Combineを利用した SwiftUI・UIKitのどちらにも対応する Unidirectionalな設計を実現するには

```
//
// CA.swift #10
//
// Talked by @marty_suzuki on 2019/09/26
//
```

marty_suzuki

marty-suzuki

Taiki Suzuki

インターネットテレビ局「AbemaTV」を担当するiOSエンジニア。2014年サイバーエージェント新卒入社。
コミュニティサービスでサーバーサイドを担当した後、iOSエンジニアに転向しファッション通販サイト「VILECT」の立ち上げ・運営に従事。その後新感覚SNS「755」での開発を経て、2017年3月より現職。

Overview

Popular repositories

SAHistoryNavigationViewController
SAHistoryNavigationViewController realizes iOS task manager like UI in UINavigationController.
Swift ★ 1,559

ReverseExtension
A UITableView extension that enables cell insertion from the bottom of a table view.
Swift ★ 1,446

SABlurImageView
You can use blur effect and it's animation easily to call only two methods.
Swift ★ 523

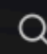
URLEmbeddedView
URLEmbeddedView automatically caches the object that is confirmed the Open Graph Protocol.
Swift ★ 549

アジェンダ

1. **まずはじめに**
2. import UIKit
3. import UIKit + Ricemill
4. import SwiftUI
5. import SwiftUI + Ricemill

まずはじめに

MVVMの実装を縛るFrameworkを開発・導入し、チー...



Edit deck

MVVMの実装を縛るFrameworkを開発・導入し
チームでばらつきがあった実装を統一する

iOSDC Japan 2019 Reject Conference days1
@marty_suzuki

1 / 127 ページ

https://bit.ly/2m3qk0S

「MVVMの実装を縛るFrameworkを開発・導入し
チームでばらつきがあった実装を統一する」

RxSwiftでUnidirectionalな設計を実現する

https://github.com/cats-oss/Unio

Combineを利用したSwiftUI・UIKitのどちらにも対応するUnidirectionalな設計を実現する

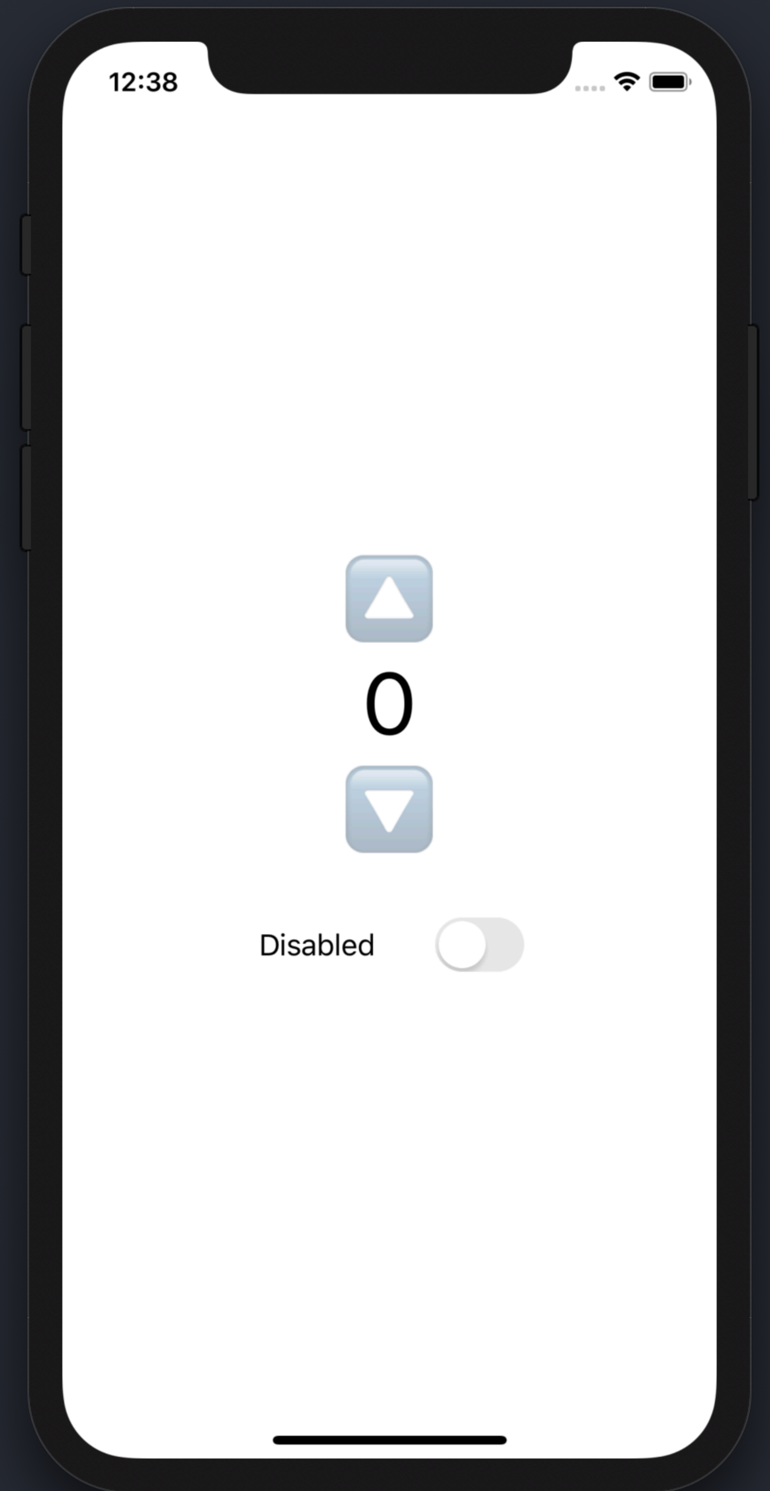
```
import Ricemill
```

https://github.com/marty-suzuki/Ricemill

```
import UIKit
```

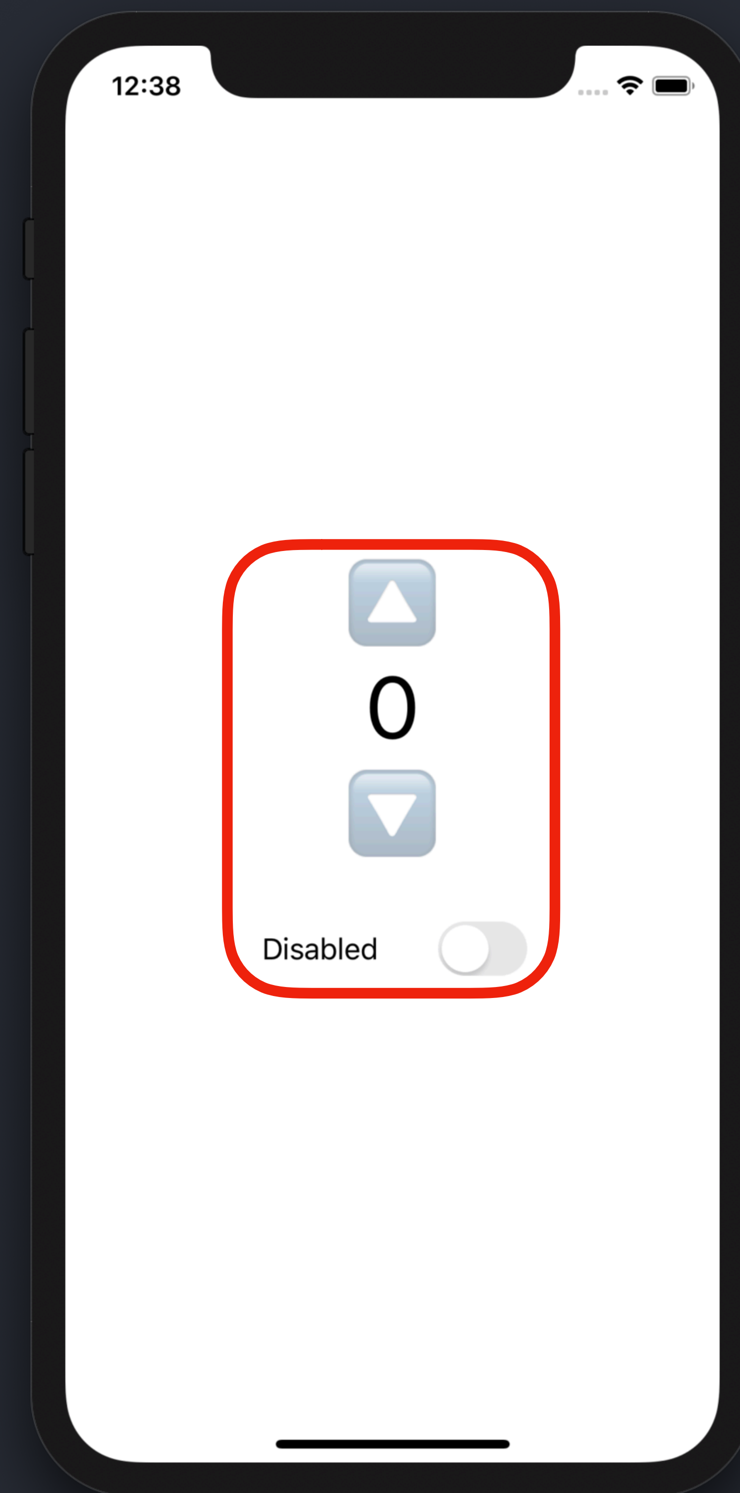
ViewControllerの定義

```
final class CounterViewController: UIViewController {

    let counterToggle: UISwitch
    let incrementButton: UIButton
    let decrementButton: UIButton
    let countLabel: UILabel
    let counterStateLabel: UILabel

    private var cancellables: [AnyCancellable] = []
    private let viewModel = ViewModel()

    ...
}
```



ViewControllerの定義

```
final class CounterViewController: UIViewController {

    let counterToggle: UISwitch
    let incrementButton: UIButton
    let decrementButton: UIButton
    let countLabel: UILabel
    let counterStateLabel: UILabel

    private var cancellables: [AnyCancellable] = []
    private let viewModel = ViewModel()

    ...
}
```



ViewControllerからの入力

```
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        incrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.increment)

        decrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.decrement)

        counterToggle.valueChanged
            .subscribe(viewModel.isOn)

        ...

    }
}
```

ViewControllerからの入力

```swift
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        incrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.increment)

        decrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.decrement)

        counterToggle.valueChanged
            .subscribe(viewModel.isOn)

        ...

    }
}
```

🔼のタップイベントをViewModelに伝える

ViewControllerからの入力

```swift
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        incrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.increment)

        decrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.decrement)

        counterToggle.valueChanged
            .subscribe(viewModel.i

        ...

    }
}
```

🔽のタップイベントをViewModelに伝える

ViewControllerからの入力

```swift
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        incrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.increment)

        decrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.decrement)

        counterToggle.valueChanged
            .subscribe(viewModel.isOn)

        ...

    }
}
```

UISwitchの値の変化をViewModelに伝える

ViewControllerへの反映

```swift
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        viewModel.count
            .assign(to: \.text, on: countLabel)
            .store(in: &cancellables)

        viewModel.isIncrementEnabled
            .assign(to: \.isEnabled, on: incrementButton)
            .store(in: &cancellables)

        viewModel.isDecrementEnabled
            .assign(to: \.isEnabled, on: decrementButton)
            .store(in: &cancellables)
    }
}
```

ViewControllerへの反映

```
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        viewModel.count
            .assign(to: \.text, on: countLabel)
            .store(in: &cancellables)

        viewModel.isIncrementEnabled
            .assign(to: \.isEnabled, on: in
            .store(in: &cancellables)

        viewModel.isDecrementEnabled
            .assign(to: \.isEnabled, on: decrementButton)
            .store(in: &cancellables)
    }
}
```

カウントをUILabelに反映

ViewControllerへの反映

```
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        viewModel.count
            .assign(to: \.text, on: countLabel)
            .store(in: &cancellables)

        viewModel.isIncrementEnabled
            .assign(to: \.isEnabled, on: incrementButton)
            .store(in: &cancellables)

        viewModel.isDecrementEnabled
            .assign(to: \.isEnabled, on: de
            .store(in: &cancellables)
    }
}
```

ViewControllerへの反映

```
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        viewModel.count
            .assign(to: \.text, on: countLabel)
            .store(in: &cancellables)

        viewModel.isIncrementEnabled
            .assign(to: \.isEnabled, on: incrementButton)
            .store(in: &cancellables)

        viewModel.isDecrementEnabled
            .assign(to: \.isEnabled, on: decrementButton)
            .store(in: &cancellables)
    }
}
```

🔽が有効かどうかを反映

ViewModelの定義

```
final class ViewModel {

    let increment: Subscribers.Sink<Void, Never>
    let decrement: Subscribers.Sink<Void, Never>
    let isOn: Subscribers.Sink<Bool, Never>

    var count: AnyPublisher<String?, Never> {
        $_count.map { Optional.some(String($0)) }
            .eraseToAnyPublisher()
    }
    var isIncrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.eraseToAnyPublisher()
    }
    var isDecrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.combineLatest($_count) { $0 && $1 > 0  }
            .eraseToAnyPublisher()
    }

    @Published private var _count: Int = 0
    @Published private var _isToggleEnabled = false

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

ViewModelの定義

```swift
final class ViewModel {

    let increment: Subscribers.Sink<Void, Never>
    let decrement: Subscribers.Sink<Void, Never>
    let isOn: Subscribers.Sink<Bool, Never>

    var count: AnyPublisher<String
        $_count.map { Optional.some
            .eraseToAnyPublisher()
    }
    var isIncrementEnabled: AnyPubl
        $_isToggleEnabled.eraseToAnyPublisher()
    }
    var isDecrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.combineLatest($_count) { $0 && $1 > 0  }
            .eraseToAnyPublisher()
    }

    @Published private var _count: Int = 0
    @Published private var _isToggleEnabled = false

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

外部からの入力:
イベントを受け付けることだけできれば良い
ので、今回の場合はSubscribers.Sinkで定義

ViewModelの定義

```
final class ViewModel {

    let increment: Subscribers.Sink<Void, Never>
    let decrement: Subscribers.Sink<Void, Never>
    let isOn: Subscribers.Sink<Bool, Never>

    var count: AnyPublisher<String?, Never> {
        $_count.map { Optional.some(String($0)) }
            .eraseToAnyPublisher()
    }
    var isIncrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.eraseToAnyPublisher()
    }
    var isDecrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.combineLatest($_count) { $0 && $1 > 0  }
            .eraseToAnyPublisher()
    }

    @Published private var _count: I   = 0
    @Published private var _isToggle

    private var cancellables: [AnyCan

    init() { ... }
}
```

外部への出力:
イベントを出力することだけできれば良いので、今回の場合はAnyPublisherで定義

ViewModelの定義

```
final class ViewModel {

    let increment: Subscribers.Sink<Void, Never>
    let decrement: Subscribers.Sink<Void, Never>
    let isOn: Subscribers.Sink<Bool, Never>

    var count: AnyPublisher<String?, Never> {
        $_count.map { Optional.some(String($0)) }
            .eraseToAnyPublisher()
    }
    var isIncrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.eraseToAnyPubl
    }
    var isDecrementEnabled: AnyPublisher
        $_isToggleEnabled.combineLatest
            .eraseToAnyPublisher()
    }

    @Published private var _count: Int = 0
    @Published private var _isToggleEnabled = false

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

内部状態:
保持した状態が変化したことも検知したいので、今回の場合は@Publishedで定義

ViewModelのInitializerの実装

```swift
final class ViewModel {

    ...

    init() {
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()
        let _isOn = PassthroughSubject<Bool, Never>()

        self.increment = .init(receiveCompletion: { _increment.send(completion: $0) },
                               receiveValue: { _increment.send($0) })
        self.decrement = .init(receiveCompletion: { _decrement.send(completion: $0) },
                               receiveValue: { _decrement.send($0) })
        self.isOn = .init(receiveCompletion: { _isOn.send(completion: $0) },
                          receiveValue: { _isOn.send($0) })

        _isOn.assign(to: \._isToggleEnabled, on: self).store(in: &cancellables)

        let increment = _increment.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \._count, on: self).store(in: &cancellables)
    }
}
```

ViewModelのInitializerの実装

```
final class ViewModel {

    ...

    init() {
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()
        let _isOn = PassthroughSubject<Bool, Never>()

        self.increment = .init(receiveCompletion: { _increment.send(completion: $0) },
                               receiveValue: { _i
        self.decrement = .init(receiveCompletion:
                               receiveValue: { _de
        self.isOn = .init(receiveCompletion: { _is
                          receiveValue: { _isOn.send($0) })

        _isOn.assign(to: \._isToggleEnabled, on: self).store(in: &cancellables)

        let increment = _increment.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \._count, on: self).store(in: &cancellables)
    }
}
```

入力を受け取ってInitalizer内でイベントを
リレーするためのPassthroughSubject

ViewModelのInitializerの実装

```swift
final class ViewModel {

    ...

    init() {
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()
        let _isOn = PassthroughSubject<Bool, Never>()

        self.increment = .init(receiveCompletion: { _increment.send(completion: $0) },
                               receiveValue: { _increment.send($0) })
        self.decrement = .init(receiveCompletion: { _decrement.send(completion: $0) },
                               receiveValue: { _decrement.send($0) })
        self.isOn = .init(receiveCompletion: { _isOn.send(completion: $0) },
                          receiveValue: { _isOn.send($0) })

        _isOn.assign(to: \._isToggleEnabled, on: self).store(in: &cancellables)

        let increment = _increment.flatMap { [weak
                self.map { Just($0._count).eraseTo
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \._count, on: self).store(in: &cancellables)
    }
}
```

入力のイベントをPassthroughSubjectに繋げる

ViewModelのInitializerの実装

```swift
final class ViewModel {

    ...

    init() {
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()
        let _isOn = PassthroughSubject<Bool, Never>()

        self.increment = .init(receiveCompletion: { _increment.send(completion: $0) },
                               receiveValue: { _increment.send($0) })
        self.decrement = .init(receiveComplet
                               receiveValue:
        self.isOn = .init(receiveCompletion
                          receiveValue: { _isOn.send($0) })

        _isOn.assign(to: \._isToggleEnabled, on: self).store(in: &cancellables)

        let increment = _increment.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \._count, on: self).store(in: &cancellables)
    }
}
```

_isOnからのイベントをもとに内部状態を更新

ViewModelのInitializerの実装

```
final class ViewModel {

    ...

    init() {
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()
        let _isOn = PassthroughSubject<Bool, Never>()

        self.increment = .init(receiveCompletion: { _increment.send(completion: $0) },
                               receiveValue: { _increment.send($0) })
        self.decrement = .init(receiveCompletion: { _decrement.send(completion: $0) },
                               receiveValue:
        self.isOn = .init(receiveCompletion:
                          receiveValue: { _i

        _isOn.assign(to: \._isToggleEnabled  on: self).store(in: &cancellables)

        let increment = _increment.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \._count, on: self).store(in: &cancellables)
    }
}
```

_incrementからのイベントをトリガーに
内部状態の_countに対して+1した値を流す

ViewModelのInitializerの実装

```swift
final class ViewModel {

    ...

    init() {
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()
        let _isOn = PassthroughSubject<Bool, Never>()

        self.increment = .init(receiveCompletion: { _increment.send(completion: $0) },
                               receiveValue: { _increment.send($0) })
        self.decrement = .init(receiveCompletion: { _decrement.send(completion: $0) },
                               receiveValue: { _decrement.send($0) })
        self.isOn = .init(receiveCompletion: { _isOn.send(completion: $0) },
                          receiveValue: { _isOn.send($0) })

        _isOn.assign(to: \._isToggleEnabled, on

        let increment = _increment.flatMap { [w
                self.map { Just($0._count).eras
                    Empty().eraseToAnyPublisher
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \._count, on: self).store(in: &cancellables)
    }
}
```

_decrementからのイベントをトリガーに
内部状態の_countに対して-1した値を0より
大きい値にして流す

ViewModelのInitializerの実装

```
final class ViewModel {

    ...

    init() {
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()
        let _isOn = PassthroughSubject<Bool, Never>()

        self.increment = .init(receiveCompletion: { _increment.send(completion: $0) },
                               receiveValue: { _increment.send($0) })
        self.decrement = .init(receiveCompletion: { _decrement.send(completion: $0) },
                               receiveValue: { _decrement.send($0) })
        self.isOn = .init(receiveCompletion: { _isOn.send(completion: $0) },
                          receiveValue: { _isOn.send($0) })

        _isOn.assign(to: \._isToggleEnabled, on: self).store(in: &cancellables)

        let increment = _increment.flatMap { [weak self] _ in
                self.map { Just($0._count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [we
                self.map { Just($0._count).erase
                    Empty().eraseToAnyPublisher(
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \._count, on: self).store(in: &cancellables)
    }
}
```

incrementとdecrementのイベントを
もとに内部状態を更新

ViewModelのInitializerに収まらなかった実装

```
final class ViewModel {

    let increment: Subscribers.Sink<Void, Never>
    let decrement: Subscribers.Sink<Void, Never>
    let isOn: Subscribers.Sink<Bool, Never>

    var count: AnyPublisher<String?, Never> {
        $_count.map { Optional.some(String($0)) }
            .eraseToAnyPublisher()
    }
    var isIncrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.eraseToAnyPublisher()
    }
    var isDecrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.combineLatest($_count) { $0 && $1 > 0  }
            .eraseToAnyPublisher()
    }

    @Published private var _count: Int = 0
    @Published private var _i

    private var cancellables:

    init() { ... }
}
```

Stored Propertyで定義したかったが
PropertyWrapperを利用しているPropertyに
アクセスするためには、すべてのPropertyの初期化
が完了している必要があるため、Computedで定義

ViewModelのInitializerに収まらなかった実装

```swift
final class ViewModel {

    let increment: Subscribers.Sink<Void, Never>
    let decrement: Subscribers.Sink<Void, Never>
    let isOn: Subscribers.Sink<Bool, Never>

    var count: AnyPublisher<String?, Never> {
        $_count.map { Optional.some(String($0)) }
            .eraseToAnyPublisher()
    }
    var isIncrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.eraseToAnyPublisher()
    }
    var isDecrementEnabled: AnyPublisher<Bool, N
        $_isToggleEnabled.combineLatest($_count)
            .eraseToAnyPublisher()
    }

    @Published private var _count: Int = 0
    @Published private var _isToggleEnabled = false

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

内部状態の_countが変更されたら
String?に変換して出力

ViewModelのInitializerに収まらなかった実装

```swift
final class ViewModel {

    let increment: Subscribers.Sink<Void, Never>
    let decrement: Subscribers.Sink<Void, Never>
    let isOn: Subscribers.Sink<Bool, Never>

    var count: AnyPublisher<String?, Never> {
        $_count.map { Optional.some(String($0)) }
            .eraseToAnyPublisher()
    }
    var isIncrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.eraseToAnyPublisher()
    }
    var isDecrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.co
            .eraseToAnyPubli
    }

    @Published private var _count: Int = 0
    @Published private var _isToggleEnabled = false

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

内部状態の_isToggleEnabledが変更されたら
そのイベントを外部に出力

ViewModelのInitializerに収まらなかった実装

```
final class ViewModel {

    let increment: Subscribers.Sink<Void, Never>
    let decrement: Subscribers.Sink<Void, Never>
    let isOn: Subscribers.Sink<Bool, Never>

    var count: AnyPublisher<String?, Never> {
        $_count.map { Optional.som
            .eraseToAnyPublisher()
    }
    var isIncrementEnabled: AnyPub
        $_isToggleEnabled.eraseTo
    }
    var isDecrementEnabled: AnyPublisher<Bool, Never> {
        $_isToggleEnabled.combineLatest($_count) { $0 && $1 > 0  }
            .eraseToAnyPublisher()
    }

    @Published private var _count: Int = 0
    @Published private var _isToggleEnabled = false

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

内部状態の_isToggleEnabledまたは_countが変更されたら、_isToggleEnabledがtrueかつ_countが0より大きいという値に変換して出力

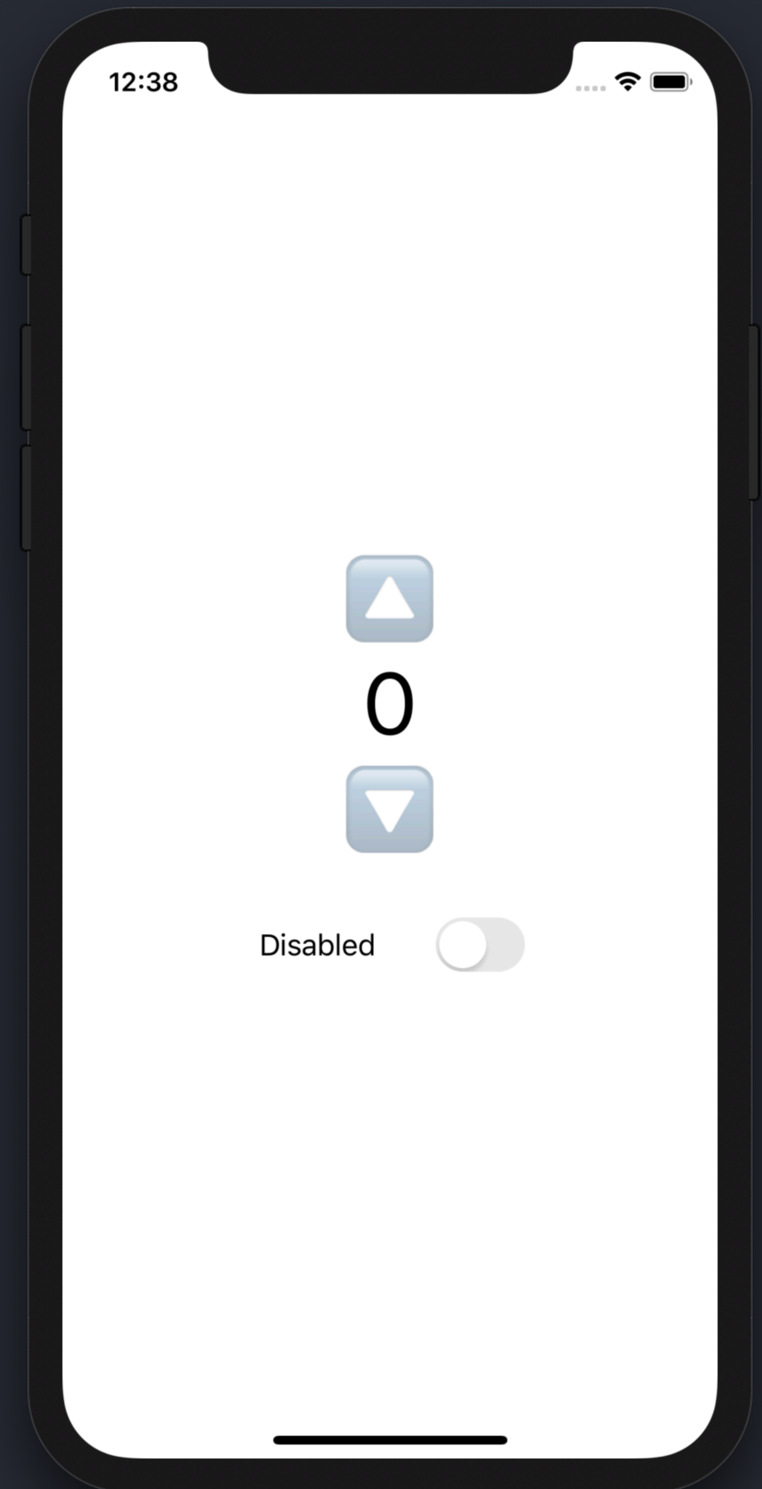
ViewControllerの実装の全体像

```
final class CounterViewController: UIViewController {

    let counterToggle: UISwitch
    let incrementButton: UIButton
    let decrementButton: UIButton
    let countLabel: UILabel
    let counterStateLabel: UILabel

    private var cancellables: [AnyCancellable] = []
    private let viewModel = ViewModel()

    override func viewDidLoad() {

        ...

        incrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.increment)

        decrementButton.tap
            .map { _ in () }
            .subscribe(viewModel.decrement)

        counterToggle.valueChanged
            .subscribe(viewModel.isOn)

        viewModel.count
            .assign(to: \.text, on: countLabel)
            .store(in: &cancellables)

        viewModel.isIncrementEnabled
            .assign(to: \.isEnabled, on: incrementButton)
            .store(in: &cancellables)

        viewModel.isDecrementEnabled
            .assign(to: \.isEnabled, on: decrementButton)
            .store(in: &cancellables)
    }
}
```

```
import UIKit
import Ricemill
```

ViewControllerの定義

```
final class CounterViewController: UIViewController {

    let counterToggle: UISwitch
    let incrementButton: UIButton
    let decrementButton: UIButton
    let countLabel: UILabel
    let counterStateLabel: UILabel

    private var cancellables: [AnyCancellable] = []
    private let viewModel = ViewModel(input: .init(),
                                      store: .init(),
                                      extra: .init())


    ...
}
```



ViewControllerからの入力

```
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        let input = viewModel.input

        incrementButton.tap
            .map { _ in () }
            .subscribe(input.increment)
            .store(in: &cancellables)

        decrementButton.tap
            .map { _ in () }
            .subscribe(input.decrement)
            .store(in: &cancellables)

        counterToggle.valueChanged
            .subscribe(input.isOn)
            .store(in: &cancellables)

        ...
    }
}
```

ViewControllerからの入力

```
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        let input = viewModel.input

        incrementButton.tap
            .map { _ in () }
            .subscribe(input.increment)
            .store(in: &cancellables)

        decrementButton.tap
            .map { _ in () }
            .subscribe(input.decrement)
            .store(in: &cancellables)

        counterToggle.valueChanged
            .subscribe(input.isOn)
            .store(in: &cancellables)

        ...
    }
}
```

ViewModelの入力が明示的になっている

ViewControllerへの反映

```
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        let output = viewModel.output

        output.count
            .assign(to: \.text, on: countLabel)
            .store(in: &cancellables)

        output.isIncrementEnabled
            .assign(to: \.isEnabled, on: incrementButton)
            .store(in: &cancellables)

        output.isDecrementEnabled
            .assign(to: \.isEnabled, on: decrementButton)
            .store(in: &cancellables)
    }
}
```

ViewControllerへの反映

```swift
final class CounterViewController: UIViewController {

    ...

    override func viewDidLoad() {

        ...

        let output = viewModel.output

        output.count
            .assign(to: \.text, on: c
            .store(in: &cancellables)

        output.isIncrementEnabled
            .assign(to: \.isEnabled, on: incrementButton)
            .store(in: &cancellables)

        output.isDecrementEnabled
            .assign(to: \.isEnabled, on: decrementButton)
            .store(in: &cancellables)
    }
}
```

ViewModelからの出力が明示的になっている

ViewModelの定義

```swift
final class CounterViewModel: Machine<CounterViewModel.Resolver> {

    struct Input: InputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        let isOn = PassthroughSubject<Bool, Never>()
    }

    struct Output: OutputType {
        let count: AnyPublisher<String?, Never>
        let isIncrementEnabled: AnyPublisher<Bool, Never>
        let isDecrementEnabled: AnyPublisher<Bool, Never>
    }

    final class Store: StoreType {
        @Published var count: Int = 0
        @Published var isToggleEnabled = false
    }

    struct Extra: ExtraType {}

    enum Resolver: ResolverType {

        static func polish(input: Publishing<Input>, store: Store, extra: Extra) -> Polished<Output> {
            ...
            return Polished(output: Output(count: count,
                                           isIncrementEnabled: incrementEnabled,
                                           isDecrementEnabled: isDecrementEnabled),
                            cancellables: cancellables)
        }
    }
}
```

ViewModelの定義

```
final class CounterViewModel: Machine<CounterViewModel.Resolver> {

    struct Input: InputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        let isOn = PassthroughSubject<Bool, Never>()
    }

    struct Output: OutputType {
        let count: AnyPublisher<String?, Never>
        let isIncrementEnabled: AnyPublisher<Bool, Never>
        let isDecrementEnabled: AnyPublisher<Bool, Never>
    }

    final class Store: StoreType {
        @Published var count: Int = 0
        @Published var isToggleEnabled = false
    }

    struct Extra: ExtraType {}

    enum Resolver: ResolverType {

        static func polish(input: Publishing<Input>, store: Store, extra: Extra) -> Polished<Output> {
            ...
            return Polished(output: Output(count: count,
                                           isIncrementEnabled: incrementEnabled,
                                           isDecrementEnabled: isDecrementEnabled),
                            cancellables: cancellables)
        }
    }
}
```

RicemillのInput

```swift
struct Input: InputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    let isOn = PassthroughSubject<Bool, Never>()
}


let input: InputProxy<Input>
let isOn: SubjectProxy<PassthroughSubject<Bool, Never>> = input.isOn
isOn.send(true)


@dynamicMemberLookup
final class InputProxy<Input: InputType> {

    private let input: Input

    init(_ input: Input) {
        self.input = input
    }

    subscript<S: Subject>(dynamicMember keyPath: KeyPath<Input, S>)
                            -> SubjectProxy<S> {
        SubjectProxy(input[keyPath: keyPath])
    }
}
```

RicemillのInput

```
struct Input: InputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    let isOn = PassthroughSubject<Bool, Never>()
}


let input: InputProxy<Input>
let isOn: SubjectProxy<PassthroughSubject<Bool, Never>> = input.isOn
isOn.send(true)


@dynamicMemberLookup
final class InputProxy<Input: InputType> {

    private let input: Input

    init(_ input: Input) {
        self.input = input
    }

    subscript<S: Subject>(dynamicMember keyPath: KeyPath<Input, S>)
                                -> SubjectProxy<S> {
        SubjectProxy(input[keyPath: keyPath])
    }
}
```

RicemillではInputがInputProxyに
ラップされた状態で公開される

RicemillのInput

```
struct Input: InputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    let isOn = PassthroughSubject<Bool, Never>()
}


let input: InputProxy<Input>
let isOn: SubjectProxy<PassthroughSubject<Bool, Never>> = input.isOn
isOn.send(true)
```

SubjectProxyにラップされた状態のSubject取得する

```
@dynamicMemberLookup
final class InputProxy<Input: InputType> {

    private let input: Input

    init(_ input: Input) {
        self.input = input
    }

    subscript<S: Subject>(dynamicMember keyPath: KeyPath<Input, S>)
                              -> SubjectProxy<S> {
        SubjectProxy(input[keyPath: keyPath])
    }
}
```

RicemillのInput

```
struct Input: InputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    let isOn = PassthroughSu
}


let input: InputProxy<Input>
let isOn: SubjectProxy<Passt
isOn.send(true)


@dynamicMemberLookup
final class InputProxy<Input: InputType> {

    private let input: Input

    init(_ input: Input) {
        self.input = input
    }

    subscript<S: Subject>(dynamicMember keyPath: KeyPath<Input, S>)
                              -> SubjectProxy<S> {
        SubjectProxy(input[keyPath: keyPath])
    }
}
```

Swift 5.1から利用可能になったtype-safeなKeyPath
ベースのdynamicMemberLookupを利用して
propertyにアクセスしているかのようなinterfaceで
型変換したインスタンスを取得する
実際のinputはprivateになっているので、外部からは
直接アクセスすることができない

RicemillのInput

```
struct Input: InputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    let isOn = PassthroughSubject<Bool, Never>()
}


let input: InputProxy<Input>
let isOn: SubjectProxy<PassthroughSubject<Bool, Never>> = input.isOn
isOn.send(true)


@dynamicMemberLoo
final class Input

    private let i

    init(_ input:
        self.input = input
    }

    subscript<S: Subject>(dynamicMember keyPath: KeyPath<Input, S>)
                              -> SubjectProxy<S> {
        SubjectProxy(input[keyPath: keyPath])
    }
}
```

SubjectProxyでは

- func send(_:)
- func send(completion:)
- func send(subscription:)

のみが公開されているので、入力に特化した型になっている

ViewModelの定義

```swift
final class CounterViewModel: Machine<CounterViewModel.Resolver> {

    struct Input: InputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        let isOn = PassthroughSubject<Bool, Never>()
    }

    struct Output: OutputType {
        let count: AnyPublisher<String?, Never>
        let isIncrementEnabled: AnyPublisher<Bool, Never>
        let isDecrementEnabled: AnyPublisher<Bool, Never>
    }

    final class Store: StoreType {
        @Published var count: Int = 0
        @Published var isToggleEnabled = false
    }

    struct Extra: ExtraType {}

    enum Resolver: ResolverType {

        static func polish(input: Publishing<Input>, store: Store, extra: Extra) -> Polished<Output> {
            ...
            return Polished(output: Output(count: count,
                                           isIncrementEnabled: incrementEnabled,
                                           isDecrementEnabled: isDecrementEnabled),
                            cancellables: cancellables)
        }
    }
}
```

RicemillのOutput

```swift
struct Output: OutputType {
    let count: AnyPublisher<String?, Never>
    let isIncrementEnabled: AnyPublisher<Bool, Never>
    let isDecrementEnabled: AnyPublisher<Bool, Never>
}


let output: OutputProxy<Output>
let count: AnyPublisher<String?, Never> = output.count
let cancellable = count.sink(receiveValue: { print(String(describing: $0)) })


@dynamicMemberLookup
final class OutputProxy<Output: OutputType> {

    private let output: Output

    init(_ output: Output) {
        self.output = output
    }

    subscript<P: Publisher>(dynamicMember keyPath: KeyPath<Output, P>)
                           -> AnyPublisher<P.Output, P.Failure> {
        output[keyPath: keyPath].eraseToAnyPublisher()
    }
}
```

RicemillのOutput

```swift
struct Output: OutputType {
    let count: AnyPublisher<String?, Never>
    let isIncrementEnabled: AnyPublisher<Bool, Never>
    let isDecrementEnabled: AnyPublisher<Bool, Never>
}


let output: OutputProxy<Output>
let count: AnyPublisher<String?, Never> = output.count
let cancellable = count.sink(receiveValue: { print(String(describing: $0)) })


@dynamicMemberLookup
final class OutputProxy<Output: OutputType> {

    private let output: Output

    init(_ output: Output) {
        self.output = output
    }

    subscript<P: Publisher>(dynamicMember keyPath: KeyPath<Output, P>)
                            -> AnyPublisher<P.Output, P.Failure> {
        output[keyPath: keyPath].eraseToAnyPublisher()
    }
}
```

RicemillではOutputがOutputProxyに
ラップされた状態で公開される

RicemillのOutput

```swift
struct Output: OutputType {
    let count: AnyPublisher<String?, Never>
    let isIncrementEnabled: AnyPublisher<Bool, Never>
    let isDecrementEnabled: AnyPublisher<Bool, Never>
}


let output: OutputProxy<Output>
let count: AnyPublisher<String?, Never> = output.count
let cancellable = count.sink(receiveValue: { print(String(describing: $0)) })


@dynamicMemberLookup
final class OutputProxy<Output: OutputType> {

    private let output: Output

    init(_ output: Output) {
        self.output = output
    }

    subscript<P: Publisher>(dynamicMember keyPath: KeyPath<Output, P>)
                            -> AnyPublisher<P.Output, P.Failure> {
        output[keyPath: keyPath].eraseToAnyPublisher()
    }
}
```

AnyPublisherにtype-eraseしたインスタンスを取得する

※今回の場合、もともとAnyPublisherなので型変換なし

RicemillのOutput

```swift
struct Output: OutputType {
    let count: AnyPublisher<String?, Never>
    let isIncrementEnabled: AnyPublisher<Bool, Never>
    let isDecrementEnabled:
}


let output: OutputProxy<Outp
let count: AnyPublisher<Stri
let cancellable = count.sink


@dynamicMemberLookup
final class OutputProxy<Output: OutputType> {

    private let output: Output

    init(_ output: Output) {
        self.output = output
    }

    subscript<P: Publisher>(dynamicMember keyPath: KeyPath<Output, P>)
                            -> AnyPublisher<P.Output, P.Failure> {
        output[keyPath: keyPath].eraseToAnyPublisher()
    }
}
```

dynamicMemberLookupを利用して、Publisherに準拠しているオブジェクト（PassthroughSubject、Publishers.○○など）をAnyPublisherに型変換したインスタンスで取得する
実際のoutputはprivateになっているので、外部からは直接アクセスすることができない

RicemillのOutput

```
struct Output: OutputType {
    let count: AnyPublisher<String?, Never>
    let isIncrementEnabled: AnyPublisher<Bool, Never>
    let isDecrementEnabled: AnyPublisher<Bool, Never>
}


let output: OutputProxy<Output>
let count: AnyPublisher<String?, Never> = output.count
let cancellable = count.sink(receiveValue: { print(String(describing: $0)) })


@dynamicMemberLookup
final class OutputProxy<Output: OutputType> {

    private let output: Output

    init(_ output: Output) {
        self.output = output
    }

    subscript<P: Publisher>(dynamicMember keyPath: KeyPath<Output, P>)
                           -> AnyPublisher<P.Output, P.Failure> {
        output[keyPath: keyPath].eraseToAnyPublisher()
    }
}
```

AnyPublisherなので、出力に特化した型になっている

ViewModelの定義 - Store

```
final class CounterViewModel: Machine<CounterViewModel.Resolver> {

    struct Input: InputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        let isOn = PassthroughSubject<Bool, Never>()
    }

    struct Output: OutputType {
        let count: AnyPublisher<String?, Never>
        let isIncrementEnabled: AnyPublisher<Bool, Never>
        let isDecrementEnabled: AnyPublisher<Bool, Never>
    }

    final class Store: StoreType {
        @Published var count: Int = 0
        @Published var isToggleEnabled = false
    }

    struct Extra: E  raType {}

    enum Resol

        static                                                                 {
            ..
            re
                                        isIncrementEnabled: incrementEnabled,
                                        isDecrementEnabled: isDecrementEnabled),
                                cancellables: cancellables)
        }
    }
}
```

値の変更を監視できるようにするため、@Publishedで定義をする
RicemilではStoreはResolverのfunc polish(input:store:extra:)
からのみ参照可能なので、internalで定義してもaccess levelは問題ない

ViewModelの定義 - Extra

```
final class CounterViewModel: Machine<CounterViewModel.Resolver> {

    struct Input: InputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        let isOn = PassthroughSubject<Bool, Never>()
    }

    struct Output: OutputType {
        let count: AnyPublisher<String?, Never>
        let isIncrementEnabled: AnyPublisher<Bool, Never>
        let isDecrementEnabled: AnyPublisher<Bool, Never>
    }

    final class Store: StoreType
        @Published var count: In
        @Published var isToggleE
    }

    struct Extra: ExtraType {}

    enum Resolver: ResolverType {

        static func polish(input: Publishing<Input>, store: Store, extra: Extra) -> Polished<Output> {
            ...
            return Polished(output: Output(count: count,
                                           isIncrementEnabled: incrementEnabled,
                                           isDecrementEnabled: isDecrementEnabled),
                            cancellables: cancellables)
        }
    }
}
```

外部依存を定義する
※今回の場合は外部依存なし

ViewModelの定義

```
final class CounterViewModel: Machine<CounterViewModel.Resolver> {

    struct Input: InputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        let isOn = PassthroughSubject<Bool, Never>()
    }

    struct Output: OutputType {
        let count: AnyPublisher<String?, Never>
        let isIncrementEnabled: AnyPublisher<Bool, Never>
        let isDecrementEnabled: AnyPublisher<Bool, Never>
    }

    final class Store: StoreType {
        @Published var count: Int = 0
        @Published var isToggleEnabled = false
    }

    struct Extra: ExtraType {}

    enum Resolver: ResolverType {

        static func polish(input: Publishing<Input>, store: Store, extra: Extra) -> Polished<Output> {
            ...
            return Polished(output: Output(count: count,
                                           isIncrementEnabled: incrementEnabled,
                                           isDecrementEnabled: isDecrementEnabled),
                            cancellables: cancellables)
        }
    }
}
```

RicemillのResolverの定義

```
public protocol ResolverType {
    associatedtype Input: InputType
    associatedtype Output: OutputType
    associatedtype Store: StoreType
    associatedtype Extra: ExtraType

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output>
}
```

Input・Output・Store・Extraを紐付けて
Input・Store・ExtraからOutputを生成する

RicemillのResolverの実装

```
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {
        var cancellables: [AnyCancellable] = []

        let increment = input.increment
            .flatMap { _ in Just(store.count) }
            .map { $0 + 1 }

        let decrement = input.decrement
            .flatMap { _ in Just(store.count) }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement)
            .assign(to: \.count, on: store)
            .store(in: &cancellables)

        input.isOn
            .assign(to: \.isToggleEnabled, on: store)
            .store(in: &cancellables)

        ...
    }
}
```

RicemillのResolverの実装

```
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {
        var cancellables: [AnyCancellable] = []

        let increment = input.increment
            .flatMap { _ in Just(store.count) }
            .map { $0 + 1 }

        let decrement = input.decrement
            .flatMap { _ in Just(store.count) }
            .map { $0 > 0 ? $

        increment.merge(with:
            .assign(to: \.count, on: store)
            .store(in: &cancellables)

        input.isOn
            .assign(to: \.isToggleEnabled, on: store)
            .store(in: &cancellables)

        ...
    }
}
```

Publishing経由で外部からの入力を、内部向けの出力としてして受け取る

RicemillのResolver – Publishing

```swift
struct Input: InputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    let isOn = PassthroughSubject<Bool, Never>()
}


let input: Publishing<Input>
let increment: AnyPublisher<Void, Never> = input.increment


@dynamicMemberLookup
final class Publishing<Input: InputType> {

    private let input: Input

    init(_ input: Input) {
        self.input = input
    }

    subscript<P: Publisher>(dynamicMember keyPath: KeyPath<Input, P>)
                                    -> AnyPublisher<P.Output, P.Failure> {
        input[keyPath: keyPath].eraseToAnyPublisher()
    }
}
```

RicemillのResolver – Publishing

```swift
struct Input: InputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    let isOn = PassthroughSubject<Bool, Never>()
}


let input: Publishing<Input>
let increment: AnyPublisher<Void, Never> = input.increment


@dynamicMemberLookup
final class Publishing<Input: InputType> {

    private let input: Input

    init(_ input: Input) {
        self.input = input
    }

    subscript<P: Publisher>(dynamicMember keyPath: KeyPath<Input, P>)
                                            -> AnyPublisher<P.Output, P.Failure> {
        input[keyPath: keyPath].eraseToAnyPublisher()
    }
}
```

AnyPublisherにtype-eraseしたインスタンスを取得する

RicemillのResolver – Publishing

dynamicMemberLookupを利用して、Publisherに準拠しているオブジェクト（PassthroughSubject、Publishers.○○など）をAnyPublisherに型変換したインスタンスで取得する
実際のinputはprivateになっているので、外部からは直接アクセスすることができない

```
struct Input: InputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = Passthro
    let isOn = PassthroughSu
}


let input: Publishing<Input>
let increment: AnyPublisher<


@dynamicMemberLookup
final class Publishing<Input: InputType> {

    private let input: Input

    init(_ input: Input) {
        self.input = input
    }

    subscript<P: Publisher>(dynamicMember keyPath: KeyPath<Input, P>)
                                -> AnyPublisher<P.Output, P.Failure> {
        input[keyPath: keyPath].eraseToAnyPublisher()
    }
}
```

RicemillのResolverの実装

```swift
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {
        ...

        let count = store.$count
            .map(String.init)
            .map(Optional.some)
            .eraseToAnyPublisher()

        let incrementEnabled = store.$isToggleEnabled
            .eraseToAnyPublisher()

        let isDecrementEnabled = store.$isToggleEnabled
            .combineLatest(store.$count)
            .map { $0 && $1 > 0 }
            .eraseToAnyPublisher()

        return Polished(output: Output(count: count,
                                       isIncrementEnabled: incrementEnabled,
                                       isDecrementEnabled: isDecrementEnabled),
                        cancellables: cancellables)
    }
}
```

RicemillのResolverの実装

```swift
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {
        ...

        let count = store.$count
            .map(String.init)
            .map(Optional.some)
            .eraseToAnyPublisher()

        let incrementEnabled = store.$isToggleEnabled
            .eraseToAnyPublisher()

        let isDecrementEnabled = store.$isToggleEnabled
            .combineLatest(store.$count)
            .map { $0 && $1 > 0 }
            .eraseToAnyPublisher()

        return Polished(output: Output(count: count,
                                       isIncrementEnabled: incrementEnabled,
                                       isDecrementEnabled: isDecrementEnabled),
                        cancellables: cancellables)
    }
}
```

内部状態の変更をもとにOutputを生成

RicemillのResolverの実装

```swift
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {
        ...

        let count = store.$count
            .map(String.init)
            .map(Optional.some)
            .eraseToAnyPublisher()

        let incrementEnabled = store.$isToggleEnabled
            .eraseToAnyPublisher()

        let isDecrementEnabled = store.$isToggleEnabled
            .combineLatest(store.$count)
            .map { $0 && $1 > 0 }
            .eraseToAnyPublisher()

        return Polished(output: Output(count: count,
                                       isIncrementEnabled: incrementEnabled,
                                       isDecrementEnabled: isDecrementEnabled),
                        cancellables: cancellables)
    }
}
```

RicemillのResolver - Machine

```swift
open class Machine<Resolver: ResolverType> {

    public let input: InputProxy<Resolver.Input>
    public let output: OutputProxy<Resolver.Output>

    private let _extra: Resolver.Extra
    private let _store: Resolver.Store
    private let _cancellables: [AnyCancellable]

    private init(input: Resolver.Input,
                 output: Resolver.Output,
                 store: Resolver.Store,
                 extra: Resolver.Extra,
                 cancellables: [AnyCancellable]) {
        self.input = InputProxy(input)
        self.output = OutputProxy(output)
        self._store = store
        self._extra = extra
        self._cancellables = cancellables
    }

    public convenience init(input: Resolver.Input, store: Resolver.Store, extra: Resolver.Extra) {
        let receivableInput = Publishing(input)
        let polished = Resolver.polish(input: receivableInput, store: store, extra: extra)
        self.init(input: input,
                  output: polished.output ?? { fatalError() }(),
                  store: store,
                  extra: extra,
                  cancellables: polished.cancellables)
    }
}
```

RicemillのResolver - Machine

```
open class Machine<Resolver: ResolverType> {

    public let input: InputProxy<Resolver.Input>
    public let output: OutputProxy<Resolver.Output>

    private let _extra: Resolver.Extra
    private let _store: Resolver.Store
    private let _cancellables: [AnyCancellable]

    private init(input: Resolver.Input,
                 output: Resolver.Output,
                 store: Resolver.Store,
                 extra: Resolver.Extra,
                 cancellables: [AnyCancellable]) {
        self.input = InputProxy(input)
        self.output = OutputProx
        self._store = store
        self._extra = extra
        self._cancellables = car
    }

    public convenience init(input: Resolver.Input, store: Resolver.Store, extra: Resolver.Extra) {
        let receivableInput = Publishing(input)
        let polished = Resolver.polish(input: receivableInput, store: store, extra: extra)
        self.init(input: input,
                  output: polished.output ?? { fatalError() }(),
                  store: store,
                  extra: extra,
                  cancellables: polished.cancellables)
    }
}
```

Machineの初期化時に紐付いているResolverの
func polish(input:store:extra)が一度だけ呼び出し
返り値のPolished内のoutputを利用する

RicemillのResolver – Machine

```swift
open class Machine<Resolver: ResolverType> {

    public let input: InputProxy<Resolver.Input>
    public let output: OutputProxy<Resolver.Output>

    private let _extra: Resolver.Extra
    private let _store: Resolver.Store
    private let _cancellables: [AnyCancellable]

    private init(input: Resolver.In
                 output: Resolver.O
                 store: Resolver.St
                 extra: Resolver.E  tra,
                 cancellables: [AnyCancellable]) {
        self.input = InputProxy(input)
        self.output = OutputProxy(output)
        self._store = store
        self._extra = extra
        self._cancellables = cancellables
    }

    public convenience init(input: Resolver.Input, store: Resolver.Store, extra: Resolver.Extra) {
        let receivableInput = Publishing(input)
        let polished = Resolver.polish(input: receivableInput, store: store, extra: extra)
        self.init(input: input,
                  output: polished.output ?? { fatalError() }(),
                  store: store,
                  extra: extra,
                  cancellables: polished.cancellables)
    }
}
```

InputProxyやOutputProxyで該当のインスタンスをラップして、初期化を完了する

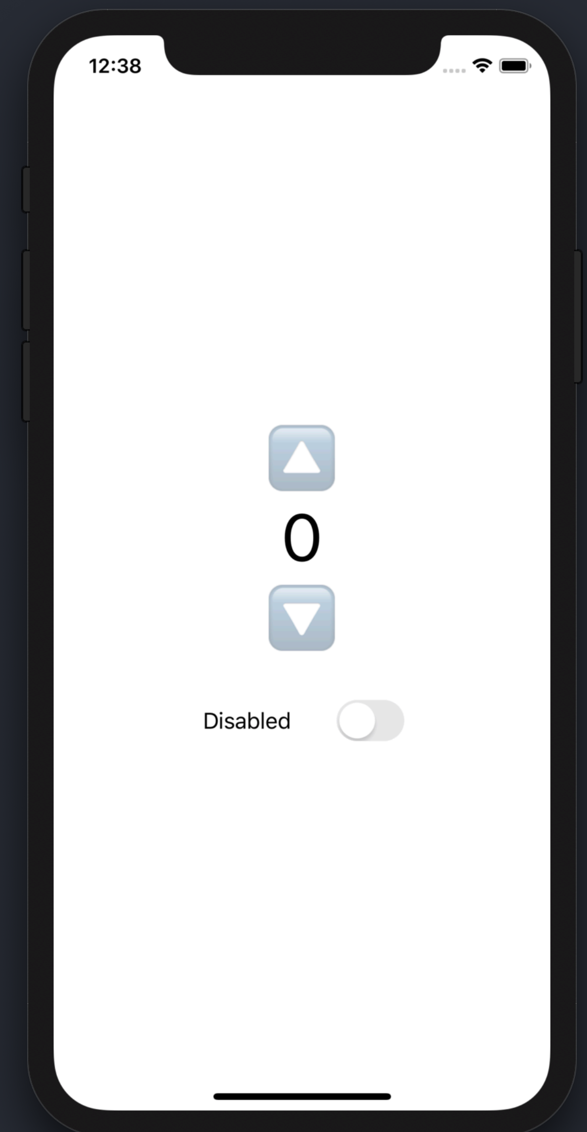
ViewControllerの実装の全体像

```swift
final class CounterViewController: UIViewController {

    let counterToggle: UISwitch
    let incrementButton: UIButton
    let decrementButton: UIButton
    let countLabel: UILabel
    let counterStateLabel: UILabel

    private var cancellables: [AnyCancellable] = []
    private let viewModel = ViewModel(input: .init(), store: .init(), extra: .init())

    override func viewDidLoad() {

        ...

        let input = viewModel.input

        incrementButton.tap.map { _ in () }.subscribe(input.increment)
            .store(in: &cancellables)

        decrementButton.tap.map { _ in () }.subscribe(input.decrement)
            .store(in: &cancellables)

        counterToggle.valueChanged.subscribe(input.isOn)
            .store(in: &cancellables)

        let output = viewModel.output

        output.count.assign(to: \.text, on: countLabel)
            .store(in: &cancellables)

        output.isIncrementEnabled.assign(to: \.isEnabled, on: incrementButton)
            .store(in: &cancellables)

        output.isDecrementEnabled.assign(to: \.isEnabled, on: decrementButton)
            .store(in: &cancellables)
    }
}
```

```swift
import SwiftUI
```

Viewの実装

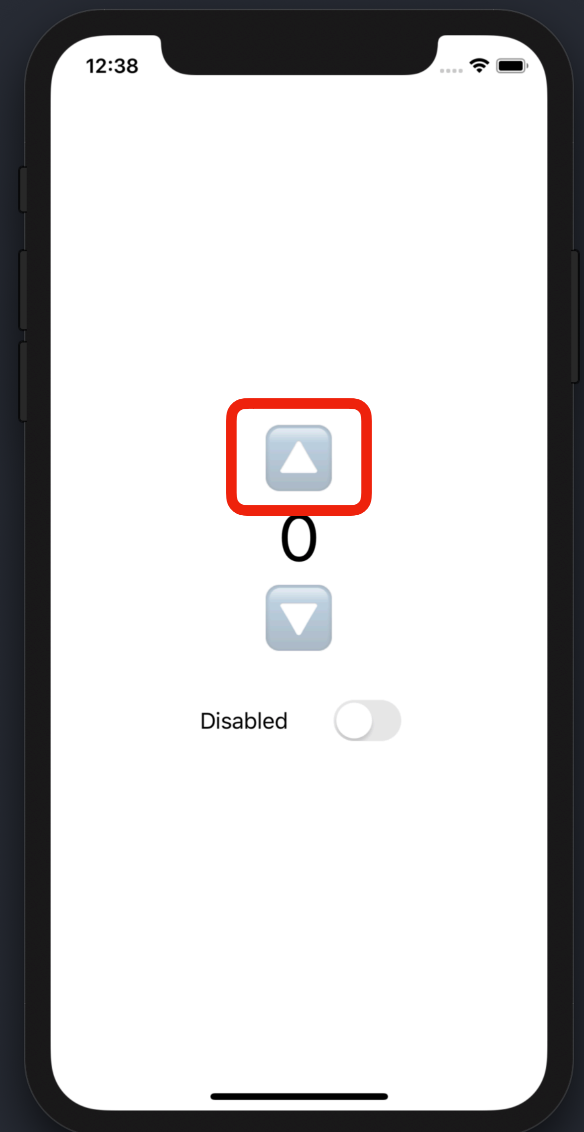
```swift
struct CounterView: View {

    @ObservedObject var viewModel = ViewModel()

    var body: some View {
        VStack {
            Button("🔼") { self.viewModel.increment() }
                .font(.system(size: 50))
                .disabled(!viewModel.isIncrementEnabled)
                .opacity(viewModel.isIncrementEnabled ? 1 : 0.5)

            Text("\(viewModel.count)")
                .font(.system(size: 50))

            Button("🔽") { self.viewModel.decrement() }
                .font(.system(size: 50))
                .disabled(!viewModel.isDecrementEnabled)
                .opacity(viewModel.isDecrementEnabled ? 1 : 0.5)

            Toggle(viewModel.toggleText, isOn: $viewModel.isOn)
                .frame(width: CGFloat(150), alignment: .center)
        }
    }
}
```



Viewの実装

```
struct CounterView: View {

    @ObservedObject var viewModel = ViewModel()

    var body: some View {
        VStack {
            Button("🔼") { self.viewModel.increment() }
                .font(.system(size: 50))
                .disabled(!viewModel.isIncrementEnabled)
                .opacity(viewModel.isIncrementEnabled ? 1 : 0.5)

            Text("\(viewModel.count)")

                .font(.system(size: 50))
                .disabled(!viewModel.isDecrementEnabled)
                .opacity(viewModel.isDecrementEnabled ? 1 : 0.5)

            Toggle(viewModel.toggleText, isOn: $viewModel.isOn)
                .frame(width: CGFloat(150), alignment: .center)
        }
    }
}
```

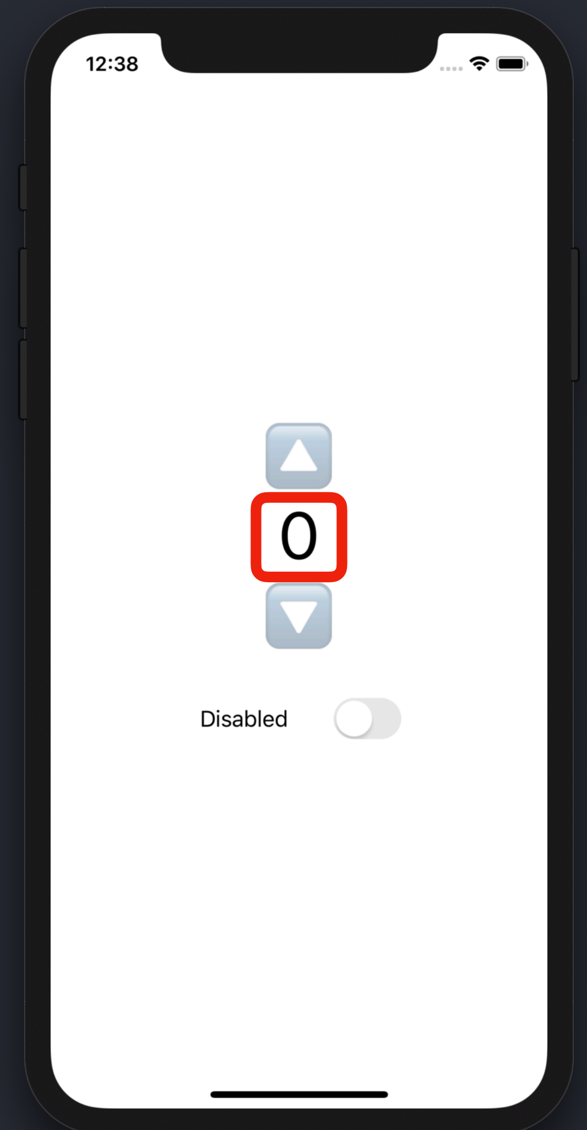
isIncrementEnabledをもとにボタンの有効・無効を反映し、ボタンがタップされるとincrement()を呼ぶ



Viewの実装

```swift
struct CounterView: View {

    @ObservedObject var viewModel = ViewModel()

    var body: some View {
        VStack {
            Button("🔼") { self.viewModel.increment() }
                .font(.system(size: 50))
                .disabled(!viewModel.isIncrementEnabled)
                .opacity(viewModel.isIncrementEnabled ? 1 : 0.5)

            Text("\(viewModel.count)")
                .font(.system(size: 50))

            Button("🔽") { self.viewModel.decrement() }
                .disabled(!viewModel.isDecrementEnabled)
                .opacity(viewModel.isDecrementEnabled ? 1 : 0.5)

            Toggle(viewModel.toggleText, isOn: $viewModel.isOn)
                .frame(width: CGFloat(150), alignment: .center)
        }
    }
}
```

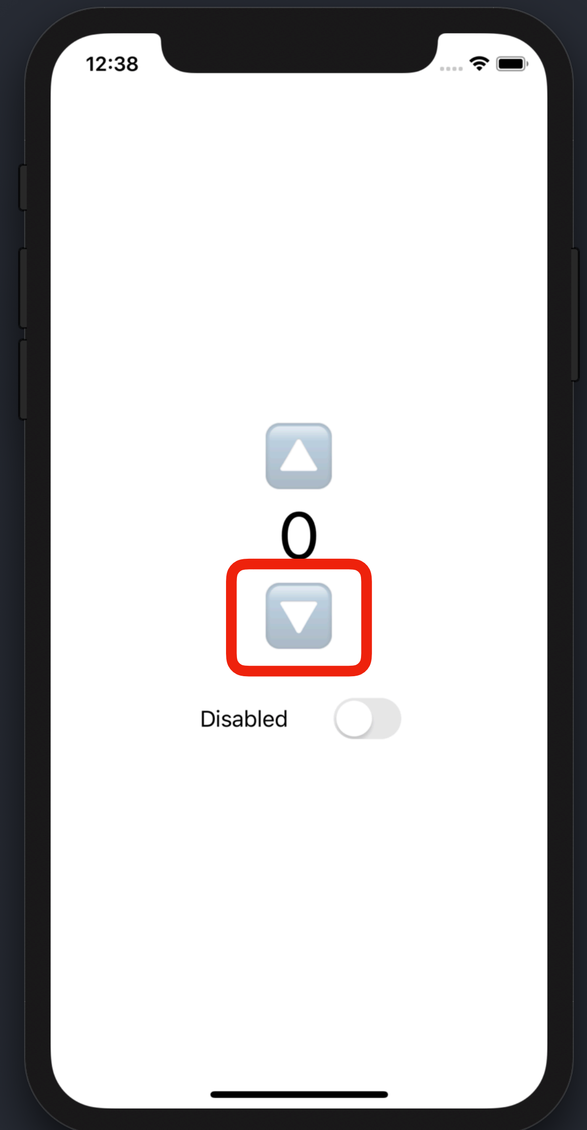
countをもとにTextを更新



Viewの実装

```swift
struct CounterView: View {

    @ObservedObject var viewModel = ViewModel()

    var body: some View {
        VStack {
            Button("🔼") { self.viewModel.increment() }
                .font(.system(size: 50))
                .disabled(!viewModel.isIncrementEnabled)
                                                        1 : 0.5)

                .font(.system(size: 50))

            Button("🔽") { self.viewModel.decrement() }
                .font(.system(size: 50))
                .disabled(!viewModel.isDecrementEnabled)
                .opacity(viewModel.isDecrementEnabled ? 1 : 0.5)

            Toggle(viewModel.toggleText, isOn: $viewModel.isOn)
                .frame(width: CGFloat(150), alignment: .center)
        }
    }
}
```

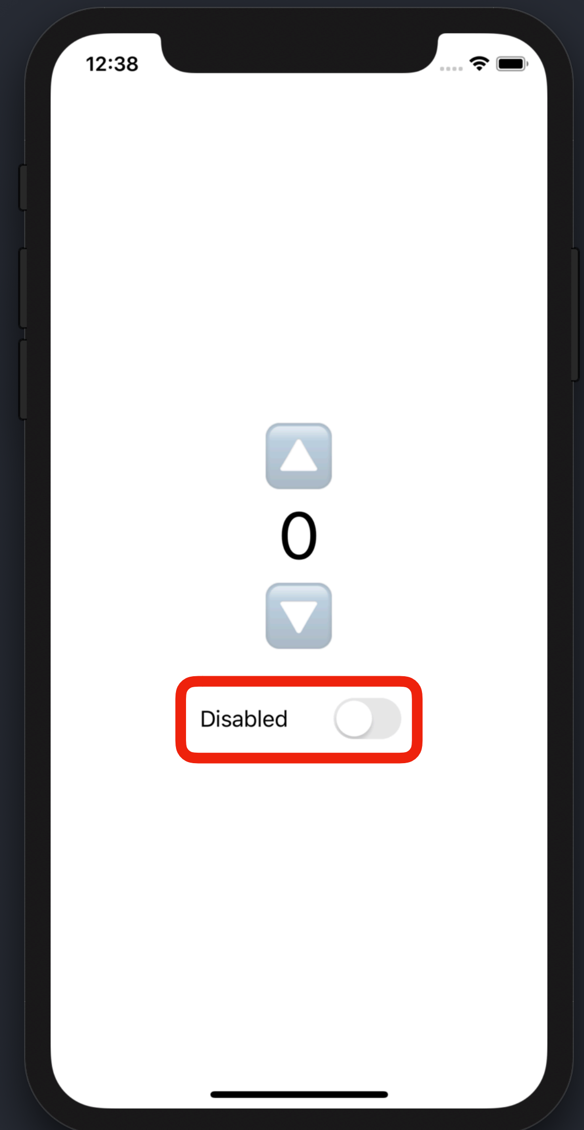
isDecrementEnabledをもとにボタンの有効・無効を反映し、ボタンがタップされるとdecrement()を呼ぶ



Viewの実装

```swift
struct CounterView: View {

    @ObservedObject var viewModel = ViewModel()

    var body: some View {
        VStack {
            Button("🔼") { self.viewModel.increment() }
                .font(.system(size: 50))
                .disabled(!viewModel.isIncrementEnabled)
                .opacity(viewModel.isIncrementEnabled ? 1 : 0.5)

            Text("\(viewModel.count)")
                .font(.system(size: 50))

            Button("🔽") { self.viewModel.decrement() }
                .font(.system(size: 50))
                                                          )
                .opacity(viewModel.isDecrementEnabled ? 1 : 0.5)

            Toggle(viewModel.toggleText, isOn: $viewModel.isOn)
                .frame(width: CGFloat(150), alignment: .center)
        }
    }
}
```

ボタンの状態がisOnによってviewModelへ流される



ViewModelの定義

```
final class ViewModel: ObservableObject {

    let increment: () -> Void
    let decrement: () -> Void

    @Published var isOn = false

    @Published private(set) var count: Int = 0
    @Published private(set) var isIncrementEnabled = false
    @Published private(set) var isDecrementEnabled = false

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

ViewModelの定義

```
final class ViewModel: ObservableObject {

    let increment: () -> Void
    let decrement: () -> Void

    @Published var isOn = false

    @Published private(set) var count: Int = 0
    @Published private(set) var isIncrementEnabled = false
    @Published private(set) var isDecrementEnabled = false

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

@ObsevedObjectとしてView側で定義できるようにする

ViewModelの定義

```
final class ViewModel: ObservableObject {

    let increment: () -> Void
    let decrement: () -> Void

    @Published var isOn = false

    @Published private(set) var cou
    @Published private(set) var isI
    @Published private(set) var isD

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

外部からの入力①:
イベントを受け付けることだけできれば良いので、今回の場合は() -> Voidで定義

ViewModelの定義

```swift
final class ViewModel: ObservableObject {

    let increment: () -> Void
    let decrement: () -> Void

    @Published var isOn = false

    @Published private(set) var count: Int = 0
    @Published private(set)
    @Published private(set)

    private var cancellables

    init() { ... }
}
```

外部からの入力②:
イベントを受け付けるためにBinding<Bool>が
必要になるので@Published internal varで定義

ViewModelの定義

```
final class ViewModel: ObservableObject {

    let increment: () -> Void
    let decrement: () -> Void

    @Published var isOn = false

    @Published private(set) var count: Int = 0
    @Published private(set) var isIncrementEnabled = false
    @Published private(set) var isDecrementEnabled = false

    private var cancellables: [AnyCancellable] = []

    init() { ... }
}
```

外部への出力と内部状態:
@Publishedで定義されているpropertyが更新されると
ObservableObjectのobjectWillChangeが発火して
Viewのbodyが更新される
そして、View側では値にアクセスできるだけで良いので
private(set)で定義して、内部でのみ更新可能とする

ViewModelの実装

```swift
final class ViewModel: ObservableObject {

    ...

    init(){
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()

        self.increment = { _increment.send(()) }
        self.decrement = { _decrement.send(()) }

        let increment = _increment.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \.count, on: self).store(in: &cancellables)

        $isOn.assign(to: \.isIncrementEnabled, on: self).store(in: &cancellables)

        $isOn.combineLatest($count).map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: self)
            .store(in: &cancellables)
    }
}
```

ViewModelの実装

```
final class ViewModel: ObservableObject {

    ...

    init(){
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()

        self.increment = { _increment.send(()) }
        self.decrement = { _decrement.send(())

        let increment = _increment.flatMap { [w
                self.map { Just($0.count).erase
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \.count, on: self).store(in: &cancellables)

        $isOn.assign(to: \.isIncrementEnabled, on: self).store(in: &cancellables)

        $isOn.combineLatest($count).map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: self)
            .store(in: &cancellables)
    }
}
```

入力を受け取ってInitalizer内でイベントを
リレーするためのPassthroughSubject

ViewModelの実装

```swift
final class ViewModel: ObservableObject {

    ...

    init(){
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()

        self.increment = { _increment.send(()) }
        self.decrement = { _decrement.send(()) }

        let increment = _increment.flatMap { [weak self]_ in
                self.map { Just($0.count).era
                    Empty().eraseToAnyPublishe
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \.count, on: self).store(in: &cancellables)

        $isOn.assign(to: \.isIncrementEnabled, on: self).store(in: &cancellables)

        $isOn.combineLatest($count).map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: self)
            .store(in: &cancellables)
    }
}
```

入力のイベントをPassthroughSubjectに繋げる

ViewModelの実装

```
final class ViewModel: ObservableObject {

    ...

    init(){
        let _increment = PassthroughSubject
        let _decrement = PassthroughSubject

        self.increment = { _increment.send
        self.decrement = { _decrement.sen ()) }

        let increment = _increment.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \.count, on: self).store(in: &cancellables)

        $isOn.assign(to: \.isIncrementEnabled, on: self).store(in: &cancellables)

        $isOn.combineLatest($count).map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: self)
            .store(in: &cancellables)
    }
}
```

_incrementからのイベントをトリガーに
内部状態の_countに対して+1した値を流す

ViewModelの実装

```swift
final class ViewModel: ObservableObject {

    ...

    init(){
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()

        self.increment = { _increment.send(()) }
        self.decrement = { _decrement.send(()) }

        let increment = _increment.flatMap {
                self.map { Just($0.count).era
                    Empty().eraseToAnyPublis
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \.count, on: self).store(in: &cancellables)

        $isOn.assign(to: \.isIncrementEnabled, on: self).store(in: &cancellables)

        $isOn.combineLatest($count).map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: self)
            .store(in: &cancellables)
    }
}
```

_decrementからのイベントをトリガーに
内部状態の_countに対して-1した値を0より
大きい値にして流す

ViewModelの実装

```swift
final class ViewModel: ObservableObject {

    ...

    init(){
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()

        self.increment = { _increment.send(()) }
        self.decrement = { _decrement.send(()) }

        let increment = _increment.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [
                self.map { Just($0.count).erase
                    Empty().eraseToAnyPublishe
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \.count, on: self).store(in: &cancellables)

        $isOn.assign(to: \.isIncrementEnabled, on: self).store(in: &cancellables)

        $isOn.combineLatest($count).map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: self)
            .store(in: &cancellables)
    }
}
```

incrementとdecrementのイベントを
もとに内部状態を更新

ViewModelの実装

```swift
final class ViewModel: ObservableObject {

    ...

    init(){
        let _increment = PassthroughSubject<Void, Never>()
        let _decrement = PassthroughSubject<Void, Never>()

        self.increment = { _increment.send(()) }
        self.decrement = { _decrement.send(()) }

        let increment = _increment.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 + 1 }

        let decrement = _decrement.flatMap { [weak self]_ in
                self.map { Just($0.count).eraseToAnyPublisher() } ??
                    Empty().eraseToAnyPublisher()
            }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement).assign(to: \.count, on: self).store(in: &cancellables)

        $isOn.assign(to: \.isIncrementEnabled, on: self).store(in: &cancellables)

        $isOn.combineLatest($count).map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: self)
            .store(in: &cancellables)
    }
}
```

_isOnからのイベントをもとに内部状態を更新

Viewの実装の全体像

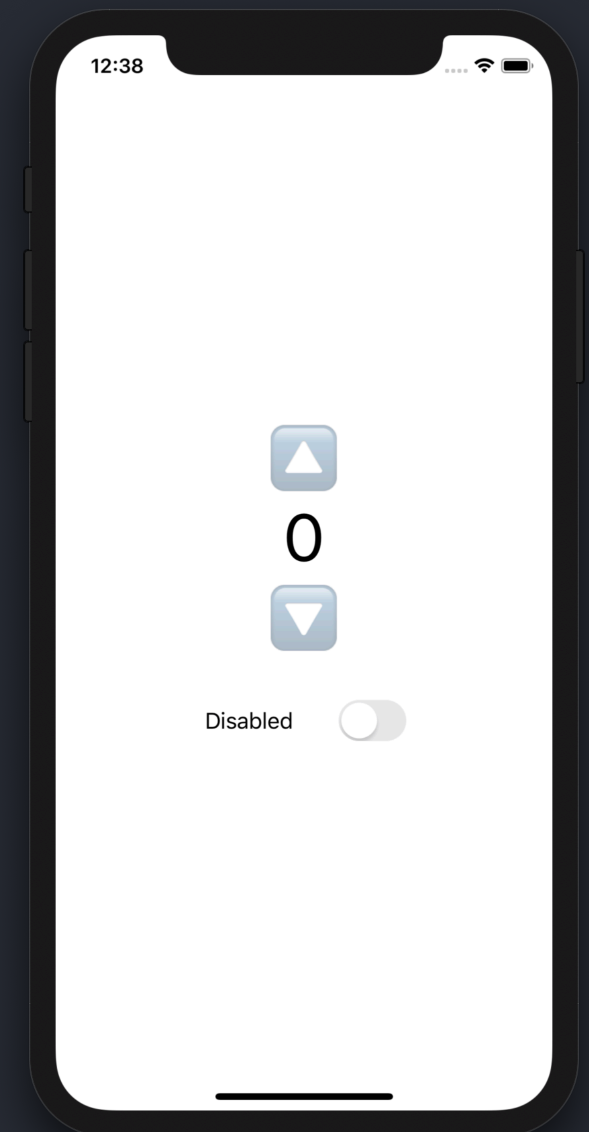
```
struct CounterView: View {

    @ObservedObject var viewModel = ViewModel()

    var body: some View {
        VStack {
            Button("⬆️") { self.viewModel.increment() }
                .font(.system(size: 50))
                .disabled(!viewModel.isIncrementEnabled)
                .opacity(viewModel.isIncrementEnabled ? 1 : 0.5)

            Text("\(viewModel.count)")
                .font(.system(size: 50))

            Button("⬇️") { self.viewModel.decrement() }
                .font(.system(size: 50))
                .disabled(!viewModel.isDecrementEnabled)
                .opacity(viewModel.isDecrementEnabled ? 1 : 0.5)

            Toggle(viewModel.toggleText, isOn: $viewModel.isOn)
                .frame(width: CGFloat(150), alignment: .center)
        }
    }
}
```

```swift
import SwiftUI
```

```swift
import Ricemill
```

Viewの実装

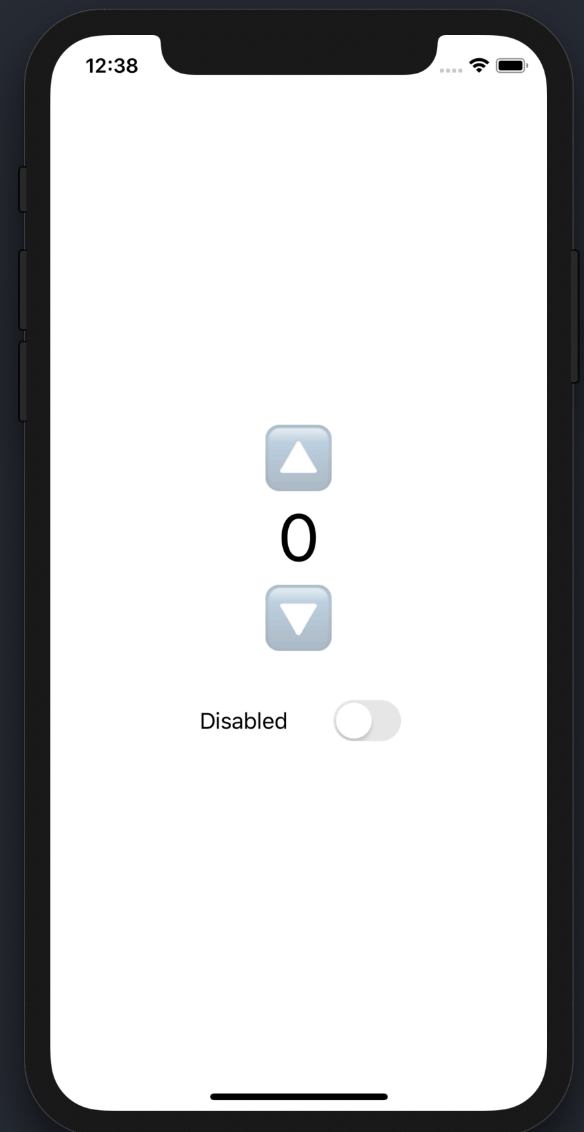
```swift
struct CounterView: View {

    @ObservedObject var viewModel = ViewModel(input: .init(),
                                              store: .init(),
                                              extra: .init())

    var body: some View {
        let input = viewModel.input
        let output = viewModel.output

        return VStack {
            Button("🔼") { input.increment.send() }
                .font(.system(size: 50))
                .disabled(!output.isIncrementEnabled)
                .opacity(output.isIncrementEnabled ? 1 : 0.5)

            Text("\(output.count)")
                .font(.system(size: 50))

            Button("🔽") { input.decrement.send() }
                .font(.system(size: 50))
                .disabled(!output.isDecrementEnabled)
                .opacity(output.isDecrementEnabled ? 1 : 0.5)

            Toggle(output.toggleText, isOn: input.isOn)
                .frame(width: 150, alignment: .center)
        }
    }
}
```



Viewの実装

```
struct CounterView: View {

    @ObservedObject var viewModel = ViewModel(input: .init(),
                                              store: .init(),
                                              extra: .init())

    var body: some View {
        let input = viewModel.input
        let output = viewModel.output

        return VStac  {
            Button("
                .font(.system(size: 50))
                .disabled(!output.isIncrementEnabled)
                .opacity(output.isIncrementEnabled ? 1 : 0.5)

            Text("\(output.count)")
                .font(.system(size: 50))

            Button("🔽") { input.decrement.send() }
                .font(.system(size: 50))
                .disabled(!output.isDecrementEnabled)
                .opacity(output.isDecrementEnabled ? 1 : 0.5)

            Toggle(output.toggleText, isOn: input.isOn)
                .frame(width: 150, alignment: .center)
        }
    }
}
```

ViewModelの入出力が明示的になっている



ViewModelの定義

```swift
final class ViewModel: Machine<ViewModel.Resolver> {

    typealias Output = Store

    final class Input: BindableInputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        @Published var isOn = false
    }

    final class Store: StoredOutputType {
        @Published var count: Int = 0
        @Published var isIncrementEnabled = false
        @Published var isDecrementEnabled = false
    }

    struct Extra: ExtraType {}

    enum Resolver: ResolverType {

        static func polish(input: Publishing<Input>,
                           store: Store,
                           extra: Extra) -> Polished<Output> {
            ...
            return Polished(cancellables: cancellables)
        }
    }
}
```

ViewModelの定義

```
final class ViewModel: Machine<ViewModel.Resolver> {

    typealias Output = Store

    final class Input: BindableInputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        @Published var isOn = false
    }

    final class Store: StoredOutputType {
        @Published var count: Int = 0
        @Published var isIncrementEnabled = false
        @Published var isDecrementEnabled = false
    }

    struct Extra: ExtraType {}

    enum Resolver: ResolverType {

        static func polish(input: Publishing<Input>,
                           store: Store,
                           extra: Extra) -> Polished<Output> {
            ...
            return Polished(cancellables: cancellables)
        }
    }
}
```

SwiftUIでのInputの振る舞い

```
final class Input: BindableInputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    @Published var isOn = false
}


protocol BindableInputType: InputType, ObservableObject {}


extension InputProxy where Input: BindableInputType {

    subscript<Subject>(
        dynamicMember keyPath: ReferenceWritableKeyPath<Input, Subject>
    ) -> Binding<Subject> {
        ObservedObject(initialValue: input).projectedValue[dynamicMember: keyPath]
    }
}
```

SwiftUIでのInputの振る舞い

```swift
final class Input: BindableInputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    @Published var isOn = false
}


protocol BindableInputType: InputType, ObservableObject {}
```

ObservableObjectに準拠したInput

```swift
extension InputProxy

    subscript<Subject>(
        dynamicMember keyPath: ReferenceWritableKeyPath<Input, Subject>
    ) -> Binding<Subject> {
        ObservedObject(initialValue: input).projectedValue[dynamicMember: keyPath]
    }
}
```

SwiftUIでのInputの振る舞い

```swift
final class Input: BindableInputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    @Published var isOn = false
}


protocol BindableInputType: InputType, ObservableObject {}


extension InputProxy where Input: BindableInputType {

    subscript<Subject>(
        dynamicMember keyPath: ReferenceWritableKeyPath<Input, Subject>
    ) -> Binding<Subject> {
        ObservedObject(initialValue: input).projectedValue[dynamicMember: keyPath]
    }
}
```

InputがBindableInputTypeの場合はdynamicMemberLookup
で@Publishedで定義されているpropertyからBindingを取得可能にする

ViewModelの定義

```swift
final class ViewModel: Machine<ViewModel.Resolver> {

    typealias Output = Store

    final class Input: BindableInputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        @Published var isOn = false
    }

    final class Store: StoredOutputType {
        @Published var count: Int = 0
        @Published var isIncrementEnabled = false
        @Published var isDecrementEnabled = false
    }

    struct Extra: ExtraType {}

    enum Resolver: ResolverType {

        static func polish(input: Publishing<Input>,
                           store: Store,
                           extra: Extra) -> Polished<Output> {
            ...
            return Polished(cancellables: cancellables)
        }
    }
}
```

SwiftUIでのStoreとOutputの振る舞い

```
final class Store: StoredOutputType {
    @Published var count: Int = 0
    @Published var isIncrementEnabled = false
    @Published var isDecrementEnabled = false
}


protocol StoredOutputType: OutputType, StoreType {}


extension Machine: ObservableObject where Resolver.Output == Resolver.Store {

    var objectWillChange: Resolver.Store.ObjectWillChangePublisher {
        return _store.objectWillChange
    }
}
```

SwiftUIでのStoreとOutputの振る舞い

```
final class Store: StoredOutputType {
    @Published var count: Int = 0
    @Published var isIncrementEnabled = false
    @Published var isDecrementEnabled = false
}


protocol StoredOutputType: OutputType, StoreType {}


extension Machine: O… == Resolver.Store {

    var objectWillChange: Resolver.Store.ObjectWillChangePublisher {
        return _store.objectWillChange
    }
}
```

StoreとOutputに準拠したprotocol

SwiftUIでのStoreとOutputの振る舞い

```
final class Store: StoredOutputType {
    @Published var count: Int = 0
    @Published var isIncrementEnabled = false
    @Published var isDecrementEnabled = false
}


protocol StoredOutputType: OutputType, StoreType {}


extension Machine: ObservableObject where Resolver.Output == Resolver.Store {

    var objectWillChange: Resolver.Store.ObjectWillChangePublisher {
        return _store.objectWillChange
    }
}
```

StoreとOutputが同じ型だった場合（StoredOutputType）に
StoreのobjectWillChangeをMachineのobjectWillChange
からアクセスできるようにし、Storeの変更をViewに伝える

ViewModelの定義

```swift
final class ViewModel: Machine<ViewModel.Resolver> {

    typealias Output = Store

    final class Input: BindableInputType {
        let increment = PassthroughSubject<Void, Never>()
        let decrement = PassthroughSubject<Void, Never>()
        @Published var isOn = false
    }

    final class Store: StoredOutputType {
        @Published var count: Int = 0
        @Published var isIncrementEnabled = false
        @Published var isDecrementEnabled = false
    }

    struct Extra: ExtraType {}

    enum Resolver: ResolverType {

        static func polish(input: Publishing<Input>,
                           store: Store,
                           extra: Extra) -> Polished<Output> {
            ...
            return Polished(cancellables: cancellables)
        }
    }
}
```

SwiftUIでのResolverの振る舞い

```
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {
        var cancellables: [AnyCancellable] = []

        let increment = input.increment
            .flatMap { _ in Just(store.count) }
            .map { $0 + 1 }

        let decrement = input.decrement
            .flatMap { _ in Just(store.count) }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: decrement)
            .assign(to: \.count, on: store)
            .store(in: &cancellables)

        ...
    }
}
```

SwiftUIでのResolverの振る舞い

```
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {
        var cancellables: [AnyCancellable] = []

        let increment = input.increment
            .flatMap { _ in Just(store.count) }
            .map { $0 + 1 }

        let decrement = input.decrement
            .flatMap { _ in Just(store.count) }
            .map { $0 > 0 ? $0 - 1 : $0 }

        increment.merge(with: de
            .assign(to: \.count,
            .store(in: &cancellables)

        ...
    }
}
```

Publishing経由で外部からの入力を、内部向けの出力としてして受け取る

SwiftUIでのResolverの振る舞い

```swift
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {

        ...


        input.$isOn
            .assign(to: \.isIncrementEnabled, on: store)
            .store(in: &cancellables)

        input.$isOn
            .combineLatest(store.$count)
            .map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: store)
            .store(in: &cancellables)

        return Polished(cancellables: cancellables)
    }
}
```

SwiftUIでのResolverの振る舞い

```
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {

        ...

        input.$isOn
            .assign(to: \.isIncrementEnabled, on: store)
            .store(in: &cancellables)

        input.$isOn
            .combineLatest(store.$count)
            .map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: store)
            .store(in: &cancellables)

        return Polished(cancellables: cancellables)
    }
}
```

Publishing経由で外部のBindingからの入力を内部向けの出力としてして受け取る

SwiftUIでのPublishingの振る舞い

```
final class Input: BindableInputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    @Published var isOn = false
}


let input: Publishing<Input>
let isOn: AnyPublisher<Bool, Never> = input.$isOn


extension Publishing where Input: BindableInputType {

    subscript<Value>(
        dynamicMember keyPath: ReferenceWritableKeyPath<Input, Value>
    ) -> Value {
        input[keyPath: keyPath]
    }
}
```

SwiftUIでのPublishingの振る舞い

```swift
final class Input: BindableInputType {
    let increment = PassthroughSubject<Void, Never>()
    let decrement = PassthroughSubject<Void, Never>()
    @Published var isOn = false
}


let input: Publishing<Input>
let isOn: AnyPublisher<Bool, Never> = input.$isOn


extension Publishing

    subscript<Value>(
        dynamicMember keyPath: ReferenceWritableKeyPath<Input, Value>
    ) -> Value {
        input[keyPath: keyPath]
    }
}
```

AnyPublisherにtype-eraseしたインスタンスを取得する

SwiftUIでのResolverの振る舞い

```swift
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {

        ...


        input.$isOn
            .assign(to: \.isIncrementEnabled, on: store)
            .store(in: &cancellables)

        input.$isOn
            .combineLatest(store.$count)
            .map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: store)
            .store(in: &cancellables)

        return Polished(cancellables: cancellables)
    }
}
```

SwiftUIでのResolverの振る舞い

```
enum Resolver: ResolverType {

    static func polish(input: Publishing<Input>,
                       store: Store,
                       extra: Extra) -> Polished<Output> {

        ...


        input.$isOn
            .assign(to: \.isIncrementEnabled, on: store)
            .store(in: &cancellables)

        input.$isOn
            .combineLatest(store.$count)
            .map { $0 && $1 > 0 }
            .assign(to: \.isDecrementEnabled, on: store)
            .store(in: &cancellables)

        return Polished(cancellables: cancellables)
    }
}
```

StoreとOutputが同一であるため、Polishedのinitializerの引数にはcancellablesのみ

SwiftUIでのPublishedの振る舞い

```
struct Polished<Output> {
    let output: Output?
    let cancellables: [AnyCancellable]
}

extension Polished where Output: StoredOutputType {

    init(cancellables: [AnyCancellable]) {
        self.output = nil
        self.cancellables = cancellables
    }
}

extension Polished where Output: OutputType {

    init(output: Output, cancellables: [AnyCancellable]) {
        self.output = output
        self.cancellables = cancellables
    }
}
```

SwiftUIでのPublishedの振る舞い

```swift
struct Polished<Output> {
    let output: Output?
    let cancellables: [AnyCancellable]
}

extension Polished where Output: StoredOutputType {

    init(cancellables: [AnyCancellable]) {
        self.output = nil
        self.cancellables = cancellables
    }
}

extension Polished where Output: OutputType {

    init(output: Output, cancellables: [AnyCancellable]) {
        self.output = output
        self.cancellables = cancellables
    }
}
```

Viewの実装の全体像

```swift
struct CounterView: View {

    @ObservedObject var viewModel = ViewModel(input: .init(),
                                              store: .init(),
                                              extra: .init())

    var body: some View {
        let input = viewModel.input
        let output = viewModel.output

        return VStack {
            Button("🔼") { input.increment.send() }
                .font(.system(size: 50))
                .disabled(!output.isIncrementEnabled)
                .opacity(output.isIncrementEnabled ? 1 : 0.5)

            Text("\(output.count)")
                .font(.system(size: 50))

            Button("🔽") { input.decrement.send() }
                .font(.system(size: 50))
                .disabled(!output.isDecrementEnabled)
                .opacity(output.isDecrementEnabled ? 1 : 0.5)

            Toggle(output.toggleText, isOn: input.isOn)
                .frame(width: 150, alignment: .center)
        }
    }
}
```

Ricemill

画像の出典元：

https://illustrain.com/?p=29718

http://www.wanpug.com/illust/illust2566.png

https://www.irasutoya.com/2017/07/blog-post_720.html

https://www.ac-illust.com/main/profile.php?id=lHshCYmV&area=1

ご静聴ありがとうございました🌾♻️🍚