

Develop Guideline

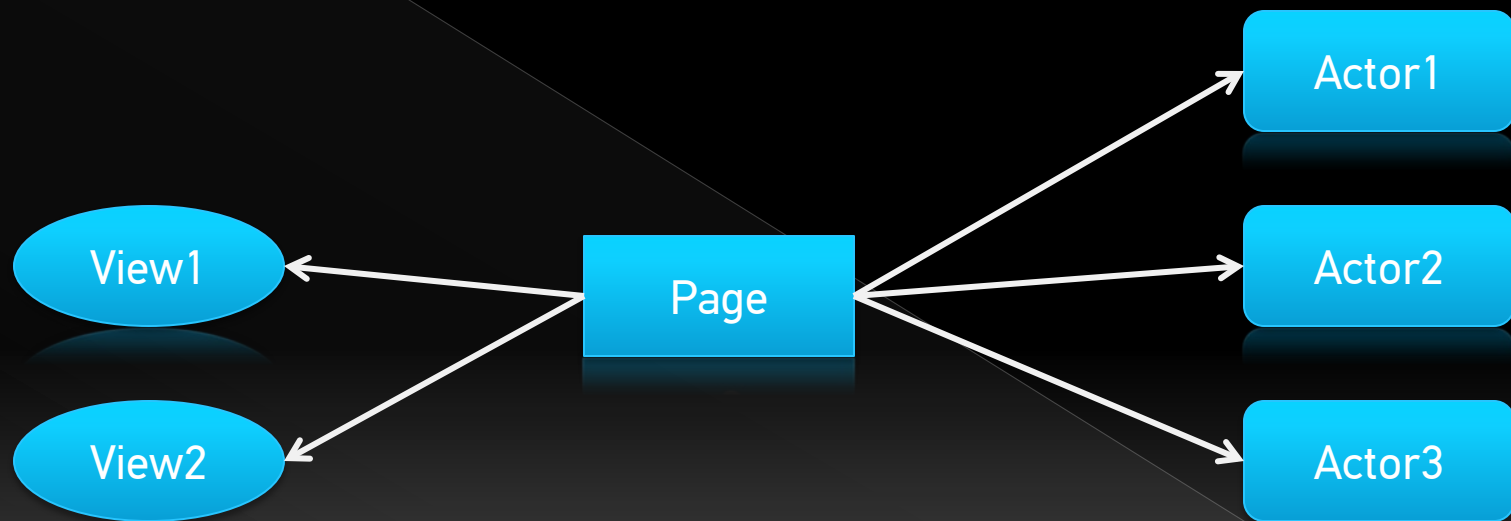
Patterns

Pattern - MVCS

- Model
- View
- Controller
- Service

Dependencies Map

依赖关系图



Static 静态类

```
private var str:String;  
str = Model.foo();
```

Singleton 单例

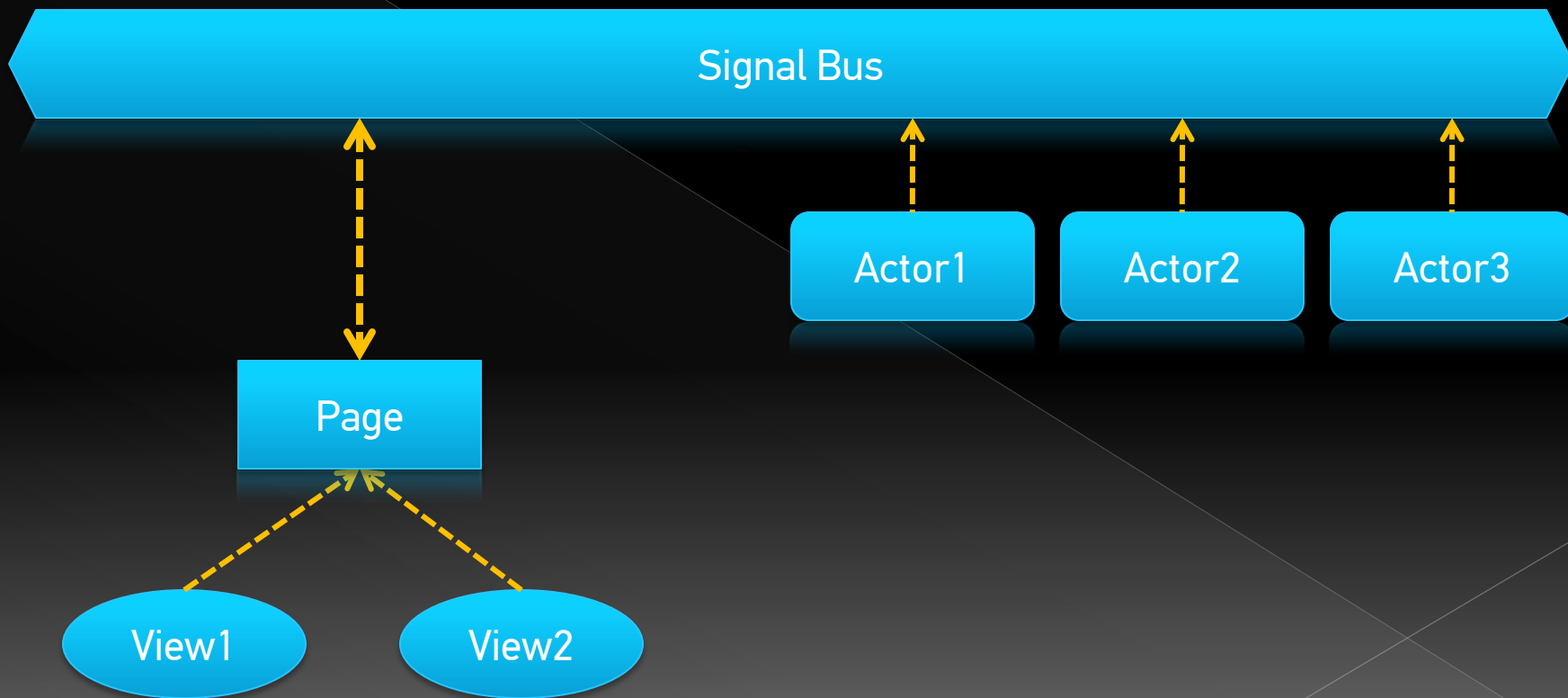
```
private var str:String;  
str = Model.getInstance().foo();
```

Dependency Injection 依赖注入

```
[Inject]  
public var model:IModel;  
  
private var str:String;  
str = model.foo();
```

Event Flow

事件流



View

- 作为视图元件(MC , Graphic , Button , 组件等)的容器。
- **监听但不处理**视图元件发出的事件，仅转发为对应的自定义信号(Signal)。

View

提供以下公开方法：

- `init()` //初始化
- `update()` //更新
- `destroy()` //销毁

Page(Mediator)

- 监听View发出的信号。
- 调用系统中其他对象(Model, Service...)的方法或数据, 或者直接发出系统信号。
- 监听系统中的信号, 包括Gaia中的事件, 并对View执行相关的操作。

Page(Mediator)

包含以下保护方法：

- `init ()` //初始化视图和代理设置
- `destroy ()` //销毁视图和代理设置

Page(Mediator)

- 访问assets对象来获取各种外部资源。
(通过Gaia框架的加载机制实现)
- 获取系统里其他对象 (包括Model, Service, Proxy和AppSignalBus等) 的引用。
(通过Robotlegs的注入功能实现)

Controller

- Controller层主要由Command(命令)模式实现。
- Robotlegs架构提供了将事件映射为一个命令(Command)的功能。通过Signals插件的扩展，信号也可以映射成为一个命令。
- 命令是一个无状态的对象。主要用于封装操作，解除视图与系统其他对象之间的依赖关系。

Controller - Signals

- 架构提供了统一的信号传递器AppSignalBus，所有系统对象可以通过它来发送信号。
- 系统信号可以由其相关联的命令(Command)来处理，或者被其他Page(Mediator)监听。
- 只有Page(Mediator)可以监听信号。

Model/Service

- Model用于存储状态，封装应用层逻辑。
- Service不保持状态，只负责外部资源的获取，并处理转换为系统内使用的数据格式。
- Model和Service通过发出系统信号通知系统内的其他对象。
- Model和Service都**不应该**监听事件。
- Model和Service应该是基于**接口**的实现。

Other Classes

- `MainContext`
- `mvcs.MediatorPage`
- `mvcs.AppActor`

MainContext

MainContext是应用程序主环境的配置类，包含以下几种依赖注入的关系定义：

- 系统中Model/Service/Proxy的映射
- 命令与事件/信号的映射
- 视图与代理的映射

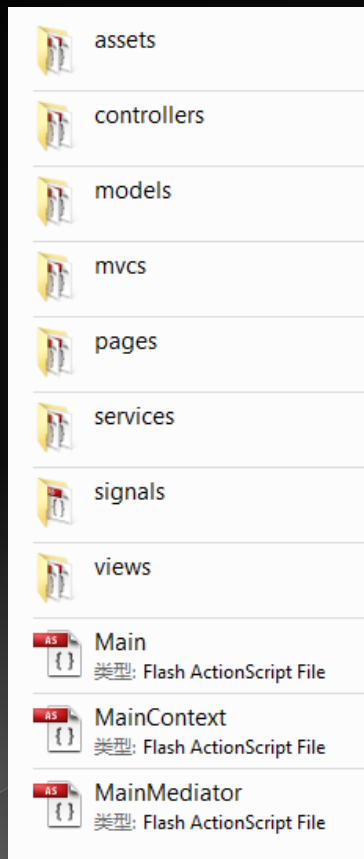
MediatorPage

MediatorPage类继承自Gaia框架中Page的基类AbstractPage，扩展了系统信号的传递功能(通过注入的AppSignalBus)。

AppActor

AppActor类继承自Robotlegs架构中的Actor，扩展了系统信号的传递功能(通过注入的AppSignalBus)。

Structure of *src* folder



- models, views, controllers, services
对应MVCS模式的包
- signals
信号类，代替内置的Event
- pages
Gaia默认的Page包
- assets
通过ActionScript编译的资源
- mvcs
MVCS模式的扩展基类