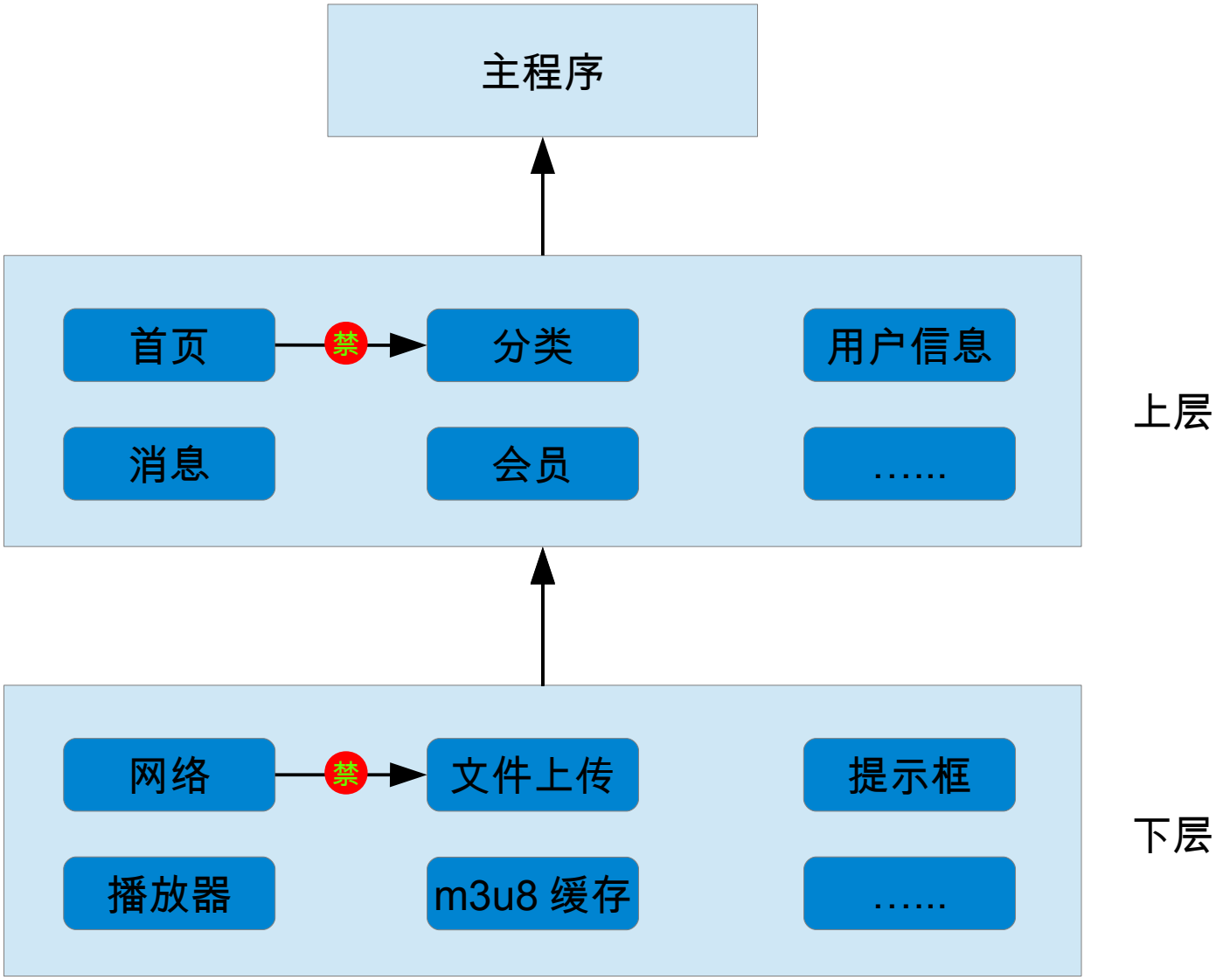
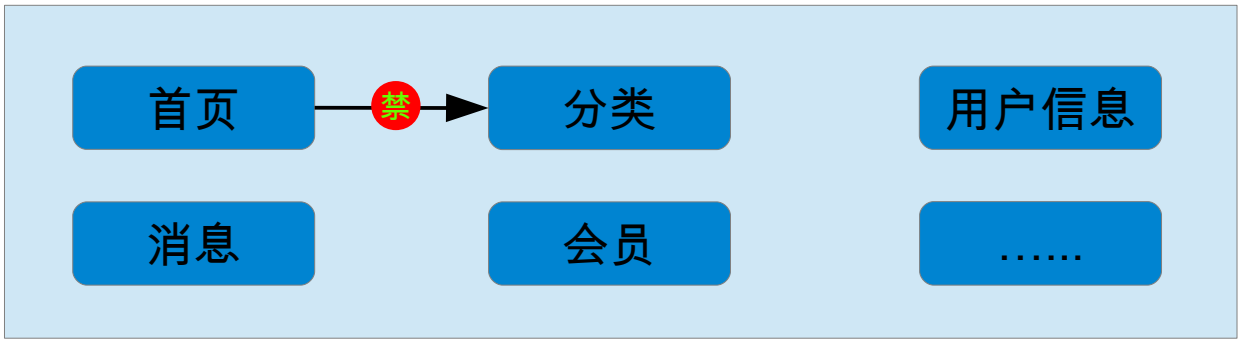
App 架构



app 典型架构

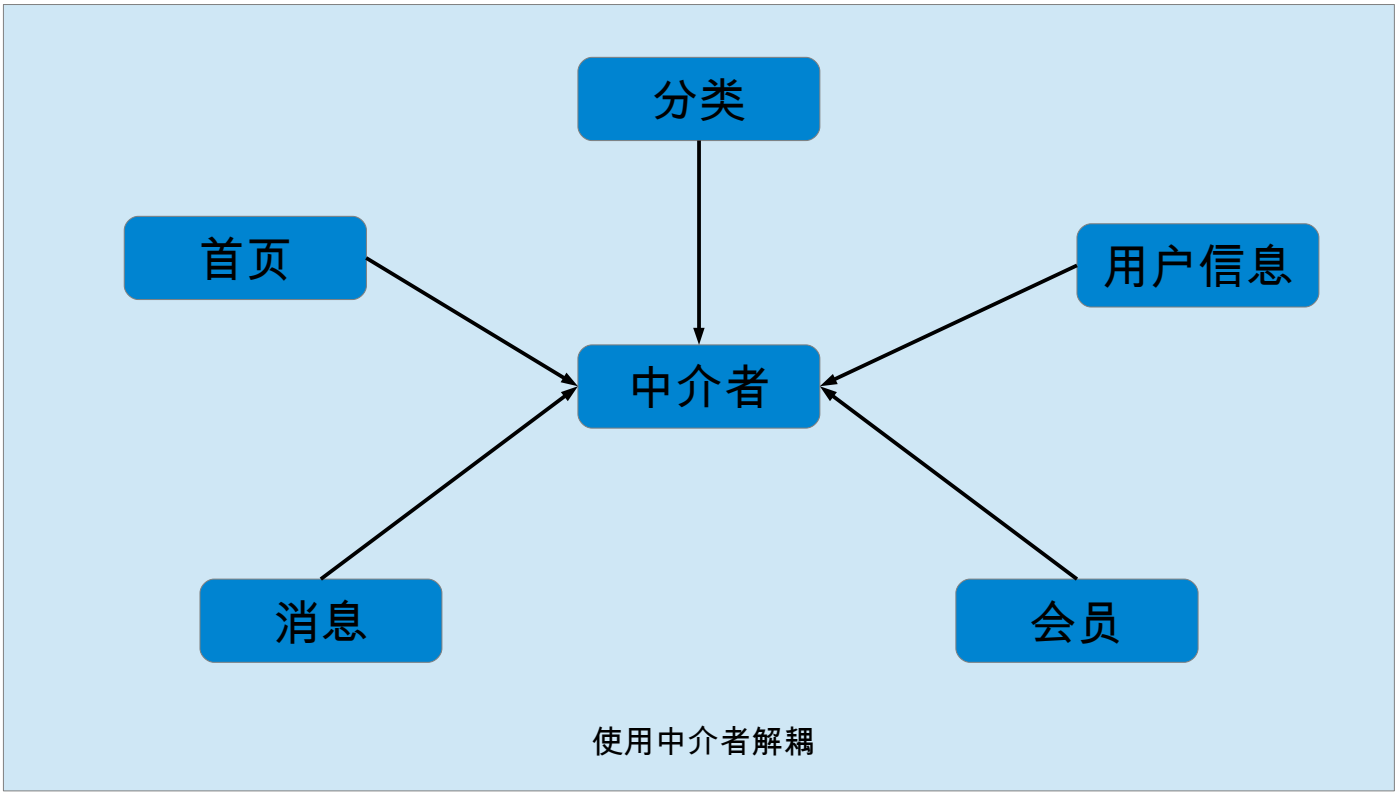
1. 下层为上层提供服务；下层不可以调用上层。
2. 同层级模块之间不能相互引用。
3. 允许跨层访问。

模块间通信



上层

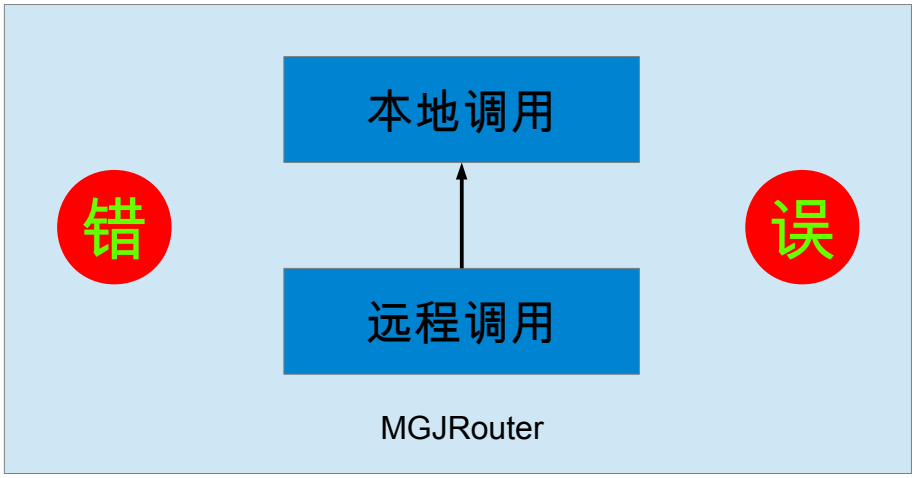
上层业务模块间难免会出现模块间相互调用，比如首页控制器 HomeVC 跳转到分类控制器 CategoryVC ，如果直接在 HomeVC 中直接引入 CategoryVC ，模块就耦合到一起了，这时候所要做的就是解藕。



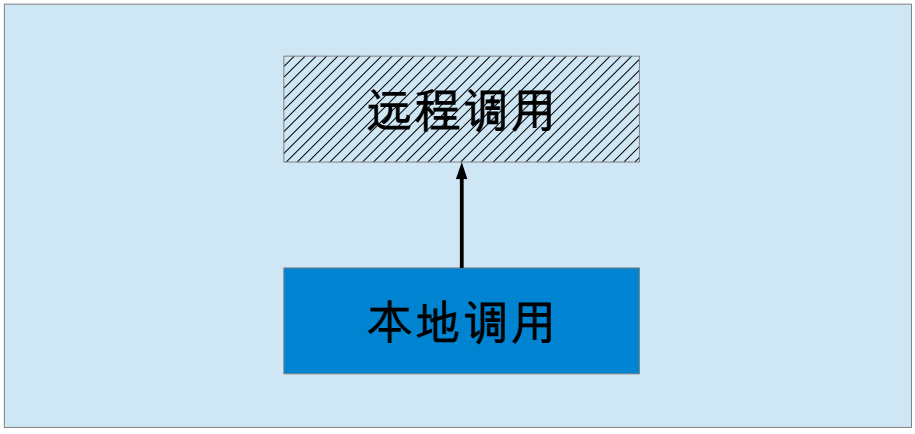
使用中介者解耦

组件化

提到组件化，很多人就会想到 URL 方式实现；例如 JRRoutes 、 MGJRouter 。
但 URL 方式不是组件化的必须方式！ 请记住：我们的目的是解耦。



蘑菇街有个思想上的错误：远程调用为本地调用提供服务。



正确的做法是：本地调用为远程调用提供服务。
但我们的目的是为了解耦，本地调用已经能够解耦，远程调用就没必要有了。

组件化实现 -1

```
@interface ViewController : UIViewController
@property (nonatomic) NSString *name;
@property (nonatomic) NSString *age;
@end
```

```
NSDictionary *dict = @{
    @"name": @"憨豆",
    @"age": @60
};
```

思想：使用 runtime 为控制器属性赋值。

```
/**
    生成控制器（非nib、非storyboard方式）

    @param  className    控制器类名
    @param  params       控制器参数，params.key与viewController.property有相同名称才会赋值
 */
+ (UIViewController *)vc:(NSString *)className params:(nullable NSDictionary *)params;
```

```
// 生成控制器（非nib、非storyboard方式）
+ (UIViewController *)vc:(NSString *)className params:(nullable NSDictionary *)params {
    Class cls = NSClassFromString(className);
    NSAssert1(cls != NULL, @"没有此控制器: %@", className);
    UIViewController *vc = [cls new];
    [self assignValueToVC:vc params:params];
    return vc;
}
```

```
// 为控制器的属性赋值
+ (void)assignValueToVC:(UIViewController *)vc params:(nullable NSDictionary *)params {
    unsigned int count = 0;
    objc_property_t *props = class_copyPropertyList([vc class], &count);
    for (unsigned int i = 0; i < count; i++) {
        objc_property_t prop = props[i];
        const char *name = property_getName(prop);
        NSString *key = [NSString stringWithUTF8String:name];
        // 没有此key，跳过;
        if (![params.allKeys containsObject:key]) {
            continue;
        }
        id value = params[key];
        [vc setValue:value forKey:key];
    }
    free(props);
}
```

有同学在想，为什么使用 NSAssert 呢？这是为了在开发时就发现所传控制器类名错误。

组件化实现 -2

```
/**
   生成控制器（非nib、非storyboard方式）

   @param  className    控制器类名
   @param  params       控制器参数，params.key与viewController.property有相同名称才会赋值
 */
+ (UIViewController *)vc:(NSString *)className params:(nullable NSDictionary *)params;
```

上面只是纯代码写的控制器；下面加上 nib 、 storyboard ，相应的实现就不给出了。

```
/**
   nib方式生成控制器。

   @param  className    控制器类名
   @param  params       控制器参数，params.key与viewController.property有相同名称才会赋值

   @note   nib的名称与类的名称保持一致
 */
+ (UIViewController *)nib:(NSString *)className params:(nullable NSDictionary *)params;
```

```
/**
   storyboard方式生成控制器。

   @param  sbName       storyboard名称
   @param  className    控制器类名
   @param  params       控制器参数，params.key与viewController.property有相同名称才会赋值

   @note   storyboard ID 与类的名称保持一致
 */
+ (UIViewController *)storyboard:(NSString *)sbName className:(NSString *)className params:(nullable
NSDictionary *)params;
```

组件化实现 -3

问题 1 ：如果次级页面要给上级页面回调怎么办？
这里我们使用 block 方式。

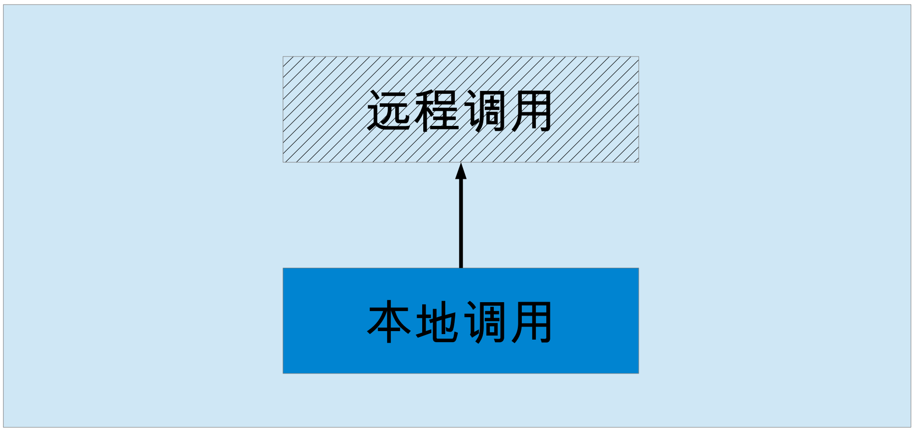
问题 2 ：如果控制器有指定的初始化方法怎么办？
在中介者使用协议。不推荐！！！

```
@protocol XSMediatorProtocol <NSObject>
+ (UIViewController *)mediatorCreateViewController:(nullable NSDictionary *)params;
@end
```

因为每个模块都引入中介者，也就都可以实现该协议，自行提供初始化。

问题 3: 如果次级页面回调采用 delegate 怎么办？
还是采用协议。不过不建议，随着 delegate 增多，协议必然增多；不推荐！！！

组件化实现 -4



本地调用为远程调用提供服务，远程调用的实现依赖于本地调用。
下面的接口几乎可以作为 url 形式实现

```
/**
   □  □  □  Anib-w□ storyboard□  □  ŷ

   @param  className  □  A Û □
   @param  params     □  A Û □ params.key viewController.property□  □  □  □  □ ∂ □  □
 */
+ (UIViewController *)vc:(NSString *)className params:(nullable NSDictionary *)params;
```

但一般不会传递控制器类名，这时候用字典映射 url 与控制器类名就可以了。
与此同时，还需要注意， url 形式只能传递常规参数；但本地调用一般都是非常规参数，例如：model 、 UIImage 之类的，这时候就要舍弃掉非常规参数。