字节跳动 字学镜像计划iOS实践篇 项目报告

字节跳动 字学镜像计划iOS实践篇 项目报告

2021.04.23 - 2021.07.01

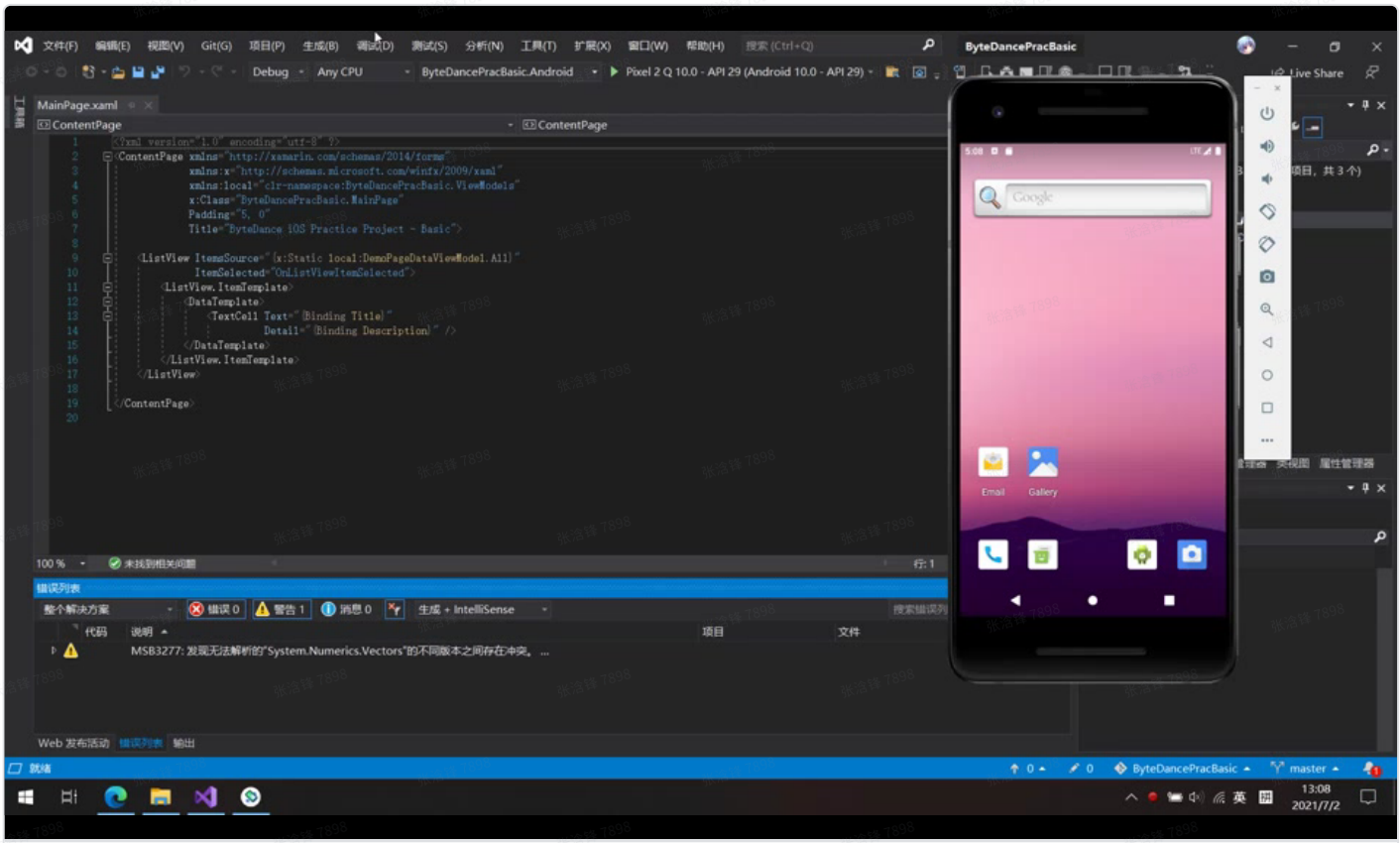
个人参赛者 张洽锋

录屏展示：

在此首先强调一下，**本项目为使用Xamarin.Forms开发的跨平台项目，由于本人的硬件限制没有苹果设备，故下面的演示均使用运行在Windows上的Android模拟器进行**（且在实体Android手机上测试可用）。如果需要生成iOS应用，则可以参考Build your first Xamarin.Forms app - Xamarin | Microsoft Docs进行远程生成。**本项目的几乎全部的代码均可跨平台运行**（除去专题研究中的LibVLC初始化代码，对此在平台相关的ByteDancePracAdvanced.iOS项目中单独添加了这部分的代码），**理论上不存在iOS无法运行的功能。**

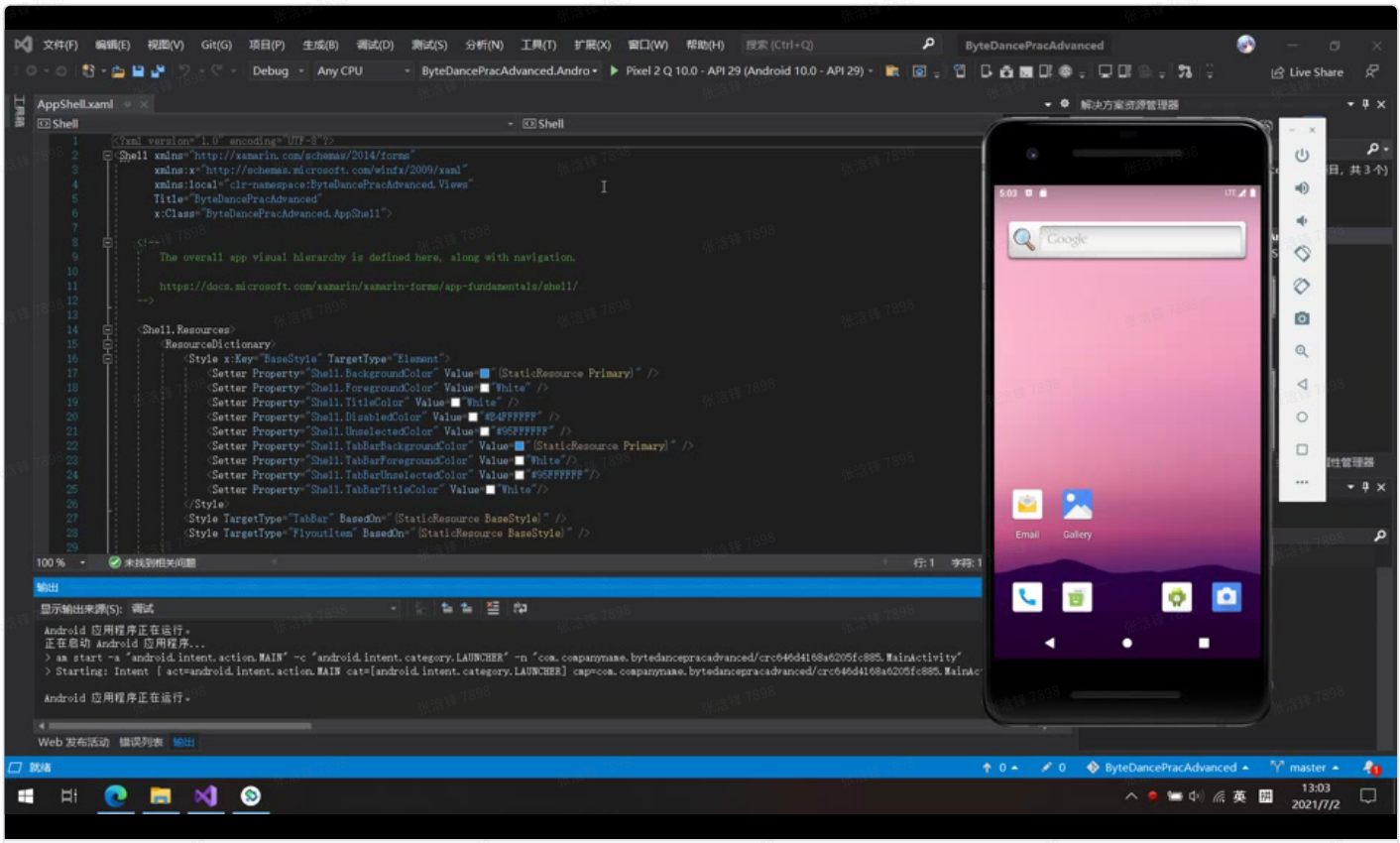
另外由于电脑性能限制，录屏时运行会很缓慢，模拟器中的音频和视频可能存在卡顿，此为正常现象，在实机中无此问题。

基础功能部分录屏演示：



2021-07-02 02-08-05.mp4

专题研究部分录屏演示：



2021-07-02 02-03-51.mp4

0x00 写在前面&对本项目的一些想法

首先感谢正在查看该报告的工程师或字节校园社群的相关工作人员，感谢您抽出宝贵的时间查看本菜鸡写的这个项目报告，祝您工作顺利，身体健康，生活幸福！

这一小节是我希望写在报告前面的一些内容，算是本报告的“说明”和一些想法吧。首先必须说明的一点是，很抱歉本报告不会有多么“正式”和“标准”，我不会用过于正式的语言，也不会像工作时会写的严谨的文档那样记录开发项目的各个方面的详细信息。与其说是“报告”不如说是“笔记”甚至是“日记”，我将会着重记录整个项目的开发期间的设计想法、心得体会和踩坑记录，而不会详细列出类的成员、接口的详细信息等等。如果校园君或者工程师在看的话，先对本报告的极不严谨的态度表示歉意哈哈哈:-)。另外必须做出澄清的是，**本报告的不严谨绝不代表我个人在正式的学习或工作中也会按照这种样式写文档**，希望不要因此对我以后可能的求职产生负面印象。

我接触字节校园主要还是2月的时候刷力扣看到了7天刷题挑战的那个广告。我是成功保研了，大四的寒假在忙研究生课题组的事情的同时，看到网上各种关于找工作的讨论，就也带着看找工作的相关知识，也偶尔刷题。但是令我感到很惊讶的是，保研/考研和找工作的准备差别真的很大，力扣上的许多题目对于好久没有刷题的我而言真的很难（我个人不喜欢竞赛，上次刷OJ是大二时CCF CSP认证，然后那届题爆难，200分排到了前5.2%，让我对算法编程题产生了严重的心理阴影……），而且一些大佬的刷题见解令我感到十分敬佩，也由此开始关注字节校园。返校后，又帮着同学面试“作弊”，也就是他们把题发来，我码好代码后再发给同学抄（当然，他们没报字节的招聘哈），然后说实话，见识到了一些同学的令人遗憾的水平（不过最后还是得到了银行的Offer）。这些经历对我的触动很大，平时的练习习惯真的对人的影响很大，大佬们熟练的刷题都是建立在海量的过往练习的基础上的，平时不注重实践、不手敲代码的话，可能会导致编程能力的退化。

进入字节校园社群之后不久，就提出了这次iOS实践项目，于是我也参加进来体验一下。大四下学期的特点就是忙，但又觉得忙了半天除了毕业论文其他啥也没搞。毕业的事情太多，也导致参与此项目的时间不多，现在又因为暑假被导师叫过去，此实践被迫提前结束。关于这个iOS实践项目，我是纯粹以兴趣为导向参加进来的，因此后面的内容中我选用了国内极其冷门的技术，也没有深入研究。我一直认为，兴趣是最好的老师，而且我习惯于什么都尝试一点，让生活更多彩一些嘛。iOS实践项目给了一些研究目标，参与进来或许会很有意思吧，于是我就参加了这个项目。

这里就说实话实说好了，我个人觉得好像确实没有很多人活跃地参与到这个项目中来，这里也比较心疼校园君，看她整天想办法活跃气氛但收效较少……没办法，线上+无收益+无强制性=用爱发电，这就很可能导致参与的同学要么很难有积极性，要么就是讨论游戏，变成水群（笑哭），突然想到本科时管理社团的日子……

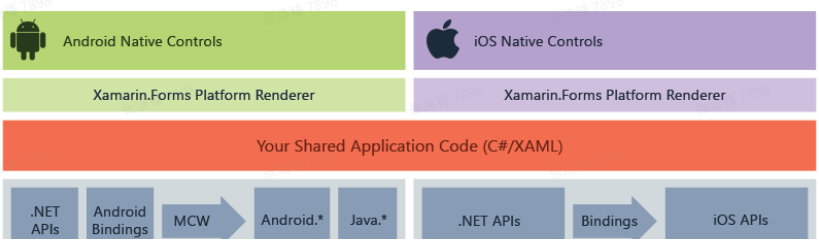
不过整个项目对我还是很有帮助的，很感激校园君和工程师们为这个项目的付出，许多资料对我有一些帮助，也督促我研究相关知识，这里对您们的付出表示感谢~

0x01 Xamarin.Forms和MVVM

事实上，我讨厌苹果的生态，参与这个项目的主要原因是学一下移动应用开发的相关知识，填补一下这个方向的空白。于是我自然想到的就是使用Xamarin.Forms进行开发，它是微软主导开发的一套框架，可以以一套共享的UI和功能代码生成原生的Android、iOS和Windows UWP应用，而且其跨平台的特点使得我可以脱离Mac开发应用，在Windows上的Android模拟器开发通过后，若有Mac硬件即可以直接生成iOS应用（我使用国外的云Mac服务器MacinCloud - Rent a Mac in the Cloud! - Mac in Cloud尝试过了，可以正常生成，但因为金钱问题，租不起那么贵的服务器，下面就全都使用Android来演示了）。另一方面Xamarin.Forms属于.NET生态，上手后可以快速转到.NET桌面应用、ASP.NET Core Web应用等，有利于后续的自学探索。

不过缺点就是，国内关于Xamarin.Forms的资料极少，几乎全部内容都需要参考英文资料进行学习。而更戏剧性的是，微软在前一阵子声明，准备在今年年末发布.NET 6，使用MAUI替代Xamarin.Forms，不仅仅是改名，而在设计模式上都有一些改变，因此比较尴尬，可能等.NET 6发布后需要更新一些知识。

按照微软官方文档的定义，Xamarin.Forms 提供了一个一致的API，用于跨平台创建UI元素。此API可以在XAML或C#中实现，支持针对模型-视图-视图模型 (MVVM) 等模式的数据绑定。在运行时，Xamarin.Forms使用平台呈现器将跨平台UI元素转换为 Xamarin.Android、Xamarin.iOS和UWP上的本机控件。这使开发人员可以获得本机外观和性能，同时实现跨平台代码共享的优势。Xamarin.Forms 应用程序通常由共享 .NET Standard 库和各个平台项目组成。共享库包含XAML或C#视图以及任何业务逻辑（如服务、模型或其他代码）。平台项目包含应用程序所需的任何特定于平台的逻辑或包。Xamarin.Forms使用 Xamarin平台，跨平台本机运行.NET应用程序。



Xamarin.Forms中，每个视图（View）由两个文件组成，一个XAML文件用于使用扩展的XML标记来定义视图的UI，另一个文件为C#编写的代码隐藏文件（code-behind），这两个文件共同构成了一个新的类定义，其中包括子视图和属性初始化。在XAML文件中，类和属性通过XML元素和属性进行引用，并建立标记和代码之间的链接。

其他详细的关于Xamarin.Forms的概念在此不再赘述，详情可参照微软的官方文档（英语）。

MVVM设计模式在Xamarin.Forms中反复被提及应用，它也是Xamarin.Forms在设计上推荐的软件设计模式（文档原文记录为：The Model-View-ViewModel architectural pattern was invented with XAML in mind.）。我个人认为软件设计模式是最难学习的知识之一，它需要大量的工程实践作为基础，而且每个人的理解不同，但差劲的设计模式会导致整个项目难以开发维护，因此在此项目开发的宝贵机会中，首先需要规划即将采用的设计模式，对其有一定理解后再上手实践。

本项目采用MVVM设计模式，关于它的一些理解记录如下：

View: View是数据的外在表现，负责视觉元素的定义、布局和风格化。View永远不会真正地包含或操纵数据，它们只是绑定了ViewModel中的属性(properties)和命令(commands)。View是、也仅仅是数据的“衣服”(the clothes that data wears)。

Model: Model是应用程序中对其业务领域的建模。例如，对于一个音乐商城应用，它的Model可能包括歌手、专辑和歌曲等对象；对于一个人力资源管理应用，它的Model可能包括管理员和雇员等对象。Model不与任何视觉表现关联，甚至可以说，Model与包含它们的应用程序都没有直接关系——Model自己就是独立且有意义的，它们是对业务领域的建模表示，而要开发的应用程序只是使用到这些模型而已。Model一般还包括对Model中的数据存取的逻辑。

ViewModel: ViewModel是对GUI应用程序的建模，也就是说，它们提供View用到的那些数据和功能，并定义整个要构建的应用程序的结构和行为。Model针对的是业务领域的建模(例如音乐商城、人力资源管理等)，而ViewModel针对的是图形化应用程序的建模，封装的是应用程序中的数据、行为等一切内容。ViewModel以属性的形式对外暴露其中的对象(objects)、集合(collections)和命令(commands)等。命令是对行为(behavior, 一个最简单的行为可以是方法调用)的封装。值得注意的是，命令将行为封装为对象的思想很重要，因为View是以数据绑定为驱动的，也就是说View中的视觉控件都要绑定到对象上。MVVM中，一个按钮是没有click handler的，但是可以将这个按钮绑定到一个命令对象上(在ViewModel中体现为它的一个属性)，就可以在命令中写出点击这个按钮时要执行的那些功能。

需要额外注意的几点：

尽管ViewModel是对图形化应用程序的建模，但它们不直接引用或包含视觉控件——那些是View所关注的事情。ViewModel只需要把必要的信息和行为暴露给将要与它绑定的那些控件即可。举例来说，如果希望在视图中加一个ListBox，那么ViewModel中只需要维护一个与之对应的集合即可；如果希望加几个button，那么ViewModel中只需要写几个与之对应的command即可。

严格来讲，一个Model是不知道ViewModel的，同时一个ViewModel是不知道View的。但凡事皆有例外，在特定的情况中，ViewModel可以考虑View的实际场景，并对一些属性的返回值进行适配。例如，如果Model的某个属性数据来源于一个使用8-bit字符的数据库，那么ViewModel中可能需要将其转换为Unicode宽字符来匹配UI界面对数据类型的需要。

0x02 基础功能任务

这里将所有的基础功能任务整合为一个项目ByteDancePracBasic。需要注意的是，由于官方文档均使用英语且无人工翻译的中文内容，再加之国内关于Xamarin.Forms的资料极少，故在软件源代码中的绝大多数注释也采用英文。

1. **设计一个类**

语言基础的学习与练习。本项目使用的语言为C#，关于对象的序列化与反序列化，这里直接使用.NET Core的System.Text.Json命名空间中的相关方法扩展实现。由于.NET命名规范使用驼峰命名法，而实践要求的输入为蛇形命名法，故需要自定义序列化/反序列化时的命名规则。

对命名规则的自定义详见Utils/SnakeCaseNamingPolicy.cs源文件。

微软官方文档中的相关更多内容详见序列化 (C#) | Microsoft Docs。

2. **完成Feed列表原型**

View：

ImageFeedPage：此视图主要由一个ListView构成，ListView中的每个元素表示一个Feed，Feed的表示由ListView.ItemTemplate中的一个ViewCell定义，其中还涉及到StackLayout、GridLayout等布局的使用。

ViewModel：

ImageFeedListViewModel：此VM用于维护一个Feed集合的列表，并包含了Feed列表的初始化操作，在本项目中，在构造函数中对Feed列表进行初始化。可以理解为ImageFeedListViewModel为ImageFeedViewModel的集合的一个wrapper。

ImageFeedViewModel：表示每个Feed，包括Feed的主体信息。可以理解为ImageFeedViewModel为ImageFeedModel的一个wrapper。

Model：

ImageFeedModel：对Feed的具体建模，各个属性维护了Feed的各项信息。

MediaInfoModel：ImageFeedModel的一个域的定义，即Feed发布者的相关信息。

数据绑定关系：

ImageFeedPage与ImageFeedListViewModel绑定，后者的列表即为前者ListView中的每个Cell的信息来源。Xamarin.Forms中的ObservableCollection为列表数据的动态更新提供了支持，即当Feed列表变化时，可以直接更新ListView的显示。

注意：

为什么要用ImageFeedViewModel再次封装ImageFeedModel?

ViewModel要与View绑定，同时ViewModel中应包含主要的业务逻辑，考虑到后续可能需要扩展每个Cell中显示的信息或者功能（例如后面的点赞效果），因此最好使用ImageFeedViewModel再次封装ImageFeedModel，后续更改时修改前者即可完成更多的功能添加和维护。

效果截图：

3. 给 Cell 卡片添加点赞动画

在前一步骤的基础上，添加点赞功能和点赞动画。

粒子效果的实现需要使用第三方库(如particle.forms等)或下降到平台相关的功能，本次实践的目标主要不是在此，因此这里没有实现这个可选目标。

ViewModel：

ImageFeedViewModel：主要修改ImageFeedViewModel，增加一个命令接口作为其一个属性，用于实现点赞的相关逻辑（这里实现为使点赞数+1，反复点击则只增加一次）。

View：

ImageFeedPageWithAnimation：在ImageFeedPage的基础上，添加动画。这里实现的逻辑为：在大拇指图片和点赞数Label的上面添加一个透明的按钮，当点击按钮时，对大拇指图片进行动画播放。由于Xamarin.Forms本身没有提供阻尼动画，故这里使用两次缩放动画来模拟此效果。

数据绑定关系：

动画的相关代码处于View中，此例为Views/ImageFeedPageWithAnimation.xaml.cs中的OnLikeFeedClicked()方法。然而，点赞的功能逻辑（增加点赞数）处于ViewModel中，此例为ViewModels/ImageFeedViewModel.cs中的LikeThisFeed成员（其类型为ICommand接口，功能逻辑以委托的形式在构造函数中写入）。

当点赞区域的Button被点击时，同时触发动画和点赞命令，在View中体现为Button的Clicked与动画方法绑定，Command与VM中的委托绑定。之所以将其拆分，是为了严格遵循MVVM的设计思想，将UI与功能逻辑分开，View只负责UI相关，而VM处理业务逻辑。这使得两者可以解耦，便于后续维护。

4. **使用NSURLSession请求数据，构建feed列表**

在前一步的基础上，添加联网拉取Feed列表。

ViewModel：

ImageFeedListViewModel：增加一个LoadFeedList()方法来完成列表的拉取创建，在构造函数中调用即可。实现时使用.NET Core提供的HttpWebRequest和HttpWebResponse方法读取列表信息（设置超时时间为3秒），然后将其反序列化为Feed列表即可。

5. **优化App的启动速度**

在前一步的基础上，添加Feed的本地数据库缓存。

ViewModel：

ImageFeedListViewModel：在LoadFeedList()中加入数据库的相关逻辑，作为示例，实现的缓存逻辑为打开页面时检查是否存在本地缓存数据库，若不存在则拉取联网信息并保存到数据库中，若存在则直接加载。

为了支持数据库相关操作，修改了Model中的一些标注（由SQLiteNetExtensions提供支持），同时引入了Utils/AsyncLazy和Data/FeedDatabase为数据库操作提供支持，实现了数据库内容的异步懒存取（虽然在本实践中没有体现出其效果，因为毕竟也就4个很简单的Feed……）。

注意：

本实践中不包含缓存校验等功能逻辑，而且由于实践指导文档中的模型设计上存在一些缺陷，因此缓存部分的实现较为“粗暴”。至于图片的缓存则由Xamarin.Forms本身的缓存机制提供原生支持，在代码中无需相关功能。

模型设计的缺陷在于，实践文档中的模型设计不太合理，但又受限于Json中定死的格式，难以自行修改。Feed和MediaInfo的对应Model均没有主键，在数据库设计时，两者又存在1-1/1-*的关系，这使得难以确定外键来维护两者的关联关系（不考虑在存储Feed时序列化或计算MediaInfo的GUID，因为这个工作本来应由外键来指定）。

这部分用语言形容有些抽象，可结合ImageFeedListViewModel.cs源文件中的FIXME注释来体会。

6. 支持视频播放

在前一步的基础上，添加视频播放支持。

使用微软官方的Xamarin Community Toolkit扩展提供媒体播放支持。实践指导中的视频链接已经失效（之前在阶段汇报中提及过，但截至此报告书写时仍未恢复），这里使用另一个视频代替。但作为替代的视频是一个mp4格式的非流媒体、外网视频，加载可能较慢。

View：

VideoFeedPage：类似于图片Feed版本，只是将图片改为视频。细节上在视频之上覆盖一个ImageButton，用于显示视频的缩略图，当用户点击后，使缩略图消时并露出下面的视频控件。

ViewModel：

VideoFeedListViewModel、VideoFeedViewModel：与图片Feed版本类似。

Model：

VideoFeedModel、VideoPlaceholderModel：对视频版本的Feed的建模，其中VideoPlaceholderModel用于存储缩略图的URL和长宽。

效果截图：

0x03 专题研究——iOS播放器实现

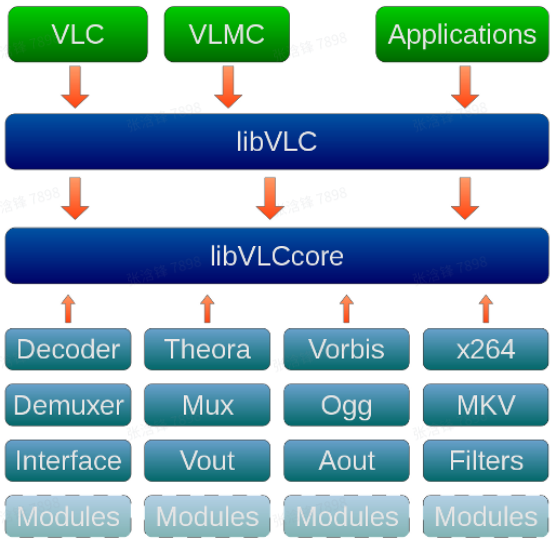
注意：因个人的时间和精力（以及兴趣）所限，没有完成首帧优化功能。

这里将所有的专题研究任务整合为一个项目ByteDancePracAdvanced。

在前面的基础研究中，使用的是Xamarin Community Toolkit来支持的视频播放功能，但是该扩展极其简单，对缓存、媒体格式等的支持均十分有限，故在专题研究中不再使用这一扩展，而是转为使用LibVLCSharp作为播放器功能的核心支持。

VLC 是一款自由、开源的跨平台多媒体播放器及框架，可播放大多数多媒体文件，以及 DVD、音频 CD、VCD 及各类流媒体协议。VLC Media Player为此项目的一个主要实现，这一媒体播放器在全平台均有大量的用户使用。LibVLC即为VLC Media Player所依赖的提供媒体相关功能的核心引擎和UI框架，而LibVLCSharp是一个新兴的项目，为.NET/Mono平台提供了一套LibVLC的API框架，由VideoLAN官方维护和更新，目前LibVLCSharp仍不是很成熟，也还在持续更新中。

LibVLC的设计架构如下：



可以理解为，LibVLC是播放器相关功能的核心实现，这个库是平台相关的，在此项目中，需要用到的是LibVLC.Android和LibVLC.iOS，在此基础上，为了调用LibVLC的相关功能，以及为了兼容Xamarin.Forms以跨平台开发应用，需要使用LibVLCSharp和LibVLCSharp.Forms，为顶层开发提供功能API的绑定。

本专题研究项目以上述基础展开，更多的有关于LibVLCSharp的信息可以参考VideoLAN / LibVLCSharp · GitLab。

除此之外，本项目还需要用到Xamarin.Forms Shell提供部分UI支持（此例中为飞出菜单和基于URI的导航模式）、以及Xamarin.Essentials提供跨平台的操作系统相关功能的支持（此例中为文件选择）。关于这两个扩展的更多信息可以参考Xamarin.Forms Shell - Xamarin | Microsoft Docs 和 Xamarin.Essentials - Xamarin | Microsoft Docs。

1. **实现基本视频格式（MP4，MOV，3GP）的视频播放**

View：

VideoPlayerPage：仅由一个MediaPlayerElement构成，用于显示播放界面。

ViewModel：

VideoPlayerViewModel：维护播放器相关信息，包括LibVLC库实例、媒体URL等，并且负责播放器资源的初始化和销毁。

数据绑定关系：

VideoPlayerPage与VideoPlayerViewModel绑定，需要注意的是播放器的创建和销毁方法在VM中，而触发时机为View的出现和消失，实现时，在View的OnAppearing和OnDisappearing方法中调用VM中对应方法完成。

在实现的细节中，遵循LibVLC的最佳实践（docs/best_practices.md · 3.x · VideoLAN / LibVLCSharp · GitLab）。另外，关于LibVLCSharp和系统相关的LibVLC的绑定，要求应用启动时初始化系统相关的渲染器，这一代码只能下降到平台相关的操作中完成，详见项目代码中Xamarin.Android项目下的MainActivity.cs和Xamarin.iOS项目下的AppDelegate.cs，此外，LibVLCSharp.Forms的初始化位于跨平台项目中的App.xaml.cs中（这一步骤没有在LibVLC的官方文档中提到，如果对各个LibVLCXXX库的关系了解不是很透彻，很容易忘记它们的初始化，导致程序无法正常运行）。

2. **支持各种网络协议，http，rtp**

此功能由LibVLC提供原生支持。

3. **实现本地播放和远程播放**

本地播放部分：

新增一个页面让用户选择本地的媒体文件，判断选择的文件是否为支持的媒体格式（这里使用最简单的后缀名判断，若需要严格判断，则需校验文件的二进制头，此处不做赘述），然后用户可以播放支持的媒体文件。

本地文件选择的相关功能支持（跨平台）由Xamarin.Essentials提供。

View：

PickFilePage：提供UI界面，允许用户选择和播放本地媒体文件。

ViewModel：

PickFileViewModel：维护用户选择文件时的相关信息，包括文件路径等。

数据绑定关系：

PickFilePage与PickFileViewModel绑定，另外，当用户选择的是合法的媒体文件时，PickFileViewModel实例在PickFileCommand内直接修改VideoPlayerViewModel的媒体路径和媒体来源类型静态属性，用户点击播放则直接跳转到VideoPlayerPage，初始化播放器并播放对应媒体。

远程播放部分：

新增一个页面，让用户输入媒体URL。功能逻辑与本地播放类似。

View：

PickNetworkStreamPage

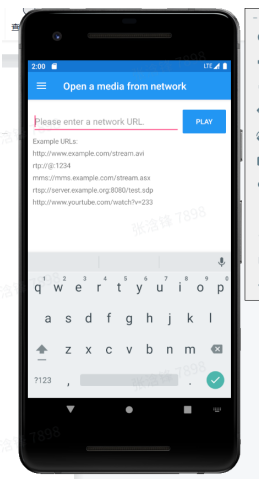
ViewModel：

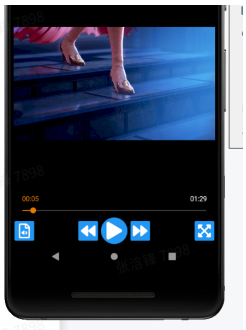
PickNetworkSteamViewModel

数据绑定关系：

PickNetworkStreamPage与PickNetworkSteamViewModel绑定，与前面的本地播放类似，PickNetworkSteamViewModel也会修改VideoPlayerViewModel的静态属性。

效果截图：





4. **实现预加载功能**

此功能由LibVLC提供原生支持，默认缓存时间为5秒。此外，LibVLC提供硬件加速支持。

5. **实现随机时间播放功能**

此功能由LibVLC提供原生支持。

6. **优化首帧（未实现）**

优化首帧这一词指代不是很明确，这里理解为优化Time To First Frame（TTFF），而这一优化通常由服务提供商完成，例如使用CDN缓存热门视频等。而在应用客户端，通常需要预先加载视频，即在用户点击播放前就缓冲，然后从MemoryStream中加载视频开始的一段时间的内容。

我个人认为优化TTFF这一工作与“实现一个iOS播放器”的目标存在偏差，播放器应做好本职工作而非缓冲用户还未确定播放的内容，而且对于本地播放器而言TTFF对用户体验的影响几乎可以忽略，故这里没有探索此功能。

注意：在基础研究（ByteDancePracBasic项目）中，Converters/ImageSourceConverter实现了类似的功能，这一转换器最初是为了解决Feed中有部分图片信息缺少部分元数据，导致Xamarin.Forms不会渲染图片的问题。而它的思想即为先缓存后再重定向至内存中读取媒体，与本功能类似。如果真的需要优化TTFF，可以参考这一思路实现。

（另外，没有实现这一功能不意味着我会拒绝部分需求，毕竟这是一个线上实践项目，纯粹以兴趣为主。学习归学习，工作归工作，兴趣归兴趣，有钱能使鬼推磨嘛哈哈哈哈）

0x04 尾声

以上即为本项目的全部情况记录。总得来说参与这次项目感觉很开心，自己学到了许多知识，也填补了自己在移动应用开发这一方面的空白，感觉收获很多。再次感谢所有在此项目项目进展中给予我帮助的人。谢谢！