[iOS] 阿里开源组件化方案 BeeHive

时间 2016-11-22 11:42:16 Github

原文 https://github.com/alibaba/BeeHive/blob/master/README-CN.md

主题 开源 iOS 开发

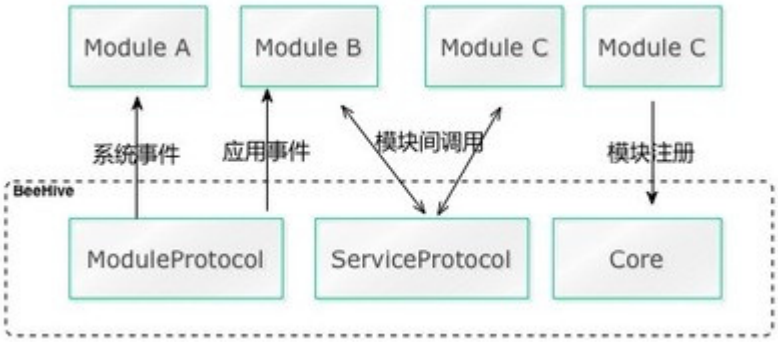
BeeHive

:book: English Documentation | :book: 中文文档

0. 概述

`BeeHive` 是用于 `iOS` 的 `App` 模块化编程的框架实现方案，吸收了 `Spring` 框架 `Service` 的理念来实现模块间的 `API` 耦合。

0.1 基本架构



0.2 实现特性

- 插件化的模块开发运行框架
- 模块具体实现与接口调用分离

- 模块生命周期管理，扩展了应用的系统事件

0.3 设计原则

因为基于 `Spring` 的 `Service` 理念，虽然可以使模块间的具体实现与接口解耦，但无法避免对接口类的依赖关系。

为什么不使用 `invoke` 以及动态链接库技术实现对接口实现的解耦，类似 `Apache` 的 `DSO` 的方式？

主要是考虑学习成本难度以及动态调用实现无法在编译检查阶段检测接口参数变更等问题，动态技术需要更高的编程门槛要求。

0.4 项目名来源

`BeeHive` 灵感来源于蜂窝。蜂窝是世界上高度模块化的工程结构，六边形的设计能带来无限扩张的可能。所以我们用了 `BeeHive` 来做为这个项目的命名。

1 模块生命周期的事件

`BeeHive` 会给每个模块提供生命周期事件，用于与 `BeeHive` 宿主环境进行必要信息交互，感知模块生命周期的变化。

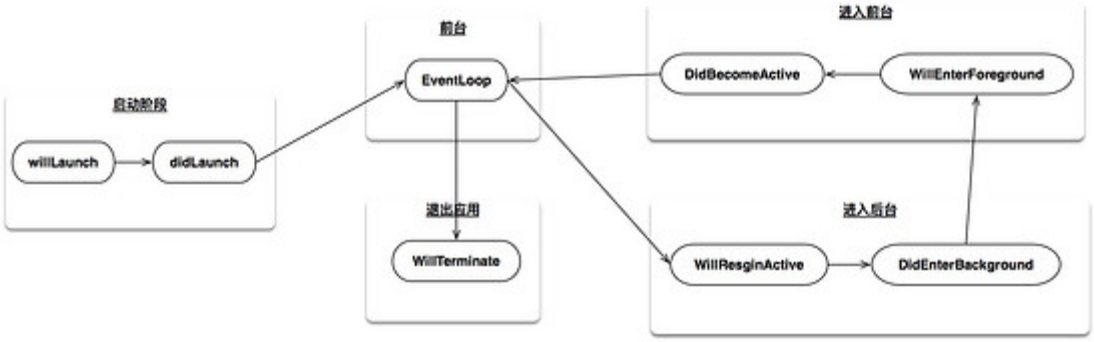
事件分为三种类型：

- 系统事件
- 通用事件
- 业务自定义事件

1.1 系统事件

系统事件通常是 `Application` 生命周期事件，例如 `DidBecomeActive` 、 `WillEnterBackground` 等。

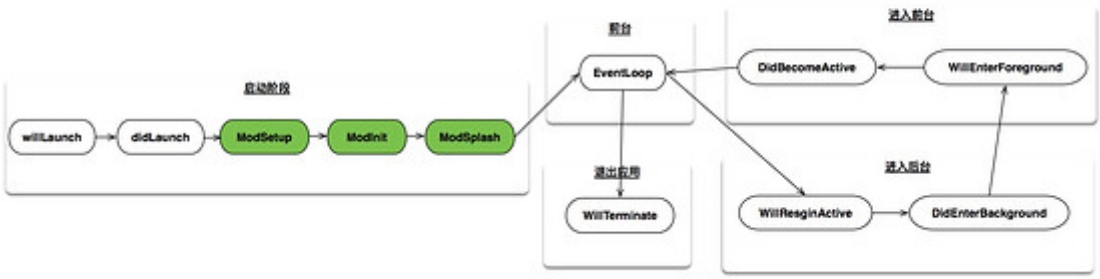
系统事件基本工作流如下：



1.2 通用事件

在系统事件的基础之上，扩展了应用的通用事件，例如 `modSetup` 、 `modInit` 等，可以用于编码实现各插件模块的设置与初始化。

扩展的通用事件如下：



1.3 业务自定义事件

如果觉得系统事件、通用事件不足以满足需要，我们还将事件封装简化成 `BHAppdelgate` ，你可以通过继承 `BHAppdelegate` 来扩展自己的事件。

2. 模块注册

模块注册的方式有静态注册与动态注册两种。

2.1 静态注册

通过在 `BeeHive.plist` 文件中注册符合 `BHModuleProtocol` 协议模块类:

2.2 动态注册

```
@implementation HomeModule



BH_EXPORT_MODULE()  // 声明该类为模块入口



@end
```

在模块入口类实现中 使用 `BH_EXPORT_MODULE()` 宏声明该类为模块入口实现类。

2.3 异步加载

如果设置模块导出为 `BH_EXPORT_MODULE(YES)`，则会在启动之后第一屏内容展现之前异步执行模块的初始化，可以优化启动时时间消耗。

3. 编程开发

`BHModuleProtocol` 为各个模块提供了每个模块可以 `Hook` 的函数，用于实现插件逻辑以及代码实现。

3.1 设置环境变量

通过 `context.env` 可以判断我们的应用环境状态来决定我们如何配置我们的应用。

```
-(void)modSetup:(BHContext *)context

{
```

```
    switch (context.env) {

        case BHEnvironmentDev:

        //....初始化开发环境

        break;

        case BHEnvironmentProd:

        //....初始化生产环境

        default:

        break;

    }

}
```

3.2 模块初始化

如果模块有需要启动时初始化的逻辑，可以在 `modInit` 里编写，例如模块注册一个外部模块可以访问的 `Service` 接口

```
-(void)modInit:(BHContext *)context

{

    //注册模块的接口服务

    [[BeeHive shareInstance] registerService:@protocol(UserTrackServiceProtocol) service:[BHUserTrackViewController class]];

}
```

3.3 处理系统事件

系统的事件会被传递给每个模块，让每个模块自己决定编写业务处理逻辑，比如 `3D-Touch` 功能

```
-(void)modQuickAction:(BHContext *)context

{

    [self process:context.shortcutItem handler:context.scompletionHandler];

}
```

3.4 模间调用

通过处理 `Event` 编写各个业务模块可以实现插件化编程，各业务模块之间没有任何依赖， `core` 与 `module` 之间通过 `event` 交互，实现了插件隔离。但有时候我们需要模块间的相互调用某些功能来协同完成功能。

通常会有三种形式的接口访问形式：

1. 基于接口的实现 `Service` 访问方式（ `Java spring` 框架实现）
2. 基于函数调用约定实现的 `Export Method` (`PHP` 的 `extension` ， `ReactNative` 的扩展机制)
3. 基于跨应用实现的 `URL Route` 模式(`iPhone App` 之间的互访)

我们目前实现了第一种方式，后续会逐步实现后两种方式。

基于接口 `Service` 访问的优点是可以编译时检查发现接口的变更，从而及时修正接口问题。缺点是需要依赖接口定义的头文件，通过模块增加得越多，维护接口定义的也有一定工作量。

3.4.1 定义接口

以为 `HomeServiceProtocol` 为例。

```
@protocol HomeServiceProtocol <NSObject, BHServiceProtocol>
```

```
- (void)registerViewController:(UIViewController *)vc title:(NSString *)title iconName:(NSString *)iconName;


@end
```

3.4.2 注册 `Service`

有三种方式：

声明式注册

```
@implementation HomeService


BH_EXPORT_SERVICE()


@end
```

`API` 注册

```
[[BeeHive shareInstance] registerService:@protocol(HomeServiceProtocol) service:[BHViewController class]];
```

`BHService.plist` 注册

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">

<plist version="1.0">
```

```
    <dict>
        <key>HomeServiceProtocol</key>
        <string>BHViewController</string>
    </dict>
</plist>
```

3.4.3 调用 `Service`

```
#import "BHService.h"



id< HomeServiceProtocol > homeVc = [[BeeHive shareInstance] createService:@protocol(HomeServiceProtocol)];
```

3.5 单例与多例

对于有些场景下，我们访问每个声明 `Service` 的对象，希望对象能保留一些状态，那我们需要声明这个 `Service` 对象是一个单例对象。

我们只需要在 `Service` 对象中实现事件函数

声明

```
-(BOOL) singleton
{
    return YES;
}
```

通过 `createService` 获取的对象则为单例对象，如果实现上面函数返回的是 `NO`，则 `createService` 返回的是多例。

```
id< HomeServiceProtocol > homeVc = [[BeeHive shareInstance] createService:@protocol(HomeServiceProtocol)];
```

3.6 上下文环境 Context

- 初始化设置应用的项目信息，并在各模块间共享整个应用程序的信息

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions

{

    [BHContext shareInstance].env = BHEnvironmentDev; //定义应用的运行开发环境

    [BHContext shareInstance].application = application;

    [BHContext shareInstance].launchOptions = launchOptions;

    [BHContext shareInstance].moduleConfigName = @"BeeHive.bundle/CustomModulePlist";//可选，默认为 BeeHive.bundle/BeeHive.plist

    [BHContext shareInstance].serviceConfigName =  @"BeeHive.bundle/CustomServicePlist";//可选，默认为 BeeHive.bundle/BHService.plist

    [[BeeHive shareInstance] setContext:[BHContext shareInstance]];



    [super application:application didFinishLaunchingWithOptions:launchOptions];
```

```
    id<HomeServiceProtocol> homeVc = [[BeeHive shareInstance] creat
eService:@protocol(HomeServiceProtocol)];



    if ([homeVc isKindOfClass:[UIViewController class]]) {

        UINavigationController *navCtrl = [[UINavigationController
alloc] initWithRootViewController:(UIViewController*)homeVc];



        self.window = [[UIWindow alloc] initWithFrame:[UIScreen mai
nScreen].bounds];

        self.window.rootViewController = navCtrl;

        [self.window makeKeyAndVisible];

    }



    return YES;

}
```

更多细节可以参考 Example 用例。

4. 集成方式

`cocoapods`

```
pod "BeeHive", '1.0.0'
```

5. 作者

- 一渡 *shijie.qinsj@alibaba-inc.com*
- 达兹 *dazi.dp@alibaba-inc.com*

6. 开源许可证