**基于移动应用的俱乐部分享和知识管理平台**

# 摘要

目前，Apple在国内各大高校开设了近百个学生创新俱乐部，各俱乐部分别进行着创新技能训练与创新项目产出，产出成果包括但不下限于各种线上的技术博客，线下的技术沙龙。最终各种形式的学生创新项目产出汇聚成一个庞大的技术与知识库。但每个人每天的只是接收是有上限的，因此，如何管理，如何推送，推送给哪些群体什么类型的信息，就成了一个不得不解决的问题。本项目旨在搭建一个各俱乐部自治的平台，独立发布各自的知识产出，亦或各自的创新活动，再根据用户的浏览情况，选择性投放相关类型的知识博客等内容。从而最终实现俱乐部集群下知识的分享与管理。

# Abstract

[摘要 1](#_Toc40970481)

[Abstract 2](#_Toc40970482)

[1.绪论 5](#_Toc40970483)

[1.1课题背景及意义 5](#_Toc40970484)

[1.1.1创新俱乐部现状 5](#_Toc40970485)

[1.1.2研究意义 5](#_Toc40970486)

[2.技术框架 6](#_Toc40970487)

[2.1 iOS开发 6](#_Toc40970488)

[2.1.1 Swift概述 6](#_Toc40970489)

[2.1.2开发模式 6](#_Toc40970490)

[3.需求分析 8](#_Toc40970491)

[3.1功能性需求建模 8](#_Toc40970492)

[3.1.1注册 8](#_Toc40970493)

[3.1.2登陆 8](#_Toc40970494)

[3.1.3 主要功模块 8](#_Toc40970495)

[3.1.4 查看新闻详情 9](#_Toc40970496)

[3.1.5查看Blog详情 10](#_Toc40970497)

[3.1.6 发布News 10](#_Toc40970498)

[3.1.7 发布Blog 11](#_Toc40970499)

[3.1.8 发布Event 11](#_Toc40970500)

[3.1.9 修改个人信息 11](#_Toc40970501)

[3.1.10查看个人发布历史 12](#_Toc40970502)

[3.1.11 查看俱乐部状态 12](#_Toc40970503)

[3.1.12 查看所有俱乐部 12](#_Toc40970504)

[3.2 非功能性需求建模 13](#_Toc40970505)

[3.2.1性能 13](#_Toc40970506)

[3.2.2 易用性 13](#_Toc40970507)

[2.2.3 安全性 13](#_Toc40970508)

[4.概要设计 14](#_Toc40970509)

[4.1 总体设计 14](#_Toc40970510)

[4.1.1 开发/运行环境 14](#_Toc40970511)

[4.1.2技术架构设计 14](#_Toc40970512)

[4.1.3 部署架构设计 15](#_Toc40970513)

[4.2 接口设计 15](#_Toc40970514)

[4.2.1用户接口 15](#_Toc40970515)

[4.2.2外部接口 16](#_Toc40970516)

[4.3界面设计 16](#_Toc40970517)

[4.3.1注册/登陆界面 17](#_Toc40970518)

[4.3.2 News/Blog 展示界面 17](#_Toc40970519)

[4.3.3 News/Blog 详情界面 18](#_Toc40970520)

[4.3.4 News发布界面 18](#_Toc40970521)

[4.3.5 Event展示界面 19](#_Toc40970522)

[4.3.6 Event发布界面 19](#_Toc40970523)

[4.3.7 发布历史界面 20](#_Toc40970524)

[5.详细设计规约 21](#_Toc40970525)

[5.1 Store & AppState模块 21](#_Toc40970526)

[5.2 News模块 23](#_Toc40970527)

[5.3 Blog模块 25](#_Toc40970528)

[5.4 Event模块 26](#_Toc40970529)

[5.5 Club模块 29](#_Toc40970530)

[5.6 User 模块 29](#_Toc40970531)

[6 33](#_Toc40970532)

[7.测试 35](#_Toc40970533)

1.绪论

1.1课题背景及意义

1.1.1创新俱乐部现状

21世纪是创新的时代，创新力决定了竞争力，创新产出衡量了技术能力。全国各高校的工程院系，作为创新全国高校创新产出群体中极为重要的一员，设立了各种各样的创新俱乐部，鼓励学生自发地进行交流研究，从而达到提高学生创新能力，培养创新意识的目的。但受限于俱乐部组成形式和高校生平时的日程安排，各俱乐部之间耦合松散，缺乏交流，跨校际的交流更是几乎为0。导致各俱乐部之间相互独立，缺乏思维碰撞和知识的渗透和互换。

1.1.2研究意义

Apple目前已在全国各地高校开设了近百个学生创新俱乐部，本文旨在搭建一个创新俱乐部交流平台，为俱乐部成员，俱乐部之间开设一个方便的交流渠道。同时支持用户编写或分享俱乐部近期的线下活动和线上成果产出，构建全国俱乐部共享的知识分享库。

为了更好的管理和利用这样一个庞大的知识库，本文将设计完善并最终上架一款线上的社交型App，支持各俱乐部用以管理本俱乐部成员，实现人员增减，俱乐部主席、俱乐部指导老师更替等管理工作。与此同时搭载News、Blog、Event三个主要模块，使得各俱乐部能够在平台内部发布近期俱乐部新闻，分享第三方博客文章、开放本俱乐部活动，并允许平台上的非本俱乐部成员参与。

经由此平台，原本极度松散，人员统计不完善，创新产出无存储，线下活动无记录，俱乐部之间无交流的状况必然将有所好转。更方便俱乐部主席和指导老师管理本俱乐部成员，也更方便苹果官方管理全国各俱乐部。

2.技术框架

2.1 iOS开发

本文基于iOS移动平台，因此在开发组件上选择以Swift构建前端，以SpringBoot + Mysql构建后端的基本框架。

2.1.1 Swift概述

在开发环境的选择上，前端采用SwiftUI + Combine 框架。SwiftUI框架与Combine框架均为WWDC2019上新提出的两种开发框架，搭配Xcode11，具有所见即所得的开发体验。只要引入SwiftUI包，并创建实现了PreviewProvider协议的Previews结构，开发者就能够实时的在XCode中看到当前文件最终呈现出的界面与动画效果。

Combine：Apple官方文档对Combine框架的介绍中提到：Combine是一种通过对事件处理操作进行组合（combine），从而实现对异步事件自定义处理的框架。这一点体现在Combine 的工作方式中。

Combine中最重要的三个角色：负责发布事件的Publisher；负责订阅并接收数据的Subscriber；以及负责对上游Publisher发布的数据进行包装和变换，使其能够被下游Subscriber使用的Operator。

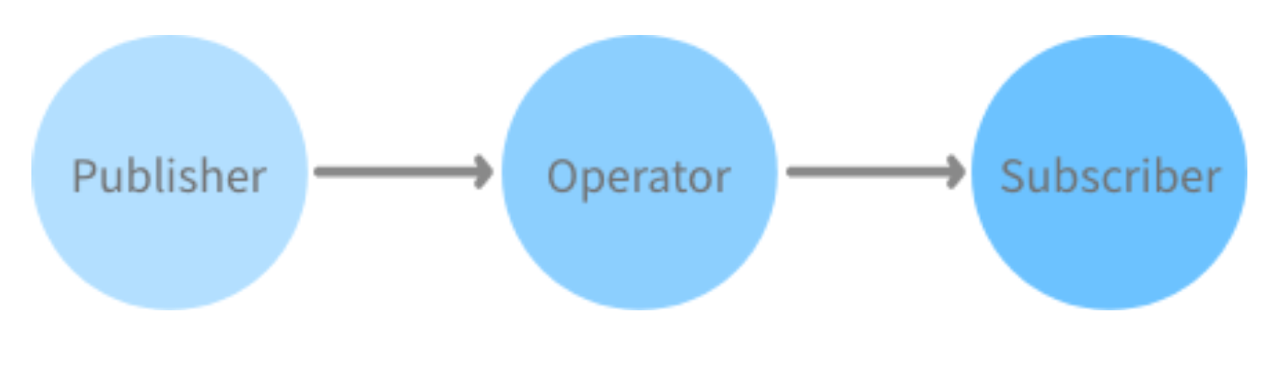


图2- 1 Combine工作流示意图

2.1.2开发模式

为了搭配较新的SwiftUI 与Combine 框架，本文将采用由王巍在《SwiftUI与Combine编程》一书中提出的类Redux框架，使用MVVM开发模式。

传统iOS开发中大多采用MVC（Model-View-Controller）模式。而该模式下“Controller过于繁重，难以维护”的问题早已饱受诟病。而MVVM（Model-View-ViewModel）则是在此基础上，经过对各组件工作量重新分配的改良。同时将View的状态和行为进行抽象化，将View视作Function(ViewModel), 即将View 视为函数产出，而ViewModel提供函数的参数输入，从而实现视图UI与业务逻辑的分离。

Model层：即数据模型层，定义数据类型，实现数据的增删改查，与后端数据库进行数据交换。

View层：即视图层，是用户最终所见的内容，展示数据，同时监听用户事件，但无法直接响应，且无法直接与Model层进行通信。

ViewModel层：连接View层与Model层，与View层实现绑定，从而在View收到用户事件时，将事件传递给Model层，反之也在Model层发生改变时，将具体的变动内容体现到View层。

其实在MVC – MVP – MVVM 三种框架中，MVP和MVVM都是MVC模式的改进版本，不同于MVP模式，MVVM能够利用数据的双向绑定，使得ViewModel变动后，View自发进行更新。而SwiftUI + Combine 框架之所以能够契合MVVM，也是由于Combine中加入了诸如“@State”、“@ObservableObject”等修饰前缀，实现了数据的双向绑定。

而由王巍提出的Redux for SwiftUI 框架，则恰到好处的运用了以上所提到的各框架与模式的优点。

在该框架下，app被视为一个状态机，所有的视图UI都由状态决定。而所有的状态都存放在一个Store对象中，View无法直接改变State，即无法直接更改视图内容，只能通过响应时间，发送Action到Store，Store中的Reducer根据Action以及当前State，生成新的State，并最终替换原有State。而在搭载了双向数据绑定的框架下，State改变意味着视图UI也自动更新，从而在单项数据流的最终实现视图View的更新。

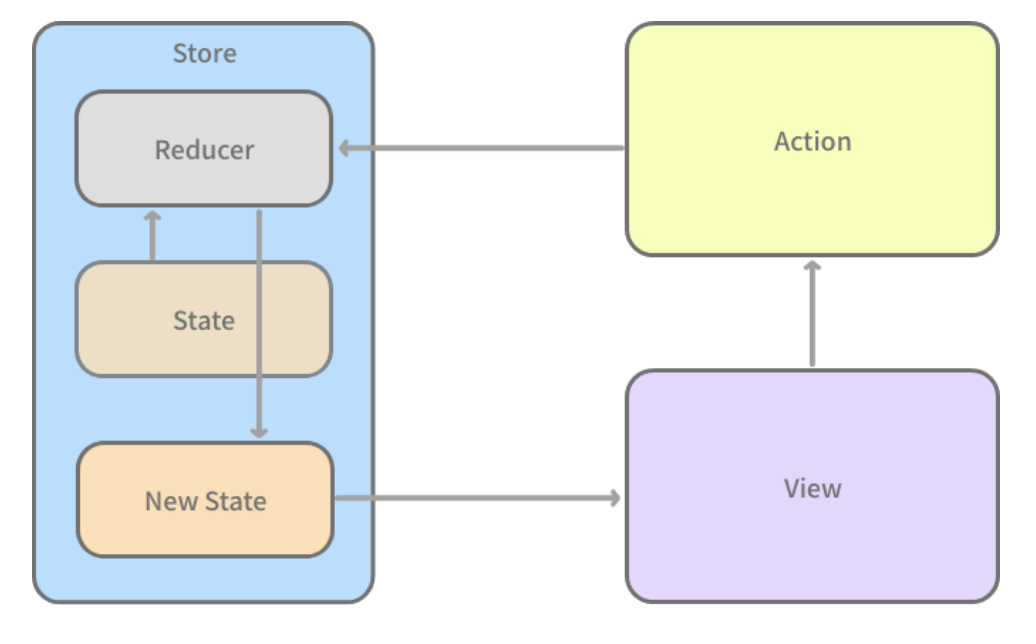


图2- 2 Redux for SwiftUI的数据流动示意图

3.需求分析

3.1功能性需求建模

3.1.1注册

业务概述：每个俱乐部都独有长为十位，仅包含数字和大写字母的**注册码**，用户在注册时需要先输入注册码，根据注册码是否有效，判定用户是否有资格注册，以及注册后归属于哪个俱乐部。

使用者：未注册的新用户

输入要素：俱乐部注册码、注册邮箱、用户昵称、登陆密码，重复登录密码

基本事件流：打开应用后，若判读为第一次使用本App，就直接跳转至注册界面，由用户主动输入所需要素后，点击注册按钮，执行注册事件。

输出要素：若注册成功，输出注册成功提示文本，跳转至登陆界面，并自动填充注册邮箱，同时将新用户信息返回至客户端，创建新用户。若注册失败，则弹出失败的本地描述文本。

异常事件流：用户给出俱乐部注册码无效；登陆邮箱格式不正确；服务器通信受阻；登陆邮箱已被占用，重复登录密码与登陆密码不相同

3.1.2登陆

业务概述：用户根据提示输入登陆要素，并按下登陆按钮；或者直接以游客身份登陆

使用者：已注册用户、游客

输入要素：登陆邮箱、登录密码

基本事件流：应用启动后，或检测当前登陆用户为空，就直接显示登陆界面，由用户输入所需要素后后，点击登陆按钮，执行登陆事件。

输出要素：若登陆成功，输出提示文本，并跳转至主界面，同时将本地User设置为登陆用户，加载主界面展示信息。

异常事件流：登陆邮箱格式不正确；登陆邮箱无记录；密码不正确；服务器通信受阻。

3.1.3 主要功模块

1. News模块

业务概述：各用户均有权发布News，发布后的News将被加入数据News库，根据用户的权限等级，从服务器中提取出对该用户开放的News内容，将其封装后展示到News主界面上。实现俱乐部交流的第一层：新闻共享

使用者：所有用户（为防止非俱乐部信息的混入，游客没有发布内容的权限）

输入要素：登陆成功，且用户为非游客身份

基本事件流：登陆成功后，登陆事件将作为Load News事件的发起节点，后台根据当前User的权限等级，加载可开放内容。

输出要素：若对该用户开放的News库不为空，则将从后台加载得到的News展示于News Page界面

异常事件流：服务器通信受阻；News数据加载失败；

2. Blog 模块

业务概述：用户同样可以发布Blog，Blog应用Apple 的PresentationLink组件，封装一个url链接，并自动加载链接内部的图片，标题等信息，实现富文本分享动态。同样的，根据用户的用户权限，后台将加载对应权限下开放的Blog内容，并将其展示给用户。从而实现俱乐部交流的第二层：知识分享

使用者：所有用户

输入要素：成功登陆，且身份为非游客用户

基本事件流：同News模块，成功登陆后，后台根据当前用户的用户信息，加载可开发内容。

输出要素：展示Blog列表

异常事件流：服务器通信受阻；Blog数据加载失败；URL无效。

3. Event模块

业务概述：用户可以在Event模块发布本俱乐部近期活动，Event模块接入Swift的EventKit组件，能够与iPhone原生日历、备忘录、reminder等应用进行直接的互动。用户发布俱乐部事件时，可以选择是否对外开放，若对外开放，则该Event对其他非本俱乐部成员也同样可见，从而达成第三层的俱乐部交流：活动开放。

使用者：所有用户

输入要素：Event活动的具体title、content、以及持续时间等

基本事件流：同上News和Blog模块，在用户后登陆后根据用户权限自动加载，主要以日历Pad方式展示，有活动的特殊日期将特别标注。

输出要素：展示Event页面

异常事件流：服务器通信受阻，Event数据加载失败。

3.1.4 查看新闻详情

业务概述：显然，由于新闻列表主要起罗列新闻信息的作用，因此用户需要选择并点击新闻Card，从而进入新闻的详情页面，查看所有图片和正文。

使用者：所有用户

输入要素：点击News Cell组件

基本事件流：点击News Cell组件后，弹出News Detail界面，覆盖当前主界面。

输出要素：展示News Detail页面

异常事件流：UI加载异常

3.1.5查看Blog详情

业务概述：Blog是由用户主动分享至平台的第三方网站链接，其他用户可以通过点击Blog Card，唤醒手机的原生Safari应用，查看完整的博客或文章

使用者：所有用户

输入要素：点击Blog Cell

基本事件流：点击Blog Cell 后，自动跳转至Safari软件，展示该Blog所包含的详情网页。

输出要素：原生Safari应用的详情界面

异常事件流：当前Blog中的url无效（过期，格式错误）；Blog加载异常，Safari唤醒异常。

3.1.6 发布News

业务概述：发布News事件对所有用户均可见，但在具体发布时，将根据用户权限，排除游客用户。正常用户可以在发布News界面编辑标题，正文，添加图片，添加News标签，最终将其发布至服务器。

使用者：非游客用户

输入要素：标题；正文；附加图片；News标签

基本事件流：

1. 点击Post按钮，唤出Post News界面
2. 输入新闻标题（必选）
3. 输入新闻正文（必选）
4. 点击添加图片按钮，添加若干图片（可选）
5. 输入新闻标签（可选）
6. 点击确认按钮，等待完成发布。

输出要素：发布成功提示文本

异常事件流：编辑缺少标题；编辑缺少正文；游客用户发布（将被拦截）；服务器通信异常

3.1.7 发布Blog

业务概述：同News，Blog发布界面和按钮对所用户开放，但游客用户将被拦截。用户可以在发布界面编辑输入文章的链接，其余信息均由系统自动补充。

使用者：非游客用户

输入要素：Blog的目标url

基本事件流：

* 1. 在系统内点击Post按钮
  2. 由外部链接通过URL Schema自动唤醒

（2）点击确认按钮，等待发布完成

输出要素：发布成功提示文本

异常事件流：游客用户发布（被拦截）；缺少目标url；服务器通信异常

3.1.8 发布Event

业务概述：非游客成员可以发布本俱乐部活动，参与其他俱乐部选择对外开放的活动，游客用户只能发布私人活动，将Event模块作为暂时日历。

使用者：所有用户

输入要素：标题；活动内容；是否对外开放

基本事件流：

1. 在Event界面选择特定日期。
2. 点击弹出的发布按钮。
3. 输入信息。

输出要素：发布成功提示文本

异常事件流：服务器通信异常；UI更新延迟

3.1.9 修改个人信息

业务概述：非游客用户在服务器存有特定数据条，登陆成功后，本人能够在Profile界面修改包括Username，gender，contact等个人信息，同时也能够更换本人头像。

使用者：非游客用户

输入要素：进入Profile界面

基本事件流：

1. 进入Profile界面
2. 选择修改信息

（3.1）输入新文本

（3.2）选择新图片

（4）点击确认按钮

输出要素：发布成功修改文本

异常事件流：游客用户（被拦截）；服务器通信异常；UI更新延迟。

3.1.10查看个人发布历史

业务概述：非游客用户能够查看本人发布后成功存储在服务器上的News和Blog

使用者：非游客用户

输入要素：点击对应“库按钮”，唤出界面

基本事件流：

1. 点击并唤醒历史界面
2. 查看历史信息
3. 选择并删除（可选）

输出要素：展示对应详情，若有删除操作，则在完成后弹出操作成功提示。

异常事件流：游客用户（被拦截）；服务器通信异常；UI更新延迟

3.1.11 查看俱乐部状态

业务概述：在本系统中，非游客用户必然属于某一特定俱乐部，该用户能够查看本俱乐部成员信息。

使用者：非游客用户

输入要素：点击“俱乐部按钮”，唤出界面

基本事件流：

1. 点击并唤醒俱乐部成员列表
2. 点击某一特定成员查看详情信息
3. 踢出俱乐部成员（仅限俱乐部主席、俱乐部指导老师、系统管理员）
4. 更换俱乐部主席（仅限指导老师和系统管理员）
5. 更换俱乐部指导老师（仅限系统管理员）

输出要素：展示对应成员列表，若有删除操作，弹出操作成功或失败提示文本。

异常事件流：游客用户（被拦截）；服务器通信异常；UI更新延迟

3.1.12 查看所有俱乐部

业务概述：非游客用户

输入要素：点击“全俱乐部按钮”，唤出界面

基本事件流：

1. 唤出界面
2. 查看俱乐部列表
3. 查看指定俱乐部的联系方式等详细信息（可选）
4. 编辑指定俱乐部的指导老师或主席（仅限系统管理员）
5. 新增或删除俱乐部（仅限系统管理员）

输出要素：展示对应俱乐部列表，若有增删改操作，弹出结果提示文本

异常事件流：越级操作（将被拦截）；服务器通信异常；UI更新延迟。

3.2 非功能性需求建模

3.2.1性能

据统计，截止2019年11月，Apple公司已经在国内包括浙江大学、同济大学等总共43所高等学校设立了创新俱乐部[[1]](#footnote-1)。考虑到目前的用户体量和总访问频率。即使不租用大型云服务器，应用的访问速度等外部限制瓶颈也将具有天然较高的阈值。

因此，程序的内部效率瓶颈将是目前的主要性能瓶颈。因此，本项目将严格按照上文提及的类Redux 架构，采用标准化的，契合Swift语言环境的MVVM设计模式，从而为程序的内部运行保驾护航，规范数据流向，保证程序响应事件的实时性。

3.2.2 易用性

本程序在界面设计过程中参考了市面上的主流社交型应用（如微信，QQ）等，参考了iOS系统原生日历等程序，但同时保留了高度的独立设计性和充足的动画过渡，保证用户在使用时既不会因为界面复杂而烦恼，也不会因为UI界面简陋而丧失使用兴趣

2.2.3 安全性

本程序在开发和日后运维中均主要租用阿里云服务器，阿里云OSS对象存储服务，自带安全保障。同时在发送请求时对敏感数据进行隐蔽化处理。保证重要信息的隐蔽性。

4.概要设计

4.1 总体设计

4.1.1 开发/运行环境

开发环境：macOS.10.15； Xcode.11.4；

服务器开发环境：阿里云轻量级服务器

开发语言：Swift 5

运行环境：iOS13+

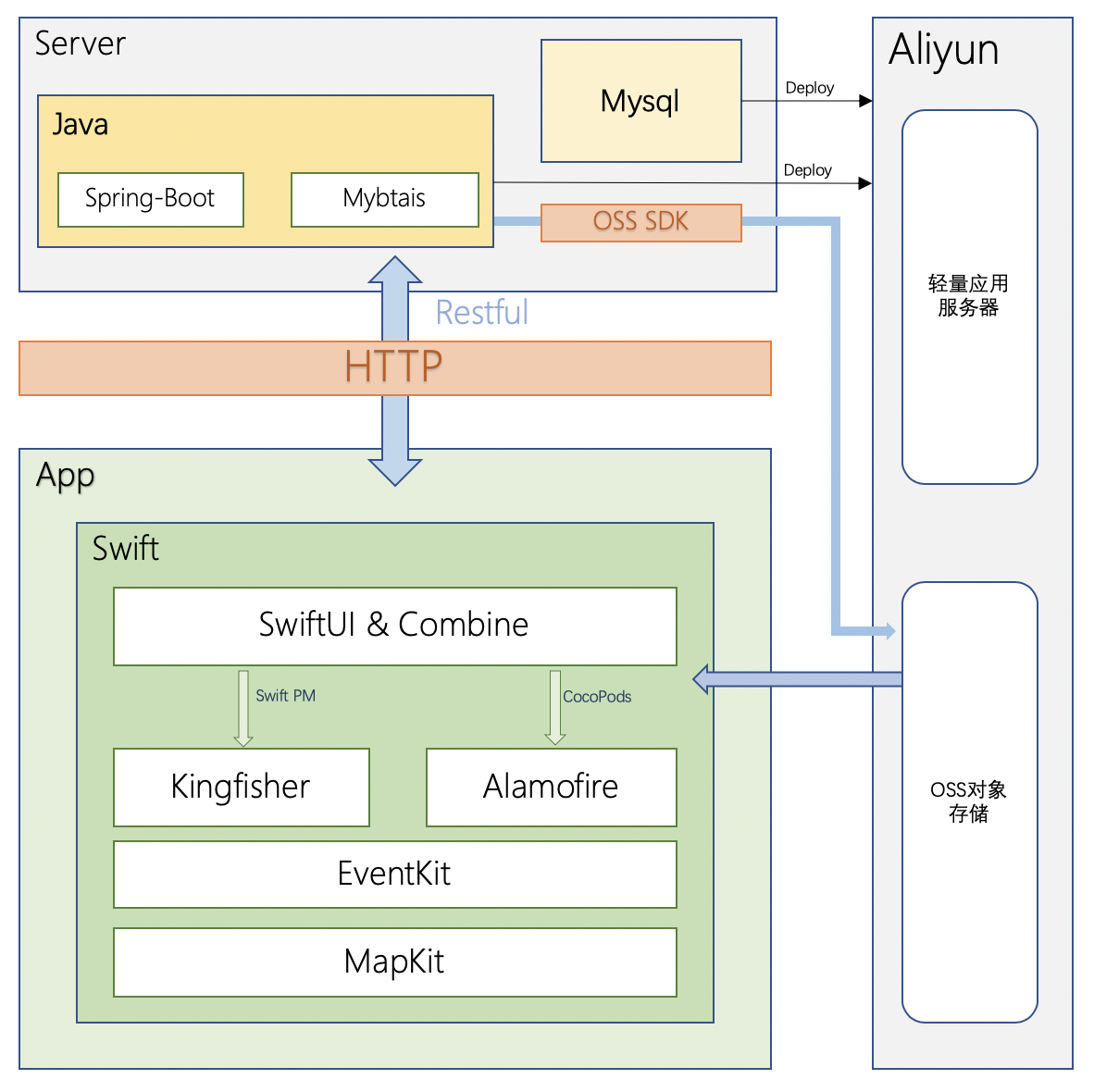
4.1.2技术架构设计

图4- 1 技术架构设计示意图

本项目是典型的基于移动应用平台，App端由Swift语言开发的应用型程序，应用公布于WWDC2019的SwitfUI和Combine框架，整合Kingfisher、Alamofire等第三方库。后端部分采用Spring-Boot框架开发，应用(SSM SpringBoot - Springboot MVC – Mybatis)模式，数据库采用Mysql关系数据库。租用阿里云旗下的轻量应用服务器做开发服务器，OSS对象存储服务器存储News模块下的新闻图片，便于手机端的上传和下载。

App端与Server端采用REST通信模式，由服务器暴露RESTful服务接口，App端通过发送HTTP请求，实现与Server的通信。

不同于第三方的CocoPods，Swift Pack Manager（Swift PM）是Apple官方刚刚推出的，且直接绑定于XCode内部的全新包管理组件，但因推出时间过短，大部分第三方库仍没有适配该组件。因此本文仅利用Swift PM整合Kingfisher库，用以加载网络图片，并使用原有的CocoPods整合Alamofire库，以求能够用更优雅的方式上传手机端本地图片到服务器。

4.1.3 部署架构设计

图片包含 游戏机

描述已自动生成

图4- 2 部署架构设计示意图

由于App端由Xcode直接打包部署到手机实体，因此需要部署的只有Server 部分和Mysql数据库部分，二者均由Docker容器部署在ip地址为47.100.54.7的阿里云服务器上。

4.2 接口设计

4.2.1用户接口

1.基本操作：用户在正式使用App前需要注册账户，注册时需要提供特定的，与俱乐部成一对一映射关系的RegisterCode，RegisterCode用来判定用户是否有注册权限，同时判断该用户注册后归属于哪个俱乐部；注册成功后用户凭借注册邮箱和密码进行登陆，否则只能以功能受到限制的游客用户进行登陆。

2.News模块：用户能够在主界面查看对自己权限开放的News列表，点击后可进入News Detail界面，查看新闻呢详情。同时能够自主发布新的News。发布时可以附带图片，话题相关的信息。

3.Blog模块，基本同于News模块，用户可进行浏览、查看详情，发布等操作。

4.Event模块，游客用户仅能当作第三方日历使用，直接关联系统日历，系统Reminder，系统Alarm。非游客用户能够查看对自己权限开放的，一年以内的俱乐部Event事件，并自主选择是否参与Event活动。Event仅可由俱乐部主席和俱乐部指导老师发布，发布后本俱乐部成员自动参与。发布时，发布人能够选择该活动是否对外开放，若开放，则其他俱乐部用户也有权限查看该活动，并可自行选择是否参与。

5.个人信息：用户有权修改包括用户名、性别、头像、联系方式等个人信息，但不能修改用户ID。

4.2.2外部接口

Server端：服务器与数据库通信中的SQL语句；OSS对象存储服务器的上传图片接口。

App端：由Kingfisher提供的图片缓存机制；OSS对象存储服务器的图片公开下载链接。

4.3界面设计

本节内容展示为项目开发初期，利用Swift自带Bundle存储与加载本地JSON文件设计了UI原型。最终产品内容可能与本节图片展示略有出入。

4.3.1注册/登陆界面

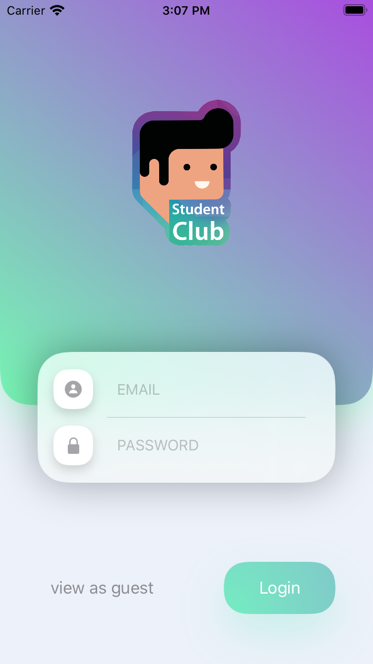
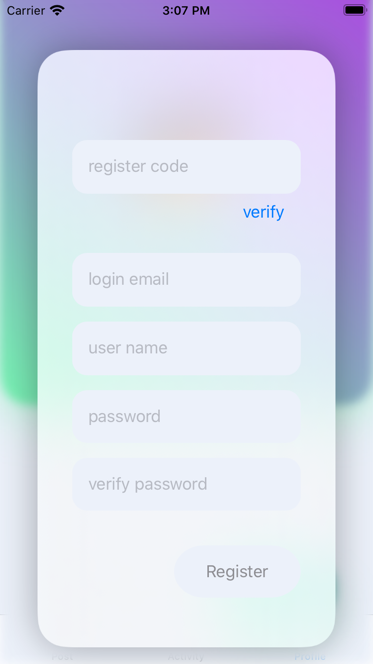


图4- 4 登陆界面设计示意图

图4- 4 注册界面设计示意图

本项目为登陆用户user属性创建了@FileStorage注解，将user属性作为持久数据，自动读写于客户端的本地文件中。同时实现了首次打开程序检测和持久登陆两项功能，提高了用户的使用体验。具体实现见下一章节。

4.3.2 News/Blog 展示界面

手机屏幕截图

描述已自动生成社交网络的手机截图

描述已自动生成在用户登陆成功后，系统将用户登陆反馈作为加载事件起点，加载对该用户开放的News和Blog分享。系统将在在登陆成功后直接跳转至该界面。

图4- 5 主界面Blog浏览示意图

图4- 6 主界面News浏览示意图

4.3.3 News/Blog 详情界面

用户若对主界面Card型News与Blog感兴趣，可点击News或Blog组件，切换至详情界面。

不同于外部浏览型Card，News详情界面详细展示News包含的所有图片和文本内容。Blog详情内嵌Swift原生PresentationLink组件，由系统先根据Blog提供的url加载MetaData内容，点击后可直接由Safari View跳转至url对应网页。



图4- 7 News详情示意图



图4- 8 Blog详情示意图

4.3.4 News发布界面

手机屏幕截图

描述已自动生成手机屏幕截图

描述已自动生成 用户可在News发布界面内嵌SwiftImagePickerView组件，支持用户从系统相册中选择图片，作为News内容图片，上传至服务器后，由服务器存储至阿里云OSS对象存储服务器中。

图4- 9 选取图片示意图

图4- 10 News发布示意图

4.3.5 Event展示界面

同News和Blog，Event也作为用户自动加载内容，在用户登陆成功后自动加载。展现形式为按月分页式日历，根据Year-Month-Week-Day 逐层分集，在日级关联当日Event活动，根据用户是否参与展示不同的状态。同时用户也能够通过点击“flag”按钮选择是否参与或退出该活动。

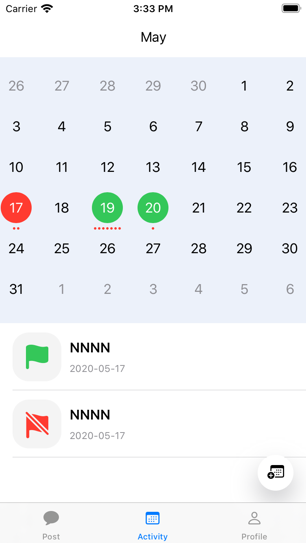


图4- 11 Event日历示意图

4.3.6 Event发布界面

Event发布模块内嵌Swift下的EventKit组件，调用EventKit所包含的EKEventEditView直接在发布事件的同时直接将事件添加至系统日历。同时能够选择是否设定系统闹钟。

手机屏幕截图

描述已自动生成

图4- 12 Event 发布示意图

4.3.7 发布历史界面

发布历史记录该用户使用至今发布过的News，用户可以在该界面查看详情，或者选择删除该条内容。内容将实时反馈至服务器，作用于数据库。



图4- 13 News发布记录示意图

5.详细设计规约

本节内容继续围绕News、Blog、Event三大模块展开，阐述各个模块的数据结构与算法设计，阐述对于上文需求部分如果满足。同时，上文已经有所提及，本文在前端设计中采用的Redux for SwiftUI设计模式以“状态驱动UI”为核心思想，因此本节在进行详细设计时，亦会兼顾这一点，以状态 横跨MVVM中三层，阐述数据如何在Model和ViewModel中流动，并最终由“双向绑定”这一概念，将数据的变化体现在UI的改变上。

5.1 Store & AppState模块

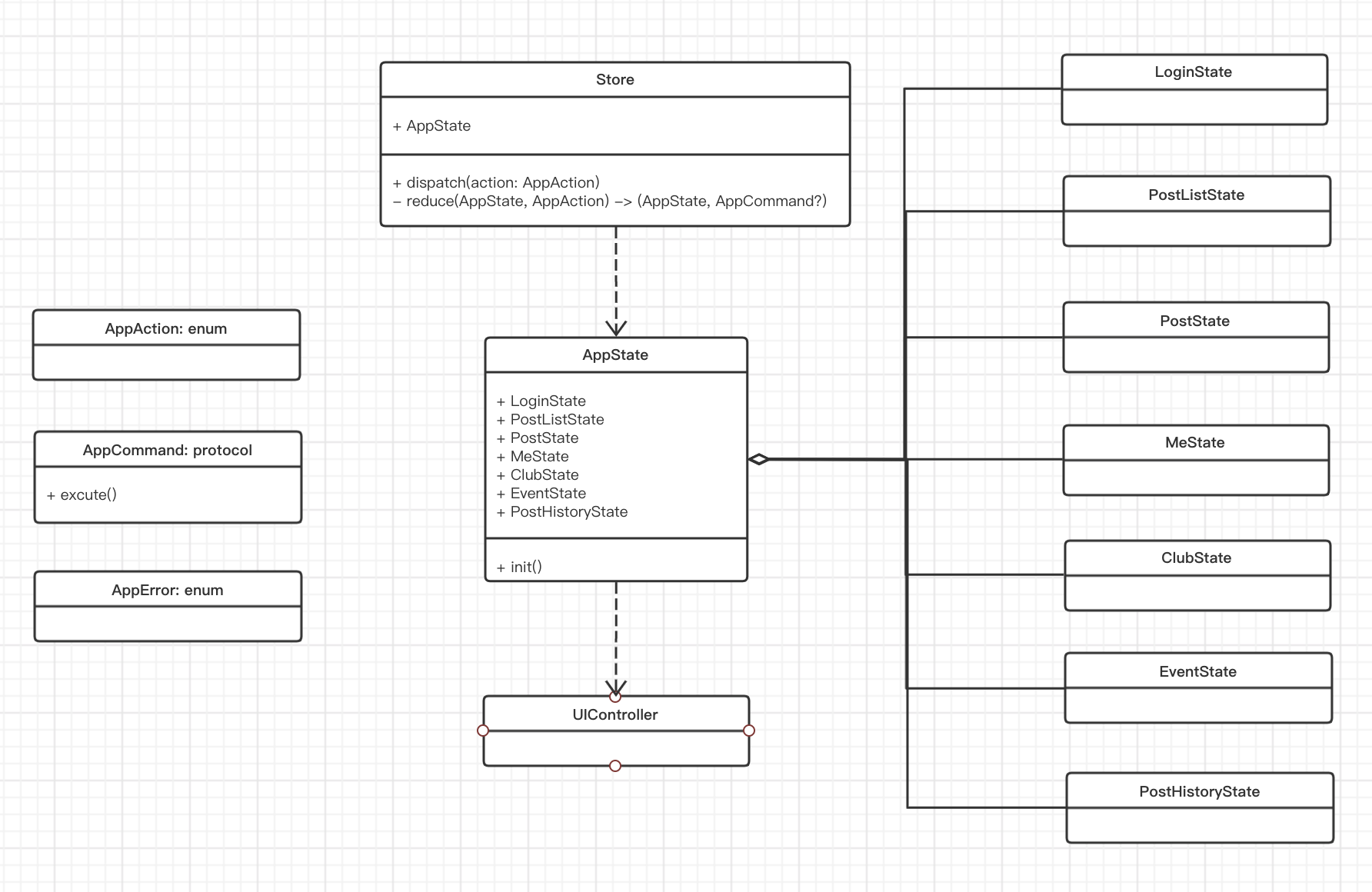
结构类图：

图5- 1 Store与AppState类图

主要功能：再次引用本框架的核心设计思想：“在Swift for Redux框架中，UI以状态驱动，相当于是状态的函数”。

为了高效，有序地实现以上中心思想，本框架将所有的State都存储在Store这一大类中，Store在单独项目中只有一个实例。而Store中的各种功能的实现依赖于所有State的协同作用。Store依赖于各个State。

Store具有两个主要函数：

1. dispatch函数：同字面意思，dispatch函数作为Store所有数据化事件的接收中转枢纽。所有通过枚举类型的AppAction和AppError，以及实现了AppCommand协议的各种Command来进行数据化处理后的事件，最终被全都被归纳为AppAction中的一个动作。Store实例在接收AppAction的动作请求后，将其本身（包括所携带的数据）统一调配至事件处理的下一站：reduce 函数。
2. reduce函数：reduce 原本在Swift中是一个常用的内置后调闭包，用于对各种类型的数据进行归纳和累加处理。在此框架中，经引申后的reduce函数接收两个参数：
   1. state: AppState[[2]](#footnote-2)。即AppState类型的state实例。Swift中的struct中各变量不允许改变，因此对某struct实例struct1.param1 进行赋值相当于直接进行struct1=struct2，因此该框架直接在reduce函数中将旧状态整体进行复制传递。
   2. action: AppAction。如上文所讲，该变量及其所携带的数据作为事件的描述者，以动作的方式告诉Store此时需要响应的事件类型和具体内容。

reduce在获取到以上参数后，根据情况，在旧AppState实例的基础上产生新的AppState状态实例，并在完成内容补正后直接将新的AppState与副作用动作副作用AppCommand打包为为tuple返回至枢纽站dispatch。

dispatch根据返回的数据详情，完成剩下的工作：首先将store实例中的旧状态实例state替换为reduce返回值中的新状态实例，完成系统状态迁移。其次为reduce返回的副作用指令AppCommand开启新的执行流。因为在Swift中，结构体struct具有值语义，在所有的传递过程中均以复制数据内容进行处理，因此reduce函数中的行为并不会影响到store实例中的原本状态，仅做状态计算机。限于篇幅，关于Swift的值语义和引用语义这里不详细展开。

上文中提到了AppCommand概念，AppCommand实际上是仅包含一个函数：excute(Store)的协议（protocol）类型。实际内容由实现该协议的各类具体AppCommand实现，下文将会有详细实例。在store的调度功能执行范畴中，AppCommand定义了实现某动作需要完成的“副作用”，如Http请求发送，UI绑定属性更新等。这里的“副作用”之所以要单独提取出来是因为：非功能性需求之一的实时性在移动性应用实际应用中对用户体验具有极大的影响。而AppCommand中携带的大部分是需要在非主线程完成的工作，如果从服务器或本地Bundle加载数据等。如果将这些动作纳入主线程，作为后果，首先是程序的实时性无法得到保证，服务器负载过高造成数据传输瓶颈等任何情况都将大幅增加程序的响应时间。其次，因为作用结果具有不确定性，若将其置于主响应流，将会产生大量的多余工作用于进行容错设计。

因此，这些动作内容统一归入AppCommand，将真正的执行放置在dispatch函数更新AppState之后。在后台，AppCommand执行成功后，将调用代表动作执行结束的新的AppAction，引发新一轮的dispatch – reduce – 更新state工作流。

不过，并不是所有的事件在响应时都有副作用需要执行，因此在结构类图中，我将reduce函数返回值中的AppCommand中做了(?)可选值处理。可选值Optional在Swift中是一个非常重要的概念，因为Swift是强类型语言，普通变量在正常情况下不允许为空（nil），对又需要的情况，采用在类型名后加“？”，将该实例标记为可选值类型，在使用时，如果不进行解包，该值作为可选值类型，或者等于空值nil，或者等于一个Optional包含的具体类型值。同样，关于Swift的解包行为，以及AppCommand的纯副作用详情，在此均不做赘述。

5.2 News模块

结构类图：

图5- 2 News模块类图

模块功能概述：News模块主要涉及两个State：PostListState 和PostState。PostListState中用PostListViewModel记录并维持当前展示列表的状态Model，即在本MVVM框架中真正驱动UI的模型部分，而在PostListViewModel下又包含了以NewsViewModel作为元素类型的数组，存储所有待展示的News数据。同时PostListViewModel存储当日日期，作为News展示部分的header内容，在ViewModel的初始化过程中自动更新。

PostListState在保存所有待展示内容的同时，也会存储作为状态属性存在的detailNews：NewsViewModel?，用于标记当前正于详情页面展示的是哪一条具体News。该值为nil时表示用户并没有打开新闻详情界面。

之所以这样设计，是因为在声明ViewModel等变量时，需要加上诸如@Published的修饰符。得益于Combine框架提供的这些特殊修饰符，被该修饰符修饰的属性，将自动获得一对附加的getter和setter函数，在该值进行变更时，系统将自动调用程序生命周期中处于较为底层的，用来对SwiftUI框架包裹下的UI界面进行更新呢。这也就是SwiftUI与Combine框架之所以能够实现UI与ViewModel双向绑定的核心内容。底层具体实现不在此章进行详细解读。

回到PostListState部分，抛开View层，该模块在数据层上的功能实现主要可以归纳为加载News数据，发布News数据以及查看详情News。

1. 加载News依赖于AppAction下的LoadNews详细case。该action捕获当前用户的权限等级，创建LoadNewsAppCommand，最终通过在底层订阅LoadNewsRequest内的publisher（关于Combine框架下的Publisher等组件的实现与订阅等操作将在下一章详细阐述），该publisher通过发布HTTP的GET请求，获取当前服务器中对该用户权限开发的News数据条，将其返回，并在publisher内部有Swift原生JSONDecoder进行解析。解析后返回经打包后的NewsViewModel列表，PostListViewModel由此更新其属性viewmodel，在经由双向绑定构建的自发数据流对UI进行更新。

以上就是SwiftUI与Combine基本任务流的概述。

1. 发布News的状态由PostState保存，事件依赖于AppAction下的PostNews case。事件打包PostState下的title等数据，将其封装为完整的POST请求参数Body，并通过PostNewsCommand将其传递给PostNewsRequest，最终通过POST请求的方式将News发布至服务器，再经由服务器，将News中包含的图片等文件，存储在阿里云OSS对象存储服务器内，并设置权限为公开访问。
2. 展示News详情的状态同样保存在PostListState中，根据上文提及，由可选值属性detailNews: Optional(NewsViewModel)记录。View层在捕获到News预览组件的点击事件后，将在响应事件流的中段定位该News在NewsViewModelList中的存储位置，并将其复制传送最终赋值给State下的detailNews属性。再一次，得益于双向绑定，detailNews 的赋值事件引发了View层的更新行为，View层将唤出详情展示界面，并展示当前详情新闻的具体内容。

在实现以上功能的同时，PostListState和PostState分别存储了一个Bool类型的属性：isLoading和isPosting，这两个值表征当前程序是否正在执行加载和发布的事件。在执行期间，即两Bool值分别为true期间，View层仍然会捕获新的事件流发起事件，但会在具体action的reduce处理函数部分被拦截，从而实现请求的去重化处理，提高系统容错，提升用户的实际使用体验。

此外，PostListState和PostState分别都还存储了一个AppError属性，用于记录该State管辖范畴下，某个动作的响应过程中是否出现了某种错误。所有的错误均由枚举类型AppError定义并封装。对于Error的处理，同detailNews，具体的error属性是一个经可选值包装的AppError类型，在大部分State的大部分error属性从nil转为详细AppError时，View层将会自动弹出Swift原生的Alert组件，展示对应error的本地化文本描述。从而实现错误处理，进一步提高用户体验。

5.3 Blog模块

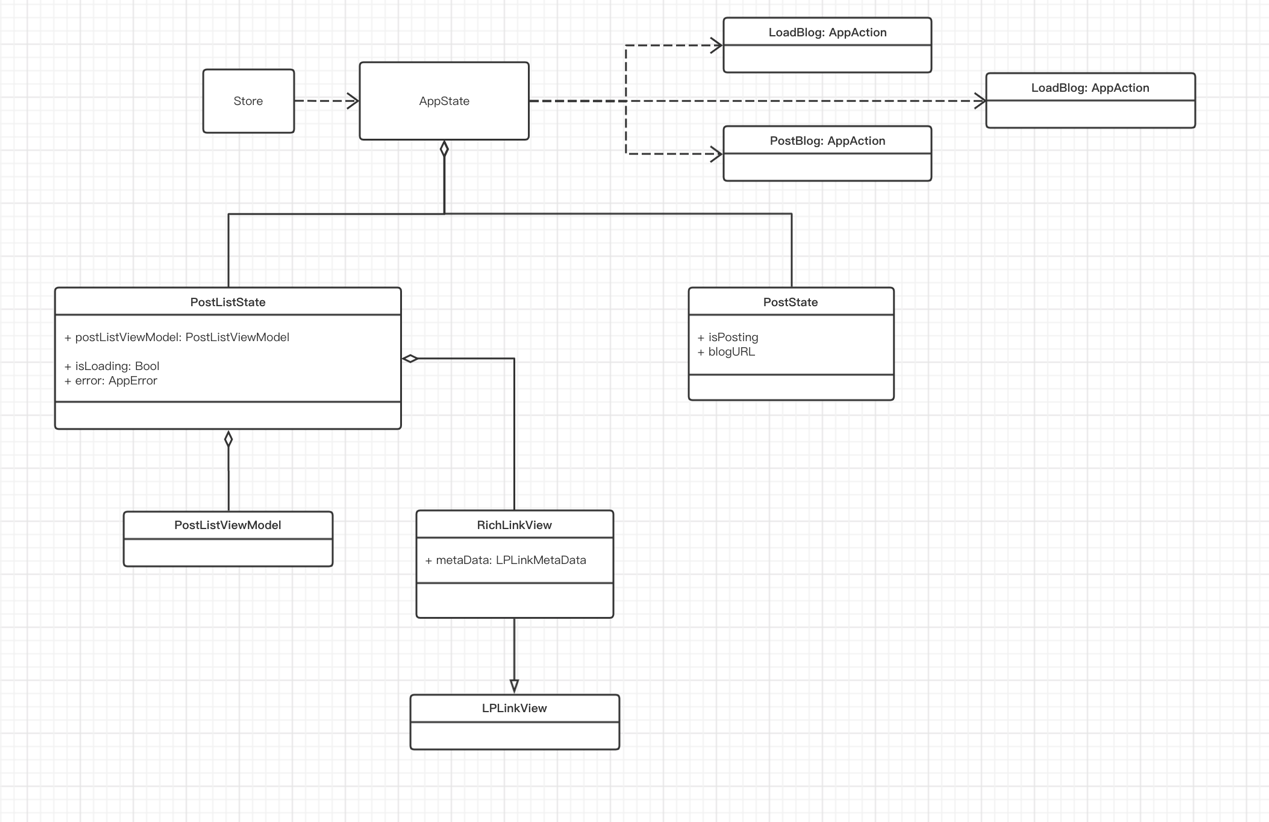
结构类图：

图5- 3 Blog模块类图

模块功能概述：Blog模块与News模块整体实现类似，都由PostListState和PostState进行管理与存储。并且同样能够在数据层归纳为三个主要功能方面：1）加载Blog数据。2）发布Blog。3）展示Blog详情

1. 加载Blog数据：该动作由AppAction下的LoadBlog case进行封装，同样捕获当前已登陆用户的权限等级，通过LoadBlogAppCommand调用LoadBlogRequest中的publisher，订阅该服务使得PostListState在提供了当前用户权限后能够获得服务器上存储的，对该用户开放的Blog内容。
2. 发布Blog：Blog本质上属于第三方技术性文章或博客分享，在数据层的最底部封装的是对应网站的url链接。同发布News时的各种title等信息，该数据将由PostState捕获并提供，具体提供方式，是经由与View层的Textfield组件绑定的具体属性值存储，再由PostBlog Action捕获。相比于News，Blog中用户信息的封装占比更大，而用户信息又是存储在与PostState同一层级的LoginState属性值中的。因此，获取用户的信息就必须由PostState和LoginState共同的父节点：AppState来提供，具体体现形式就是在PostBlog Action的执行过程中，通过当时的AppState实例来进行捕获。至此，这套框架在数据的流动上完成了闭环，实现了数据的单向流通规范化。
3. 展示Blog详情：在2）中就有提及，Blog的底层数据模型中，抛开用户的信息，主要是一个指向博客或文章对应地址的url链接，因此，展示该文章的的最好方式就是跳转至链接所指的内容。本文在经过调研与实际测试后，最终在“内嵌SafariView，将浏览器嵌入应用”与“应用Swift内置新组件LinkPresentation”中选择了后者。因为在SwiftUI与Combine框架中，View层不再显式地提供以往iOS版本中各生命周期节点的调用函数，因此同一iOS系统下不同应用间的跳转行为变得更加安全，其次，LinkPresentation中封装的metadata数据能够直接被上层组件调用，在可扩展性方面具有更加强大的优势。最终，本文在LinkPresentation的基础上，根据SwiftUI提供的Representable协议，实现了客制化的RickLinkView，字面意思，在实现了富文本内容展示的同时，支持点击跳转，直接与系统原生Safari接力。不过，为了实现富文本内容的单独加载，需要在原有Load请求事件流中隐式的新增一系列Load请求，且具体加载事件流的发起事件也与上述两种有所不同，这一部分也将在下一章中详细阐述。

5.4 Event模块

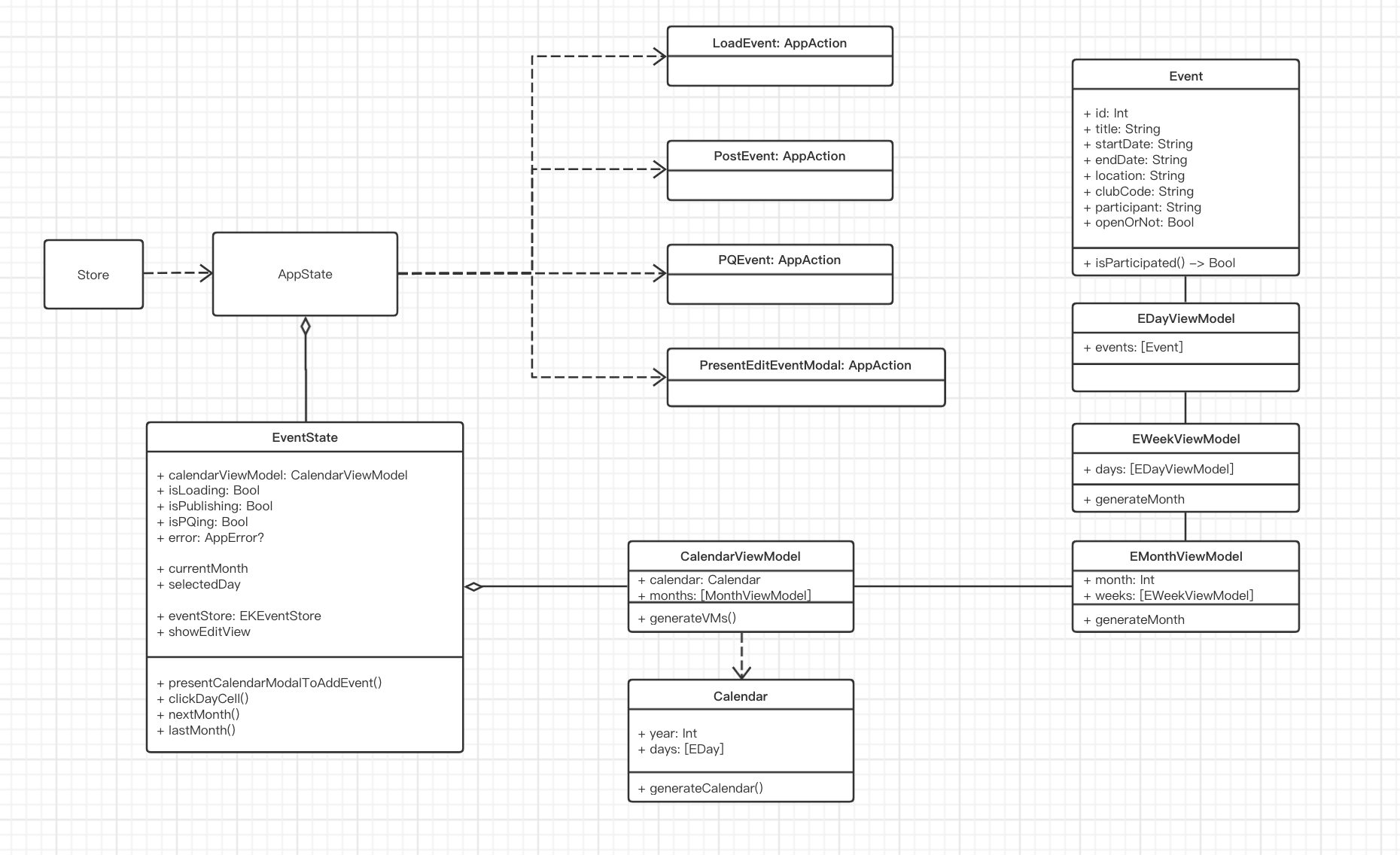
结构类图：

图5- 4 Event模块类图

模块功能概述：虽然Event和News、Blog一样在数据层都是加载、发布、详情三种情况，但因为实现上的不同方式，Event相比于News和Blog会复杂的多。

一方面，为了契合活动事件所特有的高时间相关性，本项目将Event的展示形式定为按月分页的日历界面，以至于权限不足以发布活动的用户也能够将该界面作为日历来使用。另一方面，在底层实现的中，Event模块应用了Swift原生EventKit组件，在添加事件的时候又自然内嵌了MapKit组件，层层相扣下，不得不将event的存储放在独立于其他两模块的EventState状态中。

1. 日历的展示。在旧版的Swift生态环境中，有很多基于Objective-C或原生Swift的优秀第三方日历与许多其他的组件内容。但由于SwiftUI与Combine框架在View层的接口部分有较大改动，因此若想调用这部分组件，需要额外的工作量来进行接口转换，如上一节中提到的Representable协议。且转换后View层功能受限，因此出于对可扩展性和可维护性的考虑，本文的日历系统是采用现有框架设计的客制化日历，具体数据Model为图中Calendar类。Calendar类根据当年的闰平规则，采用两组不同的日期计数数组，计算出对应平闰年中具体日期的对应工作日状态，日历显示状态等。在View层基于原生ScrollView，实现页式滑动界面，并实现对当前日期的监控与捕捉。与Calendar模型同层级的是12个MonthViewModel，MonthViewModel包含WeekViewModel，WeekViewModel包含DayViewModel，层层递进，最终在DayViewModel中精准的记录当日的View状态与Event事件。

在具体展示时，为了描述用户与当前活动的参与关系，根据Event实例中的participant属性，判断用户是否参与当前活动，并将具体的事件参与信息展示到EventCell上（见4.3.5节图4-11），同时用户能够点击对应按钮，轻松的选择是否参与该活动。

1. 事件的加载。Event的加载同样依赖于由LoadEventAppCommand提供的LoadEventRequest，对结果的订阅方式与News和Blog基本没有区别。但在服务端对数据进行提取时，根据业务逻辑的不同，比起由权限等级设计规范概括的News和Blog筛选机制，Event的拉取在权限上讲究的是该活动是否对外开放。因此，在从数据库提取数据时，提供申请用户的俱乐部代码，根据俱乐部代码，获取1）本俱乐部内的活动；2）其他俱乐部中，在发布时被选择为对外开放的活动。同样，Load行为的主体由EventState的父节点AppState承担，提供用户信息。
2. 事件的发布：为了能够联动手机上的原生Calendar系统，在事件发布这一动作中，不可避免的需要整合EventKit组件。EventKit是iOS原生Kit之一，提供系统日历、reminder以及Alarm等系统，同时内部调用MapKit。2）中有所提及：传统Swift MVC框架下，以ViewController为核心，通过将父ViewController作为子View组件代理的方式，实现子View的所有数据层交互。而当前版本的MVVM模式直接剥离了ViewController层，将“把数据变动更新到View层”这一行为交给后台与底层的框架自发完成。但可惜的是，截止本文撰写，Apple仍然没有为其原生各组件提供从Swift到SwiftUI的接口迁移。因此要想使用EventKit等组件，仍需要将其作为第三方组件来看待，通过UIViewRepresentable和UIViewControllerRepresentable进行接口桥接，详细实现在下一章阐述。回到事件发布，虽然EventKit的引入带来了一些额外工作量，但好处是我们能够调用与EventKit共享数据流的EventKitUI包，在不添加更多的代码的情况下，使用原生的，自然符合iOS界面风格的EKEventEditView（见4.3.6节，图4-12）。我们要做的就是在实现Representable接口的事件流中段，捕获用户信息，通过共享的EventStore组件，在发布本地日历事件之后，根据Event数据，将数据流转码并创建新的Event事件，调用PublishEvent动作，将事件发布至服务器。
3. 事件的参与和退出：事件的参与退出动作依赖于AppAction下的PQEvent（participate & quit Event）。据图4-11，用户在日历系统中选中具体的某一天后，View层捕获点击事件，系统调用EventState下的clickDayCell 函数，该函数定位被点击日期，为EventState下的selectedDay赋值，最终View层通过selectedDay内容对展示表单进行更新。在当前状态下，系统自发调用Event的结构体检测函数：isParticipated，检测用户是否在该事件中的参与状态，并将其以Logo Flag的形式展现到View层内。用户可以根据事件描述，点击Logo Flag，切换自己的参与状态。在非参与状态下点击该按钮，将发起“参与事件流”，将自己列入Event实体的Participant属性中。反之亦然，在已参与状态下点击，将通过“退出事件流”，将自己从该属性中剔除。因为这一事件的事件流起点触发简单，比起数据的传输时间，View层的UI更新时间相对更长一些，所以需要更加严格的监听动作状态，限制事件流的冗余发起，保护服务器负载。因此，不同于其他大部分事件在Action响应过程中的过滤形式，该事件将在View层的On Tap Gesture中进行拦截，为该点击组件设立对状态属性“isPQing”的监听，仅在该值为false时对用户行为进行响应。

相比于News和Blog中大范围将数据模型包裹进ViewModel的设计，因为增加了与独立纯数据模型Calendar的互动和绑定，Event模块总体上比前两个模块更为复杂，这一复杂度主要体现在如何有限度的、规范的设立Calendar和ViewModel之间的耦合程度，如何将数据内容进行提取和无副作用绑定，又如何在进行数据复制的时候使用有限内存。而在实现了EventState的自身生态后，为了拓宽业务范畴，又需要对EventKit组件进行整合，在整合的过程中完成数据流的把控。

5.5 Club模块

图片包含 游戏机

描述已自动生成结构类图：

图5- 1 Club模块类图

模块功能概述：虽然在本项目的实际应用阶段中，用户视角下包括登录注册的各项行为功能都于俱乐部挂靠，但事实上用户在使用时与俱乐部的直接互动功能是比较少的。因为在本项目中，俱乐部这一概念很大程度上可以视作对用户实体的一条制约规则，更多的互动需要体现在设计规范和服务器实现规则上。

作为实际的功能，俱乐部模块可以归纳为两个分支：本俱乐部详情，以及其他俱乐部信息。在本俱乐部详情分支下，普通俱乐部成员可以查看自己所在俱乐部内的成员信息，身份被系统识别为俱乐部学生主席和俱乐部指导老师的用户能够对本俱乐部成员进行踢除等操作。在其他俱乐部信息分支下，用户能够查看当前系统录入的所有俱乐部缩略信息，包括名称、图标以及俱乐部联系方式等，为各俱乐部之间的现实交流提供接入口。

在技术实现角度，这两部分分别被封装为LoadMyClubMembersAction和LoadClubListAction两个Action，分别向服务器发送对应请求，加载数据，将其展示到View层，从而开放下一层级的用户交互领域。

5.6 User 模块

引言：归根结底，作为一款社交型应用，所有的动作响应、事件流动、数据结构最终都会归结到程序的核心结构：User模型上。所有的请求发起需要提供用户的信息，所有的数据上传需要绑定到用户词条，程序的所有行为都避无可避的需要User实体的参与。而且，程序的启动与运行，在起点处就依赖于用户的注册和登录。

结构类图：

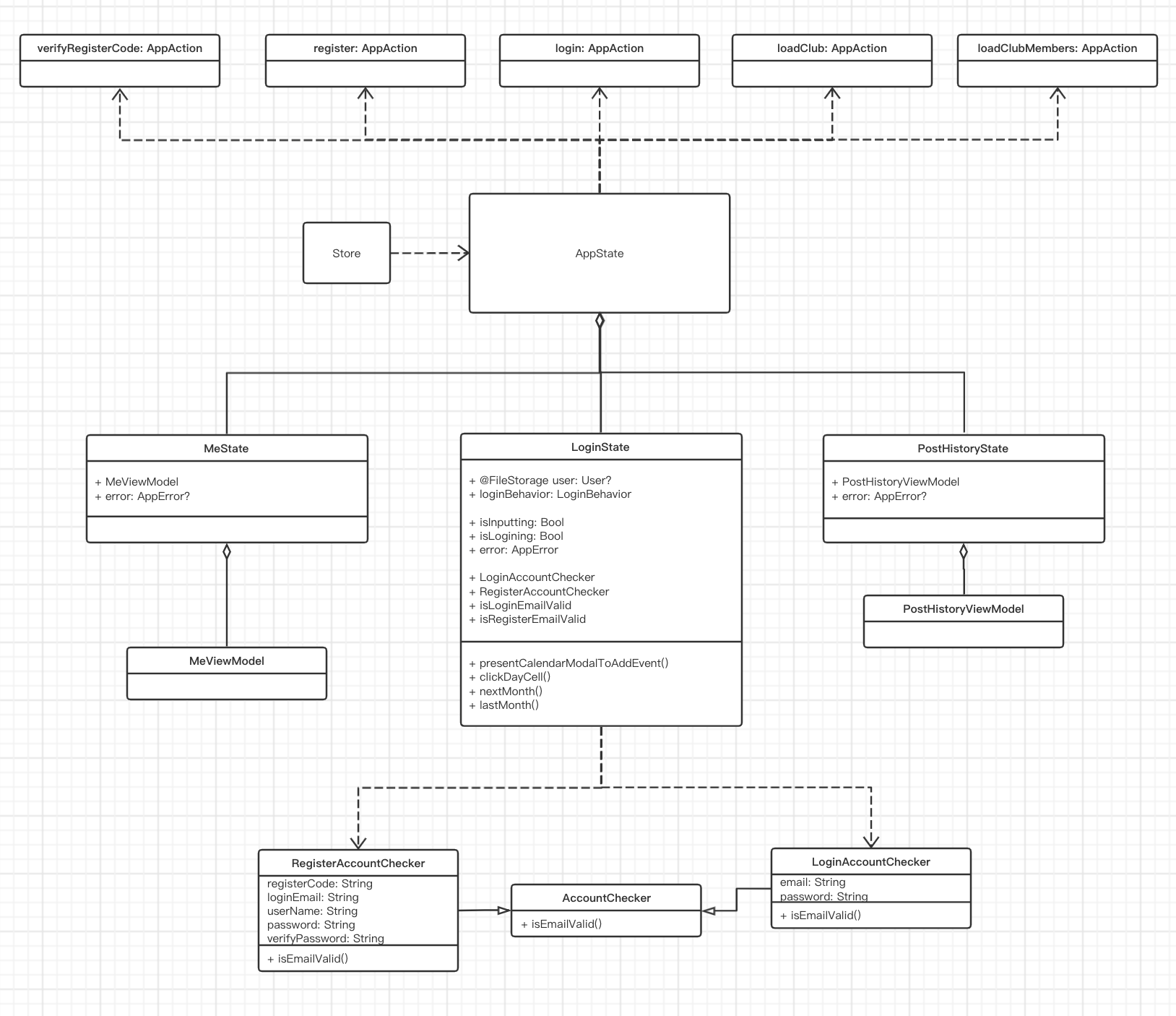


图5- 5 User模块类图

模块功能概述：作为核心结构，User的行为涉及AppState中剩余的四个State：Me State、LoginState、Club State、Post History State。其实如引言所述：User的行为渗透在程序中所有行为的方方面面，抛开View层的独立部分，所有的状态迁移都需要当前的User实例作为数据提供者和事件流发起人，不过以上几个State是User所有主动行为的承担者和执行环境，因此在这里将其都归并在User模块。

User模块本身功能内容涵盖较广，主要可以概括为登录/注册，加载历史信息，修改用户个人信息这几个方面。

1）登陆：

本项目目前没有在服务端设立登陆设备上限检测，因此登陆功能本质上可以概括为提供登陆邮箱和密码，发起请求，服务器收到请求验证登陆信息，服务器返回用户的id，用户名、权限等级等数据。并在事件流范畴中，作为后续加载事件的起点，唤醒后续加载事件流。具体登陆操作由LoginAction封装，Action携带当前View层监控所得的登陆邮箱和密码数据。系统封装Action将其交给Store的dispatch枢纽，在Store的reduce函数中，系统调取Action封装的登陆邮箱和密码数据，并将其再度封装给LoginAppCommand，由该Command对LoginRequest请求内的publisher进行底层订阅。

服务器根据邮箱是否存储在数据库中、邮箱与密码是否匹配，返回成功与否的结果数据，如果失败，则返回失败的详细情况，如果成功，则封装该登陆邮箱下对应用户的详细用户数据，其中包括用户的头像地址、用户描述、用户权限等，将其返回客户端。由于LoginAppCommand与LoginRequest之间是订阅关系，因此，在Request获得服务器返回结果后，Command实体将自动根据返回结果的成功与否，封装接下来的LoginDoneAction，开启新一轮的dispatch-reduce行为流，实现具体的响应性状态迁移，即用户实例的赋值等操作。

2）注册：

注册行为在用户视角的行为流中是所有事件的起点，但本项目作为一个俱乐部际的社交型应用，规范用户的身份和权限等级、实现用户与俱乐部之间的双向绑定，是非常重要的规范性事件。本项目在设计实现时采用了“注册码”式注册方式，在创建俱乐部实体时，需要为该俱乐部绑定一个独有的，十位长度，纯由大写字母和数字组成的注册码。并将其单独存储在新的表项里，以俱乐部Code作为键值。注册码由俱乐部指导老师和俱乐部经理进行管理，并有权编辑或重设该值。

用户在注册时，首先需要提供注册码，并点击注册码输入文本框下方的verify register code字样（见图4-4），封装并启动verifyRegisterCode动作，向服务器发送验证注册码。服务器在上文提到的[club code-register code]表中对该注册码进行查询操作，若成功，则返回该注册码对应的学校名称，若失败，则捕获并返回失败的错误处理信息。

在客户端，根据验证的结果情况，注册码文本框上方将出现绿色或红色的提示字样，告知用户注册码是否有效，已经有效注册码绑定的俱乐部名称，防止出现用户注册时绑定俱乐部错误的情况。

验证成功后，用户需要提供登陆邮箱、登陆密码、用户名等新的文本数据。相较于登陆操作中“登陆邮箱是否有效”的事件筛选，注册模块在登陆邮箱验证的基础上又增加了上文的“注册码是否有效”，以及“密码与重复确认密码是否一致的”拦截行为。只有在输入文本满足“注册码有效”、“邮箱格式有效”、“密码与登录密码一致”的情景下，该界面View层的注册确认按钮才会对用户的点击行为解除拦截。

注册是典型的可以在主线程进行请求的行为之一。从程序功能并行的角度考虑，大部分程序很少能有和注册功能并行的事件流存在，毕竟大部分程序的功能都要在登陆之后才开放。另一方面，从服务器的响应角度来看，注册功能的请求和回复的数据传输体量都相对较小，因此注册功能的响应时间内大多能控制在用户可接受的范围内。

本项目内，用户在经过上述三层验证，核对信息无误后，即可正式发送请求，等待请求响应结束后，若请求成功，则系统将自动跳转至登陆界面，并用自动填充登陆邮箱。

3）发布历史：

这里历史信息指的是用户曾经发布的News和Blog，类似于其他加载动作，由PostHistoryState提供数据源，归纳为LoadNewsHistory和LoadBlogHistory两个Action，经由各自的AppCommand订阅下层Request的发布者。不同于前序章节提到的加载动作，加载发布历史在封装HTTP请求时提供当前用户的ID，服务器根据用户ID在数据库中查找出该用户发布的数据条目，并打包返回。

展示历史界面主体是Swift的Navigation View，能够较为方便地实现条目浏览和详情查看，且能够利用原生Navigation view提供的删除UI效果，但具体的删除操作还是需要和加载动作一样，经过Action订阅底层Request服务进行实际运行。

1. 修改用户个人信息：

可见的用户个人信息主要包括系统ID、头像、用户名、联系电话、联系邮箱，除ID不可变之外，用户能够自行对其他信息进行修改，系统也提供了对应的修改接口，整体的实现框架基本同于前述方案。有所不同的是在View层通过.sheet()接口，以Modal形式封装了对应修改界面。

6 项目实现

概述：具体的项目实现细节在前序章节的详细设计部分已有提及，本章将着重于前序章节没有提及的更加深入的技术细节部分。同时从整体的视角，补完本文对Redux for Swift这一框架的实现和应用，详细阐述整体和部分两个视角下的两个数据流闭环。同时补充服务器设计规范。

6.1服务端实现细节

6.1.1 数据设计规范：

1）权限系统：

本系统从前期开发到后期上线的使用和运维过程中，用户身份可以概括为6类，按其权限等级升序排序为：游客-普通俱乐部成员-俱乐部学生主席-俱乐部指导老师-系统管理员/运维-开发。对应权限等级为0-5。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 权限等级 | 0 | 1 | 2 | 3 | 4 | 5 |
| 用户身份 | 游客 | 普通成员 | 俱乐部主席 | 俱乐部指导老师 | 管理员/运维人员 | 开发 |

图6 - 1 权限身份参照表

不同

权限等级 >= 1可以进行所有的发布行为

权限等级 >= 2 可以对本俱乐部成员进行增删改

权限等级 >= 3 可以更换俱乐部经理

权限等级 >= 4 可以更换俱乐部指导老师，实现俱乐部实例的增删改

权限更高的用户有权删除权限较低用户发布的News和Blog，以实现俱乐部主席和指导老师对发布内容的审核。

2）User\_id的定义范围：

【一位数】：0-9 为保留位，目前将游客用户id定为0，所有从登陆界面，以游客身份登陆的用户在客户端的系统中ID标识均为0，0号用户的权限等级为0，无用户名、头像等身份信息且不可添加。1号为开发人员。

【两位数】：10-99为管理员预留位，日后的运维人员在获取帐号时id都默认设为该范围内数字。

【三位数】：100-999范围内数字为指导老师保留ID

【四位数】：1000及以上为俱乐部主席和俱乐部普通成员ID保留位，俱乐部主席和普通成员之间的不同主要在于权限等级不同，在ID上不做区别。以求简化俱乐部成员换届时的操作规范。

3）OSS存储服务器的命名规范

文件夹名称：[/UserAvatar] [/ClubIcon] [/NewsImage] [/test]四个对象文件夹分别存放用户头像文件、俱乐部图标文件、新闻内容图片以及开发测试素材。

图片标识title：用户头像绑定用户id，具体格式为[UserId.suffix]（suffix指代文件后缀名，目前服务器支持的图片后缀名包括：.bmp、.jpg .jpeg .

1） 持久化登陆：

登陆注册功能主要由LoginState驱动，在前序章节中略有涉及，核心属性值user被声明为Optional(User)类型，当user为nil时，系统判定当前应用处于未登陆状态，将调用登陆注册界面，引导用户进行登陆操作。

但在iOS系统下的app生命周期中，退出应用将使应用自动进入后台运行状态。当内存空间不足，或系统负载较重时，系统将自动释放该app进程，这就使得没有持久化存储环境的app将在后台闲置一段时间后自动被重置到第一次使用状态。体现在本项目就是登陆后的User实例被重置为nil，加载的数据被清空。数据清空暂且不论，User频繁重置毫无疑问将严重破坏用户的体验。甚至可能会出现以下使用情景：用户在浏览Blog模块RichLink所指向的外链时，Safari运行占用资源过多，导致程序被重置，浏览结束回到应用时，原本已登陆的用户实例被清空，需要重新登陆。

为了改善并杜绝这一潜在破坏体验的系统行为，本文采用实现@FileStorage类型的propertyWrapper，为wrapper Value设立默认的getter和setter函数。由@FileStorage修饰后的User实例在被声明时，将自动调用下层FileHelper中的LoadJSON函数，尝试读取并解码指定位置的JSON文件，如果返回结果不为空，就将该解析实例自动赋值给当前声明的变量中。实现登陆的持久化。相对的，不论user实例是否为空，在对该变量进行赋值时，系统将会自动调用修饰符中所包含的set函数，通过下层FileHelper中的writeJSON函数将当前User数据作为持久化的JSON文件写入硬件层。这一行为主要发生在退出登陆，或登陆的过程中。

此外，为了防止为user变量赋空值时造成的可疑且日后不可触的硬件空间占用，在@FileStorage的set闭包调用中，如果检测当前值为空值，就直接删除对应文件。

2）邮箱格式的本地检测：

Combine框架下，数据之间的交流基于发布者订阅者模式，事件流的起点在于发布者发布新的事件信息，下游的订阅者将自动接收数据并触发数据处理行为。这就使得在一些特定的情境下，实现一个长期的观察监视系统的工作量变得相对简单，同时也不会占用过多的内存。

邮箱的格式检测就是基于本模式设计实现的。作为事件流起点Publisher的分别是LoginAccountChecker和RegisterAccountChecker内部的isEmailValid属性。Swift允许为声明变量计算赋值的特性允许我们将isEmailValid定义为一个只能返回Bool，无法返回错误信息的Publisher（Publisher的详细介绍将在下一章展开，本章暂不做阐述）。该Publisher的数据源头是对应AccountChecker下经Binding封装的email属性，这意味着每当email改变时，对应isEmailValid的Publisher就会被激活，并抓取当前email值，将其发布到下游的变形函数中。中游的Operator根据email的格式，应用在Helpers中为String定义的拓展类型：isValidEmailAddress，判别当前email是否为符合邮箱格式规范的字符串，再将表征结果的Bool值继续发布给最下游的Subscriber，完成事件的发布、操作与订阅。在这一邮箱格式检测事件流中，最下游的Subscriber指的是LoginState中isLoginEmailValid和isRegisterEmailValid这两个分别表征当前登陆注册邮箱是否符合格式的最终节点，同时也是直接与View层进行绑定的节点。至此，数据的流动再一次在Publisher-Operator-Subscriber三者之间形成闭环。

当该值被赋为false时，规定原文本框中的字体颜色显示为红色，系统将拦截登陆界面和注册界面内确认按钮的登陆/注册事件发起行为。而该值为true时，字体颜色对应设置为绿色，并开发按钮的事件响应行为。

同时，上文中略有涉及，该事件流是典型的“执行时间覆盖整个程序生命周期”，同时也“并不占有过多系统资源”的独立事件流闭环。因此，最好的启动方式就是在所有State的启动节点：Store的初始化行为组中，声明一个set Observer函数，在该函数中绑定P-O-S流的起点和终点，并显式地将设定最终节点isLoginEmailValid和isRegisterEmailValid这个两个值的Action绑定进来。

7.测试

1. [↑](#footnote-ref-1)
2. 在Swift中，instance: Class/Struct 是声明变量的基本语法，在此使用。 [↑](#footnote-ref-2)