A black and white cat with orange patches is sitting on a dark, textured surface. The cat's head is slightly tilted, and its eyes are looking towards the camera. The background is a soft, out-of-focus grey.

# iOS/Macアプリ開発の AutoLayoutプログラミング について考えてみた

# 自己紹介

- 藤本尚邦 (@fhisa)
- <https://github.com/fhisa>
- フリーランスプログラマー
- RubyCocoaフレームワーク原作者
- Mac開発歴、薄く長く約25年
- iOS開発歴、1年弱

# AutoLayoutの基本

あるビューの位置やサイズを、**別の何かとの関係式**として定義すること:

$$\text{ビューXの縦幅} = \text{ビューYの縦幅} \times A + B$$

$$\text{ビューXの縦幅} = \text{ビューXの横幅}$$

$$\text{ビューXの上端} = \text{上部マージン} + A$$

$$\text{ビューXの横幅} = A$$

(※  $A$  と  $B$  は定数)

# AutoLayoutの基本

別の何かは、ビューの位置やサイズとは限りません:

- 自身または他のビューの位置や高さ
- マージン(UILayoutSupportプロトコル)
- なし(nil)

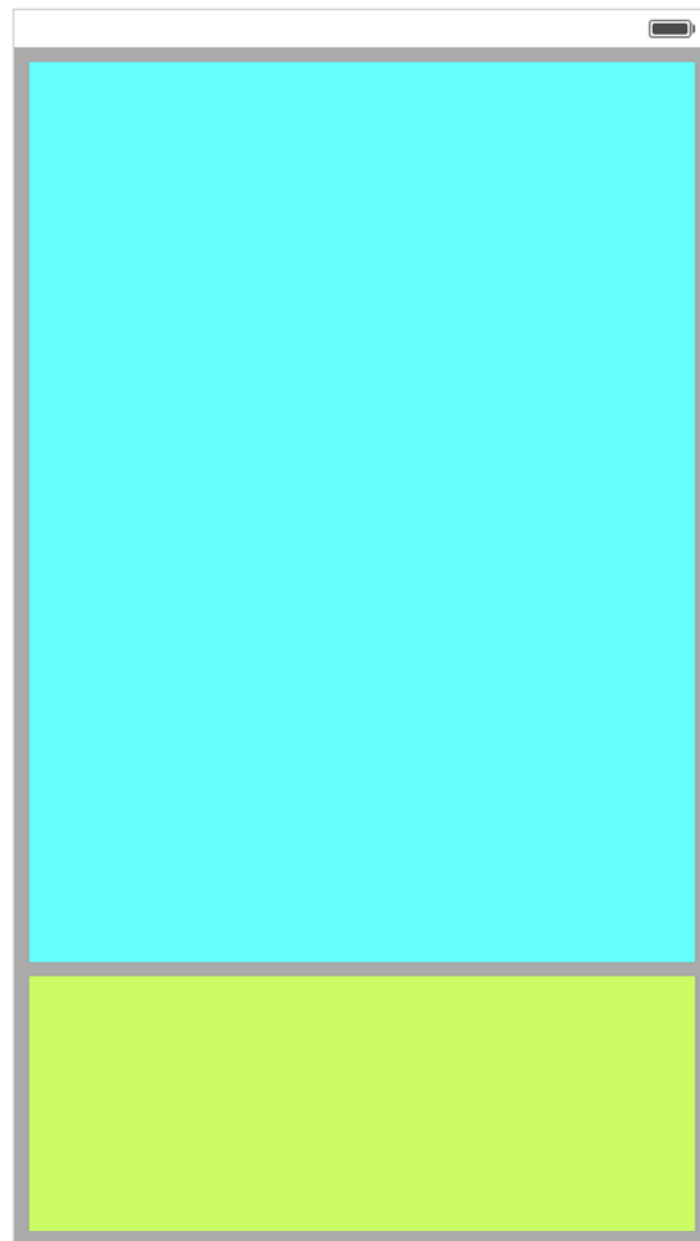
# AutoLayoutの基本

iOS/Mac OS Xプログラミングでは、この関係式をNSLayoutConstraintクラスのオブジェクトとして扱います。

つまり、NSLayoutConstraint(レイアウト制約)を定義することが、AutoLayoutプログラミングの基本となります。



# レイアウト例



baseView  
mainView  
infoView

- mainViewの高さ = baseViewの高さ  $\times$  3/4
- mainViewの上端 = baseViewの上端 + 8
- mainViewの左端 = baseViewの左端 + 8
- mainViewの右端 = baseViewの右端 - 8
- infoViewの上端 = mainViewの下端 + 8
- infoViewの左端 = baseViewの左端 + 8
- infoViewの右端 = baseViewの右端 - 8
- infoViewの下端 = baseViewの下端 - 8

# AutoLayoutを定義する方法

1. storyboard/xib ファイルをXcodeのGUIで編集
2. プログラムでNSLayoutConstraintを生成

# 質問です

- AutoLayoutを、Xcode GUIではなくプログラムで定義することはありますか？
- どんなときにプログラムで定義しますか？



この発表ではプログラムで  
のNSLayoutConstraint定義  
に的を絞ります

# NSLayoutConstraintの生成方法

1. ビジュアル言語で制約を記述し、文字列引数として与えて生成
2. 制約に関係する全ての値を引数に与えて生成

# ビジュアル言語

- プログラム実行時に構文がチェックされる
  - つまり走らせてみなければわからない
- 覚えるのも面倒
- 実用に難ありなので、ここでは触れません

# ならば全ての値を引数渡し

```
class NSLayoutConstraint: NSObject {  
    convenience init(item view1: AnyObject,  
        attribute attr1: NSLayoutConstraintAttribute,  
        relatedBy relation: NSLayoutConstraintRelation,  
        toItem view2: AnyObject?,  
        attribute attr2: NSLayoutConstraintAttribute,  
        multiplier multiplier: CGFloat,  
        constant c: CGFloat)  
}
```

引数多すぎワロたwww

実際に使うとこんな感じ...

# NSLayoutConstraint

```
baseView.addConstraints([
    NSLayoutConstraint(
        item: mainView, attribute: .Height, relatedBy: .Equal,
        toItem: baseView, attribute: .Height, multiplier: 3.0 / 4.0, constant: 0),
    NSLayoutConstraint(
        item: mainView, attribute: .Leading, relatedBy: .Equal,
        toItem: baseView, attribute: .Leading, multiplier: 1, constant: 8),
    NSLayoutConstraint(
        item: mainView, attribute: .Top, relatedBy: .Equal,
        toItem: baseView, attribute: .Top, multiplier: 1, constant: 8),
    NSLayoutConstraint(
        item: mainView, attribute: .Trailing, relatedBy: .Equal,
        toItem: baseView, attribute: .Trailing, multiplier: 1, constant: -8),
    // 以下、長いので略
])
```



文字多すぎワロた

\(^o^)/

ワロえない  
(´・ω・`)

**NSLayoutConstraint なんて**

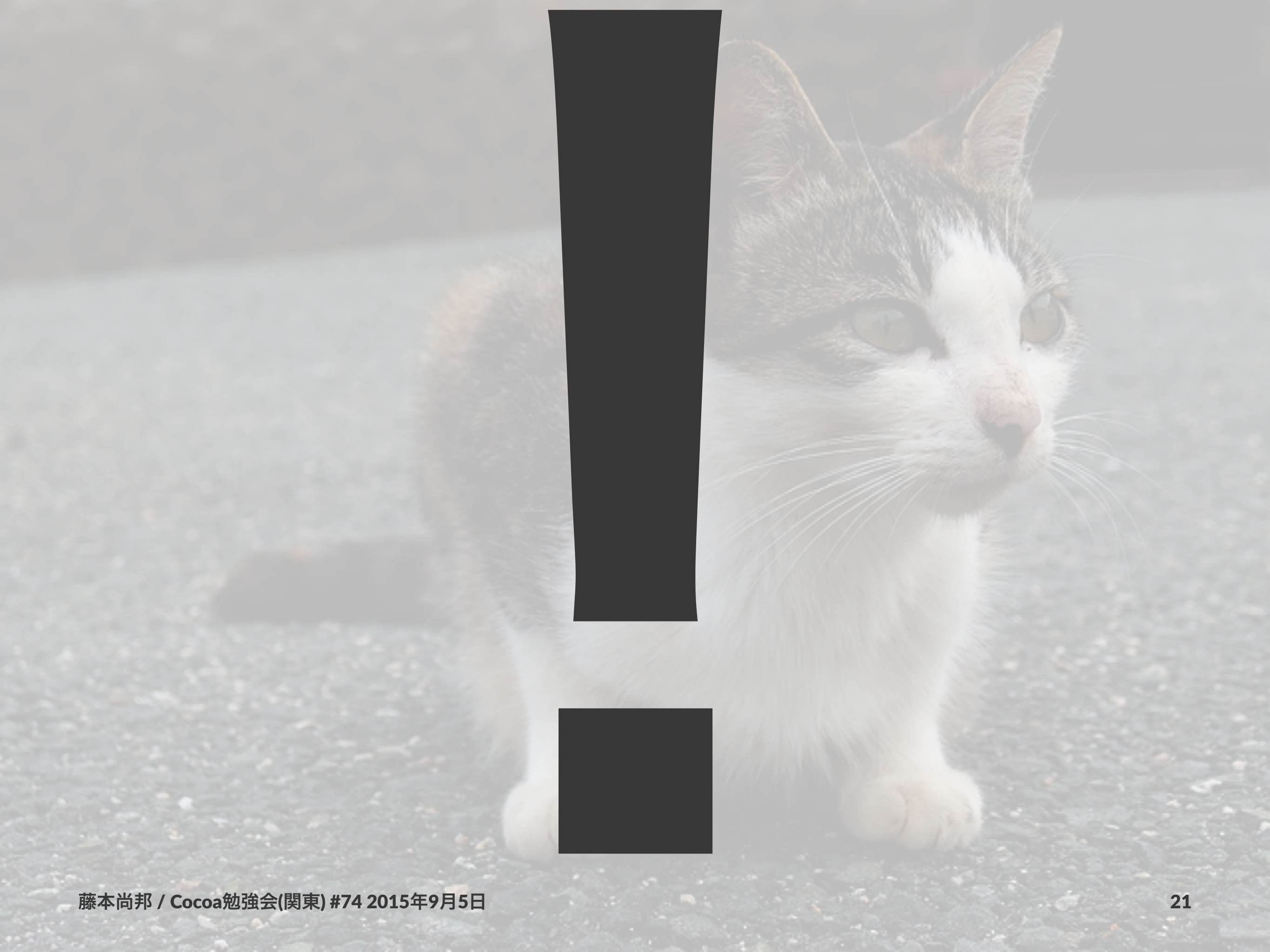
**大っ嫌いなんだからねっ！**

**9 ( ㊤ ^ ^ ㊤ ) ㇏**









# こんな風に書けたら読み書きしやすいなあ

```
baseView.addConstraints([
    mainView[.Height] * 4 == baseView[.Height] * 3,
    mainView[.Leading] == baseView[.Leading] + 8,
    mainView[.Top] == baseView[.Top] + 8,
    mainView[.Trailing] == baseView[.Trailing] - 8,
    // 以下も略さないよ

    infoView[.Top] == mainView[.Bottom] + 8,
    infoView[.Leading] == baseView[.Leading] + 8,
    infoView[.Trailing] == baseView[.Trailing] - 8,
    infoView[.Bottom] == baseView[.Bottom] - 8,
])
```



そこで奥さん！

# FormulaStyleConstraint

ですよ！

# FormulaStyleConstraint

NSLayoutConstraintを等式・不等式などの数式で定義できるようにするSwift用のフレームワーク

<https://github.com/fhisa/FormulaStyleConstraint>

# FormulaStyleConstraintの使用例

```
baseView.addConstraints([
    mainView[.Height] * 4 == baseView[.Height] * 3,
    mainView[.Leading] == baseView[.Leading] + 8,
    mainView[.Top] == baseView[.Top] + 8,
    mainView[.Trailing] == baseView[.Trailing] - 8,
    infoView[.Top] == mainView[.Bottom] + 8,
    infoView[.Leading] == baseView[.Leading] + 8,
    infoView[.Trailing] == baseView[.Trailing] - 8,
    infoView[.Bottom] == baseView[.Bottom] - 8,
])
```

# FormulaStyleConstraintの特徴

- NSLayoutConstraintの数式による定義に目的を絞ったシンプルな構成
- ソースコードはたったの154行、実装の理解が容易です (バージョン1.2)

# FormulaStyleConstraintの課題

- UILayoutSupportプロトコルのサポート
  - マージン(topLayoutMargin, bottomLayoutMargin)を扱えない
- Swift 2.0 の protocol extension でおそらく対応可能
- Mac OS Xのサポート

# FormulaStyleConstraintの競合品

制約を数式で定義するというアイディアは、誰かがすでに作ってる可能性大。しかし、自作する楽しみを味わいたかったので調べずに作りました。

ひとまず完成してから調べたところ、やっぱりありました(´・ω・`)

# Cartography

Using Cartography, you can set up your Auto Layout constraints in declarative code and without any stringly typing!

<https://github.com/robb/Cartography>




# Cartographyの使用例

```
layout(baseView, mainView) {  
    $1.height == $0.height * (3.0 / 4.0)  
    $1.Leading == $0.leading + 8  
    $1.top == $0.top + 8  
    $1.trailing = $0.trailing - 8  
    // 以下略  
}
```

# Cartographyの特徴

- 制約の定義をブロック内に記述するDSLタイプ
- 整列(align)などの拡張機能あり
- ソースコード約1400行(バージョン0.5)
- 名前が短くてかっこいい
- GitHubのスター数約3000 (FormulaStyleConstraintは0、ゼロ、Nothing)

# 開発にあたっての感想など

- 初めてTravis CIを使ってみた( バッジを付けてみたかっただけ)
- 初めてCarthageに対応してみた(これは便利)
- 初めてプルリクをもらった (ただしボットにw)
- ひとり焼き肉、ひとりディズニーランド、ひとり美ら海水族館、ひとりGitHub

# まとめ

FormulaStyleConstraintにせよ、Cartographyにせよ、素でNSLayoutConstraintのコードを書くよりはるかに楽ちんなので、AutoLayoutをプログラムで書いている人にはたいへんオススメです！

- <https://github.com/fhisa/FormulaStyleConstraint>
- <https://github.com/robb/Cartography>

A black and white cat with orange patches is sitting on a dark, textured surface. The cat is looking towards the right. The text "Thank you!" is overlaid in a large, bold, black font across the center of the image.

# Thank you!