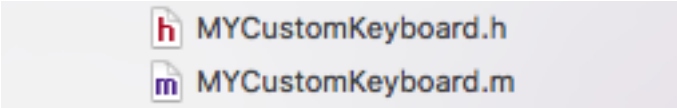
如何以 xib 方式创建自定义 view

1. 简介：记得在成为一名程序猿头的一年里，我习惯使用代码编程，甚至最初时连 stroyboard 都不用了(还好后来有利用链式编程对系统的 autolayout 进行了封装 ,不然光页面布局就会把我累死掉 T T) ,那时候的想法是 :嗯 ,整个项目都是纯代码的，好像很吊的样子。。。过了几个月后，发现，以 storyboard 作为界面布局工具相比纯代码布局可以加快开发效率，而且界面一目了然。。。然后，之后用了 svn 进行团队开发，才发现，多个人使用 main.storyboard 容易产生冲突。这时候，考虑到应该按功能模块划分创建 storyboard，由指定开发人员进行开发管理，这样问题就解决了。。。就这样一年过去了，第二年，和其他同行分享交流经验后，我对 xib 文件产生了兴趣：团队开发，一个页面对应一个 xib，这有效地避免了 svn 的文件冲突，一个模块，可以多个人同时进行开发了。。。再到后来，我又想着怎么来封装自定义 view 快 ,或者说 ,一个 controller.xib 的 view 能做到公用(比如给其他项目用)吗？然后我发现，xib 方式创建的自定义 view，非常直观的可以看出我想要封装的视图的效果 ,这种以 xib 结合代码而非纯代码式的自定义 view 的封装 ,个人认为，是个非常不错的 idea。以下就对 xib 自定义 view 整个封装过程做介绍，其中涉及 xib 自动布局、代码 frame 快速适配等知识。

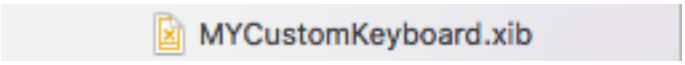
2. 实现：

(1) xib 的创建

首先，我们新建一个 view，发现，纳尼？xib 的勾选项是灰的！没事，我们先不建 xib，比如我们创建了一个自定义键盘的 UIView，它长这样：



ok，接下来，我们要创建 xib 了，右键，新建 UserInterface 栏下的 View，我们输入和自定义类名一样的，它长这样：



(2) xib 的布局

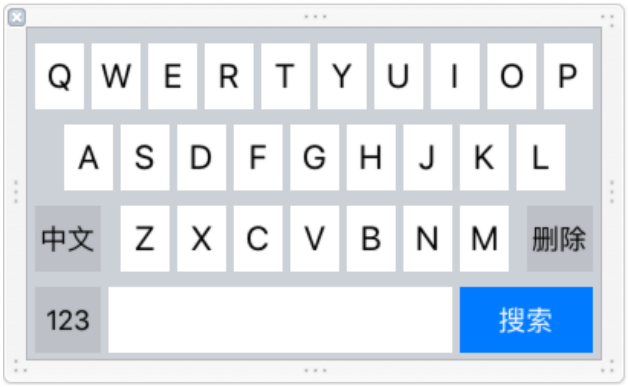
1)先看下效果

嗯，我们看到了三个”很漂亮”的自定义键盘，注意左边那栏，每个键盘都绑定了一个类。这里我要讲下这个三个类，不是对应三个类文件，为什么不是呢？很多人会理解，一个文件就应该是一个类？为啥？扯蛋！都是自定义的键盘，为啥要用三个文件来创建类呢？所以，我们应该把 MYCustomKeyboard 作为基类，而 xib 几个 view 就创建几个对应的子类。对于 MYCustomKeyboard 类的介绍，后面会讲，我们继续：

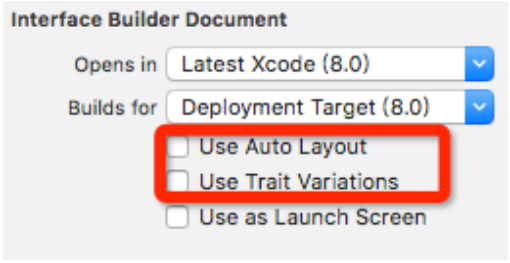
xib 里创建的 view 有的布局是很复杂的，如果用 frame 进行布局的话，那效率太低下了，我们需要借用 autolayout 进行 xib 中自定义视图内控件的布局。

2)注意，autolayout 的核心思想是：相对布局，尽量少出现多个独立的分支依赖，什么意思呢，就是布局完毕后，比如我想改一组同样大小的控件的宽，如果你修改了 n 个参数的话，那 ok，你直接用 frame 父布局好了，autolayout 完全没用对嘛～使用 autolayout 的效果应该我只要改一个控件的宽度，其他相同宽度的同类型控件自动适配完成布局！还有一个布局要点就是，按 command+option 可以对一个或一排或一列控件进行复制！这个是 xib 快速自动布局的关键，有了这个思想，可以节省你大量的布局时间！

3)好了，假如我们布局完了，它长这样：



这时候，我们执行一个关键的操作，为了后面能”完美适配”这个键盘。



我们把这两个项勾掉，别问我在哪？右边选项卡自己找，目的嘛，禁用 autolayout 布局，并取消所有自动布局的约束。勾掉后，你发现，想继续进行自动布局，是不行了，之前做好的约束好像"全不见"了：纳尼！我要继续修改怎么办？？？别慌，继续修改前，把那两栏再勾选上就好，你看看，是不是所有约束又回来了~

(3) 代码方面介绍

(1) 我们先瞅一瞅我们的 MYCustomKeyboardl 类，先上一段.h 代码：

```
#define ShiPei(a)  [UIScreen mainScreen].bounds.size.width/375.0*a
#define HexColor(hexValue) [UIColor colorWithRed:((float)((hexValue & 0xFF0000) >> 16)) / 255.0 green:((float)((hexValue & 0xFF00) >> 8)) / 255.0 blue:((float)(hexValue &
  0xFF)) / 255.0 alpha:1.0]

typedef NS_ENUM(NSUInteger, KeyboardType){
  KeyboardTypeUnKnow,//未知
  KeyboardTypeChinese,//中文键盘
  KeyboardType123,//股票数字键盘
  KeyboardTypeABC,//首字母键盘
  KeyboardTypeEmoji//表情键盘
};

@protocol MYCustomKeyboardDelegate <NSObject>

- (void)onClickKeyboardButtonTypeInputString:(UIButton *)inputStringBtn;//点击会输入字符串的按钮
- (void)onClickKeyboardButtonTypeCommand:(UIButton *)commandBtn;//点击会执行指定命令的按钮

@end

//-----------------------------------------------------------------MYCustomKeyboard----------------------------------------------------------

@interface MYCustomKeyboard : UIView//自定义键盘基类，高度基于ihpne6进行适配的216高度，即ShiPei(216)

@property (assign, nonatomic) id <MYCustomKeyboardDelegate> delegate;

@property (assign, nonatomic) KeyboardType type;

+ (instancetype)customKeyBoardWithType:(KeyboardType)type;

@end
```

首先，我们看到的是基类，之前 xib 中创建的视图都是属于子类，对吗？是的，基类这边我们定义了一个协议，子类继承了这个协议，外部只要实现了，就可以拿到所有键盘的相关事件监听；同样，type 是我们定义的一个枚举，标识着我这个子类是哪个键盘，子类实例化的时候进行赋

值，说明”我是什么键盘”就好。这是两点，再有，我们看到它没有对象创建方法，只有以下方法：

```
+ (instancetype)customKeyBoardWithType:(KeyboardType)type;
```

是不是很眼熟？看到这里，你应该猜到了，没错，因为对于没使用过这个键盘的人来说，根本不知道有几个键盘，有什么键盘，因为我们把这些键盘写成了子类了，这里使用工厂方法，外界只需传入想要键盘对应的枚举，我们基类就可以实力化出对应的键盘传给外界。也就是说，外界根本不需要知道有哪些键盘子类！

以上三点对基类的描述，大家看完后应该会有所体会，基类一般都用来干些啥～

(2) 接下来我要介绍下基类中的工厂方法 ,因为我们是使用 xib 创建的 view，怎么工厂制作对应的实例给外界：

```
//----------------------------------------------------------MYCustomKeyboard----------------------------------------------------------

@implementation MYCustomKeyboard

+ (instancetype)customKeyBoardWithType:(KeyboardType)type{
    MYCustomKeyboard *view;
    switch (type) {
        case KeyboardTypeChinese:
            return [[MYCustomKeyboardChinese alloc] init];
            break;
        case KeyboardType123:
            return [[[NSBundle mainBundle] loadNibNamed:@"MYCustomKeyboard" owner:nil options:nil] firstObject];
            break;
        case KeyboardTypeABC:
            return [[[NSBundle mainBundle] loadNibNamed:@"MYCustomKeyboard" owner:nil options:nil] objectAtIndex:1];
            break;
        case KeyboardTypeEmoji:
            return [[[NSBundle mainBundle] loadNibNamed:@"MYCustomKeyboard" owner:nil options:nil] objectAtIndex:2];
            break;
        default:
            break;
    }
    return view;
}
```

没错，它就是这样子滴！我们注意到：这个 xib 的 owner 是 nil，因为我们根本就没有给 xib 指定 owner，注意到，xib 创建的自定义 cell 也是一样没有 owner，它是通过 cell 绑定的类和注册 identifier 的类找到对应的文件的。我目前只看到 xib 创建 controller 有个 view 的 owner 是对应的 controller，这个很好理解，因为 controller 有个 self.view 嘛，所以那个 view 是属于那个 controller 的。

(3) 接下来，我们举一个键盘的例子，看下 xib 和代码的怎么结合完成屏幕适配和事件处理，以自定义键盘 123 来说明：

```
//----------------------------------------MYCustomKeyboard123----------------------------------------

@interface MYCustomKeyboard123 : MYCustomKeyboard//数字键盘

@property (strong, nonatomic) IBOutletCollection(UIButton) NSArray *numberBtns;//数字按钮，如600、1、2等
@property (strong, nonatomic) IBOutletCollection(UIButton) NSArray *commandBtns;//命令按钮，如删除、清空、搜索等

@end
```

(1)头文件中，首先看到了之前两个 outlet 数组，一个是数字按钮，一个是命令按钮，很容易理解，就是对 xib 中的按钮进行了区分，分为输入型和命令型按钮。这个类头文件很简单，为啥，因为它是子类，也就是说除了这几个属性外 ,它还有基类的 delegate 属性和 type 属性 ,别忘了 ~

我们再看看.m 实现：

```
//----------------------------------------MYCustomKeyboard123----------------------------------------

@implementation MYCustomKeyboard123

- (void)awakeFromNib{
    [super awakeFromNib];
    self.type = KeyboardType123;

    [self cwn_makeShiPeis:^(UIView *maker) {
        maker.shiPeiSubViews();
    }];

    [self.numberBtns enumerateObjectsUsingBlock:^(UIButton *obj, NSUInteger idx, BOOL * _Nonnull stop) {
        obj.layer.borderWidth = 1.0 / [UIScreen mainScreen].scale * 0.5 ;
        obj.layer.borderColor = HexColor(0xaaaaaa).CGColor;
    }];
    [self.commandBtns enumerateObjectsUsingBlock:^(UIButton *obj, NSUInteger idx, BOOL * _Nonnull stop) {
        obj.layer.borderWidth = 1.0 / [UIScreen mainScreen].scale * 0.5 ;
        obj.layer.borderColor = HexColor(0xaaaaaa).CGColor;
    }];
}

#pragma mark 事件处理

- (IBAction)onClickNumberButtons:(UIButton *)sender {
    if(self.delegate && [self.delegate respondsToSelector:@selector(onClickKeyboardButtonTypeInputString:)]){
        [self.delegate onClickKeyboardButtonTypeInputString:sender];
    }
}

- (IBAction)onClickCommandButtons:(UIButton *)sender {
    if(self.delegate && [self.delegate respondsToSelector:@selector(onClickKeyboardButtonTypeCommand:)]){
        [self.delegate onClickKeyboardButtonTypeCommand:sender];
    }
}

@end
```

(2)我们看到了一个 awakeFromNib 文件 ,为啥不是 initWith…？别问我，我也不知道，xib 加载方式，loadWithNib 后就会执行 xib 中绑定的类的 awakeFromNib 方法，这时候可以做一些适配啊啥的。

下面具体看下实现的方法：

```
- (void)awakeFromNib{
    [super awakeFromNib];
    self.type = KeyboardType123;

    [self cwn_makeShiPeis:^(UIView *maker) {
        maker.shiPeiSubViews();
    }];

    [self.numberBtns enumerateObjectsUsingBlock:^(UIButton *obj, NSUInteger idx, BOOL * _Nonnull stop) {
        obj.layer.borderWidth = 1.0 / [UIScreen mainScreen].scale * 0.5 ;
        obj.layer.borderColor = HexColor(0xaaaaaa).CGColor;
    }];
    [self.commandBtns enumerateObjectsUsingBlock:^(UIButton *obj, NSUInteger idx, BOOL * _Nonnull stop) {
        obj.layer.borderWidth = 1.0 / [UIScreen mainScreen].scale * 0.5 ;
        obj.layer.borderColor = HexColor(0xaaaaaa).CGColor;
    }];
}
```

没错，就是这样子滴！我估摸着可能会引起你的兴趣的就是 frame 快速适配了，如果想知道详情的，可以参阅我的另一篇文章：

“链式编程原理与应用(Autolayout)”里面有详细介绍

(3)这里要再做一下说明，如果你打断点再 awakeFromNib,然后，外界通过基类工厂方法创建了 n 个键盘，你会发现单这个数字键盘的 awakeFromNib 就执行了 n 次，明明我就只用一个数字键盘啊？会不会造成内存浪费啊啥的，有这个想法很正常，作为第一次摸索的我也同样有这个疑惑。在此说明，以打消你们心中的疑虑：

```
- (void)setDatasource:(id<MYStockKeyboardViewDatasource>)datasource{
    _datasource = datasource;
    _numberOfKeyboards = [_datasource numberOfCustomKeyboards];

    if([_customKeyBoards count] == 0){
        _customKeyBoards = [NSMutableArray array];

        for (int i = 0; i < _numberOfKeyboards; i ++) {
            KeyboardType type = [_datasource customKeyboardAtIndex:i];
            if(type == KeyboardTypeUnKnow)
                return;

            MYCustomKeyboard *view = [MYCustomKeyboard customKeyBoardWithType:type];
            view.delegate = self;
            CGRect frame = view.frame;
            if(type == KeyboardType123){
                frame.origin.y = self.needToolBar == YES ? (44 - 1.0 / [UIScreen mainScreen].scale) : 0;
            }else{
                frame.origin.y = self.needToolBar == YES ? 44 : 0;
            }
                view.frame = frame;
                [self addSubview:view];
                view.hidden = i == 0 ? NO : YES;
               [self.customKeyBoards addObject:view];
        }
    }
}
```

以上就是我外部使用到的创建键盘的方法

假如 i=4 ,那么会执行四次工厂方法 ,传入四种键盘的 type ,对嘛？没错，注意，工厂方法一共执行了四次也就是说有四次 loadWithNib 调用，每次调用都会将 xib 中的所有键盘都 load 以下，也就是都执行下 awakeFromNib ,只是我们最终只返回了 type 对应的 xib 中的 view 而已，返回后因为被强引用了，所以我们拿到了一个键盘，然后其他剩余创建的键盘因为没有被强引用，所以在工厂方法执行结束后，内存又被回收了，对嘛？没错，它就是这样滴！