nil

映射关系为一对多，即该关系允许关联多个实体

Core Data 使用：关联删除

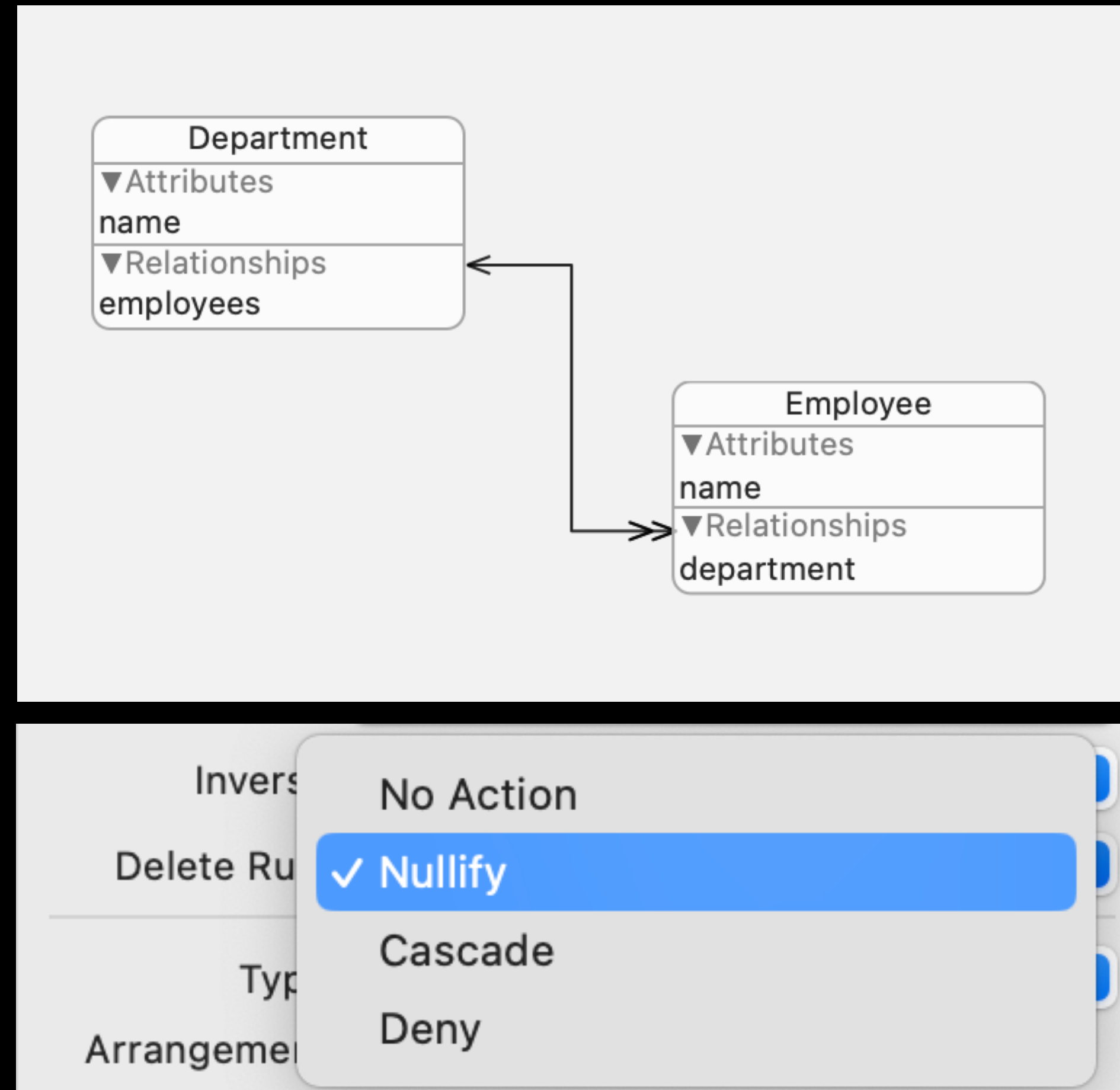
Introduction to Core Data



浙江大学
ZHEJIANG UNIVERSITY

Delete Rule:

- **Nullify**: 关联对象的指针设为null
- **Cascade**: 关联者一起被删除
- **Deny**: 除非关联对象已经被删除, 否则将无法被删除



Core Data 使用

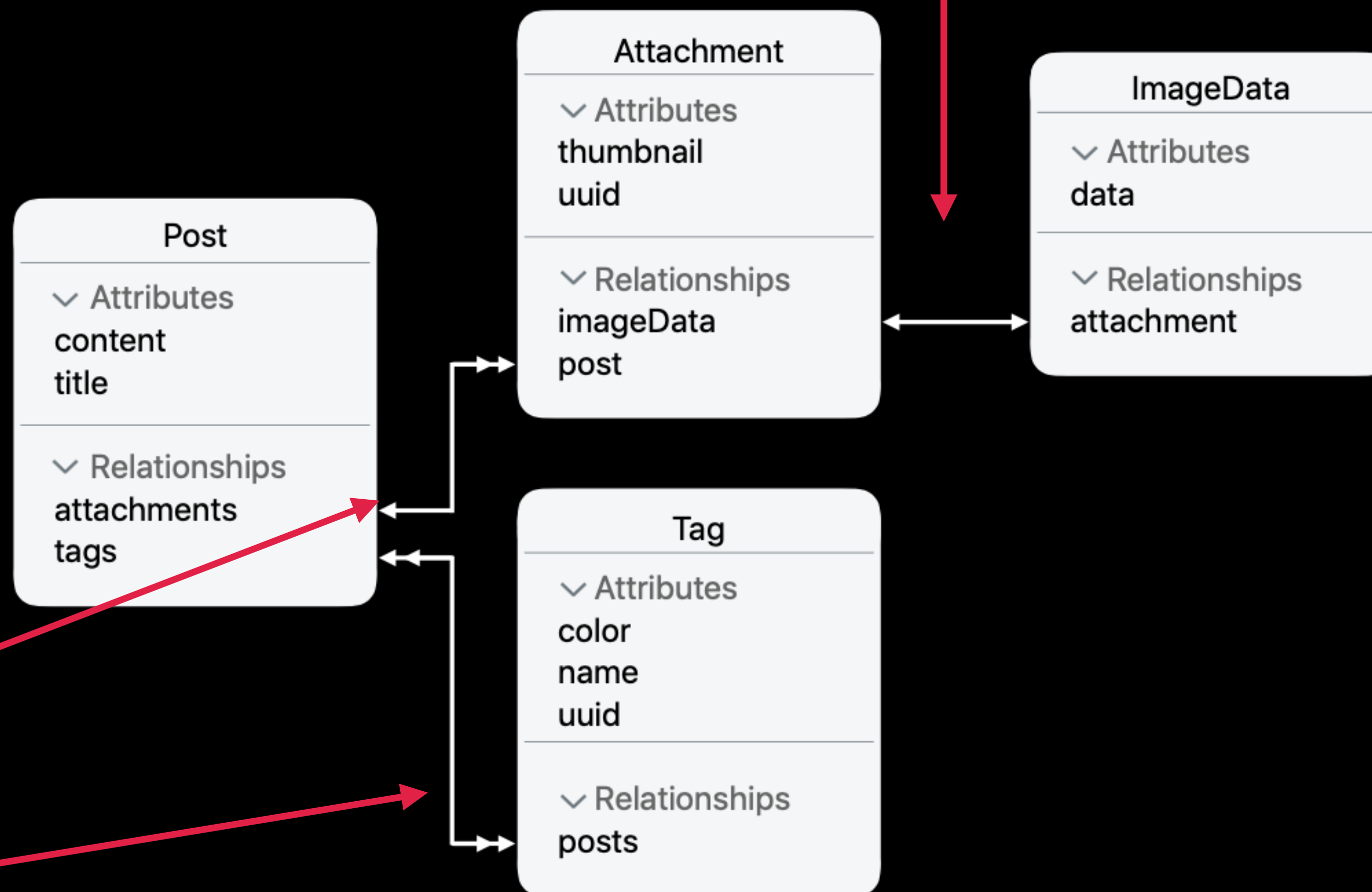
Introduction to Core Data

一个简单的提问

右图是一个备忘录APP的
CoreData定义，请说明不同数
据之间的对应关系是什么类型？

一对多

多对多



一对一



浙江大学
ZHEJIANG UNIVERSITY

Core Data 使用

Introduction to Core Data



浙江大学
ZHEJIANG UNIVERSITY

数据库定义后，可以用数据库生成代码
在Entity的界面中，有一个codegen属性，这是用来调整帮你生成什么样的代码来管理实体的。共有三种选择：

- Manual
- Class Definition (默认)
- Category

The screenshot displays the Xcode Core Data Entity Editor interface. It is divided into three main sections: Entity, Class, and Constraints.

- Entity Section:**
 - Name:** Item
 - Abstract Entity:** Unchecked checkbox
 - Parent Entity:** No Parent Entity (dropdown menu)
- Class Section:**
 - Name:** Item
 - Module:** Global namespace (dropdown menu)
 - Codegen:** Class Definition (dropdown menu)
- Constraints Section:**
 - Contains the text "No Content".
 - Has expand/collapse buttons (+ and -) at the bottom.

Core Data 使用

Introduction to Core Data



浙江大学
ZHEJIANG UNIVERSITY

- **Manual**
 - 需要手动生成对应的Class和Class Extension
- **Class Definition**
 - XCode将为你的Entity在编译时自动生成Class，但是该类不会在文件夹中出现，即你无法修改该类的代码
- **Category**
 - 只生成该类的Extension文件，你可以在这个文件中写相关逻辑

The screenshot displays the Xcode interface for configuring a Core Data entity. The 'Entity' section includes a text field for 'Name' containing 'Item', an unchecked checkbox for 'Abstract Entity', and a dropdown menu for 'Parent Entity' set to 'No Parent Entity'. The 'Class' section features a text field for 'Name' with 'Item', a dropdown for 'Module' set to 'Global namespace', and a dropdown for 'Codegen' set to 'Class Definition'. The 'Constraints' section is currently empty, showing 'No Content' and a '+' button to add constraints.

Core Data 使用

Introduction to Core Data



- 一般情况下选择Category作为代码生成工具。这是为了方便我们对这个类增加其他成员函数。
- 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 options for your new file:

Class:

Subclass of: ▼

☐ Also create XIB file

Language: ▾

Cancel Previous Next

Core Data 使用

Introduction to Core Data



- 我们需要在Swift中定义一个空的实体。属性本身的定义由Swift自动生成在原始定义中。
- 如果不想生成这个Extension，则需要前面选择 Class Definition

```
3
9 import UIKit
0 import CoreData
1
2 class Tweet: NSObject {
3
4 }
5
```

A large red arrow points from the right side of the code block towards the 'import CoreData' line, highlighting the requirement to import the CoreData framework.

Core Data 使用

Introduction to Core Data



- 使用举例：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 tweet = Tweet(context: context)
tweet.unique = twitterInfo.identifier
tweet.text = twitterInfo.text
tweet.created = twitterInfo.created as NSDate
tweet.tweeter = try? TwitterUser.find(matching: twitterInfo.user,
                                       in: context)
```

Core Data: Persistent Container



Introduction to Core Data

```
private func createPersistentContainer() {  
    let container = NSPersistentContainer(name: "Model")  
    container.loadPersistentStores { (description, error) in  
        if let error = error {  
            fatalError("Error: \(error)")  
        }  
        print("Load stores success")  
    }  
}
```

NSPersistentContainer 是一个仓库类。**container** 中包含了各种成员：有操纵表格的 **managedObjectModel**、有操纵数据的 **viewContext**（进行增删改查）

当我们创建了 **NSPersistentContainer** 对象时，仅仅完成了基础的初始化，而对于一些性能开销较大的初始化，比如本地持久化资源的加载等，都还没有完成，我们必须调用成员函数 **loadPersistentStores**

Core Data 基础操作：插入与删除

Basic operations of Core Data: Insert and Delete



- 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

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

Core Data 基础操作：插入与删除



Basic operations of Core Data: Insert and Delete

```
let context = container.viewContext
let book = NSEntityDescription.insertNewObject(forEntityName: "Book",
                                                into: context) as! Book

book.name = name
book.page = Int32(pageCount)
if context.hasChanges {
    do {
        try context.save()
        print("Insert new book \(name) successful.")
    } catch {
        print("\(error)")
    }
}
```

Core Data 基础操作：查询

Basic operations of Core Data: Query



- 进行数据库查询时，我们只需要新创建一个 `fetchRequest`，`fetchRequest` 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` 筛选条件
- `sortDescriptors` 排序

```
let request: NSFetchedRequest<Tweet>
    = Tweet.fetchRequest()
request.entity ...
request.predicate ...
request.sortDescriptors ...
```

Core Data 基础操作：查询

Basic operations of Core Data: Query



数据获取的条件筛选：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: "tweet.screenName = %@",  
                  "Guofeng Zhang")  
  
let predicate2 =  
    NSPredicate(format: "text contains[c] %@",  
                  "Haha")  
  
let predicate = NSCompoundPredicate(  
    andPredicateWithSubpredicates: [predicate1,  
                                     predicate2])
```

Core Data 基础操作：查询

Basic operations of Core Data: Query



设置排序结果 `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(_:))  
    )
```

Core Data 基础操作：查询

Basic operations of Core Data: Query



完整的例子

右图展示了一组查询的方法：

- 新建一个 `fetchRequest`
- 为 `fetchRequest` 添加排序方法和查询内容
- 获取查询结果。返回的结果即查询结果，类型为 `Array<NSManagedObject>`

```
let request:NSFetchRequest<Tweet> = Tweet.fetchRequest()

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

let predicate = NSPredicate(format: "tweeter.screenName = %0",
    "Guofeng Zhang")
request.sortDescriptors = [sortDescriptor]
request.predicate = predicate

let res = try? context.fetch(request)
```

Core Data 与 SwiftUI

Core Data with SwiftUI



浙江大学
ZHEJIANG UNIVERSITY

```
struct SongList: View {
    @FetchRequest(entity: Song.entity(),
                  sortDescriptors: [
                      NSSortDescriptor(keyPath: \Song.name, ascending: true)
                  ],
                  predicate: NSPredicate(format: "isFavorite = %@", true))
    var songs: FetchedResults<Song>

    var body: some View {
        List(songs, id: \.id) { song in
            SongRow(song: song)
        }.environment(\.managedObjectContext, CoreDataStack.shared.viewContext)
    }
}
```

SwiftUI中，FetchRequest 方法可以写成属性包装器的形式。在这个例子中，songs列表会随着FetchRequest自动更新。

Core Data 与 CloudKit 结合

Core Data with CloudKit

WWDC 2019 引入了

`NSPersistentCloudKitContainer`。这意味着无需编写大量代码，使用 `Core Data with CloudKit` 可以让用户在他所有的苹果设备上无缝访问应用程序中的数据。

优势：

- 免费：开发者不用为网络服务支付额外费用。存储使用的是用户个人的iCloud空间。
- 安全：苹果内置了沙盒、加密字段、数据隔离等多种手段保护用户数据安全。鉴权、分发都是无感的，用户体验好。



Choose options for your new project:

Product Name:

Team:

Organization Identifier:

Bundle Identifier:

Interface:

Language:

☒ Use Core Data

☒ Host in CloudKit

☐ Include Tests

Cancel Previous Next

Core Data 与 CloudKit 结合

Core Data with CloudKit



在应用程序中启用Core Data with CloudKit功能，只需要以下几步：

使用 `NSPersistentCloudKitContainer`

在项目 Target 的 Signing & Capabilities 中添加 `CloudKit` 支持

为项目创建或指定 CloudKit container

在项目 Target 的 Signing & Capabilities 中添加 background mode 支持

配置 `NSPersistentStoreDescription` 以及 `viewContext` **合并冲突策略选择**

检查 Data Model 是否满足同步的要求

Core Data 与 CloudKit 结合

Core Data with CloudKit



以下几步:

CloudKit 支持

background mode 支持

text

合并冲突策略选择

`NSMergeByPropertyStoreTrumpMergePolicy`

逐属性比较, 如果持久化数据和内存数据都改变且冲突, 持久化数据胜出

`NSMergeByPropertyObjectTrumpMergePolicy`

逐属性比较, 如果持久化数据和内存数据都改变且冲突, 内存数据胜出

`NSOverwriteMergePolicy`

内存数据永远胜出

`NSRollbackMergePolicy`

持久化数据永远胜出

基础概念

CoreData使用

LeanCloud使用

基础概念

CoreData使用

LeanCloud使用

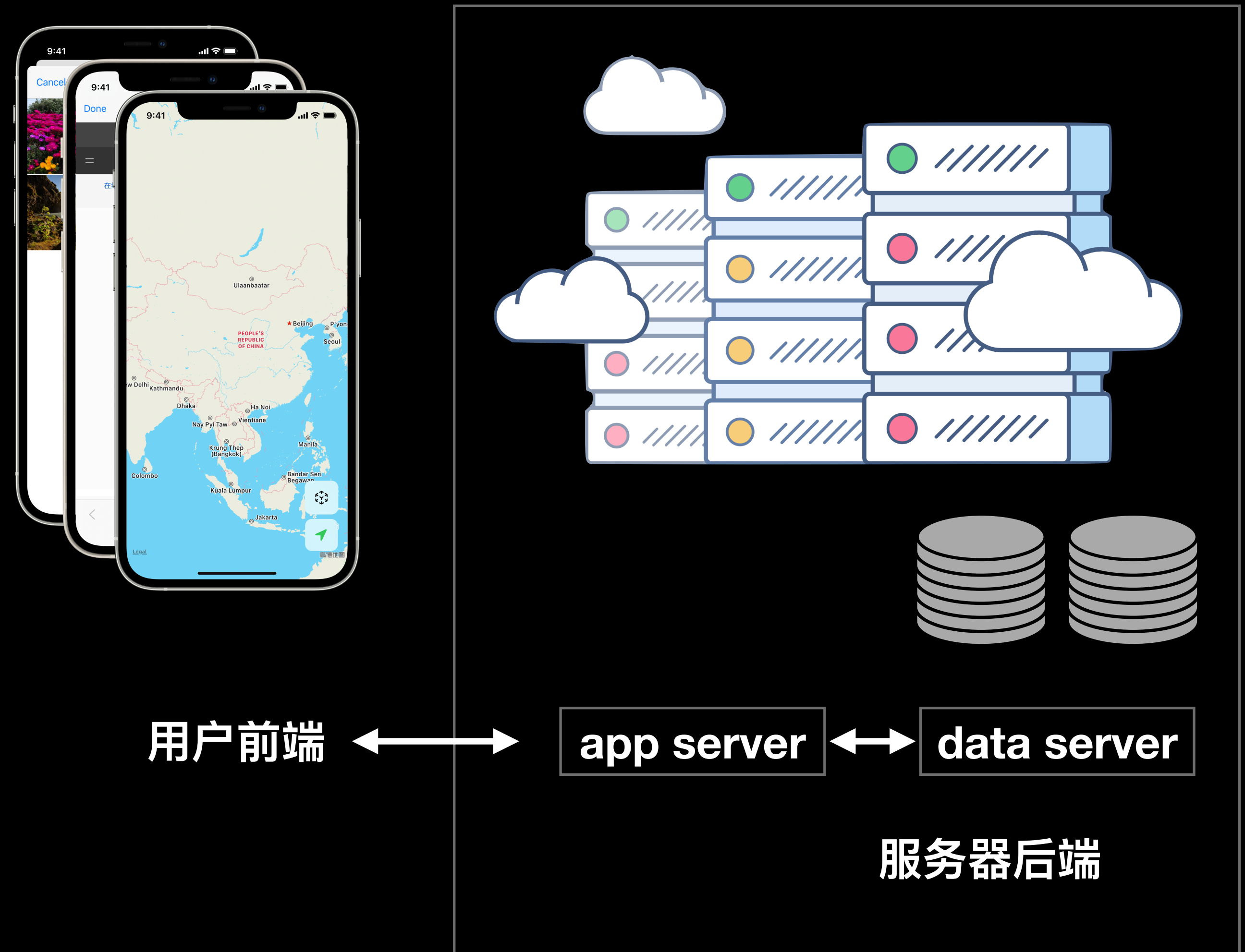
LeanCloud 简介

Introduction to LeanCloud



回顾一下大部分APP的业务逻辑：

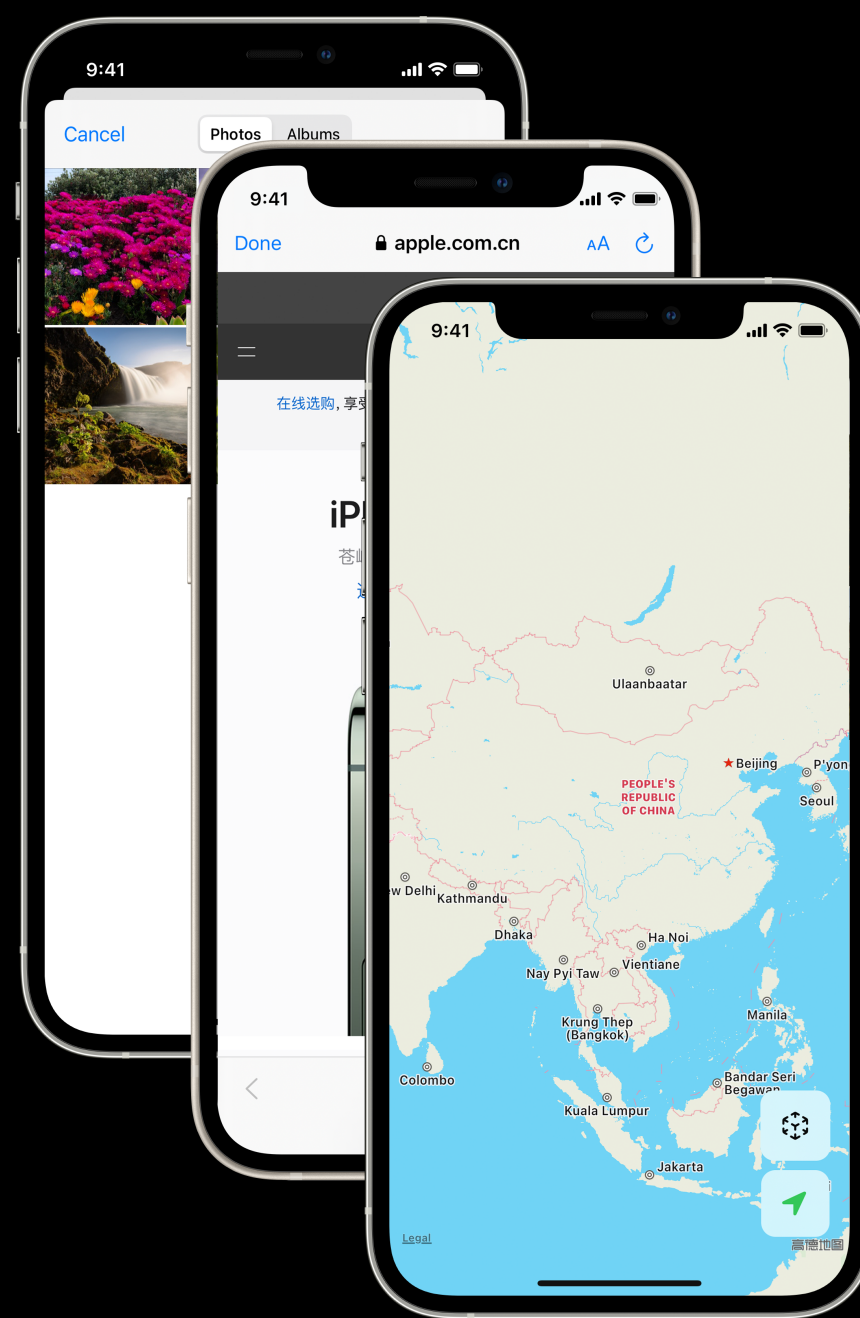
- 前端负责数据展现和用户交互处理，与后端的 app server 通过网络来交换需要的数据；
- app server 负责业务逻辑处理，生成核心数据存储在 data server，或者聚合 data server 查询到的数据返回给客户端；
- data server 负责核心数据的存储和备份。



LeanCloud 简介

Introduction to LeanCloud

LeanCloud提供了云端存储功能，能够高效存取海量级 JSON 对象、二进制文件、地理位置等数据。开发者可以使用各类语言的SDK访问云数据库，**一行后端代码都不用写**，而快速完成一个产品（网站或应用）的开发和发布。



用户前端

网络请求



LeanCloud



浙江大学
ZHEJIANG UNIVERSITY

LeanCloud 特点

Features of LeanCloud



LeanCloud 其他功能特点

- 面向对象的海量数据库
- 结构化数据存储
- 大文件存储分发
- 多端实时同步
- 权限控制
- ...

LeanCloud 相比 CoreData 的优势

- 有多种语言接口可以支持跨平台 (Web、Android)
- 大文件存储更友好



LeanCloud 简介

Introduction to LeanCloud



LeanCloud并非关系型数据库

类似文档型数据库MongoDB

RDBMS	MongoDB	LeanCloud
Database	Database	Application
Table	Collection	Class
Row	Document	Object
Index	Index	Index
JOIN	Embedded, Reference	Embedded Object, Pointer

LeanCloud 简介

Introduction to LeanCloud



LeanCloud并非关系型数据库

- 不需要任何 JDBC/ODBC 的驱动，直接通过 HTTP 协议来传输 JSON Object 即可
- Scheme Free: LeanCloud 对于数据的唯一格式要求是满足 JSON Object 的形式，存储新的对象类型时不需要预先在云端定义任何「表结构」，而且同一种数据类型里的键值也是允许随时增加的。

RDBMS	MongoDB	LeanCloud
Database	Database	Application
Table	Collection	Class
Row	Document	Object
Index	Index	Index
JOIN	Embedded, Reference	Embedded Object, Pointer

LeanCloud 简介

Introduction to LeanCloud

Person_ID	Surname	First_Name	City
0	柳	红	北京
1	杨	真	北京
2	王	新	苏黎世

Car_ID	Model	Year	Value	Person_ID
101	大众迈腾	2015	180000	0
102	丰田汉兰达	2016	240000	0
103	福特翼虎	2014	220000	1
104	现代索纳塔	2013	150000	2

关系型数据库



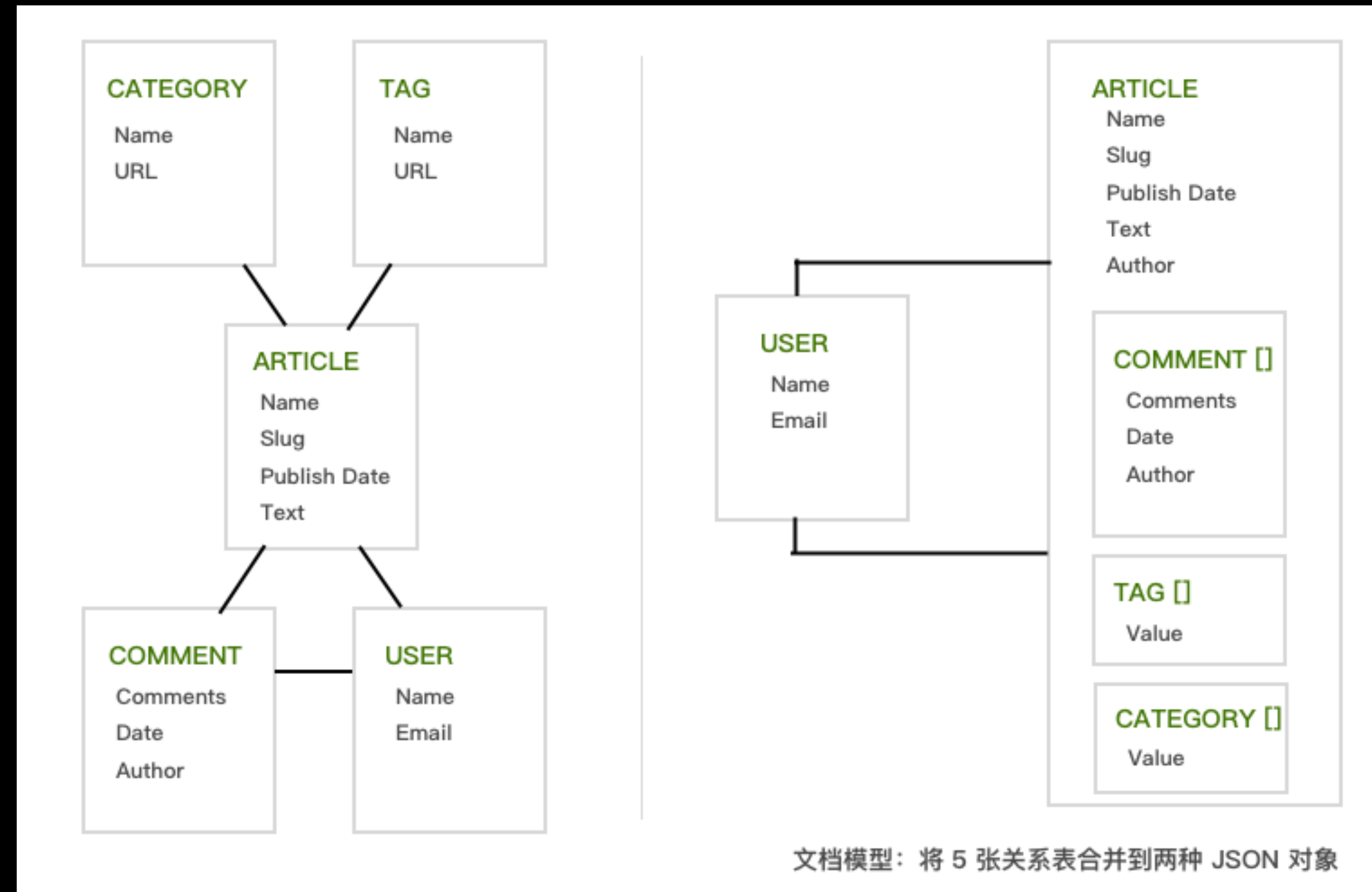
浙江大学
ZHEJIANG UNIVERSITY

```
{
  "first_name": "红",
  "surname": "柳",
  "city": "伦敦",
  "location": [
    45.123,
    47.232
  ],
  "cars": [
    {
      "model": "大众迈腾",
      "year": 2015,
      "value": 180000
    },
    {
      "model": "丰田汉兰达",
      "year": 2016,
      "value": 240000
    }
  ]
}
```

LeanCloud

LeanCloud 简介

Introduction to LeanCloud



关系型数据库

LeanCloud

LeanCloud 简介

Introduction to LeanCloud



在 Xcode 项目中安装 LeanCloud Swift SDK

安装

使用 CocoaPods 包管理工具安装

CocoaPods 是 macOS 和 iOS 应用开发中的「包管理工具」，可以非常方便地管理使用第三方库、或者发布自己的库。

CocoaPods 简介

Introduction to CocoaPods

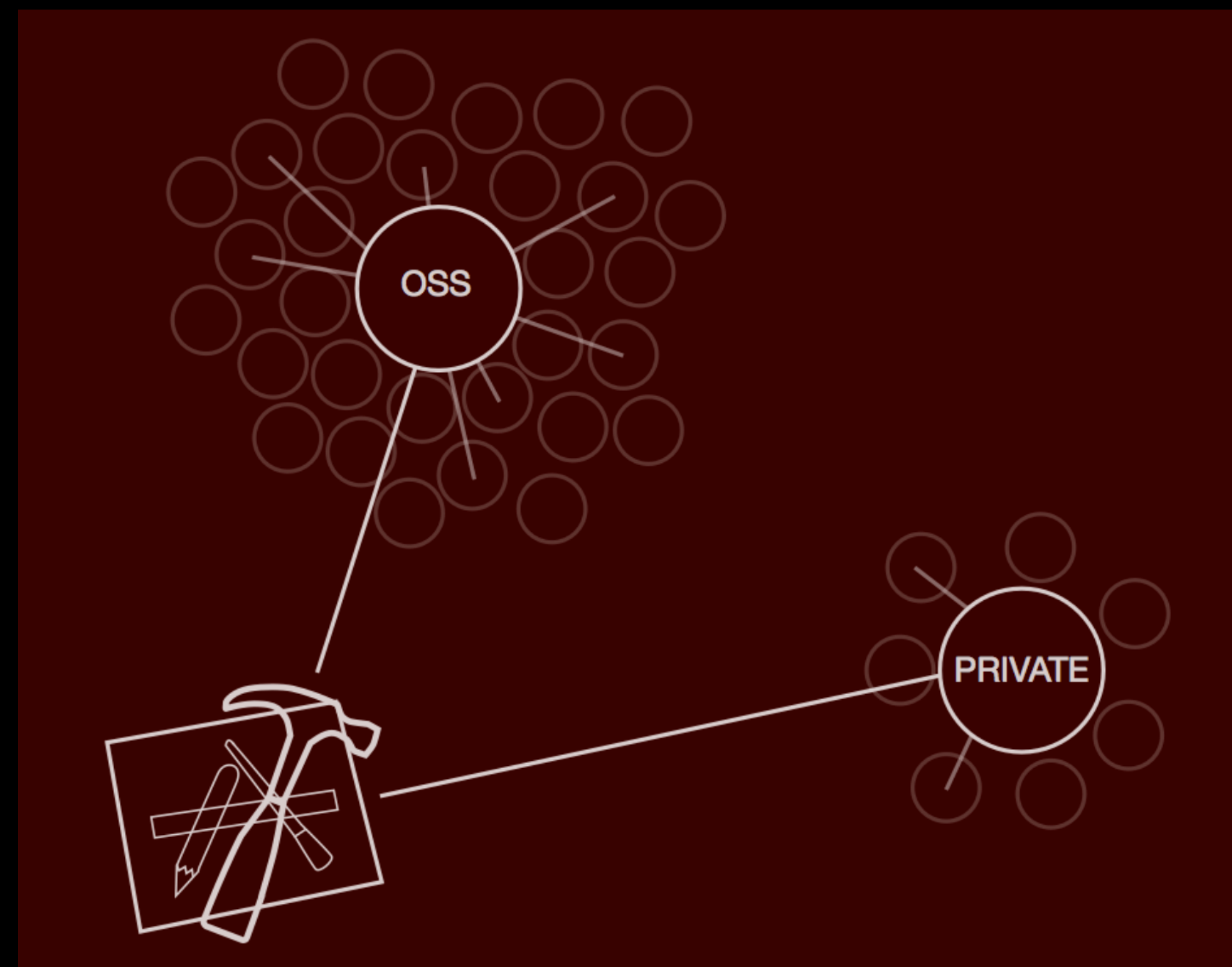
CocoaPods 是 macOS 和 iOS 应用开发中的「包管理工具」，可以非常方便地管理使用第三方库、或者发布自己的库。

安装

```
$ sudo gem install cocoapods
```



浙江大学
ZHEJIANG UNIVERSITY



CocoaPods 简介

Introduction to CocoaPods



浙江大学
ZHEJIANG UNIVERSITY

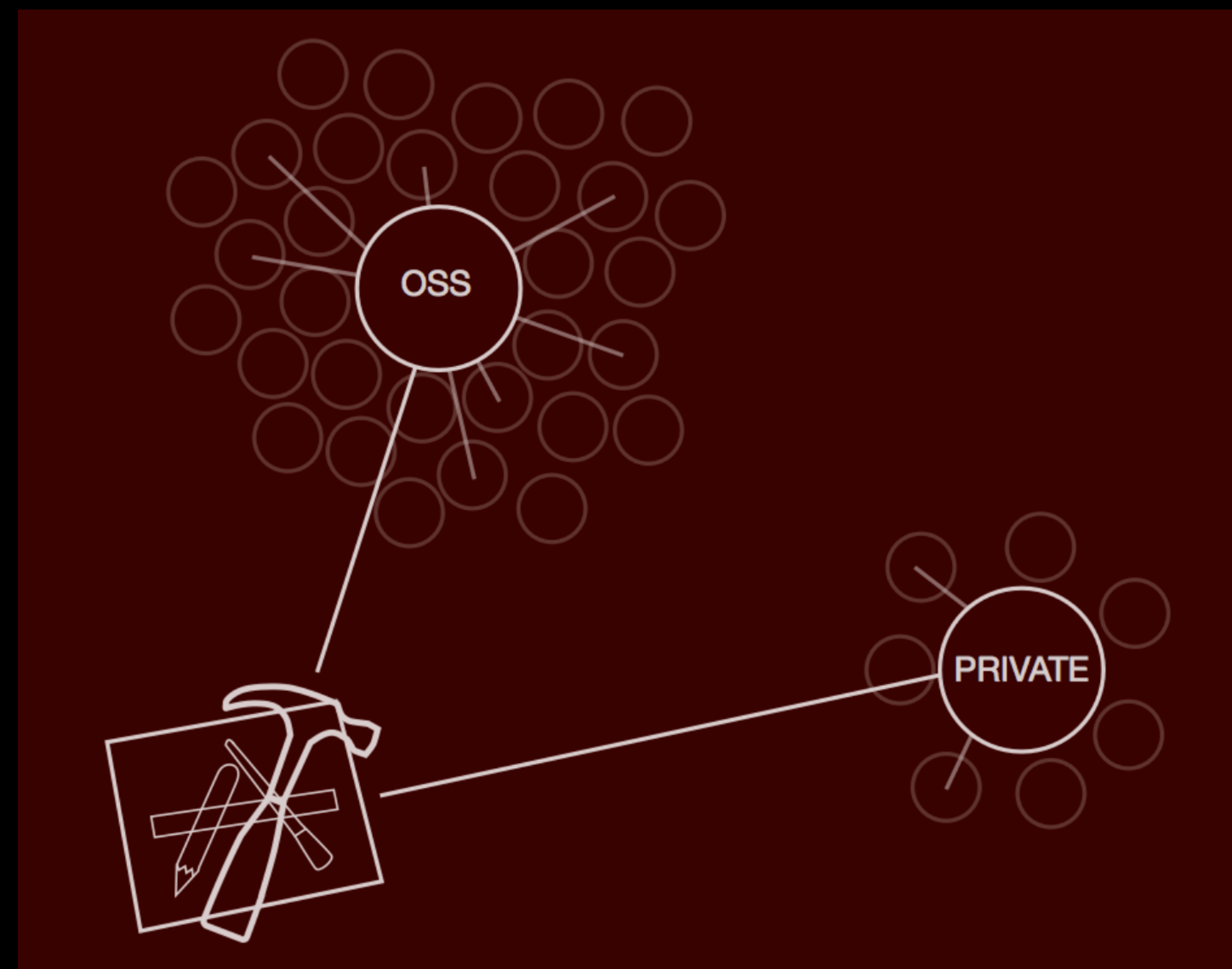
CocoaPods 是 macOS 和 iOS 应用开发中的「包管理工具」，可以非常方便地管理使用第三方库、或者发布自己的库。

安装

```
$ sudo gem install cocoapods
```

查看

```
$ pod --version
```



CocoaPods 简介

Introduction to CocoaPods



CocoaPods 是 macOS 和 iOS 应用开发中的「包管理工具」，可以非常方便地管理使用第三方库、或者发布自己的库。

安装

```
$ sudo gem install cocoapods
```

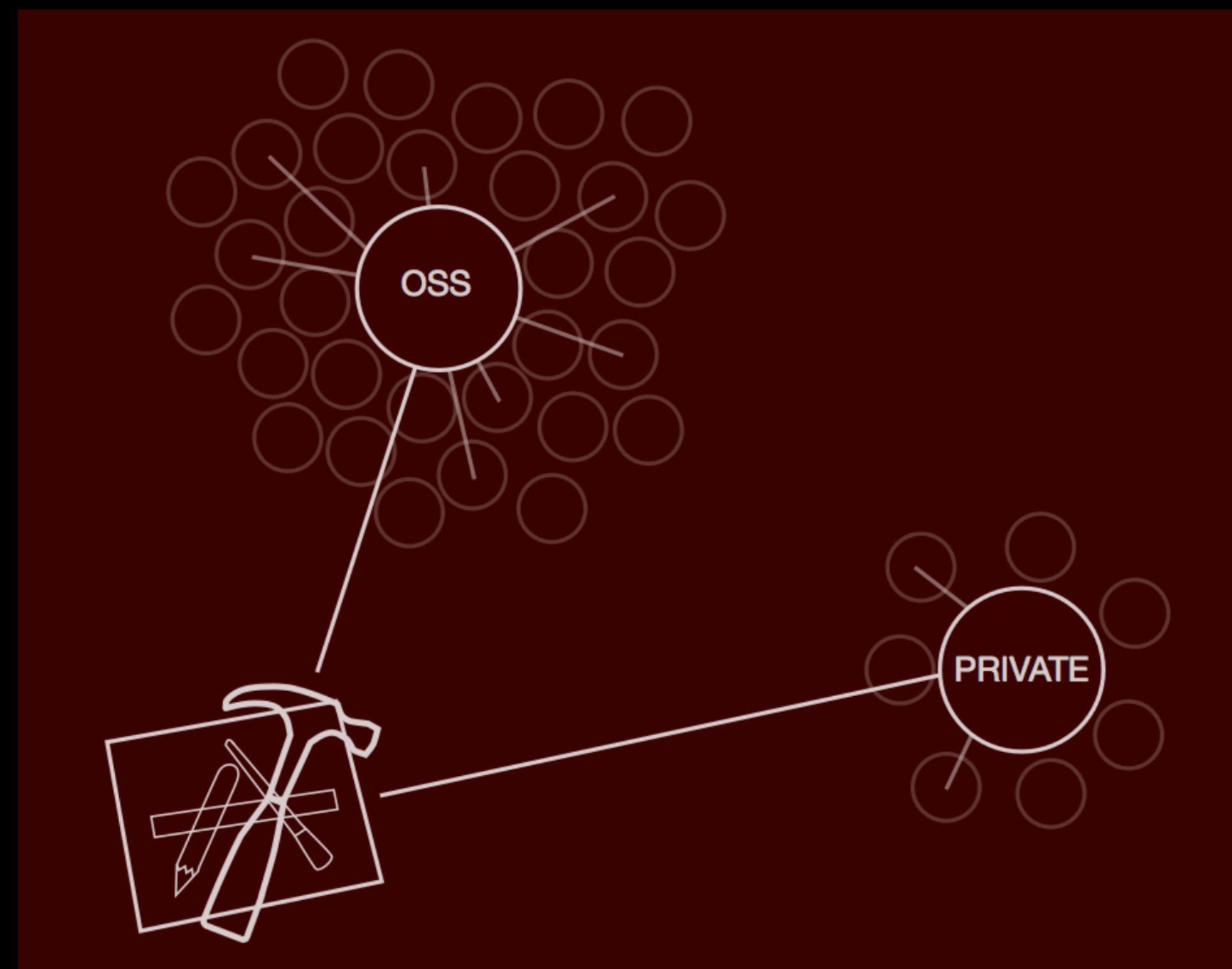
查看

```
$ pod --version
```

初始化

```
$ pod init
```

会在项目根目录创建一个Podfile文件



CocoaPods 简介

Introduction to CocoaPods



Podfile文件

source: 存放三方库 .podspec 文件的索引库, pod 搜索三方库就是从 source 仓库里搜索的。

- CocoaPods 支持添加多个源

target: 指定以下的依赖配置所针对的 target。
Podfile 允许你对 workspace 下不同 target 进行不同的依赖配置。target是我们工程中的最小可编译单元, 每一个 target 对应一个编译输出

pod: 依赖项, 可以指定版本、最低版本等。

PODFILE

```
1 # open source
2 source 'https://github.com/CocoaPods/Specs.git'
3
4 # my work
5 source 'https://github.com/Artsy/Specs.git'
6
7 target 'App' do
8
9     pod 'Artsy+UIColors'
10    pod 'Artsy+UIButtons'
11
12    pod 'FLKAutoLayout'
13    pod 'ISO8601DateFormatter', '0.7'
14    pod 'AFNetworking', '~> 2.0'
15
16    target 'AppTests' do
17        inherit! :search_paths
18        pod 'FBSnapshotTestCase'
19        pod 'Quick'
20        pod 'Nimble'
21    end
22 end
```

CocoaPods 简介

Introduction to CocoaPods



Podfile文件

安装依赖

```
$ pod install
```

读取 **Podfile**，根据podfile.lock进行增量式安装，并自动进行版本控制、解决冲突

加载源文件：每个.podspec文件都包含一个源代码的索引，这些索引一般包裹一个git地址和git tag。可以为依赖库单独指定podspec，或者使用默认的source。随后，CocoaPods会下载对应依赖库，并生成一系列文件。

PODFILE

```
1  # open source
2  source 'https://github.com/CocoaPods/Specs.git'
3
4  # my work
5  source 'https://github.com/Artsy/Specs.git'
6
7  target 'App' do
8
9      pod 'Artsy+UIColors'
10     pod 'Artsy+UIButtons'
11
12     pod 'FLKAutoLayout'
13     pod 'ISO8601DateFormatter', '0.7'
14     pod 'AFNetworking', '~> 2.0'
15
16     target 'AppTests' do
17         inherit! :search_paths
18         pod 'FBSnapshotTestCase'
19         pod 'Quick'
20         pod 'Nimble'
21     end
22 end
```

CocoaPods 简介

Introduction to CocoaPods



Podfile文件

安装依赖

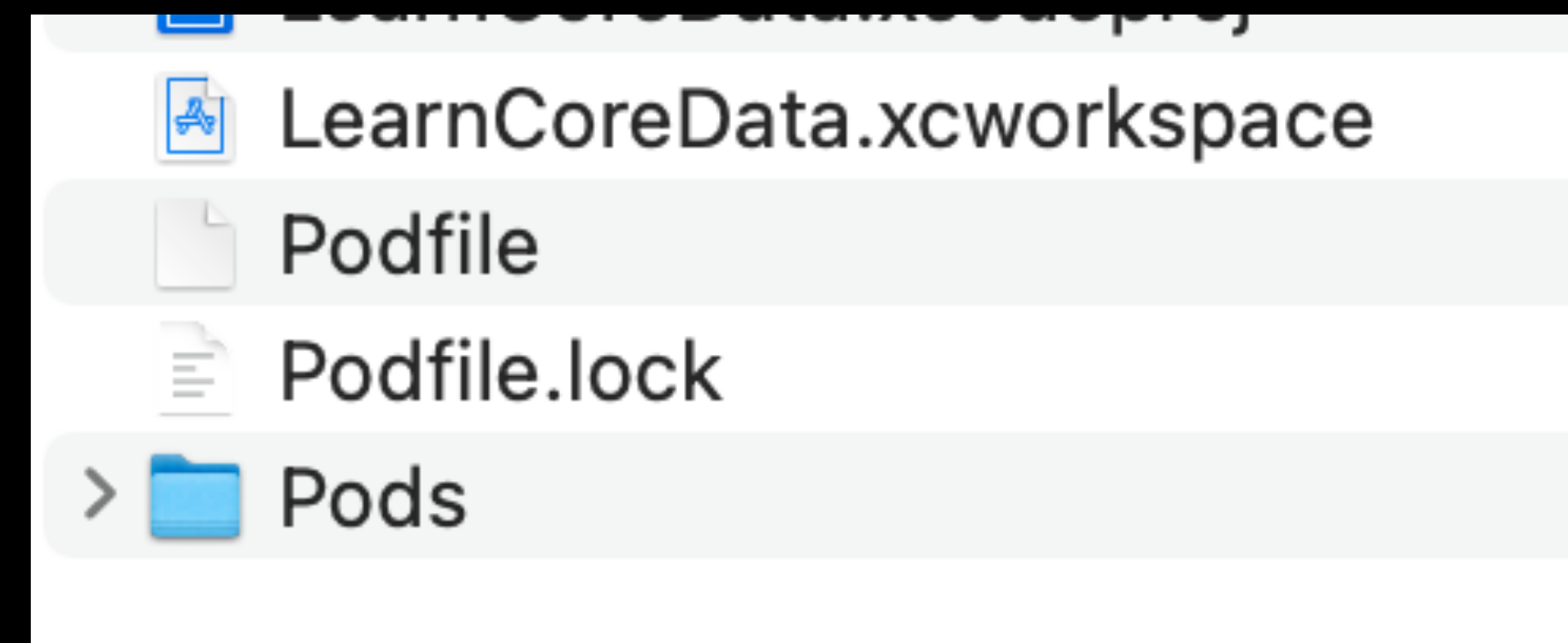
```
$ pod install
```

读取 **Podfile**，根据podfile.lock进行增量式安装，并自动进行版本控制、解决冲突

加载源文件：每个.podspec文件都包含一个源代码的索引，这些索引一般包裹一个git地址和git tag。可以为依赖库单独指定podspec，或者使用默认的source。随后，CocoaPods会下载对应依赖库，并生成一系列文件。

```
[!] Please close any current Xcode sessions and use
`LearnCoreData.xcworkspace` for this project from now on.
Pod installation complete! There is 1 dependency from the
Podfile and 1 total pod installed.
```

原来的工程设置已被更改，如果直接打开原来的工程文件去编译可能会报错，需要使用新生成的workspace来进行项目管理。



执行后项目文件夹中会多出的文件：

Podfile.lock：它记录了需要被安装的pod中每个已安装的版本。如果你想知道已安装的CocoaPods是哪个版本，可以查看这个文件

Pods.xcodeproj：CocoaPods相关库和设置保存的文件夹

...xcworkspace：和xcproj不同，workspace是纯粹的容器，不参与任何编译链接过程，它主要管理Xcode中的projects，记录它们在Finder中的引用位置、一些用户界面的自定义信息（窗口的位置，顺序，偏好等等）。

CocoaPods 简介

Introduction to CocoaPods



常用指令

安装依赖

```
$ pod install
```

更新库

```
$ pod update
```

将所有第三方框架更新到最新版本, 并且创建一个新的Podfile.lock文件

拉取索引

```
$ pod setup
```

将所有第三方的Podspec索引文件更新到本地的~/.cocoapods/repos目录下,更新本地仓库

查找

```
$ pod search xx
```

查找某一个开源库。查找开源库之前, 默认会执行pod repo update指令

回到 LeanCloud

Introduction to LeanCloud



AVObject数据对象

每种「AVObject」对应 LeanCloud 云端的一张「表」、或者说是一个LeanCloud的Class。

数据类型

AVObject 中的 键，必须是由字母、数字或下划线组成的字符串；开发者自定义的键，不能以 _（双下划线）开头。AVObject 中的 值，可以是字符串、数字、布尔值，或是数组和字典。在平台内部，LeanCloud 将数据存储为 **JSON**。

回到 LeanCloud

Introduction to LeanCloud



AVObject数据对象

每种「AVObject」对应 LeanCloud 云端的一张「表」、或者说是一个LeanCloud的Class。

数据类型

AVObject 中的 键，必须是由字母、数字或下划线组成的字符串；开发者自定义的键，不能以 _（双下划线）开头。AVObject 中的 值，可以是字符串、数字、布尔值，或是数组和字典。在平台内部，LeanCloud 将数据存储为 **JSON**。

数据关联

AVObject 模型与传统的关系型数据库有一个很大的不同，就是没有了主键、外键的概念。LeanCloud 中有 4 种方式来构建对象之间的关系：

- Pointer：将一个对象存为另一个对象的属性值，处理一对一关系
- Array：将多个对象储存为另一个对象的属性，处理一对多关系
- AVRelation：关联类，处理多对多关系
- 中间表设计：类似传统关系数据库，另外增加中间表来丰富关系表示

LeanCloud 后台页面

Online Dashboard of LeanCloud

LeanCloud 中国版注册后需要实名验证，随后可以在网页控制台创建应用，选择开发版。



LeanCloud

华北 ▾ 应用 ▾

欢迎使用 LeanCloud

完整、成熟的后端解决方案，不论是有状态组件还是无状态组件都能轻松搞定。

创建应用

快速入门 [↗](#)

创建应用

应用名称 *

应用的 [计价方案](#) [↗](#) *

☒

开发版

适用于项目开发阶段及原型项目，有用量限制。

☐

商用版（每日最低消费 30 CNY）

适用于正式上线的商业项目，高性能、高可用。

LeanCloud 后台页面

Online Dashboard of LeanCloud



LeanCloud

应用概览

数据存储

结构化数据

数据仓库

文件

LiveQuery

全文搜索

导入导出

用量统计

API 访问日志

服务设置

内建账户

云引擎

即时通讯

华北 LearnLeanCloud 开发版

创建 Class

Filter Classes

_Conversation	0
_File	0
_Followee	0
_Follower	0
_Installation	0
_Role	0
_User	0

如何管理我的数据?

- 从左侧选择一个 class 可以查看相关数据
- 您也可以创建新的 class, 或导入已有数据
- 通过快捷键更方便进行数据操作
 - Arrow/Tab 切换单元格
 - Enter 当前单元格名称为 objectId 时, 打开视图窗口
 - Enter 当前单元格类型为 File 时, 打开文件窗口
 - Enter 当前单元格可编辑时, 进入编辑模式
 - E 当单元格数据较多时, 更方便的编辑当前单元格
 - Del 删除当前单元格

结构化数据是最常用的功能

LeanCloud 后台页面

Online Dashboard of LeanCloud



LeanCloud

应用概览

数据存储

结构化数据

数据仓库

文件

LiveQuery

全文搜索

导入导出

用量统计

API 访问日志

服务设置

内建账户

云引擎

即时通讯

华北 LearnLeanCloud 开发版

创建 Class

Filter Classes

_Conversation	0
_File	0
_Followee	0
_Follower	0
_Installation	0
_Role	0
_User	0

如何管理我的数据?

- 从左侧选择一个 class 可以查看相关数据
- 您也可以创建新的 class, 或导入已有数据
- 通过快捷键更方便进行数据操作
 - Arrow/Tab 切换单元格
 - Enter 当前单元格名称为 objectId 时, 打开视图窗口
 - Enter 当前单元格类型为 File 时, 打开文件窗口
 - Enter 当前单元格可编辑时, 进入编辑模式
 - E 当单元格数据较多时, 更方便的编辑当前单元格
 - Del 删除当前单元格

选择创建Class

LeanCloud 后台页面

Online Dashboard of LeanCloud



浙江大学
ZHEJIANG UNIVERSITY

创建时候可以选择访问
控制等权限设置信息

创建 Class

Class 名称 *

Students

☐ 创建为 日志表 (仅对商用版 / 企业版应用开放)

Class 访问权限

add_fields

所有用户

create

所有用户

delete

所有用户

update

所有用户

☒ 所有用户

☐ 登录用户

☐ 指定用户

LeanCloud 后台页面

Online Dashboard of LeanCloud



添加行

age

name

String

ACL

设置 ACL 权限

高级安全选项

☒ 使用 masterKey ☐ 跳过 Hooks

取消 添加

添加列

列名称 *

列名称

列类型 *

String

☐ 默认值

权限 ☐ 只读 ☐ 客户端不可见

其他 ☐ 必填

列注释

列注释

取消 保存

创建Class之后可以通过添加列的方式添加属性、添加行的方式添加数据

LeanCloud 后台页面

Online Dashboard of LeanCloud



添加行

age

name

String

设置 A

☒ 使用

数据 权限 性能与索引 Hooks 离线数据分析

添加行 添加列 刷新 按条件过滤 批量操作 编辑单元格 ...

☐	objectId STRING	ACL ACL	age NUMBER ↑	name STRING	cre
☐	623bc2fc1691ee2cf43c0dbe	☐ {"*":{"read":true}}	18	Alice	202
☐	623bc28118a2bb05a55e1852	☐ {"*":{"read":true}}	19	Bob	202

添加列

列注释

取消 保存

创建Class之后可以通过添加列的方式添加属性、添加行的方式添加数据

LeanCloud Swift SDK

LeanCloud Swift SDK



获取对象

对于已经保存到云端的 LCObject,
可以通过它的 objectId 将其取回
底层实现仍然是REST API

```
let query = LCQuery(className: "Todo")
let _ = query.get("582570f38ac247004f39c24b")
{ (result) in
    switch result {
    case .success(object: let todo):
        let title      = todo.get("title")
        let priority    = todo.get("priority")

        // 获取内置属性
        let objectId    = todo.objectId
        let updatedAt    = todo.updatedAt
        let createdAt    = todo.createdAt
    case .failure(error: let error):
        print(error)
    }
}
```

LeanCloud Swift SDK

LeanCloud Swift SDK



保存对象

使用save方法可以把数据保存到云端

更新对象

如果对象设置了对应的ObjectID, 使用save方法即为更新数据

```
do {  
    let account = LCObject(className: "Account",  
objectId: "5745557f71cfe40068c6abe0")  
    // 对 balance 原子减少 100  
    let amount = -100  
    try account.increase("balance", by: amount)  
    account.save() { (result) in  
        switch result {  
        case .success:  
            if let balance = account["balance"] {  
                print("当前余额为: \(balance)")  
            }  
        case .failure(error: let error):  
            if error.code == 305 {  
                print("余额不足, 操作失败! ")  
            }  
        }  
    }  
} catch {  
    print(error)  
}
```

LeanCloud Swift SDK

LeanCloud Swift SDK



浙江大学
ZHEJIANG UNIVERSITY

保存对象

使用save方法可以把数据保存到云端

更新对象

如果对象设置了对应的ObjectID, 使用save方法即为更新数据

有条件更新对象

save时传入一个query对象

```
do {
    let account = LCObject(className: "Account",
objectId: "5745557f71cfe40068c6abe0")
    // 对 balance 原子减少 100
    let amount = -100
    try account.increase("balance", by: amount)
    // 设置条件
    let query = LCQuery(className: "Account")
    query.whereKey("balance", .greaterThanOrEqualTo(-amount))
    let options: [LCObject.SaveOption] = [
        .query(query),
        // 操作结束后, 返回最新数据。
        // 如果是新对象, 则所有属性都会被返回,
        // 否则只有更新的属性会被返回。
        .fetchWhenSave
    ]
    account.save(options: options) { (result) in
        switch result {
        case .success:
            if let balance = account["balance"] {
                print("当前余额为: \(balance)")
            }
        case .failure(error: let error):
            if error.code == 305 {
                print("余额不足, 操作失败! ")
            }
        }
    }
} catch {
    print(error)
}
```

LeanCloud Swift SDK

LeanCloud Swift SDK



同步对象

当云端数据发生更改时，你可以调用 fetch 方法来刷新对象，使之与云端数据同步。

Fetch会覆盖本地数据；刷新时也可以指定特定属性

```
let todo = LCObject(className: "Todo", objectId:
"582570f38ac247004f39c24b")
todo.fetch(keys: ["priority", "location"])
{ (result) in
    switch result {
    case .success:
        // 只有 priority 和 location 会被获取和刷新
        break
    case .failure(error: let error):
        print(error)
    }
}
```

LeanCloud Swift SDK

LeanCloud Swift SDK



浙江大学
ZHEJIANG UNIVERSITY

删除对象

```
let todo = LCObject(className: "Todo", objectId: "582570f38ac247004f39c24b")
_ = todo.delete { result in
    switch result {
    case .success:
        break
    case .failure(error: let error):
        print(error)
    }
}
```

LeanCloud Swift SDK

LeanCloud Swift SDK



查询对象

1. 构建 `LCQuery`;
2. 向其添加查询条件;
3. 执行查询并获取包含满足条件的对象的数组。

```
let query = LCQuery(className: "Student")
query.whereKey("lastName", .equalTo("Smith"))
_ = query.find { result in
    switch result {
    case .success(objects: let students):
        // students 是包含满足条件的 Student 对象的数组
        break
    case .failure(error: let error):
        print(error)
    }
}
```

LeanCloud Swift SDK

LeanCloud Swift SDK



浙江大学
ZHEJIANG UNIVERSITY

查询对象

1. 构建 `LCQuery`;
2. 向其添加查询条件;
3. 执行查询并获取包含满足条件的对象的数组。

可以添加不同的条件来改变获取到的结果。

```
let query = LCQuery(className: "Student")
query.whereKey("lastName", .equalTo("Smith"))
_ = query.find { result in
    switch result {
    case .success(objects: let students):
        // students 是包含满足条件的 Student 对象的数组
        break
    case .failure(error: let error):
        print(error)
    }
}
```

```
query.whereKey("age", .greaterThan(18));
query.whereKey("age", .greaterThanOrEqualTo(18));
query.whereKey("priority", .ascending)
query.whereKey("createdAt", .descending)
```

Any Questions ?