AutoLayout - by Dorayo

AutoLayout是什么?

> AutoLayout是一种基于约束的，描述性的布局系统。`Auto Layout Is a Constraint-Based, Descriptive Layout System.`

Keyword:

- 基于约束
    - AutoLayout之前是通过Frame来进行绝对地定位及确定大小
    - AutoLayout是通过约束来相对地定位和确定大小
    - 比如说，A视图距离其父视图B的上、左、下、右的距离这四个约束都确定了，实际上A视图的定位及大小也就确定了
- 描述性
    - 约束的定义或视图之间的相对位置关系的描述用接近自然语言或可视化语言的方法来描述
- 布局系统
    - AutoLayout这项技术主要是用来解决布局问题的

> AutoLayout说白了也一样是去计算一个view的Frame，只不过Frame的计算从之前的直接设置，变成现在的相对地、间接地计算，也就是说AutoLayout本质上就是一种相对布局。

如何理解相对布局，如何理解基于约束的描述性布局？举个例子： 键盘上 `Q` 键的长、宽均为 1 厘米，左边距离键盘的左边缘 10 厘米，上边距离键盘的顶部 5 厘米。这句描述就可以定位 `Q` 键在键盘中的位置，很轻松就可以计算出 `Q` 键的 frame 为 {{10.0, 5.0}, {1.0, 1.0}}。

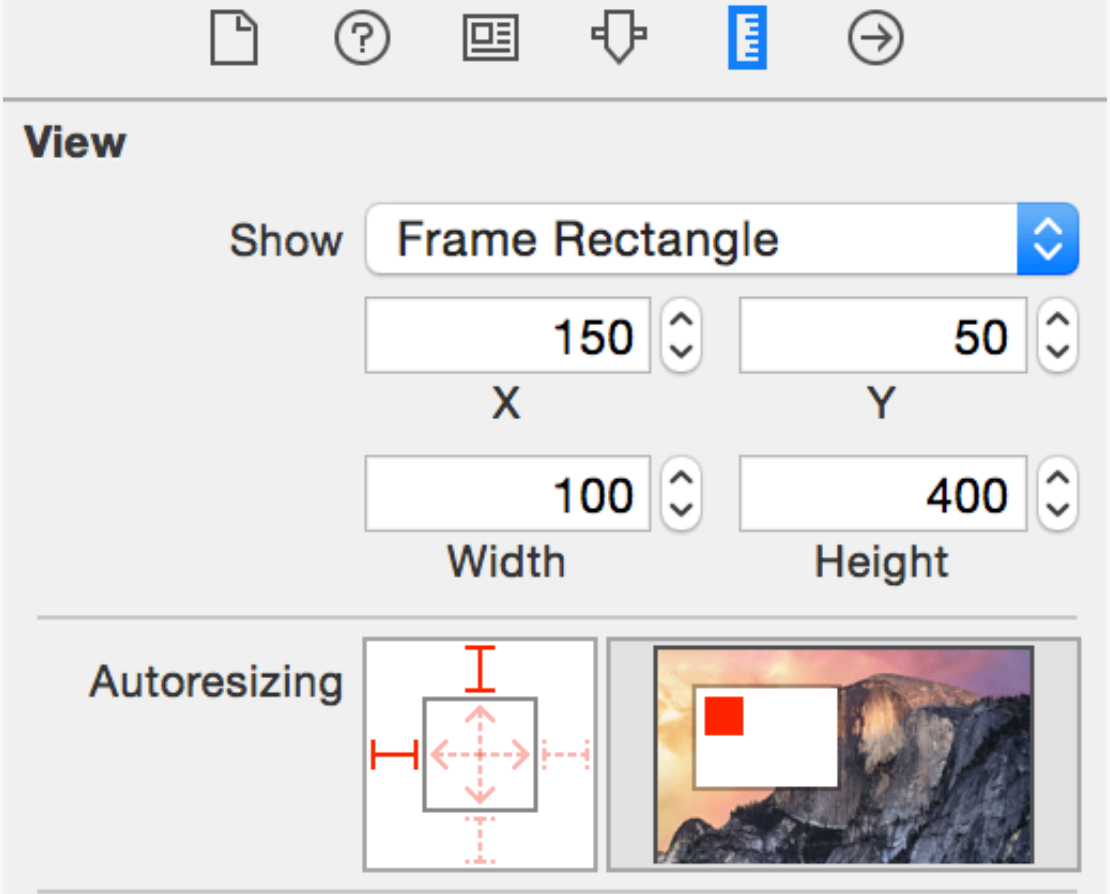
现在 `Q` 键的坐标已经确定，那么 `W` 键的坐标可以这样描述：顶部和 `Q` 键对齐，大小和 `Q` 键相等，位于 `Q` 键右侧 0.5 厘米处。仔细想想，这句描述中包含了元素间的关系，关系间的约束，因此，也一样可以得出 `W` 键的 frame。

AutoLayout 和 AutoResizingMask的关系

`AutoResizingMask`

- 代码方式：UIView 的 autoresizingMask属性

- IB方式：



- 不足：
  - 只能设置父子视图之间的大小对应变化情况，不能是任意两视图
  - 关系只能是相等的约束，不能指定非相等约束(大于或者小于等于)
  - 约束功能单一，不灵活

`AutoLayout`

- AutoLayout可以解决所有AutoresizingMask的不足
- AutoLayout还提供约束的优先级和可视化约束语言
- 通过AutoLayout可以实现类似如下的需求：
  - 一个视图尺寸与另一个视图尺寸匹配，使两个视图保持相同的宽度
  - 无论父视图的形状如何改变，都将一个视图相对于父视图居中
  - 摆放一行视图时，将几个视图底部对齐
  - 显示按钮时文本四周不要有太多的补白

> 在使用 Auto Layout 时，首先需要将视图的

> `translatesAutoresizingMaskIntoConstraints` 属性设置为 NO。这个属性默认为 YES。当它为 YES 时，运行时系统会自动将 Autoresizing Mask 转换为 Auto Layout 的约束，这些约束很有可能会和我们自己添加的产生冲突。

Auto Layout 基础知识

约束

Auto Layout 中约束对应的类为 `NSLayoutConstraint`，一个 `NSLayoutConstraint` 实例代表一条约束。

`NSLayoutConstraint` 有两个方法，

第一个方法是：

```
+ (id)constraintWithItem:(id)view1
               attribute:(NSLayoutAttribute)attribute1
               relatedBy:(NSLayoutRelation)relation
                  toItem:(id)view2
               attribute:(NSLayoutAttribute)attribute2
              multiplier:(CGFloat)multiplier
                constant:(CGFloat)constant;
```

不要被这个方法的参数吓到，实际上它只做一件事，就是让 view1 的某个 attribute 等于 view2 的某个 attribute 的 multiplier 倍加上 constant，

这里的 attribute可以是上下左右宽高等等。

精简后就是下面这个公式：

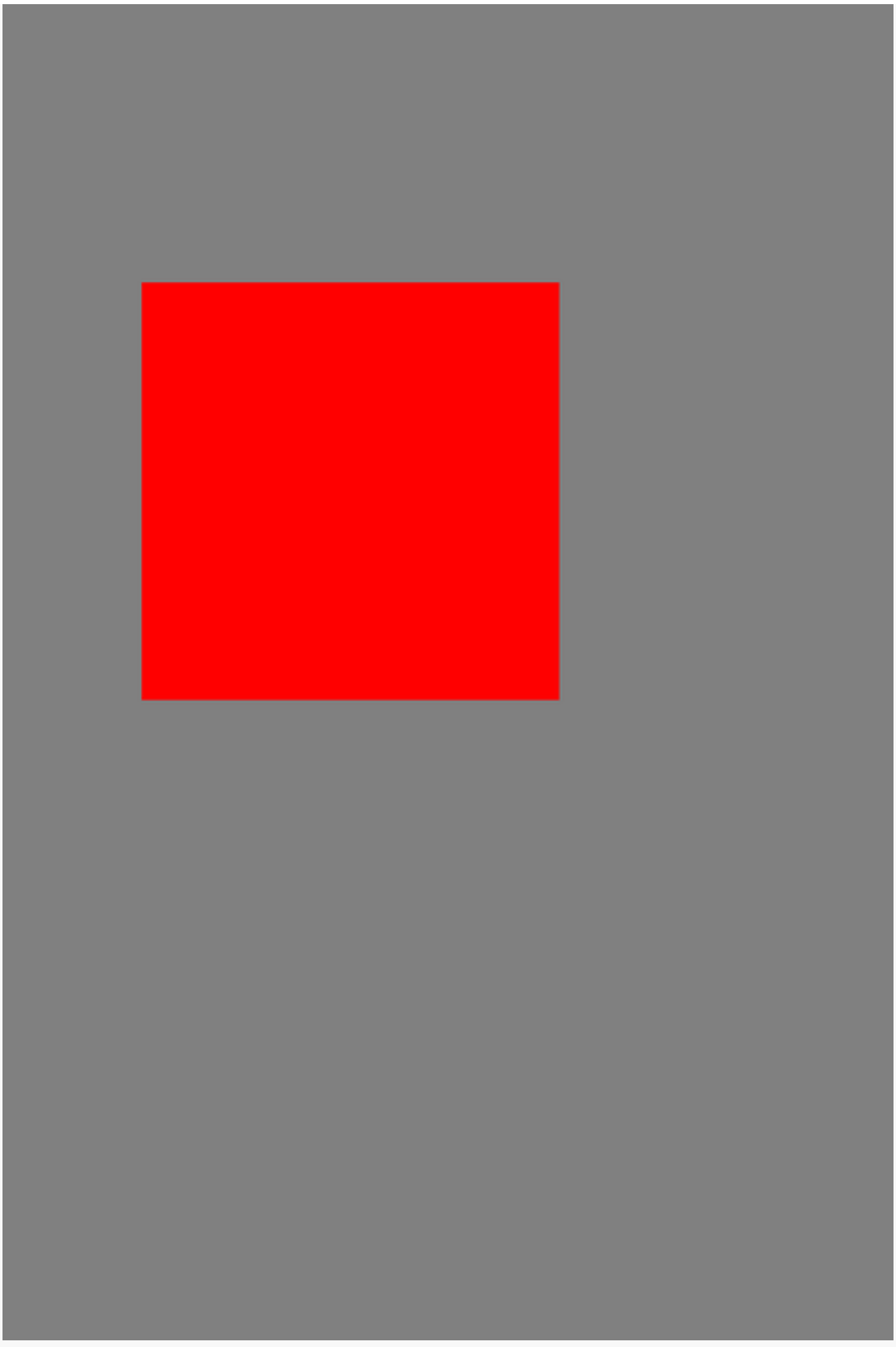
`view1.attribute1 = view2.attribute2 × multiplier + constant`

还有一个参数是 relation，这是一个关系参数，它标明了上面这个公式两边的关系，它可以是小于等于 (≤)，等于 (=)或大于等于 (≥)。上面的公式假定了这个参数传入的是 `=`，根据参数的不同，公式中的关系符号也不同。

需要注意的是，≤ 或 ≥ 优先会使用 = 关系，如果 = 不能满足，才会使用 < 或 >。例如设置一个 ≥ 100 的关系，默认会是 100，当视图被拉伸时，100 无法被满足，尺寸才会变得更大。

举个例子：

1、我们要实现一个如下图的布局。



布局代码如下：

```
UIView *view = [[UIView alloc] init];
```

```
[view setBackgroundColor:[UIColor redColor]];
[self.view addSubview:view];

CGRect viewFrame = CGRectMake(50.f, 100.f, 150.f, 150.f);

// 使用 Auto Layout 布局
[view setTranslatesAutoresizingMaskIntoConstraints:NO];

// `view` 的左边距离 `self.view` 的左边 50 点.
NSLayoutConstraint *viewLeft = [NSLayoutConstraint constraintWithItem:vie
w
                                                             attribute:NSL
ayoutAttributeLeading
                                                             relatedBy:NSL
ayoutRelationEqual
                                                                toItem:sel
f.view
                                                             attribute:NSL
ayoutAttributeLeading
                                                            multiplier:1
                                                              constant:CGR
ectGetMinX(viewFrame)];
// `view` 的顶部距离 `self.view` 的顶部 100 点.
NSLayoutConstraint *viewTop = [NSLayoutConstraint constraintWithItem:view
                                                             attribute:NSLa
youtAttributeTop
                                                             relatedBy:NSLa
youtRelationEqual
                                                                toItem:self
.view
                                                             attribute:NSLa
youtAttributeTop
                                                            multiplier:1
                                                              constant:CGRe
ctGetMinY(viewFrame)];
// `view` 的宽度 是 60 点.
NSLayoutConstraint *viewWidth = [NSLayoutConstraint constraintWithItem:vi
ew
                                                              attribute:NS
LayoutAttributeWidth
                                                              relatedBy:NS
LayoutRelationGreaterThanOrEqual
                                                                 toItem:ni
l
                                                              attribute:NS
```

```
LayoutAttributeNotAnAttribute
                                                                  multiplier:1
                                                                    constant:CG
RectGetWidth(viewFrame)];
// `view` 的高度是 60 点.
NSLayoutConstraint *viewHeight = [NSLayoutConstraint constraintWithItem:v
iew
                                                                   attribute:N
SLayoutAttributeHeight
                                                                   relatedBy:N
SLayoutRelationGreaterThanOrEqual
                                                                      toItem:n
il
                                                                   attribute:N
SLayoutAttributeNotAnAttribute
                                                                  multiplier:1
                                                                    constant:C
GRectGetHeight(viewFrame)];
// 把约束添加到父视图上.
[self.view addConstraints:@[viewLeft, viewTop, viewWidth, viewHeight]];
```

WTF，如此简单的布局，竟要写这么多的代码，这简直是个灾难！于是 UIKit 团队发明了另外一种更简便的表达方式进行布局，这个我们后面再讲，现在先看看这段代码的解析。

1. 首先我把 view 的 translatesAutoresizingMaskIntoConstraints 设为了 NO，禁止将 Autoresizing Mask 转换为约束。
2. 然后在设置 viewLeft 这个约束时，attribute 参数使用了 NSLayoutAttributeLeading 而不是 NSLayoutAttributeLeft，这两个参数值都表示左边，但它们之间的区别在于 NSLayoutAttributeLeft 永远表示左边，但 NSLayoutAttributeLeading 是根据习惯区分的，例如在某些文字从右向左阅读的地区，例如阿拉伯，NSLayoutAttributeLeading 表示右边。换句话说，NSLayoutAttributeLeading 是表示文字开始的方向。在英文、中文这种从左往右阅读的文字中它表示左边，在像阿拉伯语、希伯来语这种从右往左阅读的文字中它表示右边。通常情况下，除非你明确要限制在左边，否则你都应该使用 NSLayoutAttributeLeading 表示左边。相对的，表示右边也类似这样。这对于我们的本地化工作有很大的帮助。
3. 然后在设置 viewWidth 和 viewHeight 这两个约束时，relatedBy 参数使用的是 NSLayoutRelationGreaterThanOrEqual 而不是 NSLayoutRelationEqual。

   因为 Auto Layout 是相对布局，所以通常你不应该直接设置宽度和高度这种固定不变的值，除非你很确定视图的宽度或高度需要保持不变。

   如果一定要设置高度或宽度，特别是宽度，在没有显式地设置内容压缩优先级（Content Hugging Priority，后面会讲到）和内容抗压缩优先级（Content Compression Resistance Priority，后面会讲到）的情况下，尽量不要使用 NSLayoutRelationEqual 这种绝对的关

系，这会带来许多潜在的问题：

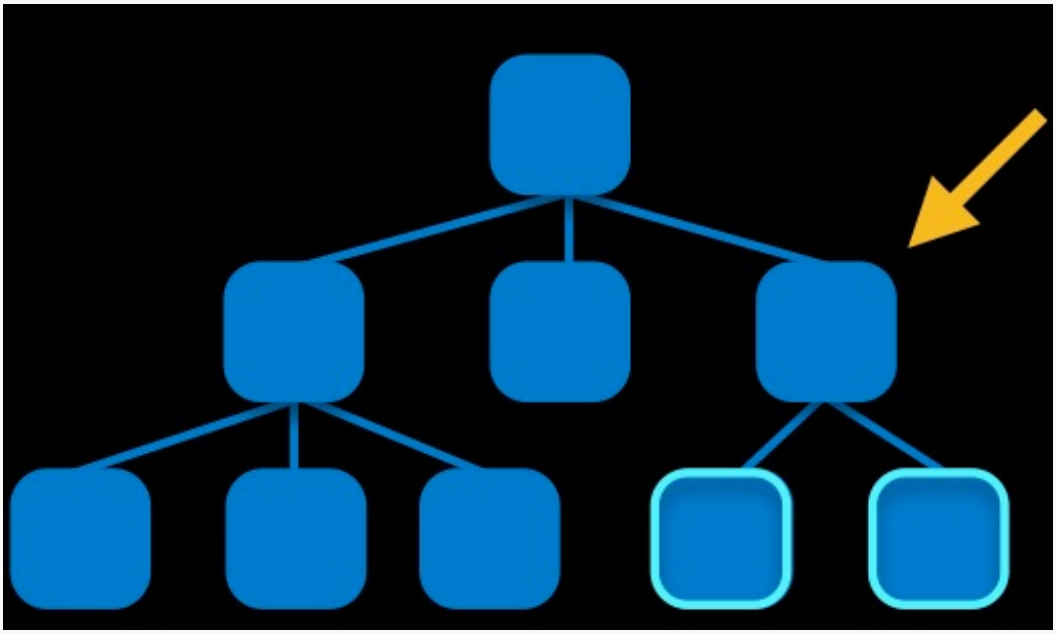
- 根据内容决定宽度的视图，当内容改变时，外观尺寸无法做出正确的改变
- 在本地化时过长的文字无法显示，造成文字切断，或文字过短，宽度显得过宽，影响美观
- 添加了多余的约束时，约束之间冲突，无法显示正确的布局

所带来的问题不仅仅局限与这几条，这里只是简单列出几条。

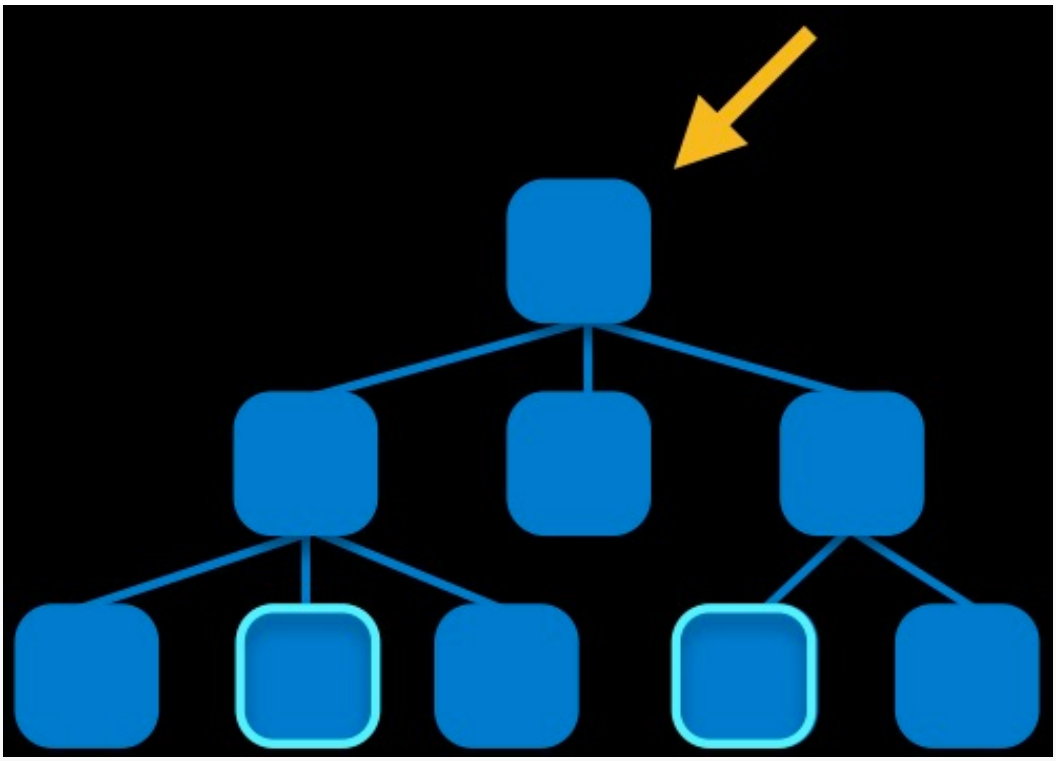
如何正确的设置宽度或高度？给出一些 Tips：

- 如果宽度和高度布局可以改变，使用固有内容尺寸（Intrinsic Content Size，后面会讲到）设置约束（即 size to fit size）
- 如果宽度和高度布局不可以改变，设置约束的关系为 ≥
- 调整压缩优先级和内容抗压缩优先级

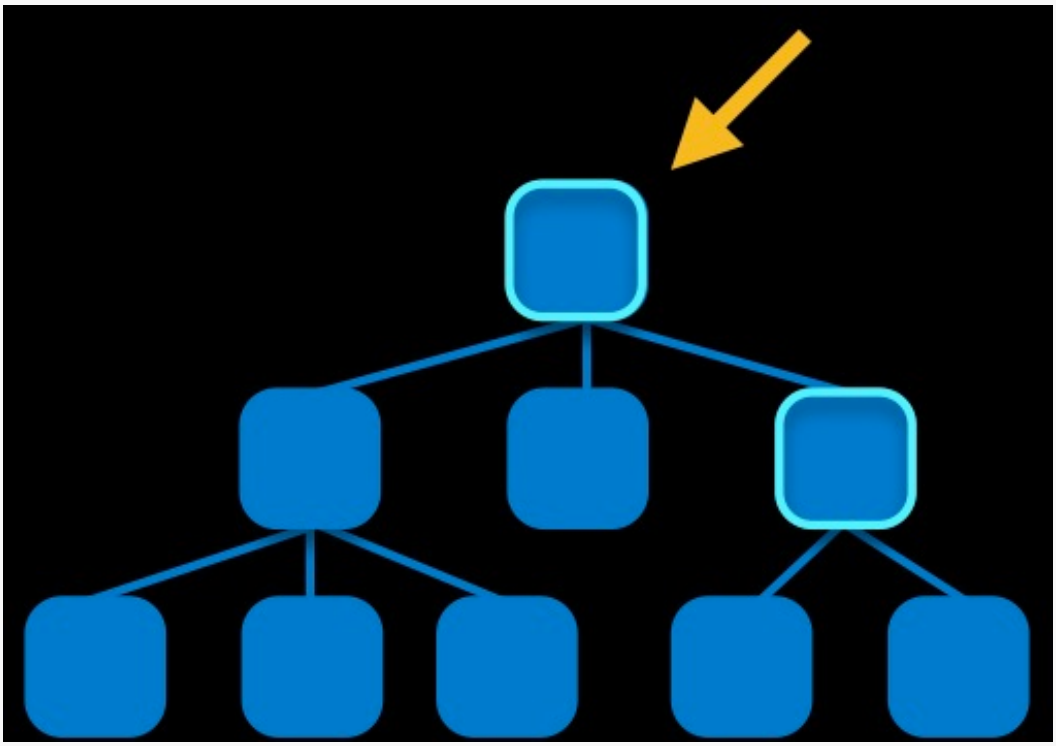
4. 最后我把所有约束都添加到了 view 的父视图 self.view 上。view 的约束为什么不添加到自身而添加到别的视图上去呢？这是由于约束是根据视图层级自下而上更新的，也就是从子视图到父视图。所以 Auto Layout 添加约束有一套自己的规则，如下：
    - 两个同层级间视图的约束，添加到它们共同的父视图上



    - 两个不同层级间视图的约束，添加到它们最近的共同的父视图上



- 两个有层级关系的视图的约束，添加到层次较高的视图上（父视图）上



因为我们属于最后一种情况，所以子视图 view 的约束添加到了父视图 self.view 上。

第二个方法是：

```
+ (NSArray *)constraintsWithVisualFormat:(NSString *)format
                                 options:(NSLayoutFormatOptions)opts
                                 metrics:(NSDictionary *)metrics
                                   views:(NSDictionary *)views;
```

这个方法是我们实际编程中最常用的方法。它会根据我们指定的参数返回一组约束。 这个方法很重要，所以接下来会详细解释每个参数的用途。

`<<format>>`

这个参数存放的是布局逻辑，布局逻辑是使用 可视化格式语言 (`VFL`) 编写的。实际编程中我们也是使用 `VFL` 编写布局逻辑，因为第一个方法明显参数过多，一个简单的布局要写很多代码。

上一个布局使用 `VFL` 来重构的话，代码如下：

```
[view setTranslatesAutoresizingMaskIntoConstraints:NO];

NSDictionary *views = NSDictionaryOfVariableBindings(self.view, view);

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:|-50-[view(>=150)]" options:0 metrics:nil views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-100-[view(>=150)]" options:0 metrics:nil views:views]];
```

解释一下：

1. 首先我们使用 NSDictionaryOfVariableBindings(...) 宏创建了一个字典 views，这个宏会自动把传入的对象的键路径作为字典的键，把对象作为字典的值。所以 `views` 字典的内容就像这样：`{@"self.view": self.view, @"view", view}`
2. 接下来两句代码涉及到VFL

   `H:|-50-[view(>=150)]`

   `V:|-100-[view(>=150)]`

第一句是在水平方向布局，表示 `view` 左边距离父视图左边 50 点，宽度至少 150 点。（水平方向是宽度）

第二句是在垂直方向上布局，表示 `view` 顶部距离父视图顶部 100 点，宽度至少 150 点。（垂直方向是高度）

分解说明如下：

- `H` / `V` 表示布局方向。`H` 表示水平方向（Horizontal），`V` 表示垂直方向（Vertical），方

向后要紧跟一个 `:`，不能有空格

- `|` 表示父视图。通常出现在语句的首尾
- `-` 有两个用途，单独一个表示标准距离。这个值通常是 8； 两个中间夹着数值，表示使用中间的数值代替标准距离，如第一句的 `-50-`，就是使用 50 来代替标准距离
- `[]` 表示对象，括号中间需要填上对象名，对象名必须是我们传入的 views 字典中的键。对象名后可以跟小括号 `()`，小括号中是对此对象的尺寸和优先级约束。水平布局中尺寸是宽度，垂直布局中尺寸是高度。如第一句中的 `(>=150)` 就是对 view 尺寸的约束，因为是水平方向布局，所以它表示宽度大于或等于 150 点。而 150 前面的 `>=` 就是我们上面第一个方法中提到的关系参数。至于为什么这里使用 `>=`，上面已经解释过了。括号中可以包含多条约束，如果我们想再加一条约束，保证 view 的宽度最大不超过 200 点，我们可以这样写：`H:|-50-[view(>=150,<=200)]`。还可以添加优先级约束，这个我们后面再讲

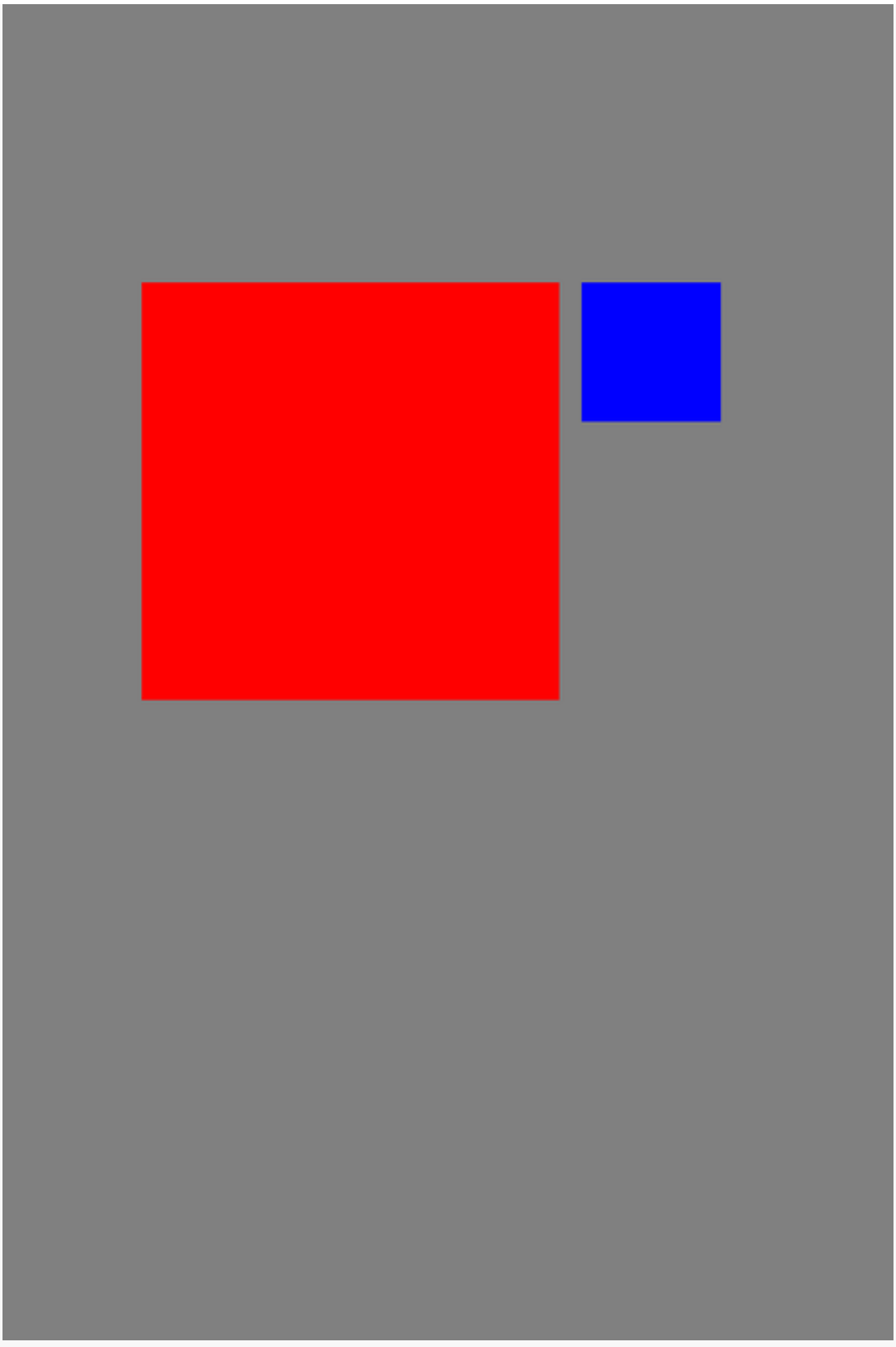
`VFL` 语法有几点需要注意：

- 布局语句中不能包含空格
- 和关系一样，没有 `>`、`<` 这种约束

再看两个例子:

例一：

我们在 `view` 右侧添加另一个视图 `view2`，效果如图：



代码如下：

```
UIView *view = [[UIView alloc] init];
[view setBackgroundColor:[UIColor redColor]];
[self.view addSubview:view];

UIView *view2 = [[UIView alloc] init];
[view2 setBackgroundColor:[UIColor blueColor]];
[self.view addSubview:view2];

[view setTranslatesAutoresizingMaskIntoConstraints:NO];
[view2 setTranslatesAutoresizingMaskIntoConstraints:NO];

NSDictionary *views = NSDictionaryOfVariableBindings(self.view, view, vie
w2);

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:|-50-[view(>=150)]" options:0 metrics:nil views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-100-[view(>=150)]" options:0 metrics:nil views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:[view]-[view2(>=50)]" options:0 metrics:nil views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-100-[view2(>=50)]" options:0 metrics:nil views:views]];
```

解释代码中出现的一条新的`VFL`语句：`H:[view]-[view2(>=50)]`

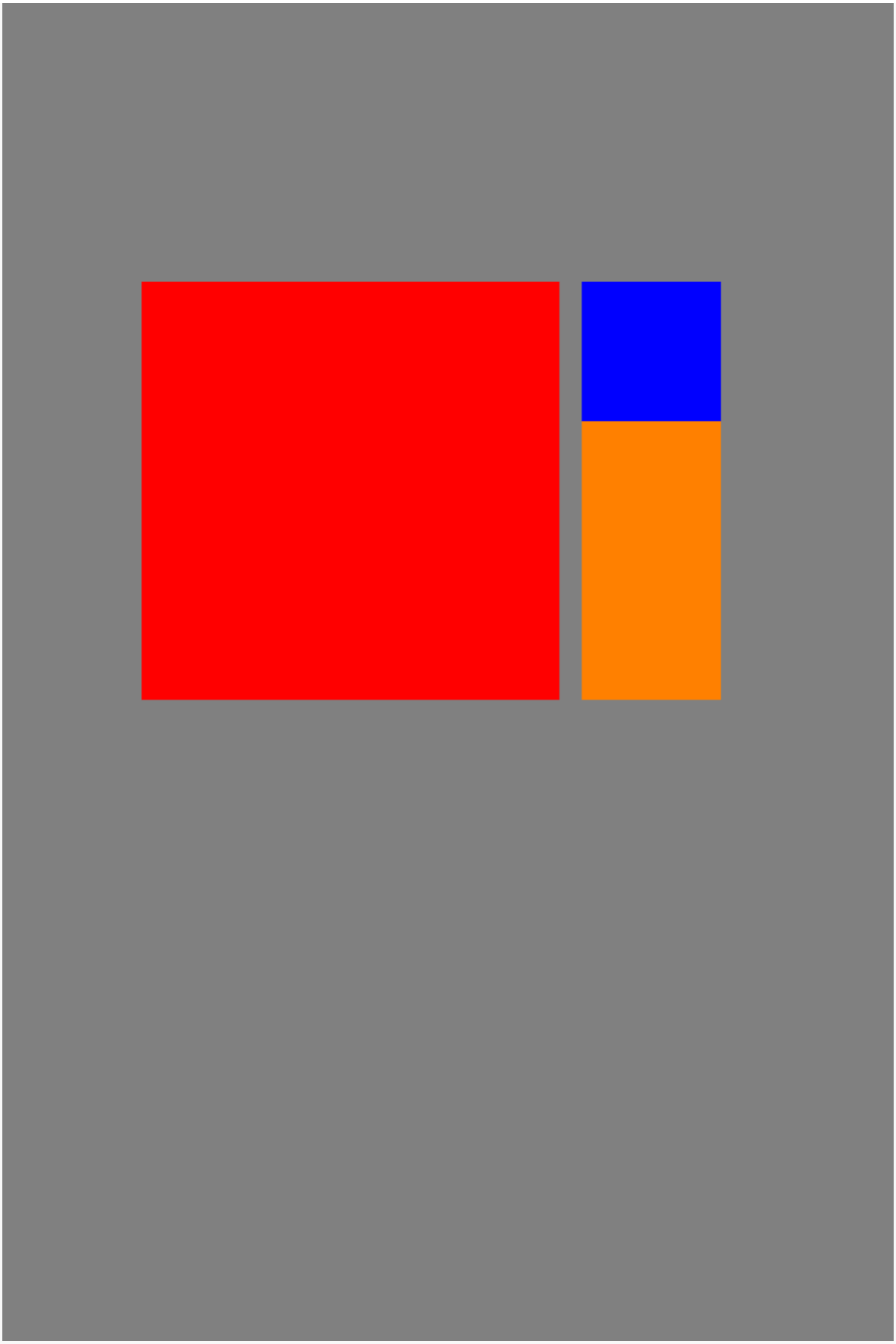
1. 从开始的 `H:` 我们可以判断出这是水平方向的布局，换句话说就是设置视图的 `x` 和 `width`
2. 接着的 `[view]`，说明后一个视图是在 `view` 的右侧；接着是 `-`，说明后一个视图和 `view` 之间有一个标准距离的间距；也就是说 `x` 等于 `view` 的右侧再加上标准距离，即 `CGRectGetMaxX(view) + 标准距离`。最后是 `[view2(>=50)]`，这里可以看出后一个视图是 `view2`，并且它的宽度不小于 50 点。整一句翻译成白话就是说：在水平方向上，`view2` 在 `view` 右侧的标准距离位置处，并且它的宽度不小于 50 点

实际上我们的代码还可以简化：

```
......
NSDictionary *views = NSDictionaryOfVariableBindings(self.view, view, vie
w2);
[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:|-50-[view(>=150)]-[view2(>=50)]" options:0 metrics:nil views:views]
];
[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-100-[view(>=150)]" options:0 metrics:nil views:views]];
[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-100-[view2(>=50)]" options:0 metrics:nil views:views]];
```

因为两个视图水平方向上是并排（从左到右）的，所以我们可以将水平方向布局的代码合并到一起。而垂直方向我们并非并排的，所以垂直方向的布局代码我们不能合并。这里所讲的并排的意思是后一个在前一个的后面，水平方向上明显是这样，但垂直方向上两个视图的 `y` 是相同的，所以无法合并在一起布局。

例二： 继续添加一个视图 `view3` 填补 `view` 右下方的空缺，效果如图：



代码如下：

```
UIView *view = [[UIView alloc] init];
[view setBackgroundColor:[UIColor redColor]];
[self.view addSubview:view];

UIView *view2 = [[UIView alloc] init];
[view2 setBackgroundColor:[UIColor blueColor]];
[self.view addSubview:view2];

UIView *view3 = [[UIView alloc] init];
[view3 setBackgroundColor:[UIColor orangeColor]];
[self.view addSubview:view3];

[view setTranslatesAutoresizingMaskIntoConstraints:NO];
[view2 setTranslatesAutoresizingMaskIntoConstraints:NO];
[view3 setTranslatesAutoresizingMaskIntoConstraints:NO];

NSDictionary *views = NSDictionaryOfVariableBindings(self.view, view, vie
w2, view3);

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:|-(50)-[view(>=150)]-[view2(>=50)]" options:0 metrics:nil views:view
s]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-(100)-[view(>=150)]" options:0 metrics:nil views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:[view]-[view3(>=50)]" options:0 metrics:nil views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-(100)-[view2(>=50)][view3(>=100)]" options:0 metrics:nil views:vie
ws]];
```

解释一下：

1. 间距用 `()` 括起来这种写法是可选的，这里仅仅是为了说明还可以这么写，但是没有必要这么写，也不建议
2. `V:|-(100)-[view2(>=50)][view3(>=100)]` 垂直方向布局，`view2` 距离父视图（顶部）100 点，并且高度不小于 50 点；`view3` 紧挨着 `view2` 底部（没有 `-`），并且高度不小于 100 点

`<<options>>`

这个参数的值是位掩码，使用频率并不高，但非常有用。它可以操作在 `VFL` 语句中的所有对象的某一个属性或方向。例如上面的例一，水平方向有两个视图，它们的垂直方向到顶部的距离相同，或者说顶部对齐，我们就可以给这个参数传入 `NSLayoutFormatAlignAllTop` 让它们顶部对齐，这样以来只需要指定两个视图的其中一个的垂直方向到顶部的距离就可以了。

代码如下：

```
......
NSDictionary *views = NSDictionaryOfVariableBindings(self.view, view, vie
w2);

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:|-50-[view(>=150)]-[view2(>=50)]" options:NSLayoutFormatAlignAllTop
metrics:nil views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-100-[view(>=150)]" options:0 metrics:nil views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:[view2(>=50)]" options:0 metrics:nil views:views]];
```

它的默认值是 `NSLayoutFormatDirectionLeadingToTrailing` (0)，根据当前用户的语言环境进行设置，比如英文中就是从左到右，希伯来语中就是从右到左。

因为是位掩码，所以我们可以使用 `|` 进行多选，例如例一，我们希望在现有约束的基础上让两个视图的高度相等，那代码可以这样写：

```
......
NSDictionary *views = NSDictionaryOfVariableBindings(self.view, view, vie
w2);

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:|-50-[view(>=150)]-[view2(>=50)]" options:NSLayoutFormatAlignAllTop
| NSLayoutFormatAlignAllBottom metrics:nil views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-100-[view(>=150)]" options:0 metrics:nil views:views]];
```

由于，指定两个视图的顶部和底部约束相同，因此，之后只设置其中一个视图的相关约束即可。

`<<metrics>>`

这是一个字典，字典的键必须是出现在 `VFL` 语句中的字符串，值必须是 `NSNumber` 类型，作用是将在 `VFL` 语句中出现的键替换为相应的值。例如本文中的第一个布局的例子，使用了这个

参数后代码就变成了这样：

```
UIView *view = [[UIView alloc] init];
[view setBackgroundColor:[UIColor redColor]];
[self.view addSubview:view];

[view setTranslatesAutoresizingMaskIntoConstraints:NO];

CGRect viewFrame = CGRectMake(50.f, 100.f, 150.f, 150.f);

NSDictionary *views = NSDictionaryOfVariableBindings(self.view, view);

NSDictionary *metrics = @{@"left": @(CGRectGetMinX(viewFrame)),
                          @"top": @(CGRectGetMinY(viewFrame)),
                          @"width": @(CGRectGetWidth(viewFrame)),
                          @"height": @(CGRectGetHeight(viewFrame))};

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:|-left-[view(>=width)]" options:0 metrics:metrics views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-top-[view(>=height)]" options:0 metrics:metrics views:views]];
```

实际上这个参数也可以使用 `NSDictionaryOfVariableBindings(...)` 宏来快速创建，代码如下：

```
......
[view setTranslatesAutoresizingMaskIntoConstraints:NO];

NSNumber *left = @50.f;
NSNumber *top = @100.f;
NSNumber *width = @150.f;
NSNumber *height = @150.f;

NSDictionary *views = NSDictionaryOfVariableBindings(self.view, view);
NSDictionary *metrics = NSDictionaryOfVariableBindings(left, top, width,
height);

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"H:|-left-[view(>=width)]" options:0 metrics:metrics views:views]];

[self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat
:@"V:|-top-[view(>=height)]" options:0 metrics:metrics views:views]];
```

`<<views>>`

又是一个字典，包含了 `VFL` 语句中用到的视图。字典的键必须是出现在 `VFL` 语句中的视图名称，值必须视图的实例。这个字典我们在讲 `format` 时已经讲过，也用过很多次，这里不再赘述

优先级 （Priority level）

约束条件有优先级，高优先级约束会比低优先级约束优先得到满足，系统内置了 4 个优先级：

```
enum {
    UILayoutPriorityRequired = 1000,
    UILayoutPriorityDefaultHigh = 750,
    UILayoutPriorityDefaultLow = 250,
    UILayoutPriorityFittingSizeLevel = 50,
};
typedef float UILayoutPriority;
```

- 优先级的取值在 0 ~ 1000 之间，取值越大，优先级越高，越会被优先满足。
- 每个约束的默认优先级就是 `UILayoutPriorityRequired`，这意味着你给出的所有约束都必须得到满足，一旦约束间发生冲突，你的应用就会 Crash。这也是在使用 Auto Layout 时经常会犯的错误：没有给约束设置适当的优先级。
- 在约束值之后加 `@` 符号，就可以为该 `VFL` 约束串设置优先级

优先级的例子如下：

```
    NSDictionary *metrics = @{@"defaultHighPriroty":@(UILayoutPriorityDef
aultHigh)};

    [self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFo
rmat:@"V:|-350-[_view2(>=200@defaultHighPriroty)]" options:0 metrics:metr
ics views:views]];

    [self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFo
rmat:@"V:|-350-[_view2(<=100@1000)]" options:0 metrics:0 views:views]];
```