iAnime 功能实现简介

前端 iOS

1. 项目文件组织结构

   项目整体划分为若干层:

Layer Name	Description
Service	处理App运行时的业务逻辑，例如登陆，获取账户信息，网络通信，数据持久化
View	放置自定义View、TableView, CollectionView的自定义Cell
ViewController	放置管理对应界面逻辑的控制器
Model	放置自定义数据结构模型、网络通信时序列化Request/反序列化Response的解析模型
Utils	放置工具类，extension，加解密模块
Storyboard	放置各个界面的故事板文件

   对于一些使用了TableView, CollectionView的ViewController，默认情况下需要在当前ViewController子类里面实现几个Delegate和DataSource的Protocol。如果当前Controller实现的功能较多，逻辑复杂，再加上几个Delegate的协议实现，就会显得非常臃肿，难以维护，有严重的代码堆砌感。因此项目组考虑了以下的结构改进:

   以显示个人主页的业务场景举例。个人主页界面需要显示当前查询用户的头像，背景图，关注他/他关注的人的数量，以及当前用户上传的全部作品。不难想到"当前用户上传的全部作品"需要考虑使用CollectionView + 自定义ItemCell的方式实现。

   因此，基于上述逻辑代码与CollectionView Delegate实现代码分离的思想，首先我们创建一个类`PersonViewController`继承自`UIViewController`，在`PersonViewController`中实现加载头像、昵称，背景图，关注人数量等等的数据；然后再创建一个类`PersonWorkGridViewController`，继承自`PersonViewController`，且在这个类中实现`UICollectionViewDelegate, UICollectionViewDataSource, UICollectionViewDelegateFlowLayout`三个协议，用来给CollectionView提供数据源和指导ItemCell的布局，获取用户发布的全部作品的代码也放到这个类中实现。最后再在Storyboard中指明当前ViewController的实现类为`PersonWorkGridViewController`，即可实现VC业务代码和数据展示相关的Provider代码分离，提高代码的可维护性。

2. 界面布局流程:

   a. 首先采用Storyboard定位、简单预设一些静态约束参数

   b. 然后使用SnapKit + 原生AutoLayout动态设置UI控件之间的约束关系 (发现页CollapsedView)

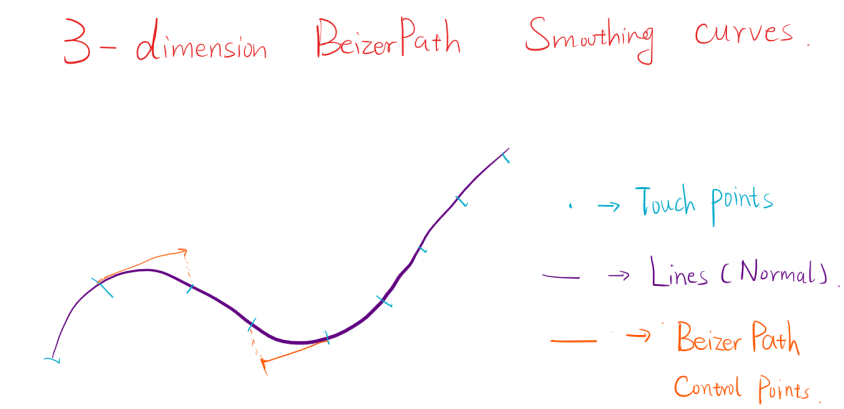
c. 对于形状大小关系复杂的控件(双指缩放图片浏览控件、画板双指缩放控件)，不采用约束定位，直接运行时手动计算frame.origin及frame.size来布局。

3. 画板具体实现

a. 对于普通触控输入设备（手指，普通电容笔），采用B 样条插值 + 三阶贝塞尔曲线近似来实现笔迹平滑。

b. 对于精确触控设备（Apple Pencil），使用`CoalescedTouches`得到更多的触控点轨迹信息，用三阶贝塞尔曲线近似。

c. 三阶贝塞尔曲线曲线近似法，以四个触控点为一组，首位相接。使用第一，第四个点为路径点、第二，第三个点位控制点，拟合贝塞尔曲线。



d. 线条压力感应经验公式

iOS在绘制一条线段时需要预设这条线段的粗细，并且要求整条线段的粗细不能发生变化，因此缺失了力度的渐变信息之后就很难决定这四个点之间的线段应该以何 种方式过渡比较均匀。经过反复验证测试，团队得到了以下控制力度的经验公式:

```
var totalForce = (0.4 * forces[2] + 0.3 * forces[3] + 0.3 * forces[4]) *
0.3 + lastForce * 0.7
```

使用此经验公式能够较好的感知触控力度变化，并将变化反映到线段的粗细变化上。

e. 颜色锚点和颜色标记点缩放控制实现

颜色锚点和颜色标记点实际上是附加在UIImageView的layer上的若干个CAShapeLayer。当UIImageView进行Scale的时候，根据iOS系统的控件绘制原理，实际上就只是控件的layer发生了scale, 这会使得表示标记点的CAShapeLayer同步进行缩放，出现图片放的越大，标记点也会放得越大的情况，从而影响对线稿细节的观察。因此需要实现在UIImageView缩放的过程中获取缩放比例，将当前缩放比例求倒应用到CAShapeLayer上，确保标记点和锚点最终的scale比例始终为1，实现在缩放过程中仍然“保持自身大小”。

```
    func FixColorPointSize(_ currentScaling : CGFloat) {
        // 在画布进行伸缩的时候同步缩放颜色提示点和颜色锚点
```

```
        for (_,an) in anchors {
            let p = an.point
            let path = an.path
            path.apply(CGAffineTransform(translationX: -p.x, y: -p.y))
            path.apply(CGAffineTransform(scaleX: 1 / currentScaling, y: 1
/ currentScaling))
            path.apply(CGAffineTransform(translationX: p.x, y: p.y))
            an.layer.lineWidth = an.layer.lineWidth / currentScaling
            an.layer.path = path.cgPath
        }

        for (_,an) in hints {
            let p = an.point
            let path = an.path

            path.apply(CGAffineTransform(translationX: -p.x, y: -p.y))
            path.apply(CGAffineTransform(scaleX: 1 / currentScaling, y: 1
/ currentScaling))
            path.apply(CGAffineTransform(translationX: p.x, y: p.y))
            an.layer.lineWidth = an.layer.lineWidth / currentScaling
            an.layer.path = path.cgPath
        }
    }
```

```
    func DrawColorPoint( _ at : CGPoint,_ type : ColorPointType, _ color :
RGB) {
        // 绘制颜色锚点/颜色提示点
        let
        up = CGPoint(x: at.x, y: at.y - 6),
        right = CGPoint(x: at.x + 6, y: at.y),
        down = CGPoint(x: at.x, y: at.y + 6),
        left = CGPoint(x: at.x - 6, y: at.y)

        let anchorLayer = CAShapeLayer(layer: layer)
        let currScaling = transform.a
        anchorLayer.frame = bounds
        anchorLayer.lineWidth = 3 * 1 / currScaling
        anchorLayer.fillColor = color.AsUIColor.cgColor
        anchorLayer.strokeColor = UIColor.white.cgColor

        switch type {
        case .Anchor:
            OneAnchorPoint = UIBezierPath()
            OneAnchorPoint?.move(to: up)
            OneAnchorPoint?.addLine(to: right)
            OneAnchorPoint?.addLine(to: down)
            OneAnchorPoint?.addLine(to: left)
            OneAnchorPoint?.close()
```

```
            OneAnchorPoint?.apply(CGAffineTransform(translationX: -at.x,
y: -at.y))
            OneAnchorPoint?.apply(CGAffineTransform(scaleX: 1 /
currScaling , y: 1 / currScaling))
            OneAnchorPoint?.apply(CGAffineTransform(translationX: at.x, y:
at.y))
            anchorLayer.path = OneAnchorPoint?.cgPath
            anchors[at.AsVector2()] = ColorAnchor(color: color, point:
at,path : OneAnchorPoint!, layer: anchorLayer)

            break
        case .Hint:
            OneAnchorPoint = UIBezierPath()
            OneAnchorPoint?.addArc(withCenter: at, radius: 5, startAngle:
0, endAngle: CGFloat.pi * 2, clockwise: true)
            OneAnchorPoint?.apply(CGAffineTransform(translationX: -at.x,
y: -at.y))
            OneAnchorPoint?.apply(CGAffineTransform(scaleX: 1 /
currScaling , y: 1 / currScaling))
            OneAnchorPoint?.apply(CGAffineTransform(translationX: at.x, y:
at.y))
            anchorLayer.path = OneAnchorPoint?.cgPath
            hints[at.AsVector2()] = ColorHint(color: color, point: at,
path : OneAnchorPoint!, layer: anchorLayer)
            break
        }
        layer.addSublayer(anchorLayer)
    }
```

4. 外部库的选用

   本App采用CocoaPods管理依赖的第三方库。

   本App只引用了一些进行网络通信、JSON解析以及增强Auto Layout DSL的著名第三方库。其他全部界面效果，画板、双指缩放、自定义数据结构持久化/重新加载、动画系统的逻辑完全自主实现。

   给出项目的Podfile:

```
platform :ios, '11.0'
use_frameworks!

target 'iAnime' do
  pod 'SwiftyJSON', '~> 4.0'
  pod 'Alamofire', '~> 4.0'
  pod 'AlamofireImage', '~> 3.5'
  pod 'SnapKit', '~> 4.0'
  pod 'CryptoSwift', '~> 0.15.0'
end
```

后端 Node.js

1. 后端架构

   Node.js 业务层

   - koa 后端Web Api框架
   - Koa-router 请求路由中间件
   - Koa-decorator-router 装饰器路由注入框架（团队自主实现，配合TypeScript的Decorator Reflect Metadata使用）
   - sequelize ORM框架
   - sequelize-typescript 装饰器实体模型定义框架
   - Amqplib 消息队列通信

   Python 上色计算节点层

   - Tensorflow GPU
   - OpenCV 图片预处理/后处理
   - Keras 深度学习框架
   - 多GPU分离加载上色神经网络模型

RabbitMQ 业务层与上色节点RPC交互

MySQL 数据库

Redis 热Key缓存, JWT吊销请求缓存

Nginx 反向代理 + 负载均衡网关

2. 部署架构

   RabbitMQ 服务端 => Docker

   上色节点 => 真机运行 + Python venv

   业务节点 => Docker

   MySQL => 真机运行

   Redis => Docker

3. 近期优化

   后端使用kong完成微服务架构改造，分离用户账户模块、作品上传发布查询模块、社交模块。

   ElasticSearch 记录服务出错日志

   服务健康度检测，计算节点双活备份。