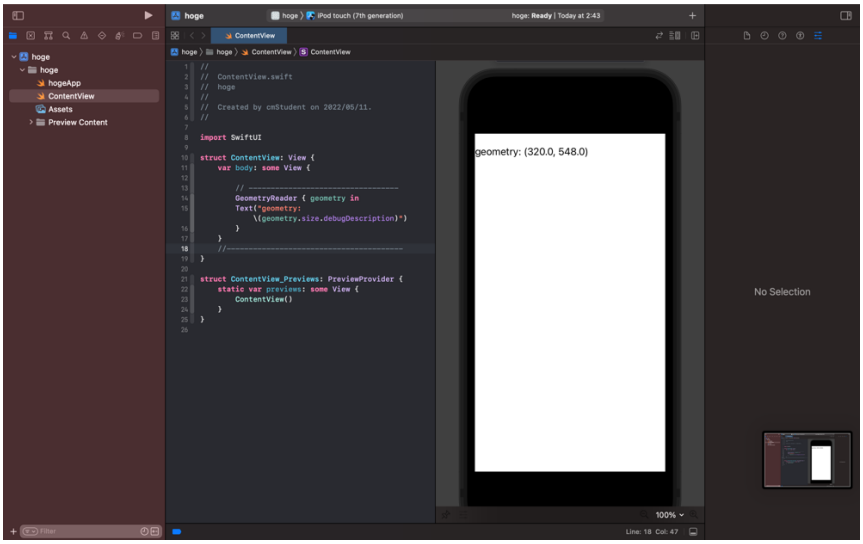
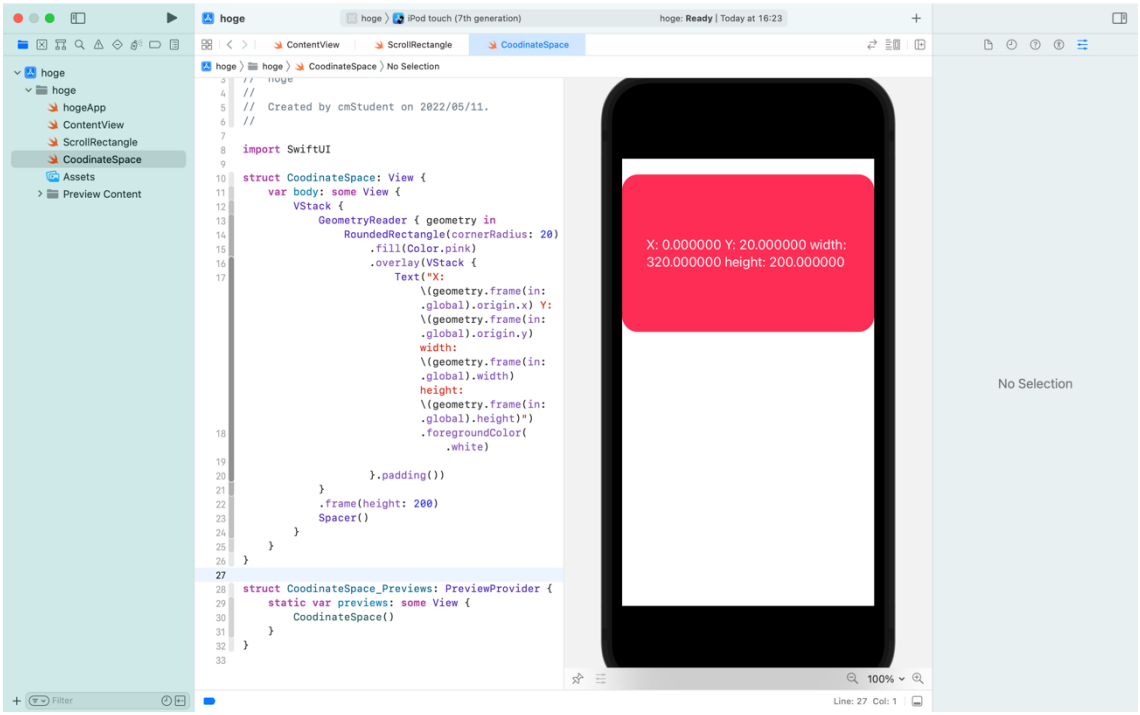
Swift 課題 GeometryReader について

GeometryReader は View の一種であり、親の View のサイズと自信の親の View に対する位置をしるためのものです。また、GeometryReader は特別な View で、自身のサイズと座標空間を返す関数をクロージャーとして保持しています。そのクロージャーを通して、自身の View や RootView のサイズ、座標位置を取得することができます。View のサイズや座標の計算が必要な場面では GeometryReader を使用します。
クロージャーの引数の geometry は GeometryProxy プロトコルに準拠したインスタンスです。geometry.size で View サイズを表示しますと、インスタンスから View のサイズや座標位置を取得できます。自身の View のサイズを知りたかったら geometry.size にアクセスをします。



さらに、GeomtryProxy プロトコルには frame メソッドがあり、RootView または自身の座標空間を取得できます。SwiftUI の座標は UIKit と同じく左上から始まります。

X 軸の右側に行くほど X 座標の値は増え、Y 軸の下側に行くほど Y 座標の値が増えます。ちなみに view に対して edgesIgnoringSafeArea をするとステータスバー含めた領域から origin 座標が計算されます。
次に edgesIgnoringSafeArea で画面を表示します。



geometry.frame(in: .global).origin で RootView の座標空間を取得しています。frame メソッドの引数には CoordinateSpace 型を指定できます。.global は RootView で.local はその View 自身を表しています。各 origin の X 座標、Y 座標、View の Width, Height の値を Text で表示しています。それぞれ次のような値になっています。・X: 0 , ・Y: 44(セーフエリアの下から計算しているためステータスバーの高さ分の値が出ています。

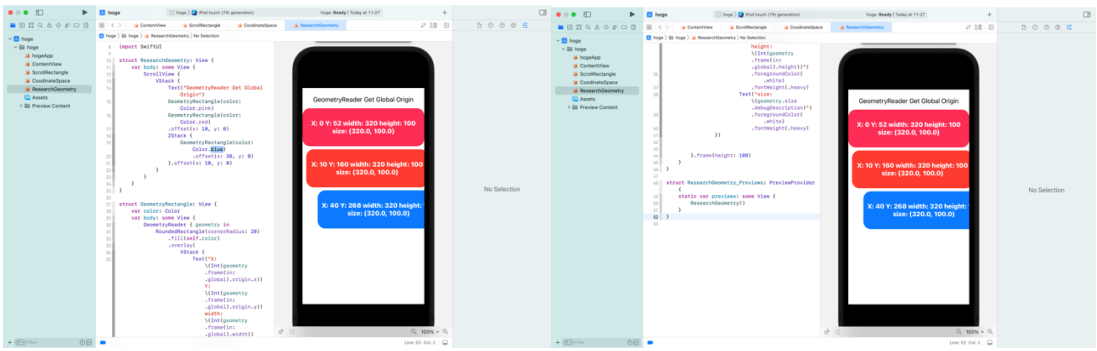
・Width: 414(自身の View の幅=端末の横幅)

・Height: 200(自身の View の高さ.frame(height: 200)を指定している)

ちなみに geometry.frame(in: .local).origin は自身の View 座標の origin なのでいつも(0,0)になります。これは UIView の bounds.origin がつねに(0, 0)であることと同じです。

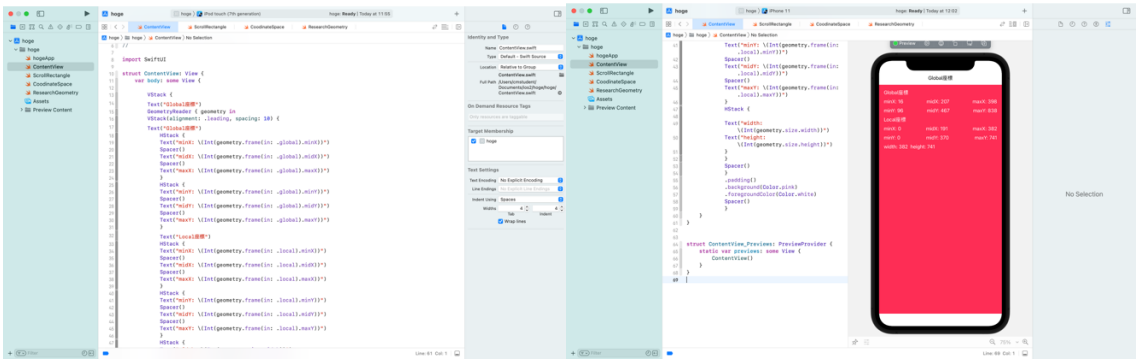
現状 CoordinateSpace 型にはドキュメントがありません。

次は、GeometryRectangle という色付きの四角の View に各 geometry の情報を出力するコードです。

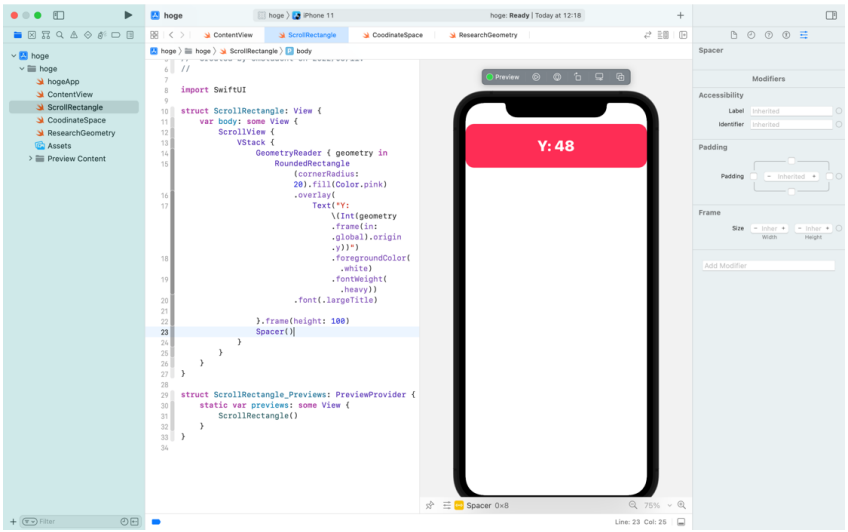


3 つの四角い View を表示しているところで、それぞれ X 座標が 0, 10, 40 となっています。

3 つ目の View は ZStack にネストしていますが ZStack は x 座標に 10、offset しており、GeometryRectangle 自身も x 座標に 30、offset しているので合計 40 の値となっています。ひと 47 つ上の親 View からの計算ではなく RootView からの座標計算になっています。ここから.global を指定すると RootView から座標計算されることがわかります。
GeometryProxy プロトコルの frame メソッドで取得できるものは origin 座標だけでなく、X、Y 座標の Min,Mid,Max 座標それぞれも取得できます。

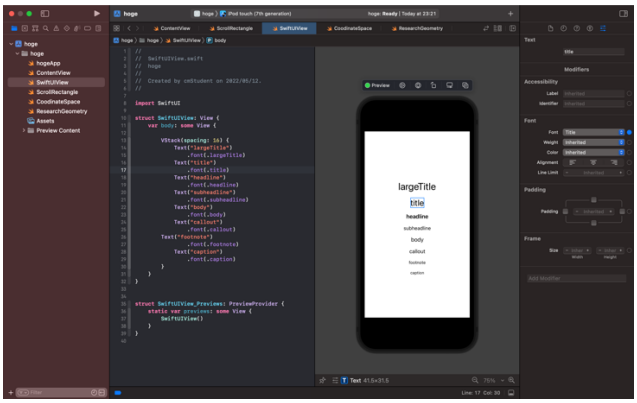


補足説明を入れると
・GeometryReader に.padding()で上下左右にデフォルトパディングを追加し、デフォルトの値は 16 です。
・Global の MinX はパディングのデフォルト値が加わって 16 です。しかし Local は 0。Global の MidX, MaxX は Local の値にパディング分の 16 を足したものになっています。
・Y 座標も Global の MinY が 92 から始まっているのに対して Local は 0 です。Global の MidY,MaxY は Local のものに 96 を足したものになっています。
さらに、GeometryReader で取得できる origin 座標は ScrollView でスクロールすると動変更されます。

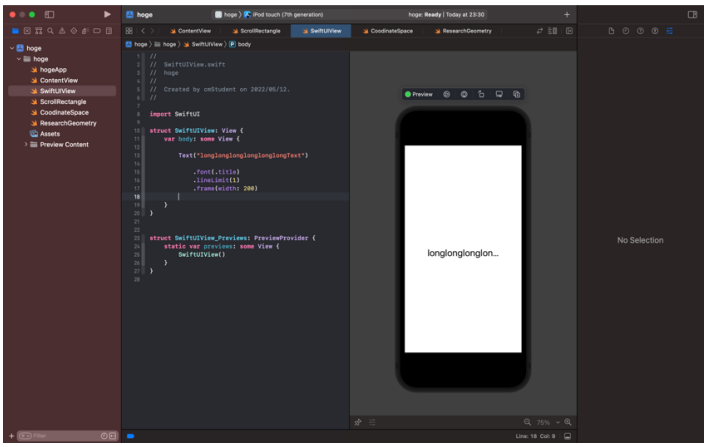


スクロールをすると geometry.frame(in: .global).origin.y の値が変更されるのが画像からわかります。これを検知することで次のような UI を実現することができます。

・ScrollView のスクロールに合わせてコンテンツを操作する、スクロールに合わせてコンテンツのパララックス効果、ヘッダーをスクロールに合わせて拡大、縮小する、横スクロールでコンテンツが中心に来たら目立たせるなどです。
次は SwiftUI のそれぞれの View についてです。まずは、Text です。一行以上の表示専用のテキストを表示する View です。Text 専用の View 修飾子が豊富にあり、テキストの修飾は簡単に設定できます。後は font です。font は Text のフォントを指定する View 修飾子です。

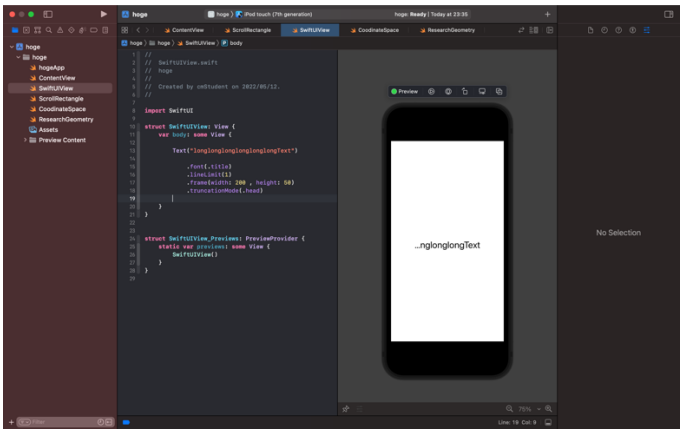


組み込みのフォントとして次のようなものが指定できます。それぞれフォントサイズとウェイトが異なるのでテキストの意味合いによって使い分けます。
表示は・largeTitle / ラージタイトル・title / タイトル・headline / ヘッドライン
・subheadline / サブヘッドライン・body / 本文・callout / コールアウト
・footnote / 脚注・caption / 見出しとなっています
次は、
Linelimat です。Text の行数を制御する View 修飾子です。引数で渡した数値が表示可能な最大行数となります。nil なら行数制限がなくなります。

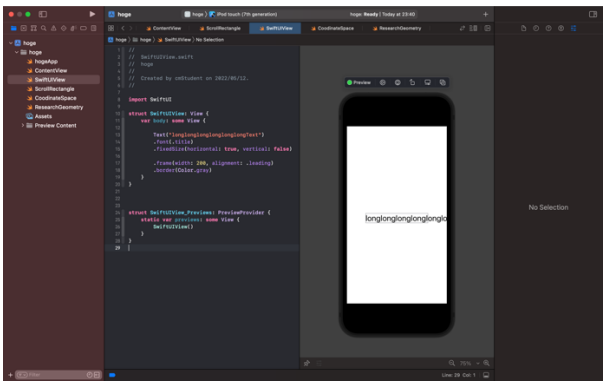


1 行指定したことで.frame(width: 200)で指定したサイズに収まりきらず「longlonglo...」と省略されまた。省略される位置はデフォルトでは文末です 。変更したい場合

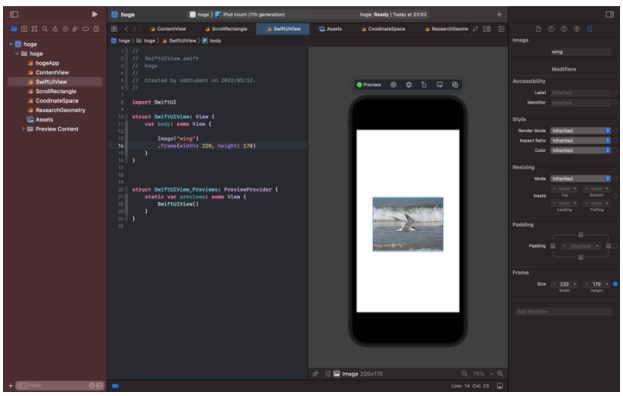
は.truncationMode を使います。truncationMode は文字を描画したときの Text の View サイズが親 View よりも大きくなる場合に省略される文字の位置を変更できます。引数に Text.TruncationMode 型の値を指定します。・.head:文頭を省略する・.middle:文中を省略する・.tail:文末を省略する(デフォルト値)。



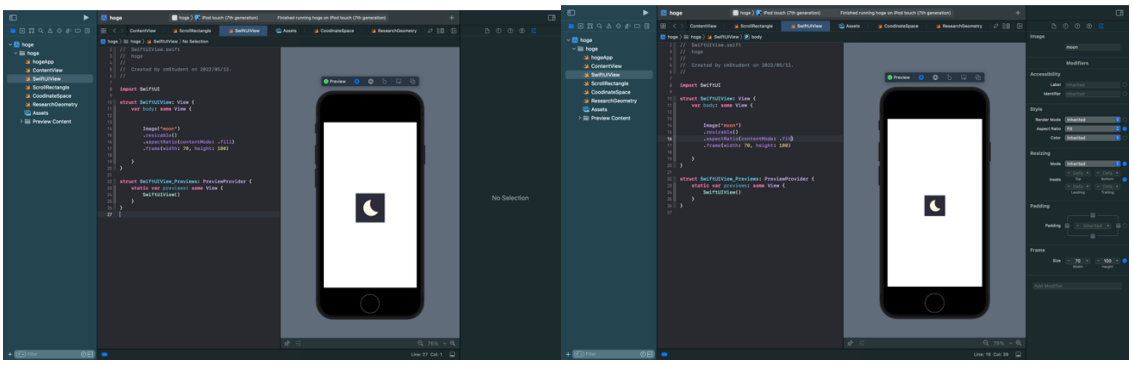
次は fixedSize です。垂直方向または水平方向に View のサイズを固定化させる View 修飾子となっています。fixedSize を指定すると表示内容にかかわらず、View のサイズが広がらなくなります。Text に指定することで、親のサイズが小さい場合に見切れたように表示することができます。



次は Image です。Image は画像を表示する View です。画像の指定はアセットカタログに登録した画像の名前を指定できるほか、SF Symbols1 のアイコンも利用できます。アセットカタログに登録した wing という画像で Image を作るには次のようにします。



次は、aspect です。aspectRatio アスペクト比を指定するには aspectRatio 修飾子を利用します。引数に ContentMode 型を指定します。fill はアスペクト比を保ったまま親 View のサイズに合わせて拡大します。親 View 全体まで拡大するので画像が上下左右見切れる可能性があります。fit は画像全体が表示されるところまでアスペクト比を保ちながら拡大します。



最後に renderingMode です。renderingMode は画像の色的に変えることができます。renderingMode に .template を指定し、.foregroundColor で色を変更すると画像がその色に変更されます。画像の色を変更したくない場合は renderingMode に.original を指定します。