UI

- VehicleServiceViewController：VehicleUIHomePageHost
  - VehicleDisplayContainerCell：头部续航里程等
    - VehicleControlBackgroundView：车辆视图
      - containerView
        - VehicleStatusView
    - VehicleManagerView 续航里程
    - MPBlessView：动画视图
  - VehicleDynamicAlertCell：状态提醒气泡
  - VehicleControlContainerCell：原子工具模块
  - HomeEntranceViewContainer：智驾

- VehicleUIHomePageHost 负责UICollectionView相关事务

首次展示

VehicleUI.HomePageItemCell

- VehicleDisplayContainerCell 车辆
- VehicleDynamicAlertCell 原状态卡片
- VehicleControlContainerCell 原子控制页面
- BubbleCell： 状态气泡
- SceneCard：场景卡片

执行路径：

- `-[VehicleUIHomePageHost configureDataSource]`
  - `-[VehicleUI.HomePageItemCell prepareForDisplay]`
    - `-[VehicleUI.HomePageItemCell reload]`

HomeEntranceViewContainer: VehicleHomeContentProvider

埋点

```
MPVehicle.report("CarControlExpo",
```

下拉刷新

继承关系：VehicleServiceViewController: VehicleUIHomePageHost :UIViewController

- `-[VehicleServiceViewController change(to newVehicle: ]`：总入口
  - `self.vehicle.serviceController.updateHomeFromCloud`： 接口刷新回调
    - `-[VehicleUIHomePageHost notifyContentToRefresh]`
      - `-[VehicleUIHomePageHost _exposeEngine].trackItems`：触发item的`refreshIfNeeded`方法
- 通知刷新：`NotificationCenter.default.addObserver(**self**, selector: **#selector**(onRequestToReload(_:)), name: "vehicle.home.reload.FsUixMqfiGnUHzLG", object: **nil**)`
- `-[VehicleServiceController updateHomeFromCloud]`
  - 车口首页下拉刷新接口：`/app/api/icar/v2/function/config`

车图

- 2D：New2DVehicleView
- 3D： Flexi3DVehicleView

接口

首页：https://app-stg.nio.com

- 车口首页下拉刷新接口：`/app/api/icar/v2/function/config`
- 查询广告-全新车型：`/app/api/icar/v2/function/section`
  - 定时刷新：每5秒请求
- 我的车列表：`/app/api/icar/v2/switch/cars`
- 车定位：`/api/1/lifestyle/map/rgeo`
  - 定时刷新：每20秒请求
- 车状态？？：`/app/api/icar/v2/user/orders`
  - 定时刷新：每20秒请求
- 北极星的输入：`/n/c/csd-config/config/group/batch`
- 旅程简要信息：`/app/api/icar/v2/lightjourney/info`

车信息：https://icar-stg.nio.com

- 报警信息：`/api/1/vehicle/${vehicleId}/alarm_for_app`
  - 定时：每30秒请求
- 车辆状态：`/api/2/rvs/vehicle/${vehicleId}/status`
  - 定时：每30秒请求
- 车辆信息：`/api/2/rvs/vehicle/${vehicleId}/info`，下拉刷新，会请求该接口
  - 接口文档

待确定

- 心跳日志相关：https://ekko-stg.nio.com/ekko/api/app/log
- https://api-fx-stg.nio.com/offlinepkg/v1/publish/list/query
  - 像是离线化相关的
- https://mercury-app-api-stg.nio.com/api/v1/otd/mer/service-aggregation/homepage/noauth/accessory/car-owner
- 领航服务订阅状态查询：https://ado-user-sub-stg.nio.com/ado-smart-driving-community-client/v1/entrance/queryEntranceV2

无法抓包

- https://sensors.nio.com
- https://stat-stg.nio.com

孟祥宝

- VehicleUIHomePost：ViewController基类
  - nio_cn 、nio_en 在用，是个壳子，包含了CollectionView
- 涉及组件：VehicleCommon/VehicleUI、VehicleCommon/MPVehicle
- VehicleServiceViewController 首页VC
  - VehicleHomeController 首页collectionView datasource
  - chargeMapCell 地图Cell
  - journeyEntranceCell 旅程入口
    - 参照这个写
- 更改示例
  internal static let alert = VehicleUI.HomePageItemRegistration(VehicleDynamicAlertCell.self, for: .vehicleAlert)

  internal static let sceneCard = VehicleUI.HomePageItemRegistration(NewVehicleDynamicAlertCell.self, for: .vehicleAlert)

获取当前车辆

```
guard let vehicle = NIOCarGarage.currentVehicle else {
    return
}
vehicle.$rawStatus
    .map { _ in }
    .sink { _ in }
vehicle.currentBLEKey
```

rawStatus // 原始数据，可监听
displayStatus // 处理后的状态

- 视图为了接收曝光，所以

首页结构

VehicleCommon.VehicleServiceViewController: VehicleCommon.VehicleUIHomePageHost

VehicleCommon.VehicleUIHomePageHost

> 管理了一个CollectionView

- Controller : VehicleHomeController： 用于为CollectionView注册Cell、更新数据等

`VehicleHomeController: NSObject`

- itemRegistrations：[元素Id: cell]
  - `[VehicleUI.HomePageItemIdentifier: VehicleUI.HomePageItemRegistration]`
- loadInitials：初始化snapshot
- reset： 空方法？？
- register，向CollectionView中注册Cell
- reload: `dataSource.apply(snapshot)`
- 给VehicleUIHomePageHost提供拓展方法
  - `-[VehicleUIHomePageHost reload]`
  - `-[VehicleUIHomePageHost apply]`

VehicleUI.HomePageItemIdentifie

```
typealias Snapshot = NSDiffableDataSourceSnapshot<VehicleUI.HomePageSectionIdentifier,
VehicleUI.HomePageItemIdentifier>
```

VehicleUI.HomePageItemIdentifier

```
struct HomePageItemIdentifier: Equatable, Hashable, ExpressibleByStringLiteral {
      public init(stringLiteral value: StringLiteralType) {
          self.init(rawValue: value)
      }
      public init(rawValue: String) {
          self.rawValue = rawValue
      }
      /// 仅存在string属性，swift4.1之后可以由系统自动实现hashable和Equatable协议
      /// 详细原因：如果 ItemId 结构体仅包含遵循 Hashable 协议的存储属性，并且你没有自定义 Equatable
协议的 == 运算符，Swift 编译器会自动为你合成 hash(into:) 方法和 == 运算符。
}
```