

# PencilKitで実装する PDFへの手書き注釈

iOSDC Japan 2024 Day2 Track B

Ras (@ras0q)

# PDF注釈がしたい！！！

- アプリの機能の一部として注釈機能がほしい
- 世のPDFアプリが絶妙にハマらないため自作したい
  - 複雑すぎる操作UI
  - ファイルのアクセス制限
  - 無限に現れる広告
  - など...

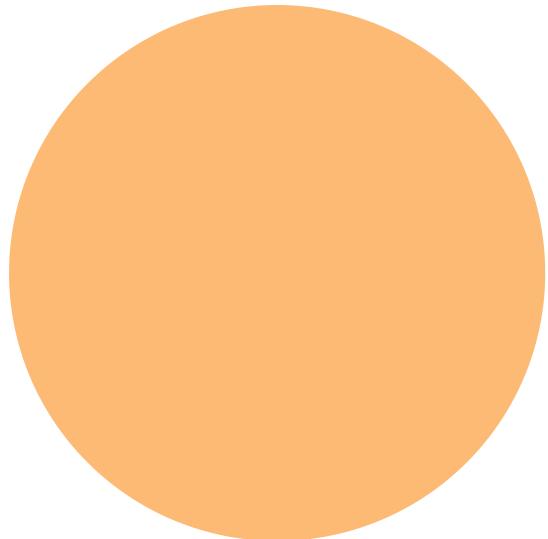
このトーケを見ることでPDF注釈アプリがイチから作れるようになります!!!

# Keynote

1. PencilKitとは？
2. PencilKitでアプリを作る
3. PencilKitをPDFに組み込む
4. PencilKitのドローイングをPDF注釈として保存する
5. PencilKitを使ってみた感想

↓↓サンプルレポジトリ↓↓

**[github.com/ras0q/iosdc2024](https://github.com/ras0q/iosdc2024)**



Ras

[ras0q.com](http://ras0q.com)



## 東京工業大学 大学院 修士1年

- 10月に大学が改名するらしい
- デジタル創作同好会traP
  - Web開発をメインに創作をしています
  - traPortfolio をリリースしました

## ピクシブ株式会社 アルバイト

- iOSエンジニア育成プロジェクト
- pixiv / pixiv Sketch / Pastela
- iOSDC Japan 参加(3) 登壇(2)

# Keynote

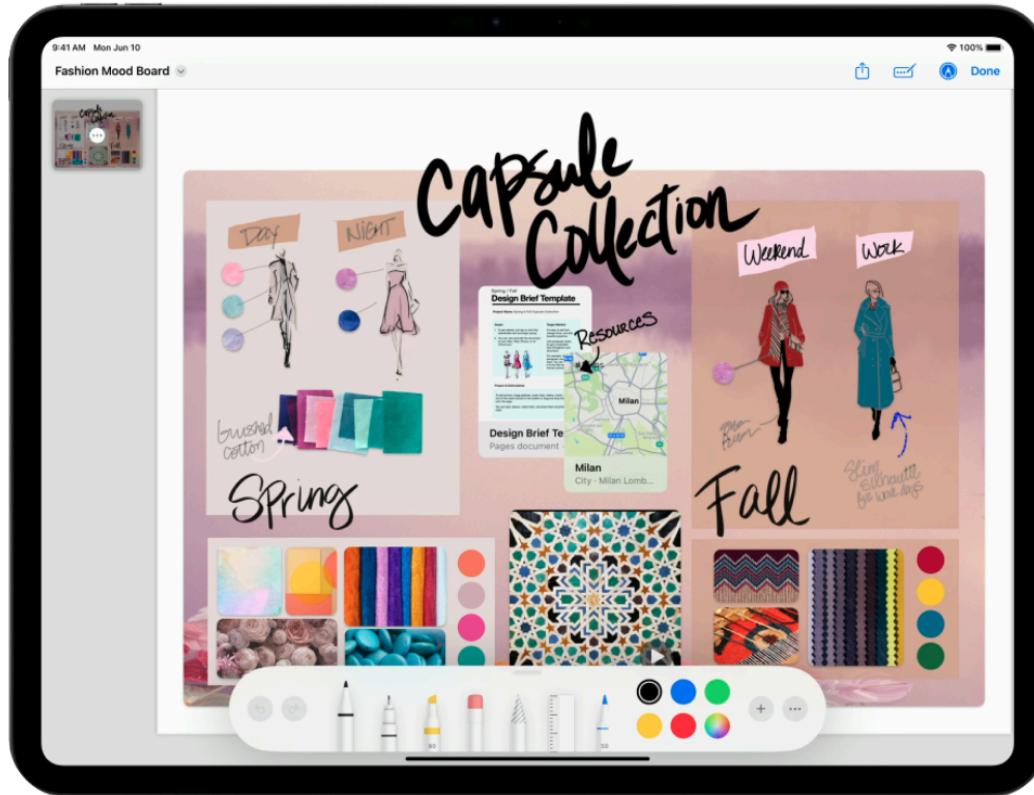
1. PencilKitとは
2. PencilKitでアプリを作る
3. PencilKitをPDFに組み込む
4. PencilKitのドローイングをPDF注釈として保存する
5. PencilKitを使ってみた感想

↓↓サンプルレポジトリ↓↓

**[github.com/ras0q/iosdc2024](https://github.com/ras0q/iosdc2024)**

# PencilKit?

# PencilKit?



引用元: WWDC19

# PencilKit

ドローイングをiOSアプリに組み込むことができる純正ライブラリ

指やApple Pencilからの入力を受け取ってアプリで使う画像データに変換する

描画ツール搭載

鉛筆のほかに  
万年筆や定規も...

純正アプリにも

ファイル, メモ,  
写真...

アプリ間の連携

別アプリへ図形の  
コピペが可能

# Keynote

1. PencilKitとは? 
2. PencilKitでアプリを作る
3. PencilKitをPDFに組み込む
4. PencilKitのドローイングをPDF注釈として保存する
5. PencilKitを使ってみた感想

↓↓サンプルレポジトリ↓↓

**[github.com/ras0q/iosdc2024](https://github.com/ras0q/iosdc2024)**

# try! PencilKit

in 1 minute

# try! PencilKit

```
import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

# try! PencilKit

```
import PencilKit
import UIKit

class ViewController: UIViewController {
    private lazy var canvasView = PKCanvasView(frame: view.frame)

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(canvasView)
    }
}
```

# try! PencilKit

```
import PencilKit
import UIKit

class ViewController: UIViewController {
    private lazy var canvasView = PKCanvasView(frame: view.frame)
    private lazy var toolPicker = PKToolPicker()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(canvasView)

        toolPicker.addObserver(canvasView)
        toolPicker.setVisible(true, forFirstResponder: canvasView)

        canvasView.becomeFirstResponder()
    }
}
```

# try! PencilKit

```
import PencilKit
import UIKit

class ViewController: UIViewController {
    private lazy var canvasView = PKCanvasView(frame: view.frame)
    private lazy var toolPicker = PKToolPicker()

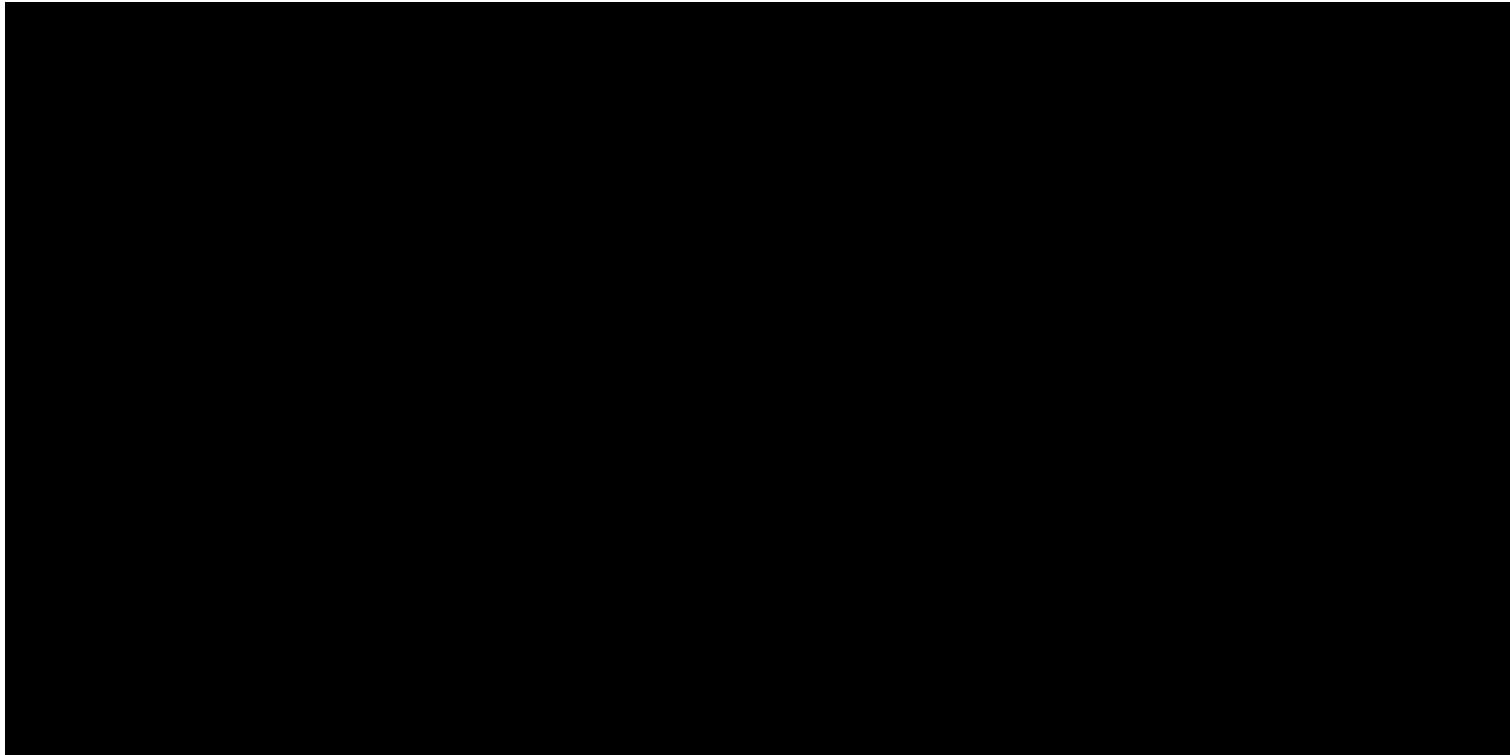
    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(canvasView)

        toolPicker.addObserver(canvasView)
        toolPicker.setVisible(true, forFirstResponder: canvasView)

        canvasView.becomeFirstResponder()
    }
}
```

 PencilKitでアプリを作る



# Keynote

1. PencilKitとは? 
2. PencilKitでアプリを作る 
3. PencilKitをPDFに組み込む
4. PencilKitのドローイングをPDF注釈として保存する
5. PencilKitを使ってみた感想

↓↓サンプルレポジトリ↓↓

**[github.com/ras0q/iosdc2024](https://github.com/ras0q/iosdc2024)**

**try! PDF Integration**

# PDFKit

```
import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

# PDFKit

```
import PDFKit
import UIKit

class ViewController: UIViewController {
    private lazy var pdfDocument = PDFDocument(somePDFURL)

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

# PDFKit

```
import PDFKit
import UIKit

class ViewController: UIViewController {
    private lazy var pdfDocument = PDFDocument(somePDFURL)
    private lazy var pdfView: PDFView = {
        let view = PDFView(frame: view.frame)
        view.document = pdfDocument
        return view
    }()
    
    override func viewDidLoad() {
        super.viewDidLoad()
        
        view.addSubview(pdfView)
    }
}
```

# PDFKit

```
import PDFKit
import UIKit

class ViewController: UIViewController {
    private lazy var pdfDocument = PDFDocument(somePDFURL)
    private lazy var pdfView: PDFView = {
        let view = PDFView(frame: view.frame)
        view.document = pdfDocument
        return view
    }()
    
    override func viewDidLoad() {
        super.viewDidLoad()
        
        view.addSubview(pdfView)
    }
}
```

どうやってPencilKitを組み込む？



# What's new in PDFKit

Conrad Carlen, PDF Engineer

PDFKit review

Live text and forms

Create PDFs from images

## Overlay views

"How can I draw on PDF pages  
using PencilKit?"--



PDFKit review

Live text and forms

Create PDFs from images

## Overlay views

the answer is to use an overlay view.



[PDFKit](#) / [PDFPageOverlayViewProvider](#)

Protocol

# PDFPageOverlayViewProvider

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst 16.0+ | macOS 13.0+ | visionOS 1.0+

```
protocol PDFPageOverlayViewProvider : NSObjectProtocol
```

---

## Topics

### Instance Methods

```
func pdfView(PDFView, overlayViewFor: PDFPage) -> UIView?
```

Required

# PDFPageOverlayViewProvider

```
import PencilKit

class ViewController: UIViewController {
    // 各ページに対してCanvasViewを作成
    private lazy var canvasViews = (0 ..< ページ数).map { _ in
        PKCanvasView()
    }

    // ページ番号をインデックスとしてcanvasViewsの要素を返す関数
    private func canvasView(for: page) → PKCanvasView { ... }
}

extension ViewController: PDFPageOverlayViewProvider {
    func pdfView(_: PDFView, overlayViewFor page: PDFPage) → UIView? {
        canvasView(for: page)
    }
}
```

# PDFPageOverlayViewProvider

```
import PencilKit

class ViewController: UIViewController {
    // 各ページに対してCanvasViewを作成
    private lazy var canvasViews = (0 ..< ページ数).map { _ in
        PKCanvasView()
    }

    // ページ番号をインデックスとしてcanvasViewsの要素を返す関数
    private func canvasView(for: page) → PKCanvasView { ... }
}

extension ViewController: PDFPageOverlayViewProvider {
    func pdfView(_: PDFView, overlayViewFor page: PDFPage) → UIView? {
        canvasView(for: page)
    }
}
```

# PDFPageOverlayViewProvider

```
import PencilKit

class ViewController: UIViewController {
    // 各ページに対してCanvasViewを作成
    private lazy var canvasViews = (0 ..< ページ数).map { _ in
        PKCanvasView()
    }

    // ページ番号をインデックスとしてcanvasViewsの要素を返す関数
    private func canvasView(for: page) → PKCanvasView { ... }
}

extension ViewController: PDFPageOverlayViewProvider {
    func pdfView(_: PDFView, overlayViewFor page: PDFPage) → UIView? {
        canvasView(for: page)
    }
}
```

# ドローイングの設定

```
class ViewController: UIViewController {
    private lazy var toolPicker = PKToolPicker()

    override func viewDidLoad() {
        // ...
        for canvasView in canvasViews {
            toolPicker.addObserver(canvasView)

            // キャンバスのドローイング用RecognizerをPDFViewに追加する
            pdfView.addGestureRecognizer(
                canvasView.drawingGestureRecognizer
            )
        }

        // 今回のfirst responderはPDFView
        toolPicker.setVisible(true, forFirstResponder: pdfView)
        pdfView.becomeFirstResponder()
    }
}
```

# ドローイングの設定

```
class ViewController: UIViewController {
    private lazy var toolPicker = PKToolPicker()

    override func viewDidLoad() {
        // ...
        for canvasView in canvasViews {
            toolPicker.addObserver(canvasView)

            // キャンバスのドローイング用RecognizerをPDFViewに追加する
            pdfView.addGestureRecognizer(
                canvasView.drawingGestureRecognizer
            )
        }

        // 今回のfirst responderはPDFView
        toolPicker.setVisible(true, forFirstResponder: pdfView)
        pdfView.becomeFirstResponder()
    }
}
```

# ドローイングの設定

```
class ViewController: UIViewController {
    private lazy var toolPicker = PKToolPicker()

    override func viewDidLoad() {
        // ...
        for canvasView in canvasViews {
            toolPicker.addObserver(canvasView)

            // キャンバスのドローイング用RecognizerをPDFViewに追加する
            pdfView.addGestureRecognizer(
                canvasView.drawingGestureRecognizer
            )
        }

        // 今回のfirst responderはPDFView
        toolPicker.setVisible(true, forFirstResponder: pdfView)
        pdfView.becomeFirstResponder()
    }
}
```

# ドローイングの設定

```
class ViewController: UIViewController {
    private lazy var toolPicker = PKToolPicker()

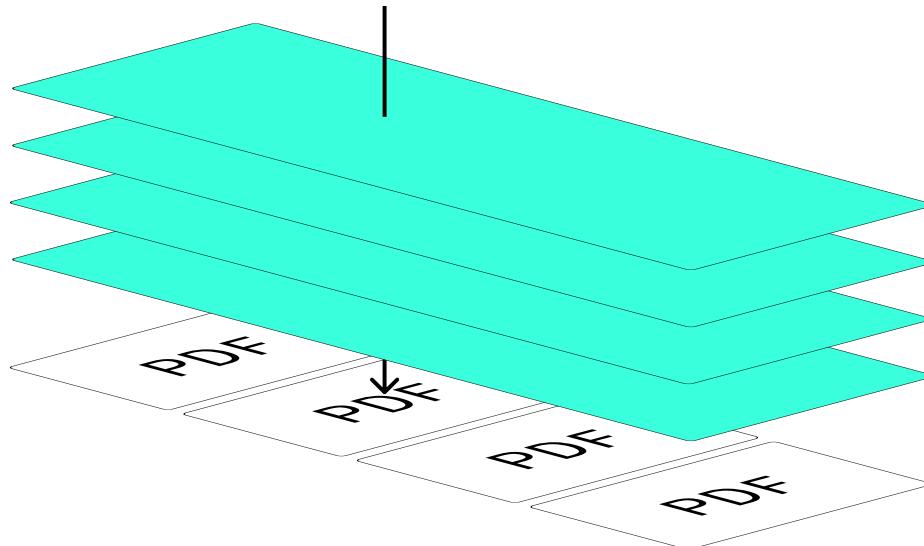
    override func viewDidLoad() {
        // ...
        for canvasView in canvasViews {
            toolPicker.addObserver(canvasView)

            // キャンバスのドローイング用RecognizerをPDFViewに追加する
            pdfView.addGestureRecognizer(
                canvasView.drawingGestureRecognizer
            )
        }

        // 今回のfirst responderはPDFView
        toolPicker.setVisible(true, forFirstResponder: pdfView)
        pdfView.becomeFirstResponder()
    }
}
```

複数のRecognizerを設定した結果...

最後に追加したRecognizerしか発火しない



canvasViews[3].drawingGestureRecognizer

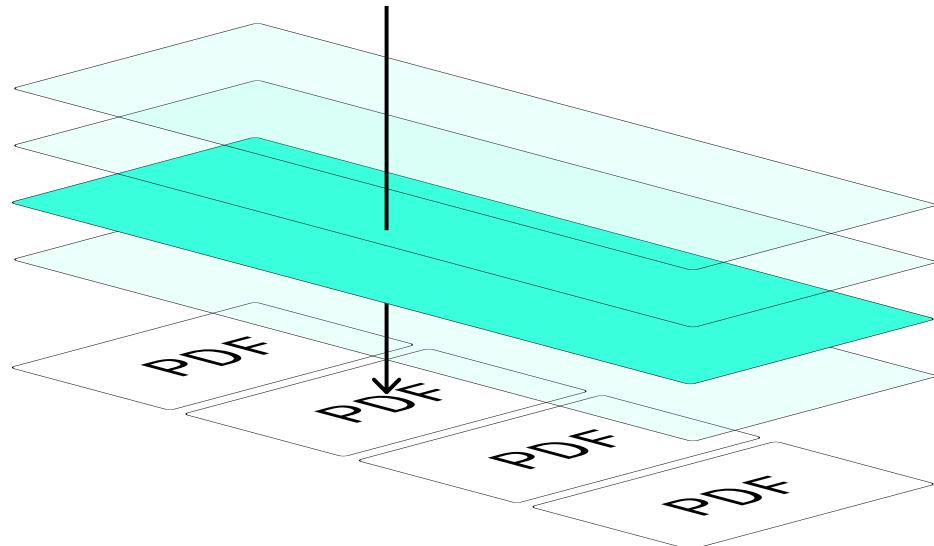
canvasViews[2].drawingGestureRecognizer

canvasViews[1].drawingGestureRecognizer

canvasViews[0].drawingGestureRecognizer

複数のRecognizerを設定した結果...

タップ時に該当ページのRecognizerのみを適切に発火させる必要がある



`canvasViews[3].drawingGestureRecognizer`

`canvasViews[2].drawingGestureRecognizer`

`canvasViews[1].drawingGestureRecognizer`

`canvasViews[0].drawingGestureRecognizer`

## Override func hitTest(\_:with:)

タップしたページのキャンバスの手書き認識のみ有効化

```
class CanvasPDFView: PDFView {
    let switchRecognizerHandler: (to: PDFPage) → Void

    override func hitTest(_ point:CGPoint,with e:UIEvent?) → UIView? {
        // pointから該当ページを探すことができる
        if let activePage = page(for: point, nearest: true) {
            switchRecognizerHandler(to: page)
        }

        return super.hitTest(point, with: e)
    }
}
```

## Override func hitTest(\_:with:)

タップしたページのキャンバスの手書き認識のみ有効化

```
class CanvasPDFView: PDFView {
    let switchRecognizerHandler: (to: PDFPage) → Void

    override func hitTest(_ point:CGPoint,with e:UIEvent?) → UIView? {
        // pointから該当ページを探すことができる
        if let activePage = page(for: point, nearest: true) {
            switchRecognizerHandler(to: page)
        }

        return super.hitTest(point, with: e)
    }
}
```

## Override func hitTest(\_:with:)

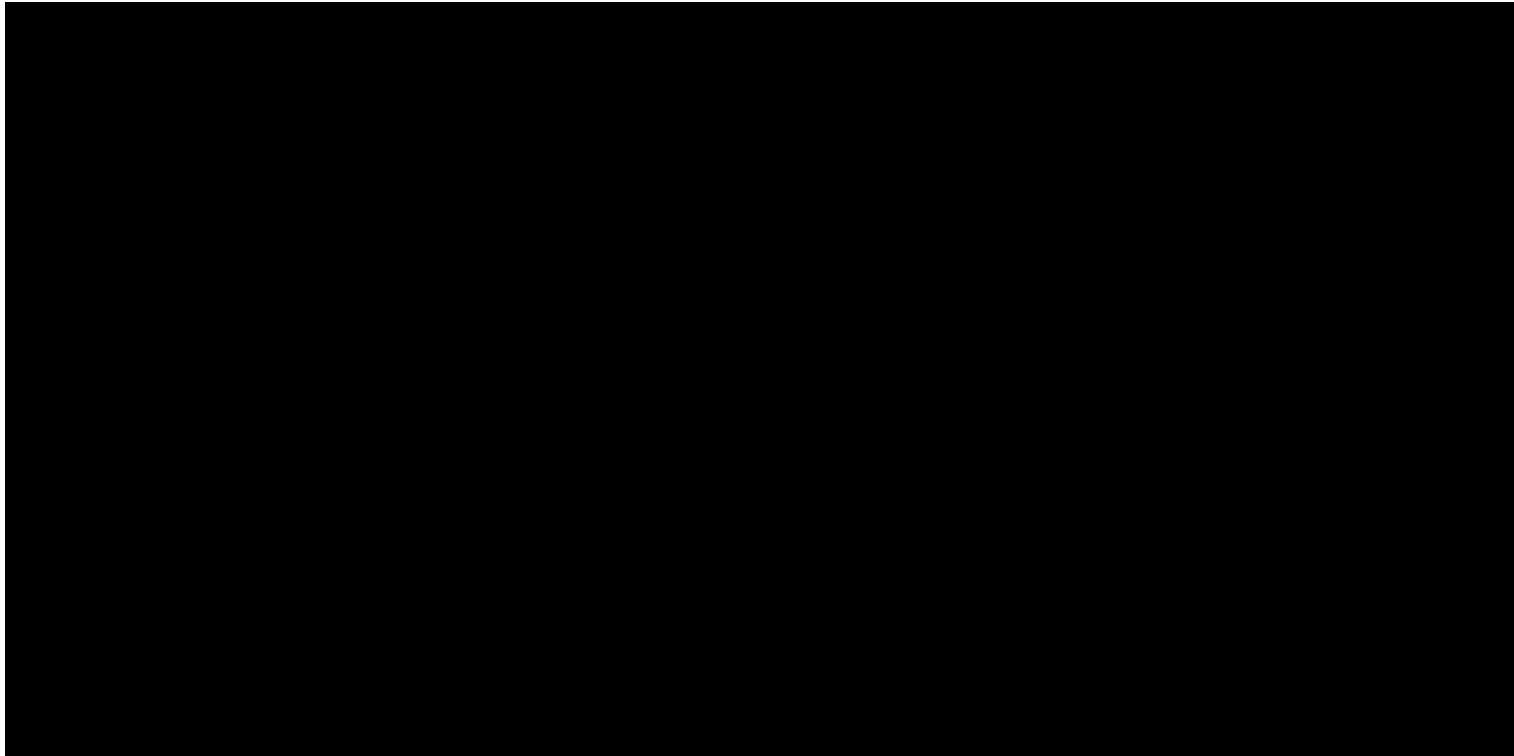
タップしたページのキャンバスの手書き認識のみ有効化

```
class CanvasPDFView: PDFView {
    let switchRecognizerHandler: (to: PDFPage) → Void

    override func hitTest(_ point:CGPoint,with e:UIEvent?) → UIView? {
        // pointから該当ページを探すことができる
        if let activePage = page(for: point, nearest: true) {
            switchRecognizerHandler(to: page)
        }

        return super.hitTest(point, with: e)
    }
}
```

 PencilKitをPDFに組み込む



# Keynote

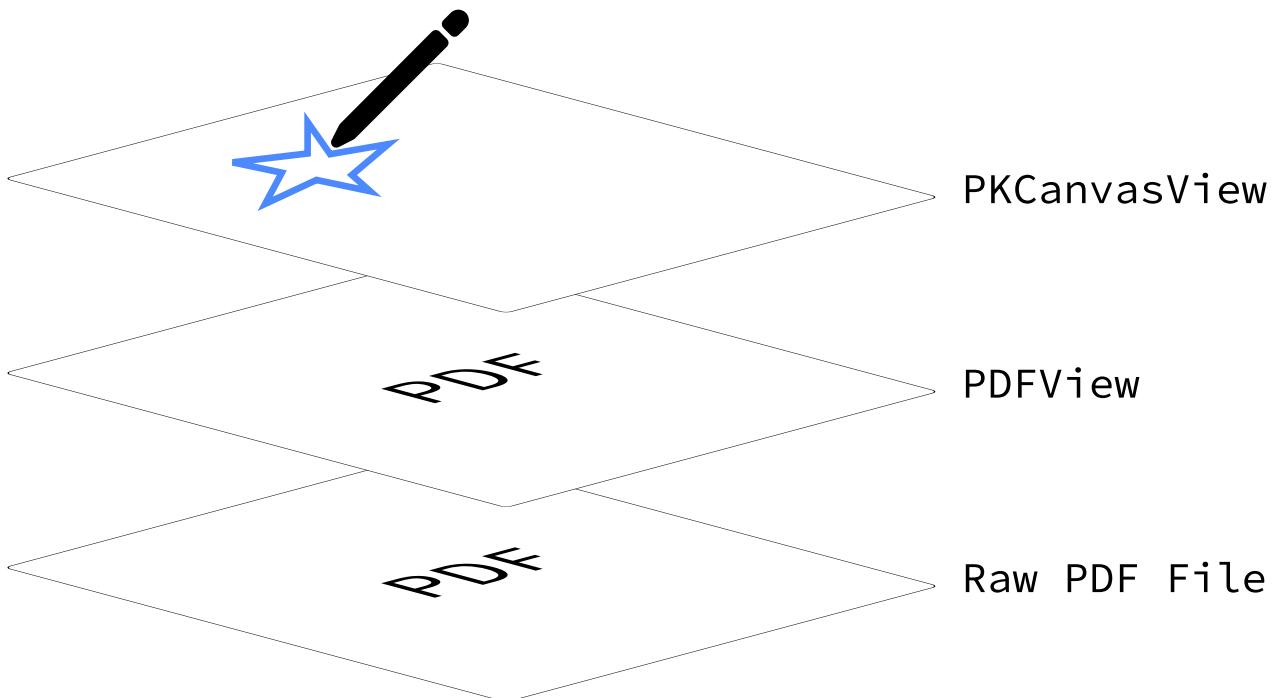
1. PencilKitとは? 
2. PencilKitでアプリを作る 
3. PencilKitをPDFに組み込む 
4. PencilKitのドローイングをPDF注釈として保存する
5. PencilKitを使ってみた感想

↓↓サンプルレポジトリ↓↓

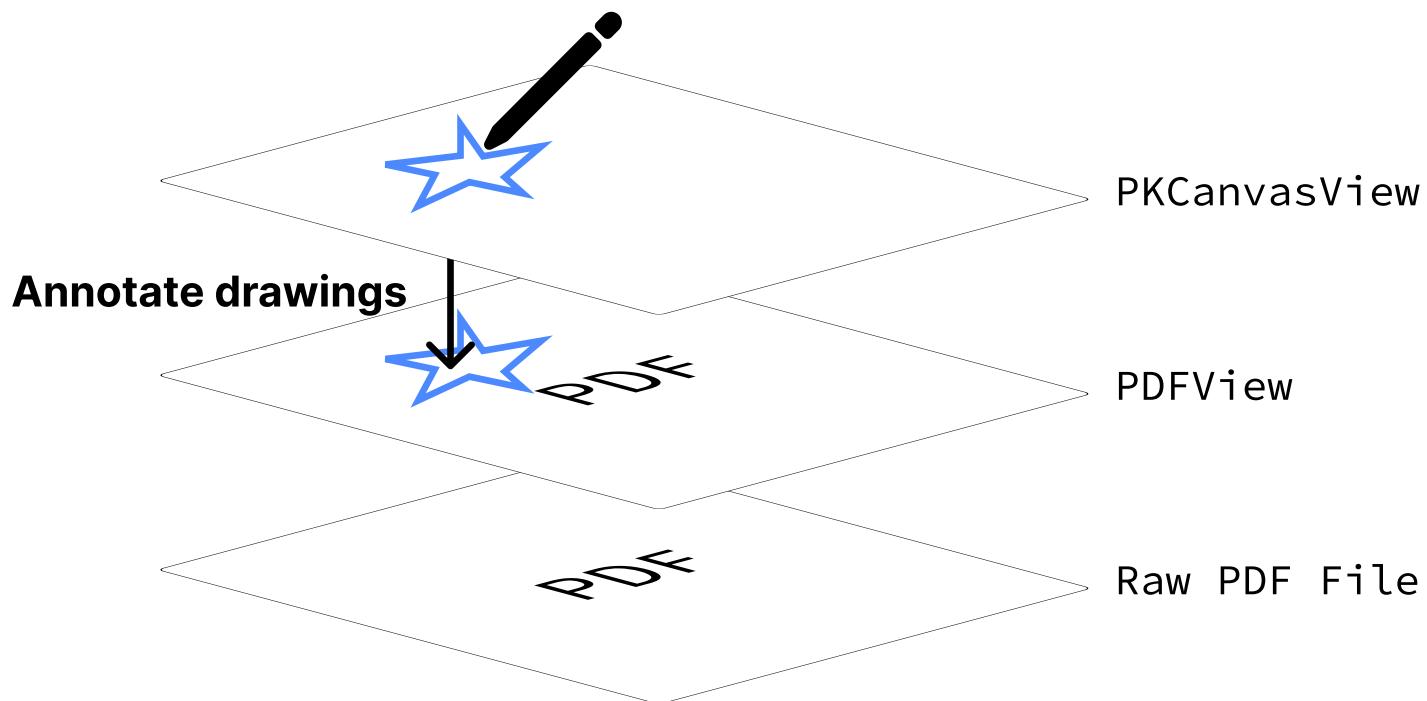
**[github.com/ras0q/iosdc2024](https://github.com/ras0q/iosdc2024)**

**try! PDF Annotation**

# PKCanvasView → PDFView



# PKCanvasView → PDFView



## ドローイングを注釈として追加する

```
class CanvasPDFAnnotation: PDFAnnotation {
    private let drawing: PKDrawing

    init(drawing: PKDrawing, page: PDFPage) {
        self.drawing = drawing

        var pdfBounds = drawing.bounds
        pdfBounds.origin.y = page.bounds(for: .mediaBox).height
            - drawing.bounds.height
            - drawing.bounds.origin.y

        super.init(bounds: pdfBounds, forType: .ink)

        self.page = page
    }
}
```

# ドローイングを注釈として追加する

```
class CanvasPDFAnnotation: PDFAnnotation {
    private let drawing: PKDrawing

    init(drawing: PKDrawing, page: PDFPage) {
        self.drawing = drawing

        var pdfBounds = drawing.bounds
        pdfBounds.origin.y = page.bounds(for: .mediaBox).height
            - drawing.bounds.height
            - drawing.bounds.origin.y

        super.init(bounds: pdfBounds, forType: .ink)

        self.page = page
    }
}
```

# ドローイングを注釈として追加する

```
class CanvasPDFAnnotation: PDFAnnotation {
    private let drawing: PKDrawing

    init(drawing: PKDrawing, page: PDFPage) {
        self.drawing = drawing

        var pdfBounds = drawing.bounds
        pdfBounds.origin.y = page.bounds(for: .mediaBox).height
            - drawing.bounds.height
            - drawing.bounds.origin.y

        super.init(bounds: pdfBounds, forType: .ink)

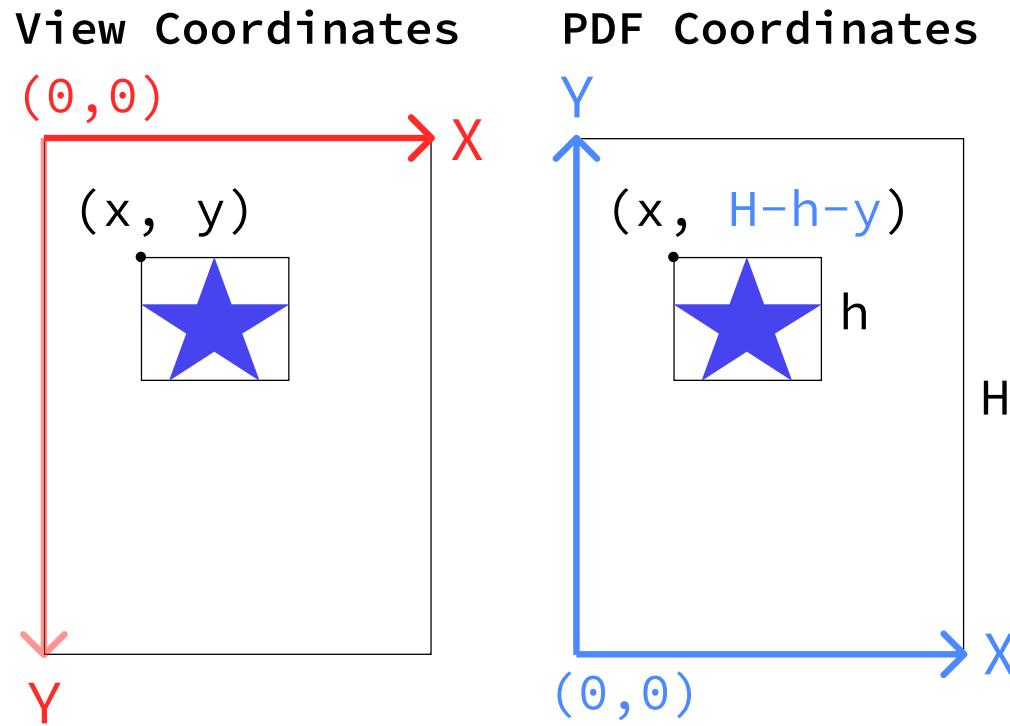
        self.page = page
    }
}
```

これは？

# PDF座標系への変換

座標系をy軸方向に反転させる必要がある

図形が反転するわけではないことに注意



# ドローイングを注釈として追加する

注釈の追加/更新時に `PDFAnnotation#draw()` が呼ばれる

```
class CanvasPDFAnnotation: PDFAnnotation {
    override func draw(with box: PDFDisplayBox, in context: CGContext) {
        super.draw(with: box, in: context)

        UIGraphicsPushContext(context)
        context.saveGState()
        defer {
            context.restoreGState()
            UIGraphicsPopContext()
        }

        let image = drawing.image(from: drawing.bounds, scale: 1.0)
        context.draw(image.cgImage!, in: bounds)
    }
}
```

# ドローイングを注釈として追加する

注釈の追加/更新時に `PDFAnnotation#draw()` が呼ばれる

```
class CanvasPDFAnnotation: PDFAnnotation {
    override func draw(with box: PDFDisplayBox, in context: CGContext) {
        super.draw(with: box, in: context)

        UIGraphicsPushContext(context)
        context.saveGState()
        defer {
            context.restoreGState()
            UIGraphicsPopContext()
        }

        let image = drawing.image(from: drawing.bounds, scale: 1.0)
        context.draw(image.cgImage!, in: bounds)
    }
}
```

# ドローイングを注釈として追加する

注釈の追加/更新時に `PDFAnnotation#draw()` が呼ばれる

```
class CanvasPDFAnnotation: PDFAnnotation {
    override func draw(with box: PDFDisplayBox, in context: CGContext) {
        super.draw(with: box, in: context)

        UIGraphicsPushContext(context)
        context.saveGState()
        defer {
            context.restoreGState()
            UIGraphicsPopContext()
        }

        let image = drawing.image(from: drawing.bounds, scale: 1.0)
        context.draw(image.cgImage!, in: bounds)
    }
}
```

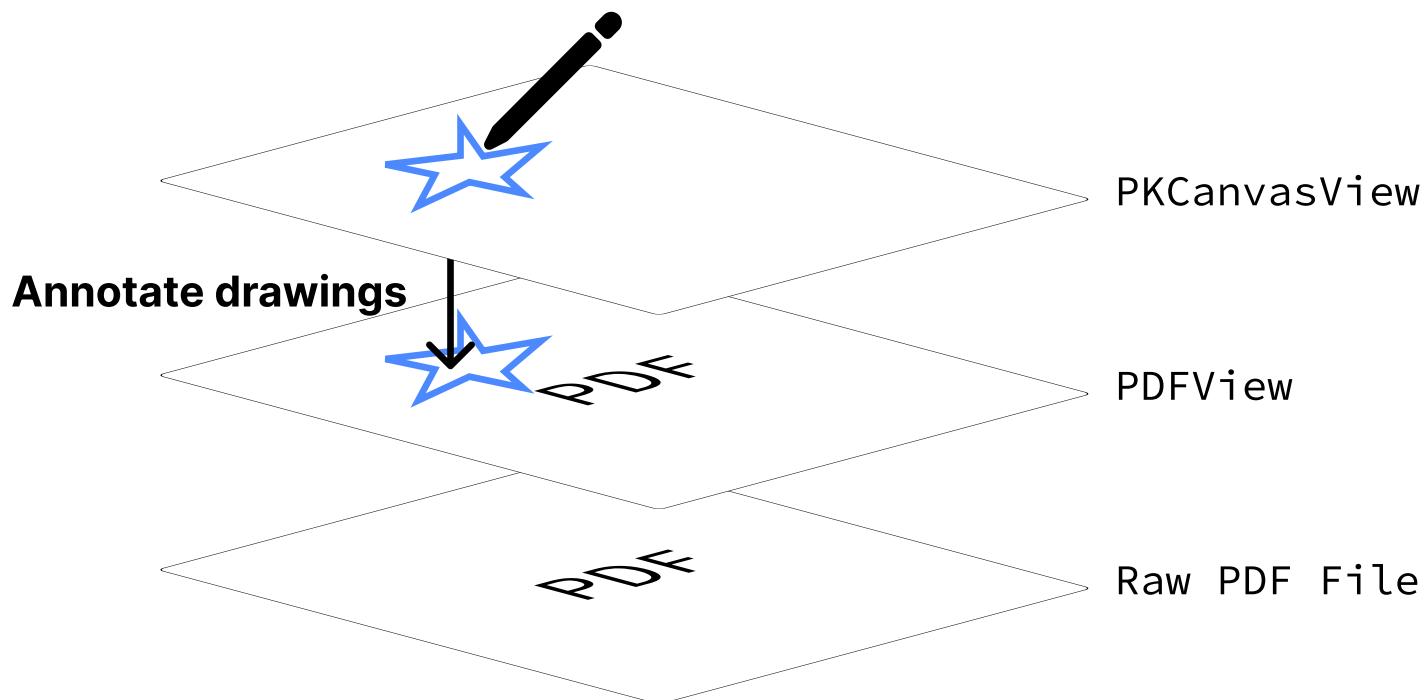
## ドローイングを注釈として追加する

```
let page = pdfView.currentPage
let canvasView = canvasView(for: page)

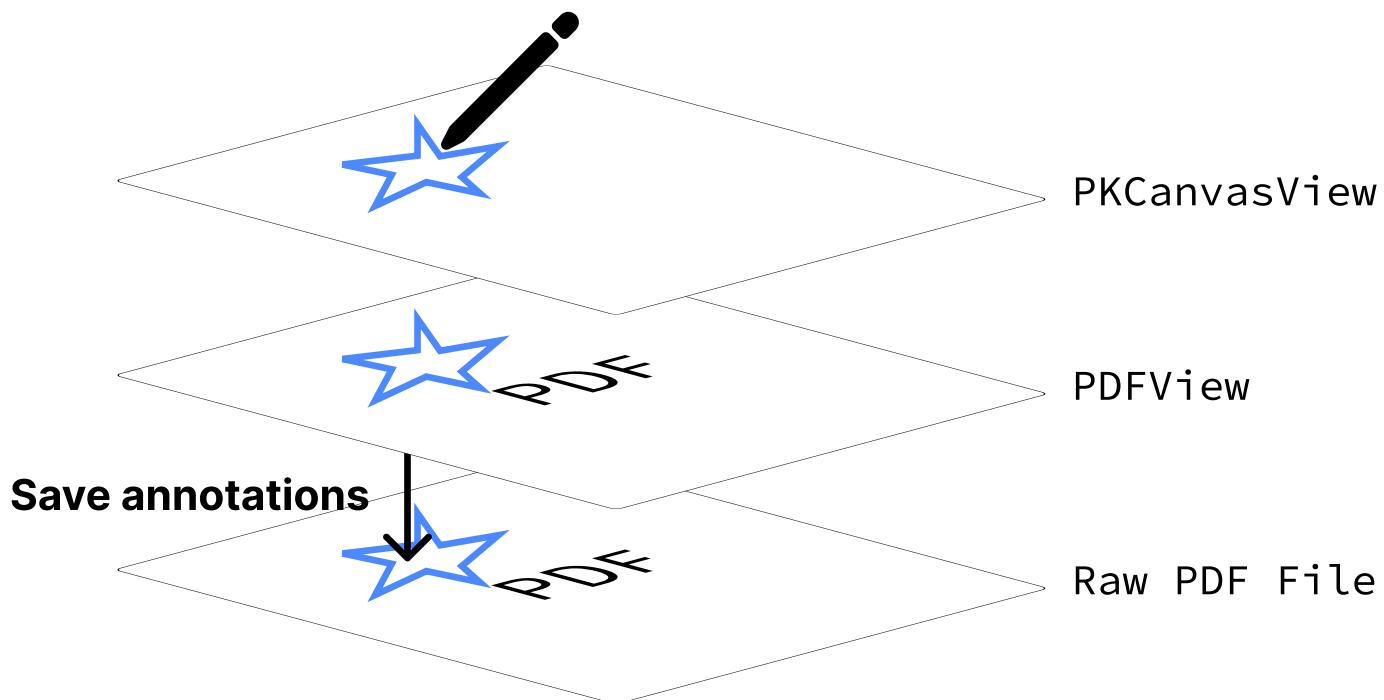
let annotation = CanvasPDFAnnotation(
    drawing: canvasView.drawing,
    page: page
)
page.addAnnotation(annotation)
```

```
for stroke in canvasView.drawing.strokes {
    let annotation = CanvasPDFAnnotation(
        drawing: PKDrawing(strokes: [stroke]),
        page: page
    )
    page.addAnnotation(annotation)
}
```

# PDFView → Raw PDF File



# PDFView → Raw PDF File

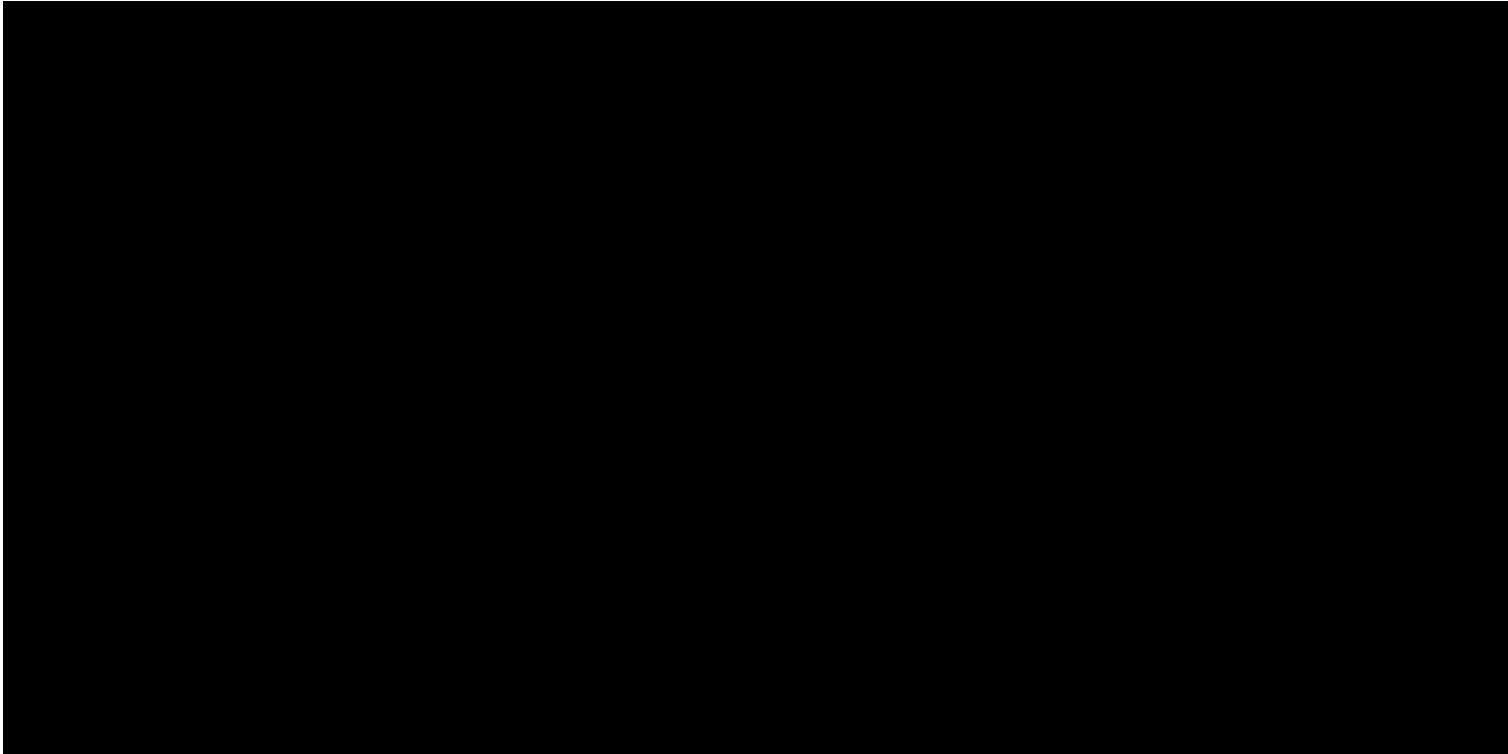


# 注釈をファイルに保存する

PDFDocument#dataRepresentation() から Data を抽出  
ファイルURLを指定し書き込む

```
let data = pdfDocument.dataRepresentation()  
let documentURL = pdfDocument.documentURL  
  
try data.write(to: documentURL)
```

PencilKitのドローイングをPDF注釈として保存する



# Keynote

1. PencilKitとは? 
2. PencilKitでアプリを作る 
3. PencilKitをPDFに組み込む 
4. PencilKitのドローイングをPDF注釈として保存する 
5. PencilKitを使ってみた感想

↓↓サンプルレポジトリ↓↓

**[github.com/ras0q/iosdc2024](https://github.com/ras0q/iosdc2024)**

## PencilKitを使ってみた感想

- Apple Pencilとの連携を数行のコードで組み込めるのはありがたい
- PDFKitとの連携が完全でない
  - 投げ縄ツールが使えない

# ありがとうございました！

サンプルレポジトリ も是非ご覧ください！



Presenter: Ras (@ras0q )

## References

- [PencilKit | Apple Developer Documentation](#)

---
- [PDFKit | Apple Developer Documentation](#)

---
- [What's new in PDFKit - WWDC22 - Videos - Apple Developer](#)

---
- [iOSのPDFKitを利用してPDFを編集する | Zenn](#)

---