

IOS学习笔记

Apple Framework

UIApplicationDelegate

当 Apple 程序运行时会创建 App 实例，UIApplicationDelegate 的本质是一个抽象接口类，创建 App 之后将 AppDelegate 绑定到 App 实例当中，App 程序去调用 AppDelegate 中对应接口的实现。

OC 入口函数

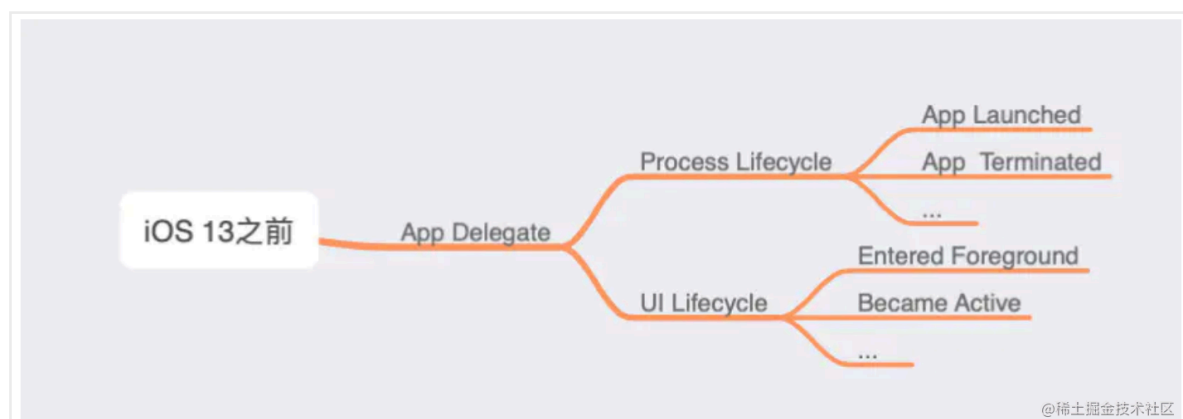
```
int main(int argc, char * argv[]) {
    NSString * appDelegateClassName;
    @autoreleasepool {
        appDelegateClassName = NSStringFromClass([MyAppDelegate class]);
    }
    return UIApplicationMain(argc, argv, nil, appDelegateClassName);
}
```

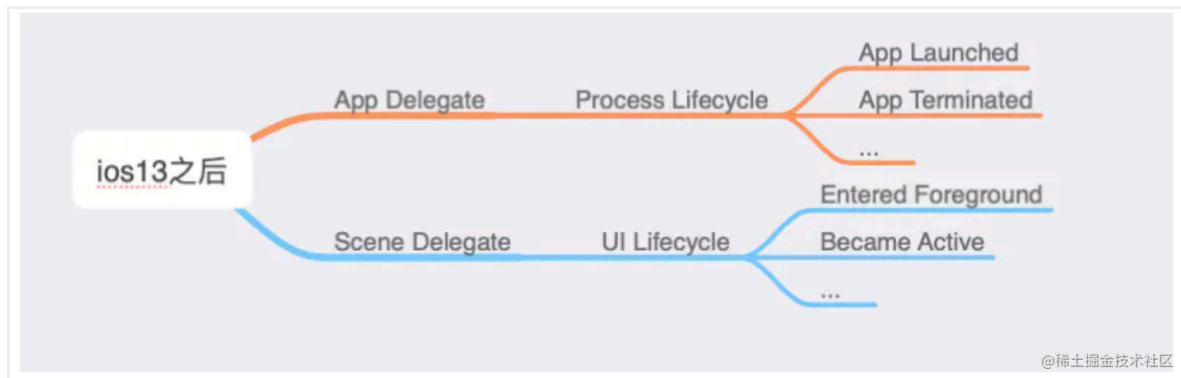
C++ 入口函数

```
int main( int argc, char* argv[] )
{
    NS::AutoreleasePool* pAutoreleasePool = NS::AutoreleasePool::alloc()->init();
    MyAppDelegate del;
    NS::Application* pSharedApplication = NS::Application::sharedApplication();
    pSharedApplication->setDelegate( &del );
    pSharedApplication->run();
    pAutoreleasePool->release();
    return 0;
}
```

UISceneDelegate

ios13 之后，Apple 官方新增 UISceneDelegate 抽象接口类用于实现 UI 生命周期，而 AppDelegate 只负责 APP 的生命周期管理，在此之前，APP 的生命周期和 UI 生命周期是由 AppDelegate 全权负责的。





在ios13 之前，负责初始化应用程序中 Window 在 UIApplicationDelegate 中的 (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions 函数中进行。

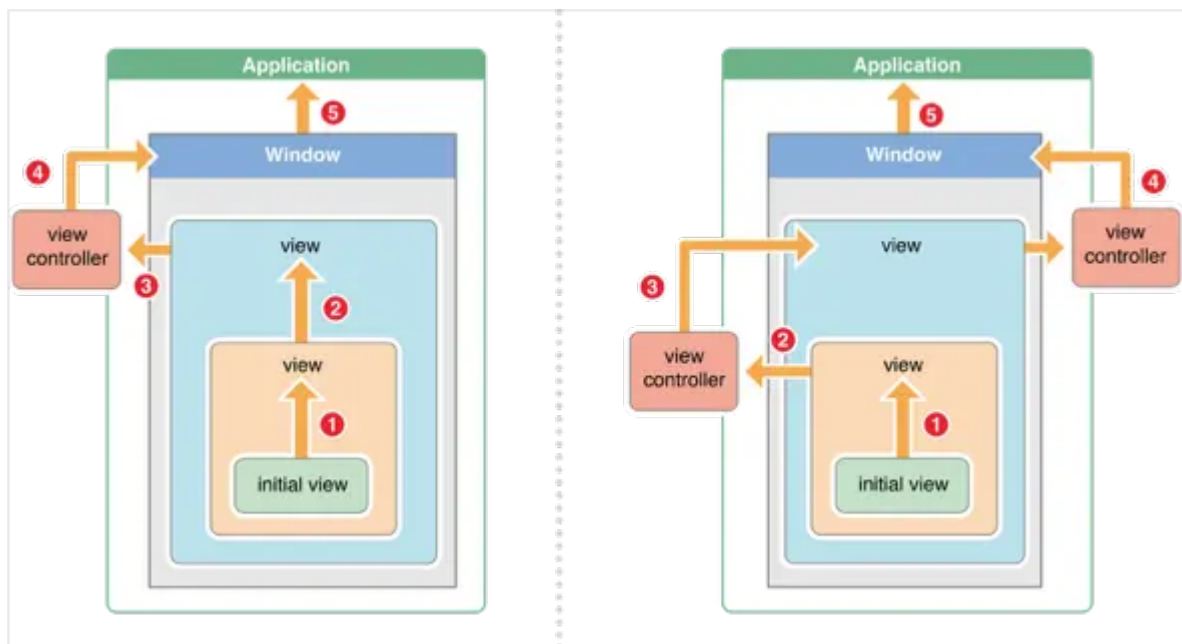
而在ios13 之后，负责初始化应用程序中 Window 在 UIResponderDelegate 中的 (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)session options:(UISceneConnectionOptions *)connectionOptions 函数中进行。

Apple Event

事件传递和响应

IOS 事件由响应链和传递链构成。一般事件先通过传递链传递下去。如果最上层不能响应，那么一层一层通过响应链找到能响应的 UIResponder。iOS 中只有继承了 UIResponder 的对象才能够接受处理事件。UIResponder 是响应对象的基类，定义了处理上述各种事件的接口。常见的子类有：UIView，UIViewController，UIApplication 和 UIApplicationDelegate。

iOS 中的 view 之间逐层叠加，当点击了屏幕上的某个 view 时，这个点击动作会由硬件层传导到操作系统并生成一个事件 (event)，这个事件将会进入 Application 中的事件队列当中，需要处理的事件会通过传递链从系统向最上层 view 传递，Application -> window -> root view -> ... -> first view，找到响应的 view 之后，通过响应链将被响应 view 向系统传递，first view -> super view -> ... -> view controller -> window -> Application -> AppDelegate



UIResponse

UIResponder 提供了许多方法来处理事件，其中事件可以分为三大类型：

- 触摸事件：
 - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
 - (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
 - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
 - (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
- 加速计事件：
 - (void)motionBegan:(UIEventSubtype)motion withEvent:(UIEvent *)event;
 - (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event;
 - (void)motionCancelled:(UIEventSubtype)motion withEvent:(UIEvent *)event;
- 远程控制事件
 - (void)remoteControlReceivedWithEvent:(UIEvent *)event;

UIEvent

UIEvent 记录事件产生的时刻和类型，每产生一个事件就会产生一个 UIEvent 对象。

UITouch

用户用手指触摸屏幕时，会创建一个与手指相关联的 UITouch 对象，一根手指对应一个 UITouch 对象。其中保存着跟手指相关的信息，比如触摸的位置、时间、阶段等。当手指移动时，系统会更新同一个 UITouch 对象，使之能够一直保存该手指在的触摸位置。当手指离开屏幕时，系统会销毁相应的 UITouch 对象。

UIKit(Apple UI 框架)

UIView

UIView 是基本类，提供 UI 的渲染，其中包含了 addSubview 等 UI 基本功能，用于与用户交互和展示界面功能，对应 MVC 框架的 V。

UIView 响应事件时，会从上层 view 向下层 view 传递被响应的 view，可以通过重写 `-(nullable UIView *)hitTest:(CGPoint)point withEvent:(nullable UIEvent *)event` 方法来修改被响应的 view。

如果 UIView 被隐藏 (`hidden == true`), 禁用用户操作 (`userInteractionEnabled == false`), 不透明度小于等于 0.01(`alpha <= 0.01`), 则 UIView 不接受事件处理。

UIStackView

继承自 UIView, 拥有自动布局 view 的功能, 支持水平布局和垂直布局, 其中多了一个添加子 view 的功能为 `addArrangedSubview`, `arrangedSubviews` 和 `Subviews` 的顺序意义和关系:

- `Subviews`: 它的顺序实际上是图层覆盖顺序, 也就是视图元素的 z 轴
- `arrangedSubviews`: 它的顺序代表了 stack 堆叠的位置顺序, 即视图元素的 x 轴和 y 轴
- 如果一个元素没有被 `addSubview`, 调用 `arrangedSubviews` 会自动 `addSubview`
- 当一个元素被 `removeFromSuperview`, 则 `arrangedSubviews` 也会同步移除
- 当一个元素被 `removeArrangedSubview`, 不会触发 `removeFromSuperview`, 它依然在视图结构中

UIScrollView

继承自 UIView, 拥有 UI 滚动功能。

UIControl

继承自 UIView, 拥有事件绑定和处理功能, 当绑定的事件触发时, 不会给调用的对象直接发消息, 而是会给 Application 发消息来调用对象, Application 有对象的引用计数来判断对象是否为空。

UITableView

继承自 UIScrollView, 通过读取数据创建 `UITableViewCell`, 加入其中来进行数据展示, 其中包含头部视图 (Header View), 底部视图 (Footer View), 多个 Section (每个 Section 包含多个 Cell, 一个 Section 头部, 一个 Section 底部)。其中有 `UITableViewDataSource` 接口成员弱指针, 用于调用并且返回数据, 实现表格的数据展示, 可以从对象池中取 `UITableViewCell` (在创建 `UITableViewCell` 时需注册标识符, 否则从对象池取出指针为空)。有 `UITableViewDelegate` 接口成员弱指针, 用于执行触发生命周期中的事件。

UICollectionView

继承自 UIScrollView, 通过读取数据创建 `UICollectionViewCell` 加入其中进行数据展示, 可自定义多种布局样式, 在创建他时必须传入样式实例 `UICollectionViewLayout`。其中有 `UICollectionViewDataSource` 接口成员弱指针, 用于调用并且返回数据, 实现视图的数据展示, 可以从对象池中取 `UICollectionViewCell` (需要在 `UICollectionView` 中注册对象池标识符, 否则从对象池取会抛异常)。有 `UITableViewDelegate` 接口成员弱指针, 用于执行触发生命周期中的事件。

UICollectionViewLayout

继承自 `NSObject`, 可以自定义实现样式, 可以重写下列函数获取对应元素的布局属性并自定义更改。

- `-(nullable NSArray<__kindof UICollectionViewLayoutAttributes *>`

- *)layoutAttributesForElementsInRect:(CGRect)rect
- - (nullable UICollectionViewLayoutAttributes *)layoutAttributesForItemAtIndex:(NSIndexPath *)indexPath;
- - (nullable UICollectionViewLayoutAttributes *)layoutAttributesForSupplementaryViewOfKind:(NSString *)elementKind atIndex:(NSIndexPath *)indexPath;
- - (nullable UICollectionViewLayoutAttributes *)layoutAttributesForDecorationViewOfKind:(NSString *)elementKind atIndex:(NSIndexPath *)indexPath;

UINavigationController

UINavigationController 是控制类，用于控制 UIView 等的逻辑功能，对应 MVC 的 C。

UINavigationController

继承自 UINavigationController，提供 UIView 界面导航功能，作为一个父控制器来管理子控制器 (控制器栈)，第一个入栈的控制器为根控制器 (RootViewController)，提供对控制器的入栈出栈等操作，提供导航条 (在 UINavigationController 中对 UINavigationController 进行了类别定义)。

UIToolBar

继承自 UIView，工具栏 UI，通过向其中添加绑定事件的 UIBarButtonItem 来添加功能按钮。

UINavigationController

继承自 UINavigationController，导航栏 UI，用于导航分层内容的控，主要属性是左按钮，中心标题和可选的右按钮。

UITabBar

继承自 UINavigationController，标签栏 UI，用于选中其中的标签来切换内容显示，通过向其中添加 UITabBarItem 来增加标签。