自动布局

`A.视图`

1 基本概念

当父视图的位置发生改变时，子视图随着移动；当父视图的大小发生变化时，如果你要子视图也调整大小以及重新定位，这被称为 **布局** 。

布局有三种主要方式

手动布局（Manual layout）
自动缩放（Autoresizing）

自动布局（Autolayout）

---

2 手动布局

手动布局 的道理很简单，每当一个视图的大小发生变化时，其 **- layoutSubviews 方法** 会被调用。你可以在其中根据自己的意愿随意调整其子视图的位置和大小。

---

3 自动缩放（Auto resizing）

要使用Autoresizing，记得 **关闭Autolayout** 。

Autoresizing是iOS 6之前提倡的布局方式。父视图的开关 **属性 autoresizesSubviews**，决定了该视图的子视图是否开启自动缩放功能（默认为YES）。如果此属性值为YES，那么子视图的 **autoresizingMask 属性** 的值决定子视图的重布局规则。

注意 自动缩放是在 layoutSubviews 被调用之前执行的，即 layoutSubviews 方法仍然会被调用。

自动缩放 是概念上分配子视图 **springs** 和 **struts** 的问题。

spring（直译为**弹簧**）寓意为可以拉伸；strut（直译为**支柱**）寓意为不可拉伸。

根据直观的形象，在IB中，“工”字型的横线表示strut；“<-->”型的横线表示spring。

spring 和 strut 可以指定在视图的“内部”，也可以在“外部”。

在代码中，通过视图的 **autoresizingMask 属性** 设置springs和struts组合。它是一个位掩码。

以 **UIViewAutoresizingFlexible...** 名称开头的选项代表spring，没有定义默认是strut。

默认值是 **UIViewAutoresizingNone**，意味着该视图不调整。

Autoresizing的“缺陷”在于：它只能指明的父子视图间的自动排布规则。

---

4 自动布局（Auto layout）

是用自动布局后，就 **不要** 再在代码中直接修改视图的 frame、bounds 和 center 。

自动布局（Auto Layout）是一种约束满足系统。所谓 **约束**，就是一系列描述iOS程序视图布局的规则。它们限定了视图之间的关系，也限定了视图的布局形式。

开发者可以在IB中以可视化的形式来指定约束，也可以在源代码中以编程的形式指定约束。

无论是在IB中，还是在代码中，约束都围绕着一个 核心公式

obj1.property1 *关系* (obj2.property2 * multiplier) + constant

---

obj1 和 obj2 是两个视图对象，与AutoResizing不同的是，它们两个 **不要求** 必须是父子视图。

---

property1 和 property2 指的是视图的属性。属性概念如下：

- left、right、top、bottom
  视图对齐矩形的左右上下边界。它们分别用于视图的最小x值、最大x值、最小y值和最大y值。
- leading、trailing
  视图对齐矩形的前边沿及后边沿。
- width、height
  视图对齐矩形的宽度及高度
- centerX、centerY

视图对齐矩形的中心点在x轴和y轴的坐标。

- baseline
  对齐矩形的基线，通常比bottom属性值小一些，而且两者之间的偏移量通常是某个定值

关系指的是两个视图对象的两个属性间的 相等/不等 的关系。

- LessThanOrEqual （ <= ）
- Equal （ = ）
- GreaterThanOrEqual （ >= ）

---

multiplier 和 **constant** 都是具体的数值。

另外，每个约束都有一个 **优先级** 的概念，它也是一个具体的数值。

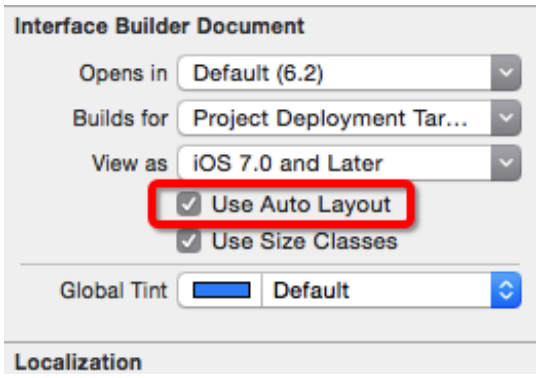
当两个约束发生冲突的时候，即，系统进行布局时，无法同时满足两个约束，此时，布局系统总是先设法满足优先级高的约束规则。

优先级的取值范围是从 0 到 1000。

- Required（默认，1000）
- DefaultHigh（750）
- DefaultLow（250）
- FittingSizeLevel（50）

4.1 在IB中使用autolayout

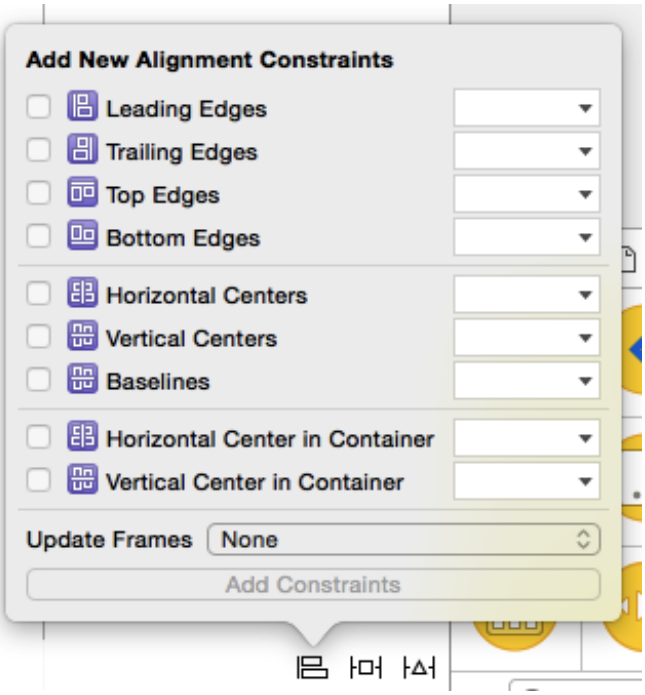
在IB中使用autolayout，首先要确保开启了autolayout布局功能



反之，则意味着关闭autolayout（即使用autoresizing布局）

约束分为两大类：关于**对齐**（Align）的，和关于**布局**（Pin）的。

4.2 对齐（Align）约束



在界面中，对齐约束上中下分为：边缘对齐、中心对齐和容器中心对齐。

由于各国文化不同，Xcode中不说“左右”，而按照写字方向来说**Leading**和**Training**。对应于我国（绝大多数国家都是如此），Leading为左，Training为右。

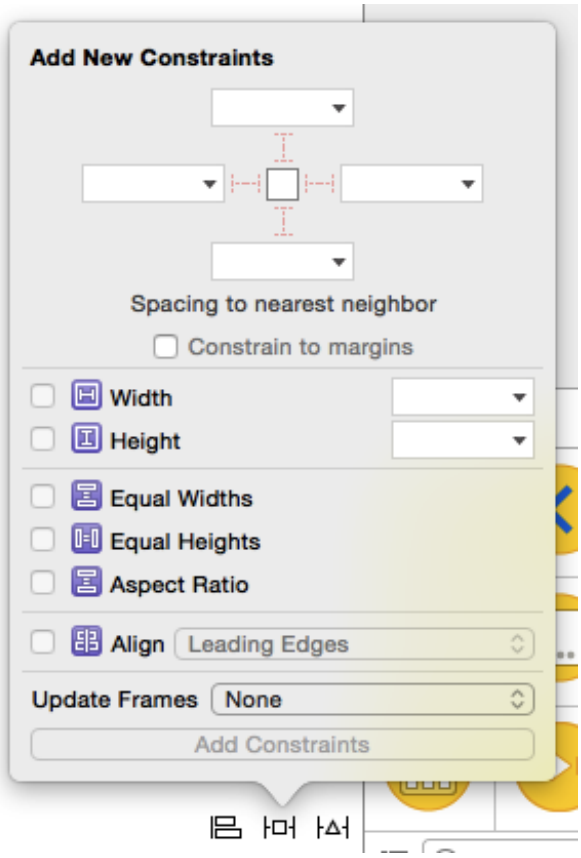
对于**边缘对齐**来说，又分为：**左边缘对齐**、**右边缘对齐**、**顶端对齐** 和 **底部对齐**。这些选线用

于让选中多个元素的边缘进行对齐，一般以最后选中的元素为准。

对于**中心对齐**来说，又分为 **水平中心对齐**、**垂直中心对齐** 和 **基准线对齐**。这些选项用于让选中的多个元素的中心进行对齐，一般以最后选中的元素为准。基准线对齐会让元素按最后选中元素的基准线对齐。

容器中心对齐相对简单，只分为 **容器水平居中** 和 **容器垂直居中**。这些选项会让选中的元素在父视图中居中。

4.3 布局（Pin）约束



布局约束主要用于设定元素的具体位置，包括边缘位置、尺寸以及横纵比等。

边缘位置 指的是该元素的边缘与另外某个元素边缘的距离。它还有一个选项：Contrain to margins，这是iOS 8之后提供的特性。

margin（边缘）指的是屏幕周围一个特殊的空白区域，一般为16px。勾选这个选项后，距离

屏幕边缘的距离会相应地减去这个margin的值。

尺寸包括Width和Height，指的是该元素的尺寸大小，可以设定为具体数值。此外，还可以对多个元素设定等高/等宽，这是用 Equal Widths 和 Equal Heights 选项来设定。

横纵比指的是该元素的宽高比。Aspect Ration（宽高比）选项勾选后，会然让元素无论怎么变化，都始终保持宽高比例。

4.4 其他

选中相应对象后，在“尺寸查看器”中会显示约束，我们可以查看和编辑这些约束的具体行为。

如果出现错误的约束，会以特殊颜色显示：黄色线中显示的是意思不明确的约束；红色线显示的是有冲突的约束。

点击界面设计器下方的“Resolve Auto Lyout Issues 按钮”，可对视图和约束进行修正

操作	含义
Update Frames	更新所选项目，是指匹配属性查看器i的约束
Update Constraints	改变属性查看器中的值，是指匹配所选的项目
Add Missing Constraints	给选择的项目添加必要的约束
Reset to Suggested Constraints	把约束重置为推荐的约束值
Clear Constraints	删除选定的项目的布局约束，然后定义新的约束

---

5 屏幕分类

Xcode 6 引入了 **Size Class**（屏幕分类）。

屏幕分类对UI进行了一次全新的抽象：将各种设备的屏幕尺寸及其旋转状态全部都抽象成以下

几种级别
Width（宽）：Regular（正常）、Any（任意）、Compact（紧凑）
Height（高）：Regular（正常）、Any（任意）、Compact（紧凑）

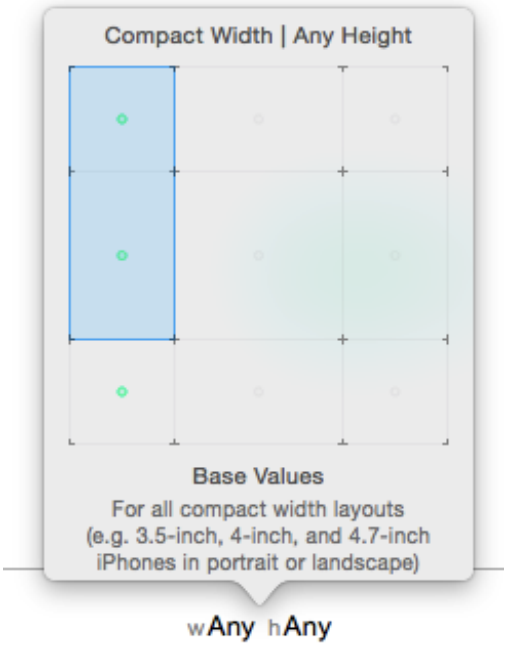
理论上这几种类别任意组合，可以合成9种不同类型。不过实际上，设备的尺寸只能组合出4种类型，也就是正常和紧凑的相互组合。

任意，意味着无视这个分类条件。

Size Classes 与 设备 的对应列表

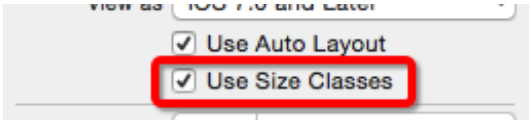
设备名称	对应分类
iPhone 4S	Compact Width | Regular Height
iPhone 5/5S	Compact Width | Regular Height
iPhone 6	Compact Width | Regular Height
iPhone 6 Plus	Compact Width | Regular Height
iPhone 4S（横屏）	Compact Width | Compact Height
iPhone 5/5S（横屏）	Compact Width | Compact Height
iPhone 6（横屏）	Compact Width | Compact Height
iPhone 6 Plus（横屏）	Regular Width | Compact Height
各种iPad	Regular Width | Regular Height

其实，你并不需要记住这张表，因为在Xcode中会有直观明确的提示信息。



启用 Size Classes 之后，Xcode会自动给Storyboard另外创建子文件，分别对应各种类别的选择。这样做的好处是，约束、控件对于 Size Class 来说是相互独立的，不同 Size Class 之间的约束、空间不会互相更影响（Any除外）。

在文件检查器里面选中“Use Size Classes”选项可开启该功能。



6 在代码中使用autolayout

6.1 创建约束规则

视图的autoresizingMask会自动转换成约束形式并发挥作用，如果你不希望如此，则你必须显示地关闭该行为，通过各个子视图的
- setTranslatesAutoresizingMaskIntoConstraints: 方法 置为NO进行关闭。不设置，默认为YES。

为某个添加/删除约束使用如下方法：

- addConstraint:，addConstraints:
- removeConstraint:，removeConstraints:

在代码中，一个约束条件，对应着一个 **NSLayoutConstraint** 对象，制定一个约束规则，就是实例化 NSLayoutConstraint 对象的过程。

通过 NSLayoutConstraint 类的
+ constraintWithItem: attribute: relatedBy: toItem: attribute: multiplier: constant: 类方法（so long ... ） 来实例化一个 NSLayoutConstraint 对象，用以描述一个约束规则。

注意 记得使用视图对象的 addConstraint: 方法将约束添加到指定视图上。

补充 视图对象的 **constraints** 属性（数组）中存放的都是该视图下的约束。

约束绝大多数情况下，总是涉及到两个视图对象，那么一个约束对象具体添加到哪个视图对象中呢？简而言之：**找祖先** 。

如果，涉及到的两个视图，有直接的父子视图关系，那么约束添加至父视图上。

如果，涉及的两个视图，没有直接的父子关系，那么他们总是有一个或多个公共的 **祖先** 的，那么将约束，添加到离它们最近的公共祖先上。

如果某个约束描述的是绝对高度或宽度，那么item2就是nil，attribute2就是NSLayoutAttributeNotAnAttribute。

从一套约束规则切换到另一套的时候，可以用UIView的动画功能来表现这个过程。在动画块中，在改变约束规则后调用视图对象的 **layoutIfNeeded**，就可以看到展示效果。

6.2 在代码中编辑XIB中的约束

XIB（storyboard）文件中的约束线条和组件一样也是对象，所以也可以拉线到代码中，对应一个约束对象。

对于这种系统“帮”我们创建的约束对象，我们可以进行修改，但是能改的地方只有它的 **优先级** 和 **常量** 。

---

7. VFL

使用代码创建显得十分啰嗦，Apple公司提供了一种简洁的方案：**【可视化格式】**（visual format）。它是基于文本的一种缩写，类似于： @"V:|[v2(10)]|"

7.1 语法元素

在水平方向或垂直方向上排布

H:

V:

视图

[item]

上级视图

|

关系

==

≤

>=

指标

metric

紧贴对齐

[item][item]

弹性空间

[item]-(>=0)-[item]

固定空间

[item]-[item]

自定义的固定空间

[item]-gap-[item]

固定宽度或固定高度

[item(size)]

[item(==size)]

最大或最小宽度/高度

[item(>=size)]

[item(≤size)]

令视图的宽度/高度与另外一个视图相匹配

[item(==item)]

[item(>=item)]

[item(≤item)]

紧贴上级视图的边界

|[item]

[item]|

本视图与上级视图边界间留出默认空白

|-[item]

[item]-|

本视图与上级视图边界间留出一定数量的空白

|-gap-[item]

[item]-gap-|

优先级（0到1000）

@value

7.2 示例

`V:[view1]-15-[view2]`

将view2放在其顶部举例view1底部15p的位置

`[view1]`

视图绑定字典将方括号所包围的名称同一个视图实例相匹配

`H:|[view1]|`

将view1的宽度尺寸同父视图的一致

`H:[view1]-(>=20)-[view2]`

使view2的前沿举例view1后沿至少20p

`H:[view1](<=someWidth)`

someWidth必须在传递的字典中映射为NSNumber值

`H:[view1][view2]`

是view1的后沿与view2的前沿齐平

`[view1]-(>=0)-[view2]`

两视图分离，至少0p

`V:[view1]-20-[view2]`

使view1的底部距离view2的顶部20p

`[view1]-[view2]`

view1和view2距离8p（默认值）

`[view(50)]`

使view沿某个轴的范围为50p

`[view(>=50)]`

使view在某个轴上的最小范围为50p

`[view1(==view2)]`

使view1在某个轴上的尺寸和view2一致

`V:|[view1]`

使view1的顶部和父视图的顶部齐平

`|-[view1]|`

在某个轴上，在父视图和view1之间间隔20p

`H:|-15-[view1]`

view1水平方向距离父视图的前沿15p

`H:[view1](<=50@100)`

使view1在水平轴上最大尺寸为50p，优先级为100

7.3 在代码中使用VFL

通过 NSLayoutConstraint 类的
+ constraintsWithVisualFormat:options:metrics:views: 类方法，根据参数VFL字符串，会生成对应约束对象的数组。

随后可以调用 视图对象的 - addConstraints: 方法，添加整个约束数组。

8 第三方框架 Masonry

由于使用官方的方法，在代码中进行布局过于繁琐，所以，常使用第三方框架在代码中进行布局。

Masonry 是 GitHub 上使用最多的第三方布局框架。

Masonry 对官方的方法进行了封装，进一步简化了布局代码。