PencilKitで実装する PDFへの手書き注釈

iOSDC Japan 2024 Day2 Track B

Ras (@ras0q)

Keynote

- PencilKit の紹介・実装
- Apple Pencil で PDF に注釈したい！
- PencilKit の Good & More

注意事項

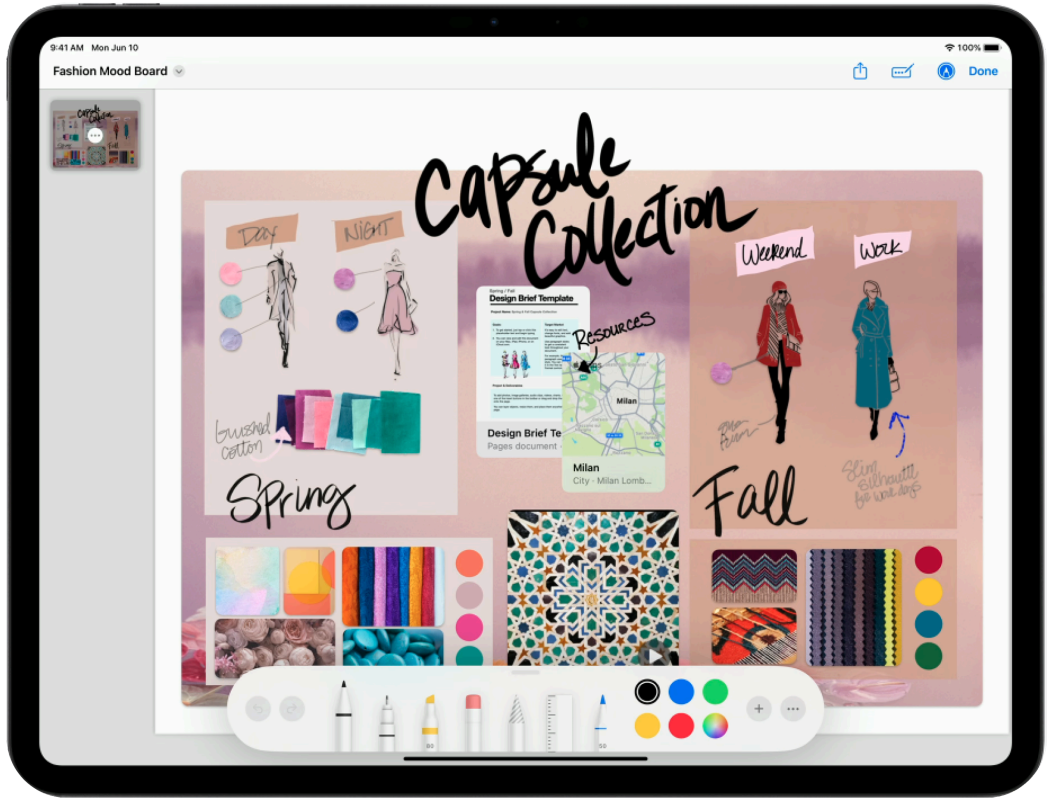
- 紹介するコードは全体の実装の一部です
  - 詳細な実装はGitHubを参照ください → ras0q/iosdc2024

Ras

@ras0q

PencilKit?

PencilKit



引用元: WWDC19

PencilKit

手書き認識をiOSアプリに組み込むことができる純正ライブラリ

- 絵・図形を描くための環境が簡単に揃う
  - ペンの種類が豊富
    - 鉛筆, 万年筆, クレヨン, 定規, …
  - ペンの設定ツールも標準で付属
- 様々な純正アプリに組み込まれている
  - メモ, 写真, ファイル, …
- PencilKitを使ったアプリ感で図形の連携ができる
  - 移動, コピー＆ペースト, …

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)

    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()
    }
}
```

完成！

try! PDF Integration

PDFKit

```
import PDFKit
import UIKit

class ViewController: UIViewController {
    private lazy var pdfDocument = PDFDocument(somePDFURL)
    private lazy var pdfView: PDFView = {
        let view = PDFView()
        view.document = pdfDocument
        return view
    }()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(pdfView)
    }
}
```

WWDC22

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.

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

PDFPageOvelayViewProvider

```
import PencilKit
import PDFKit
import UIKit

class ViewController: UIViewController {
    // ....

    // 各ページに対してCanvasViewを作成
    private lazy var canvasViews: [PKCanvasView] = (0 ..< pdfDocument!.pageCount).map { _ in
        let view = PKCanvasView()
        view.backgroundColor = .clear
        view.clipsToBounds = true
        return view
    }

    // 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)
    }
}
```

PDFPageOvelayViewProvider

```swift
import PencilKit
import PDFKit
import UIKit

class ViewController: UIViewController {
    // ....

    // 各ページに対してCanvasViewを作成
    private lazy var canvasViews: [PKCanvasView] = (0 ..< pdfDocument!.pageCount).map { _ in
        let view = PKCanvasView()
        view.backgroundColor = .clear
        view.clipsToBounds = true
        return view
    }

    // 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)
    }
}
```

PDFPageOvelayViewProvider

```
import PencilKit
import PDFKit
import UIKit

class ViewController: UIViewController {
    // ....

    // 各ページに対してCanvasViewを作成
    private lazy var canvasViews: [PKCanvasView] = (0 ..< pdfDocument!.pageCount).map { _ in
        let view = PKCanvasView()
        view.backgroundColor = .clear
        view.clipsToBounds = true
        return view
    }

    // 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)
    }
}
```

手書き認識の設定

```
// in ViewController ...
private lazy var toolPicker = PKToolPicker()

override func viewDidLoad() {
    super.viewDidLoad()

    view.addSubview(pdfView)

    for canvasView in canvasViews {
        // 各キャンバスにツールを設定する
        toolPicker.addObserver(canvasView)

        // 各キャンバスの手書き認識をPDFViewに登録する
        pdfView.addGestureRecognizer(canvasView.drawingGestureRecognizer)
    }

    // PKCanvasViewはoverlayしているだけなので、first responderはPDFViewになる
    toolPicker.setVisible(true, forFirstResponder: pdfView)
    pdfView.becomeFirstResponder()
}
```

手書き認識の設定

```
// in ViewController ...
private lazy var toolPicker = PKToolPicker()

override func viewDidLoad() {
    super.viewDidLoad()

    view.addSubview(pdfView)

    for canvasView in canvasViews {
        // 各キャンバスにツールを設定する
        toolPicker.addObserver(canvasView)

        // 各キャンバスの手書き認識をPDFViewに登録する
        pdfView.addGestureRecognizer(canvasView.drawingGestureRecognizer)
    }

    // PKCanvasViewはoverlayしているだけなので、first responderはPDFViewになる
    toolPicker.setVisible(true, forFirstResponder: pdfView)
    pdfView.becomeFirstResponder()
}
```

手書き認識の設定

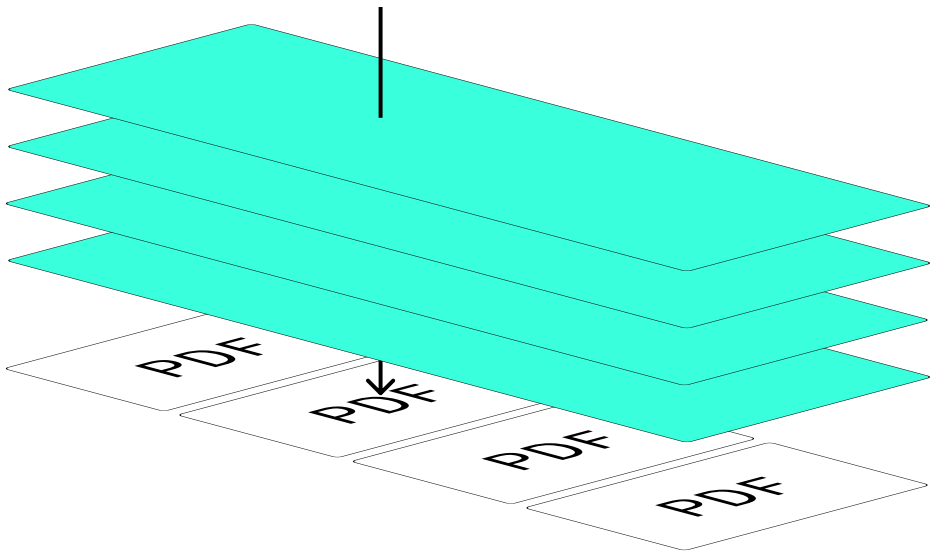
```swift
// in ViewController...
private lazy var toolPicker = PKToolPicker()

override func viewDidLoad() {
    super.viewDidLoad()

    view.addSubview(pdfView)

    for canvasView in canvasViews {
        // 各キャンバスにツールを設定する
        toolPicker.addObserver(canvasView)

        // 各キャンバスの手書き認識をPDFViewに登録する
        pdfView.addGestureRecognizer(canvasView.drawingGestureRecognizer)
    }

    // PKCanvasViewはoverlayしているだけなので、first responderはPDFViewになる
    toolPicker.setVisible(true, forFirstResponder: pdfView)
    pdfView.becomeFirstResponder()
}
```

手書き認識の設定

```swift
// in ViewController ...
private lazy var toolPicker = PKToolPicker()

override func viewDidLoad() {
    super.viewDidLoad()

    view.addSubview(pdfView)

    for canvasView in canvasViews {
        // 各キャンバスにツールを設定する
        toolPicker.addObserver(canvasView)

        // 各キャンバスの手書き認識をPDFViewに登録する
        pdfView.addGestureRecognizer(canvasView.drawingGestureRecognizer)
    }

    // PKCanvasViewはoverlayしているだけなので、first responderはPDFViewになる
    toolPicker.setVisible(true, forFirstResponder: pdfView)
    pdfView.becomeFirstResponder()
}
```

複数の手書き認識を設定した結果...

どのページにも最後に追加した`drawingGestureRecognizer`のみが発火してしまう



canvasViews[3].drawingGestureRecognizer

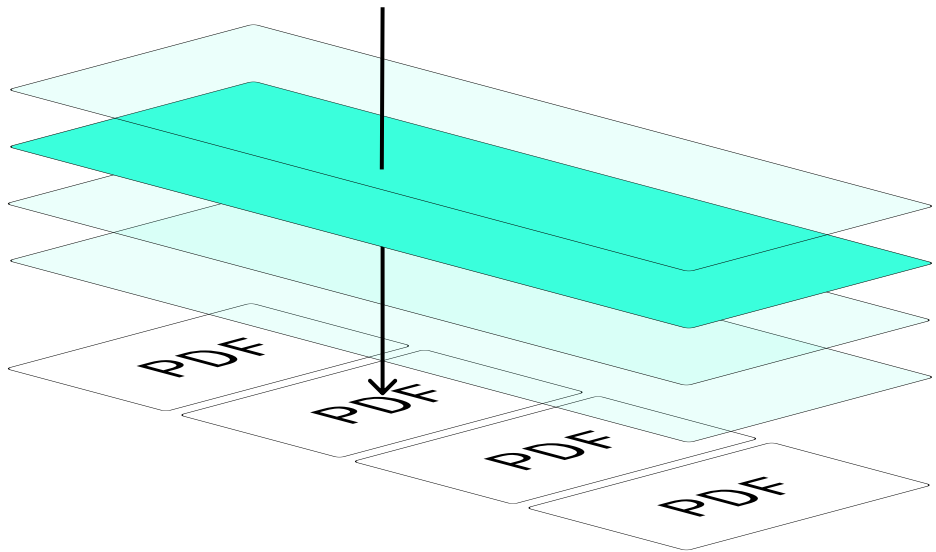
canvasViews[2].drawingGestureRecognizer

canvasViews[1].drawingGestureRecognizer

canvasViews[0].drawingGestureRecognizer

複数の手書き認識を設定した結果...

タップ時に該当ページの `drawingGestureRecognizer` 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()

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

```swift
protocol CanvasPDFViewDelegate: AnyObject {
    func switchActivePage(to page: PDFPage)
}

class CanvasPDFView: PDFView {
    var interactionDelegate: (any CanvasPDFViewDelegate)?

    // ✅
    override func hitTest(_ point: CGPoint, with event: UIEvent?) -> UIView? {
        if let activePage = page(for: point, nearest: true) {
            // 移譲先でdrawingGestureRecognizer.isEnabled を切り替える
            interactionDelegate?.switchActivePage(to: activePage)
        }

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

try! PDF Annotation

おさらい

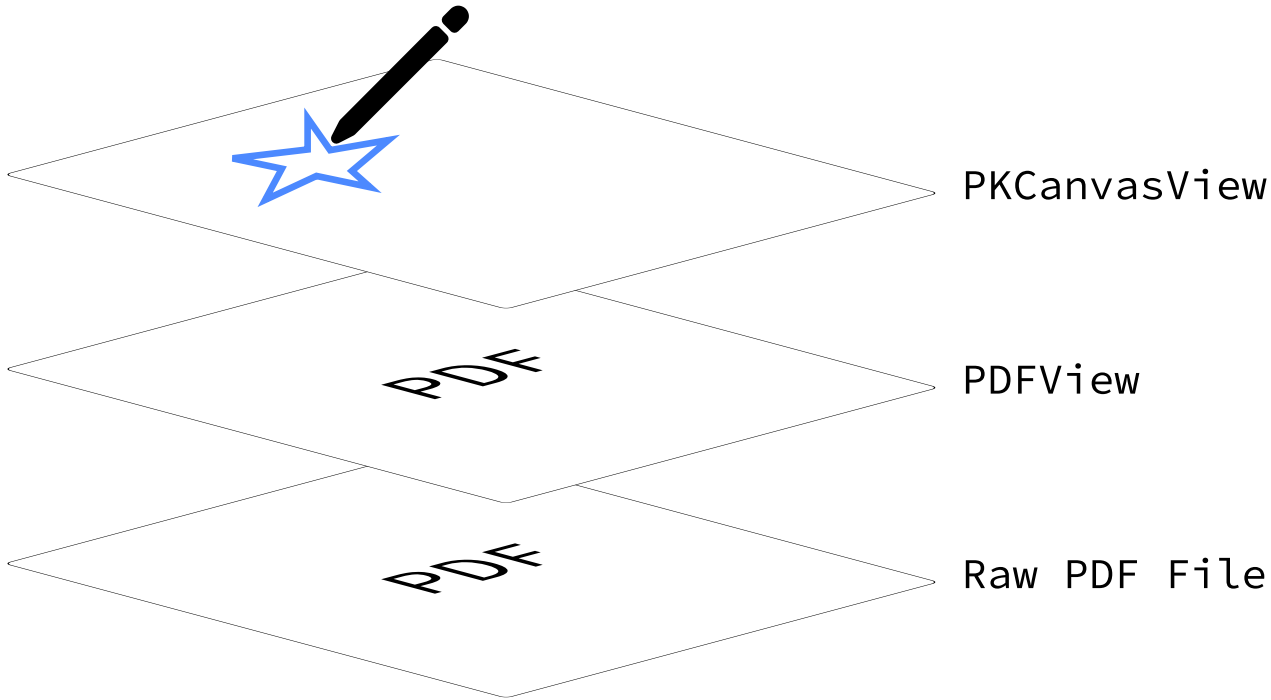
今できること

- PencilKitを使ってキャンバス画面に図形を描画する
- PDF上にキャンバスを重ねて各ページに図形を描画する

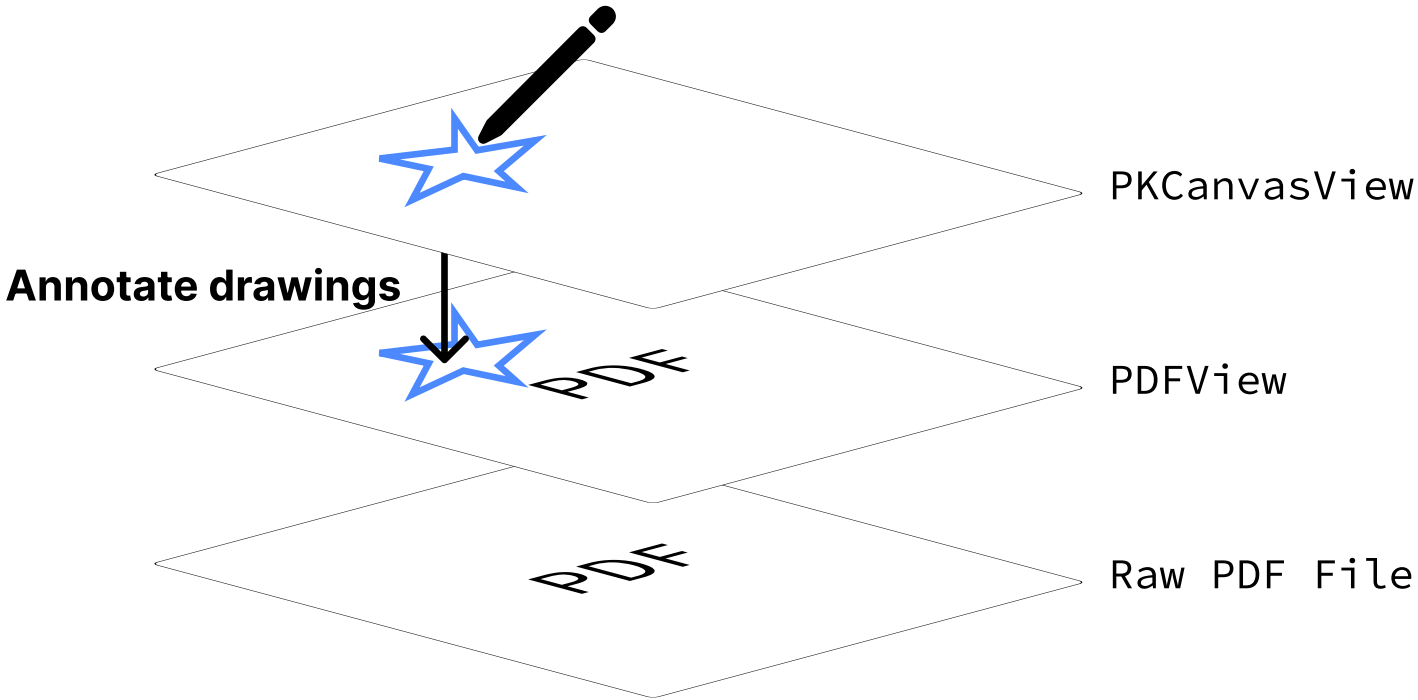
これから行うこと

- 描画した図形を注釈としてPDFに反映させる
- 注釈を反映したPDFを保存する

PKCanvasView → PDFView



PKCanvasView → PDFView



PKCanvasView → PDFView

`PDFView`への注釈は`PDFAnnotation`を使う

`PKCanvasView`の描画は`canvasView.drawing: PKDrawing`から取得できる

```
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, withProperties: nil)

        self.page = page
    }
}
```

PKCanvasView → PDFView

`PDFView`への注釈は`PDFAnnotation`を使う

`PKCanvasView`の描画は`canvasView.drawing: PKDrawing`から取得できる

```
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, withProperties: nil)

        self.page = page
    }
}
```

PKCanvasView → PDFView

`PDFView`への注釈は`PDFAnnotation`を使う

`PKCanvasView`の描画は`canvasView.drawing: PKDrawing`から取得できる

```
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, withProperties: nil)

        self.page = page
    }
}
```

これは？

座標系の変換

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

PDF coordinates

View coordinates

引用元: iOSのPDFKitを利用してPDFを編集する | Zenn

PKCanvasView → PDFView

```
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()
        }

        // Y-flip (M' = Scale * Translate * M)
        let pageHeight = page!.bounds(for: box).height
        context.translateBy(x: 0, y: pageHeight)
        context.scaleBy(x: 1.0, y: -1.0)

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

PKCanvasView → PDFView

```
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()
        }

        // Y-flip (M' = Scale * Translate * M)
        let pageHeight = page!.bounds(for: box).height
        context.translateBy(x: 0, y: pageHeight)
        context.scaleBy(x: 1.0, y: -1.0)

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

PKCanvasView → PDFView

```
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()
        }

        // Y-flip (M' = Scale * Translate * M)
        let pageHeight = page!.bounds(for: box).height
        context.translateBy(x: 0, y: pageHeight)
        context.scaleBy(x: 1.0, y: -1.0)

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

PKCanvasView → PDFView

```
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()
        }

        // Y-flip (M' = Scale * Translate * M)
        let pageHeight = page!.bounds(for: box).height
        context.translateBy(x: 0, y: pageHeight)
        context.scaleBy(x: 1.0, y: -1.0)

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

PKCanvasView → PDFView

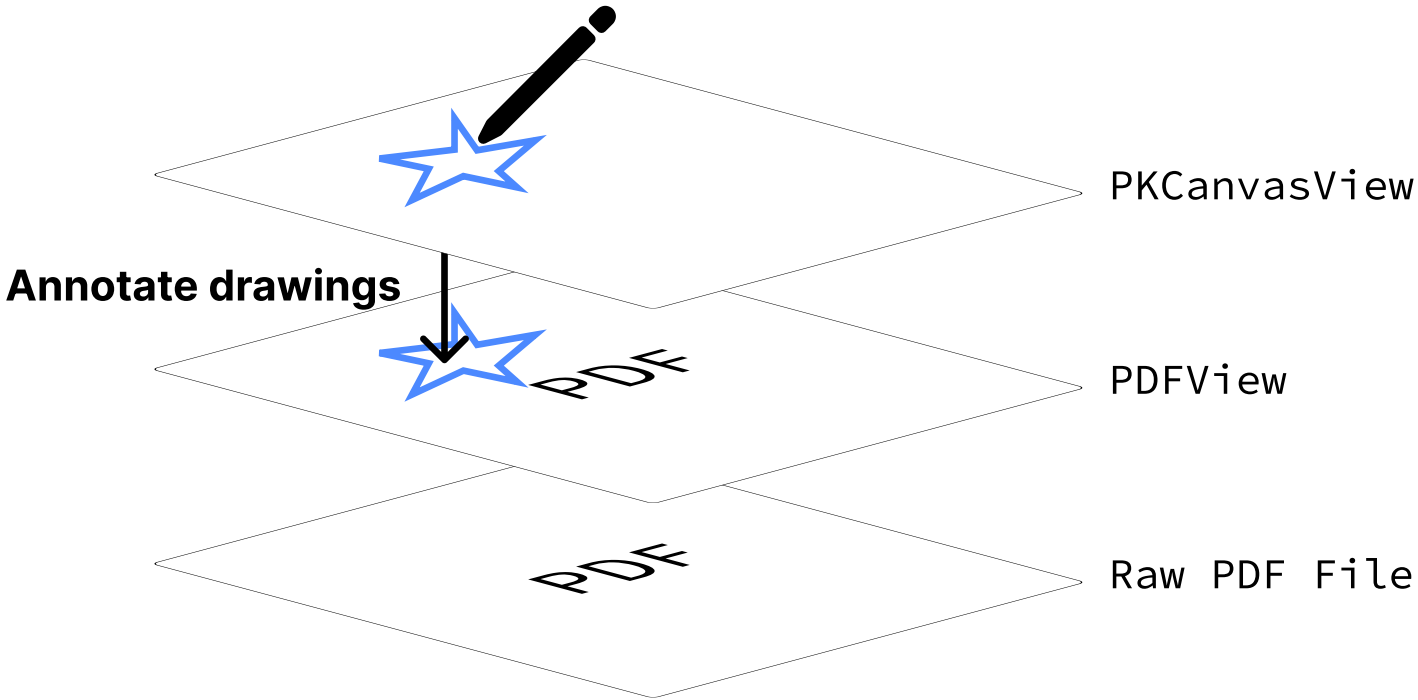
```
class CanvasPDFAnnotation: PDFAnnotation {
    override func draw(with box: PDFDisplayBox, in context: CGContext) {
        super.draw(with: box, in: context)

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

        // Y-flip (M' = Scale * Translate * M)
        let pageHeight = page!.bounds(for: box).height
        context.translateBy(x: 0, y: pageHeight)
        context.scaleBy(x: 1.0, y: -1.0)

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

使用例

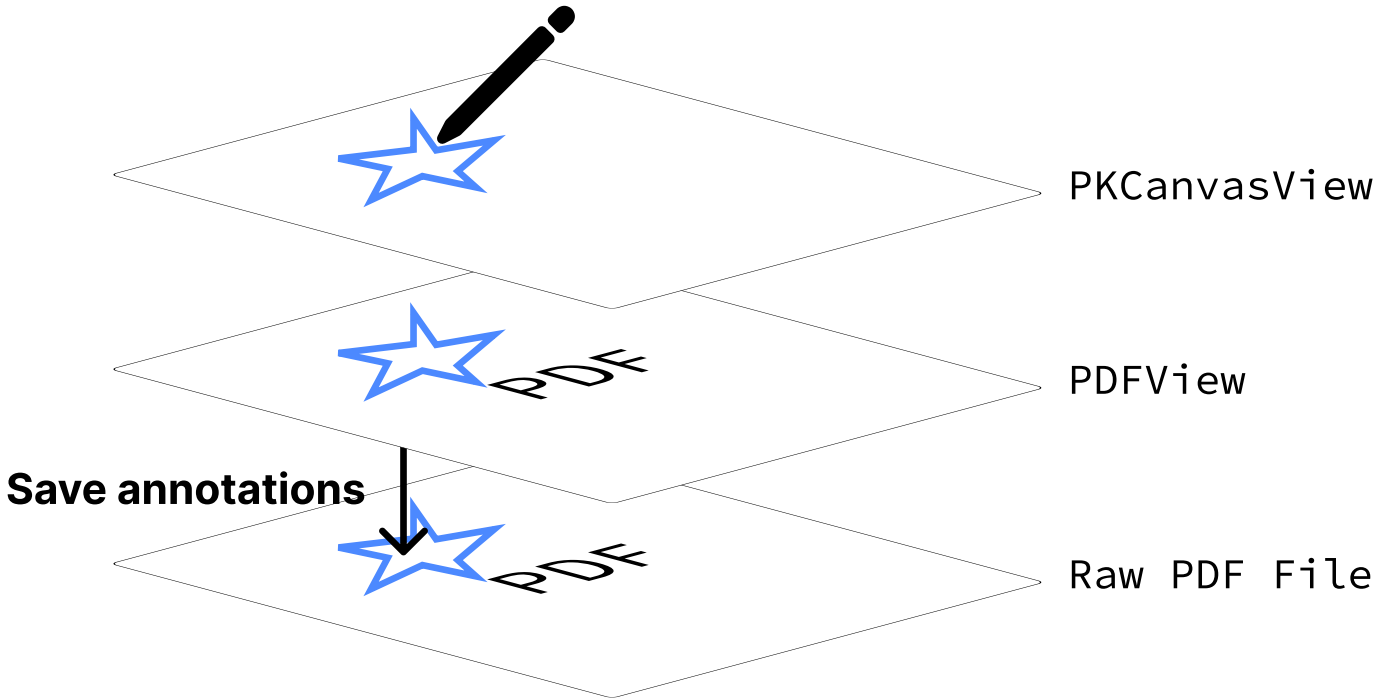
```
guard let page = pdfView.currentPage, let canvasView = canvasView(for: page) else {
    return
}

// キャンバスへの描画を1つの注釈としているため更新時にリセットが必要
let existingAnnotations = page.annotations.filter { $0 is CanvasPDFAnnotation }
for annotation in existingAnnotations {
    page.removeAnnotation(annotation)
}

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

✅ 描画した図形を注釈としてPDFに反映させる

PDFView → Raw PDF File



PDFView → Raw PDF File



PDFView → Raw PDF File

```
let data = pdfDocument.dataRepresentation()
let documentURL = pdfDocument.documentURL

try data.write(to: documentURL)
```

✅ 注釈を反映したPDFを保存する

TODO: 保存機能を実装してデモを貼る

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 𝕏)

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