I’m Zigii 👋

智杰

// Wongzigii
// Lead iOS Engineer
// @XiaoTe Tech.

Advanced Property Wrapper in Swift

Privacy Aware 💀

```
struct User {
    let username: String
    let objectId: String
    let mobilePhone: String
    let email: String
    let address: String
}

let user = User(username: "Zigii",
                objectId: UUID().uuidString,
                mobilePhone: "123456789",
                email: "wongzigii@gmail.com",
                address: "Shanghai")
```

⚠️

```
print(String(describing: user))

User(username: "Zigii", objectId: "2D564BEC-F6A6-4167-8E41-
F86F97854136", mobilePhone: "123456789", email:
"wongzigii@gmail.com", address: "Shanghai")

print(dump(user))

▿ __lldb_expr_41.User
  - username: "Zigii"
  - objectId: "2D564BEC-F6A6-4167-8E41-F86F97854136"
  - mobilePhone: "123456789"
  - email: "wongzigii@gmail.com"
  - address: "Shanghai"
```

Protocol Oriented

```
protocol SensitiveStringData: CustomDebugStringConvertible, CustomLeafReflectable {

    var rawValue: String { get }
}
```

```
extension SensitiveStringData {

    // CustomDebugStringConvertible
    var debugDescription: String { return "***" }

    // CustomLeafReflectable
    var customMirror: Mirror { return Mirror(reflecting: "***") }
}


struct Email: SensitiveStringData {

    var rawValue: String

    init(string: String) {
        rawValue = string
    }
}

struct Phone: SensitiveStringData {...}
struct Address: SensitiveStringData {...}
```

🤔

```swift
struct User {
    let username: String
    let objectId: String
    let mobilePhone: Phone
    let email: Email
    let address: Address
}
```

```swift
struct Email: SensitiveStringData {

    var rawValue: String

    init(string: String) {
        rawValue = string
    }
}

struct Phone: SensitiveStringData {...}
struct Address: SensitiveStringData {...}
```

⚠️ New types

✅ Boilerplate codes

⚠️ .rawValue

GenericType<Value>

```
struct SensitiveData<Value> {
    var value: Value

    init(_ value: Value) {
        self.value = value
    }
}


struct User {
    let username: String
    let objectId: String
    let mobilePhone: SensitiveData<String>
    let email: SensitiveData<String>
    let address: SensitiveData<String>
}
```

Property Observer

```
struct User {
    let username: String
    let objectId: String
    let mobilePhone: Phone
    let address: Address

    var email: String {
        get { return _email.value }
        set { _email.value = newValue }
    }

    private var _email: SensitiveData<String>

    init(username: String, email: String, ...) {
        self.username = username
        self._email = SensitiveData<String>(email)
        ...
    }
}
```


SensitiveData<T>
Property Wrappers
imgflip.com

```swift
@propertyWrapper
struct Sensitive {
    private var value: String

    var wrappedValue: String {
        get {
            return "***"
        }
        set { self.value = newValue }
    }

    init(wrappedValue: String) {
        self.value = wrappedValue
    }
}
```

```swift
extension Sensitive:
CustomDebugStringConvertible {
    var debugDescription: String {
        return self.wrappedValue
    }
}
```

```swift
struct User {
    let username: String
    let objectId: String

    @Sensitive var mobilePhone: String
    @Sensitive var email: String
    @Sensitive var address: String
}
```

- ✅ No new types
- ✅ No boilerplate codes
- ✅ No .rawValue
- ✅ Write once, use anywhere

History of Property wrappers

SE-0030: Property Behaviors

- Lazy
- NSCopying

main swift-evolution / proposals / 0030-property-behavior-decls.md Go to file ···

ole [Gardening] Migrate lists.swift.org links to forums.swift.org (#1266) ··· ✓ Latest commit f55963f on 7 May History

11 contributors

1060 lines (854 sloc) 33.5 KB Raw Blame

Property Behaviors

- Proposal: SE-0030
- Author: Joe Groff
- Review Manager: Doug Gregor
- Status: **Deferred**
- Decision Notes: Rationale

Introduction

There are property implementation patterns that come up repeatedly. Rather than hardcode a fixed set of patterns into the compiler, we should provide a general "property behavior" mechanism to allow these patterns to be defined as libraries.

Swift Evolution Discussion
Review

Motivation

SE-0258: Property Wrappers

Property Wrapper in SwiftUI

```
@Environment(\.widgetFamily) var family
```

```
struct PlayerView: View {
    var episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle)
            PlayButton(isPlaying: $isPlaying)
        }
    }
}
```

- View is struct (value type)
- No UIView subclass (200+ props)
- Isolating state in a clean way

wrappedValue & projectedValue

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
@frozen @propertyWrapper public struct State<Value> : DynamicProperty {

    public init(wrappedValue value: Value)

    public init(initialValue value: Value)

    public var wrappedValue: Value { get nonmutating set }

    /// A binding to the state value.
    ///
    /// Use the projected value to pass a binding value down a view hierarchy.
    /// To get the `projectedValue`, prefix the property variable with `$`. For
    /// example, in the following code example `PlayerView` projects a binding
    /// of the state property `isPlaying` to the `PlayButton` view using
    /// `$isPlaying`.
    ///
    ///     struct PlayerView: View {
    ///         var episode: Episode
    ///         @State private var isPlaying: Bool = false
    ///
    ///         var body: some View {
    ///             VStack {
    ///                 Text(episode.title)
    ///                 Text(episode.showTitle)
    ///                 PlayButton(isPlaying: $isPlaying)
    ///             }
    ///         }
    ///     }
    public var projectedValue: Binding<Value> { get }
}
```

ProjectedValue can be any type!

```swift
135
136 struct User {}
137
138 struct Example {
139
140     @State var user = User()
141
142     func debug() {
143         print(_user)
144         print(user)
145         print($user)
146     }
147 }
148
149 Example().debug()
150
```

```
"State<User>(_value: __lldb_expr_1.User(), _location: nil)\n"
"User()\n"
"Binding<User>(transaction: SwiftUI.Transaction(plist: []), loc...
```

```swift
var _user: State(getValue: { self.storage }, setValue: { self.storage = $0 })
var  user: User    { get { _user.wrappedValue   }   set { _user.wrappedValue   = newValue } }
var $user: Binding { get { _user.projectedValue }   set { _user.projectedValue = newValue } }
```

Direct Access	Indirect Access	Type
user	_user.wrappedValue	User
_user	✗	State<Value>
$user	_user.projectedValue	Binding<Value>

Examples

```swift
struct Example {

    struct RGBColor {

        @Clamping(0...255)
        var red: Double

        @Clamping(0...255)
        var green: Double

        @Clamping(0...255)
        var blue: Double
    }

    @Rounded(digits: 2) var number: Double = 3.14159

    @Trimmed var username: String

    @Expirable(duration: 3600)
    var token: String?

    @DynamicColor(light: .white, dark: .black)
    var backgroundColor: UIColor

    ...
}
```

```swift
@propertyWrapper
struct Expirable<Value> {

    var projectedValue: Expirable {
        self
    }

    var wrappedValue: Value? {
        get {
            return isValid ? storage?.value : nil
        }

        set {
            storage = newValue.map { ($0,
Date().addingTimeInterval(duration)) }
        }
    }

    init(duration: TimeInterval) {
        self.duration = duration
    }

    let duration: TimeInterval
    private var storage: (value: Value, expirationDate: Date)?

    var isValid: Bool {
        guard let storage = storage else {
            return false
        }
        return storage.expirationDate ≥ Date()
    }
}
```

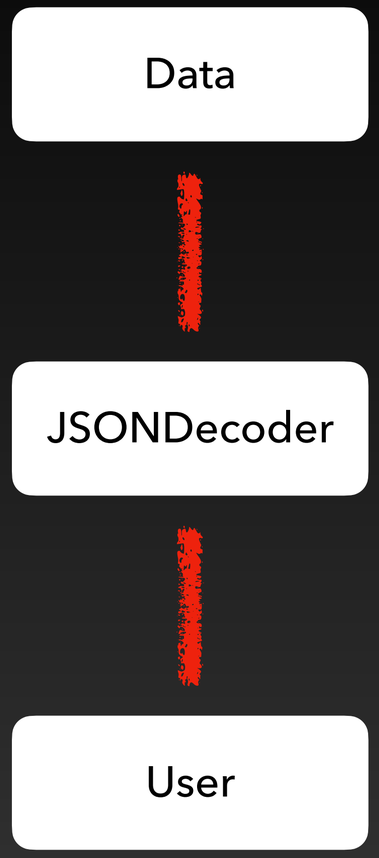
```swift
@Expirable(duration: 3600) var apiToken: String?

apiToken          // String?
$apiToken.isValid // Bool
```

With Codable

```
{
    "name": "Zigii",
    "date": "2021-06-21",
    "createdAt": "2021-06-21T12:00:00Z"
}
```

```
struct User: Decodable {
    let name: String
    let date: Date
    let createdAt: Date
}
```



```swift
struct User: Decodable {
    let name: String
    let date: Date
    let createdAt: Date
}
```

👇

```swift
struct User: Decodable {
    let name: String

    @DateValue<YearMonthDay>
    let date: Date

    @DateValue<ISO8601>
    let createdAt: Date
}
```

```swift
public protocol DateValueCodableStrategy {
    associatedtype RawValue: Codable

    static func decode(_ value: RawValue) throws -> Date
    static func encode(_ date: Date) -> RawValue
}


@propertyWrapper
public struct DateValue<Formatter: DateValueCodableStrategy>: Codable {
    private let value: Formatter.RawValue
    public var wrappedValue: Date

    public init(wrappedValue: Date) {
        self.wrappedValue = wrappedValue
        self.value = Formatter.encode(wrappedValue)
    }

    public init(from decoder: Decoder) throws {
        self.value = try Formatter.RawValue(from: decoder)
        self.wrappedValue = try Formatter.decode(value)
    }

    public func encode(to encoder: Encoder) throws {
        try value.encode(to: encoder)
    }
}
```

Pitfalls & Limitations

- Property wrappers can't be aliased.

```
@Percentage
var opacity: Double = 0.5

typealias Percentage = Clamped(0...1)   ❌
```

- Can't override property with property wrapper.

```
class SuperClass {

    @Clamping(0...255)
    var red: Double

    ...
}

class SubClass: SuperClass {

    @Clamping(0...255)
    override var red: Double ❌

    ...
}
```

- Property wrappers can’t be required in protocols.

```
protocol Clampable {
    @Clamping var value: Int { get set }
}

❌ // error: PropertyWrapper.playground:131:19: error: property
'value' declared inside a protocol cannot have a wrapper
```

- Property wrappers can’t throw error.

Swift 5.5 🥰

SE-0293: Extend Property Wrappers to Function and Closure Parameters

```swift
@propertyWrapper
struct Logged<Value> {
    init(wrappedValue: Value) {
        print(wrappedValue)
        self.wrappedValue = wrappedValue
    }

    var wrappedValue: Value {
        didSet {
            print(wrappedValue)
        }
    }
}

// Every time `runAnimation` is called, the `duration` argument
// will be logged by the property wrapper.
func runAnimation(@Logged withDuration duration: Double) { ... }
```

Recap

🟢 Reduce complexity of code

🟢 Remove duplicate boilerplate code

🟢 Write more expressive code

⚠️ Use Property wrapper not Property rapper 🤦‍♂️

Compiler

Swift Code

Expression Tree

ASTWalker

ASTContext

SILGen

ASTWalker.cpp

lib > AST > ASTWalker.cpp

```
184   }
```

Decl.cpp

lib > AST > Decl.cpp

```
6156
6157  PropertyWrapperInitializerInfo
6158  VarDecl::getPropertyWrapperInitializerInfo() const {
6159    auto &ctx = getASTContext();
6160    auto mutableThis = const_cast<VarDecl *>(this);
6161    return evaluateOrDefault(
6162        ctx.evaluator,
6163        PropertyWrapperInitializerInfoRequest{mutableThis},
6164        PropertyWrapperInitializerInfo());
6165  }
6166
6167  Optional<PropertyWrapperMutability>
6168  VarDecl::getPropertyWrapperMutability() const {
6169    auto &ctx = getASTContext();
6170    auto mutableThis = const_cast<VarDecl *>(this);
6171    return evaluateOrDefault(
6172        ctx.evaluator,
6173        PropertyWrapperMutabilityRequest{mutableThis},
```

```
208        return true;
209      }
210    }
211    return false;
212  }
213
```

Gregor, 2 years ago •

ackingInitializer);

Holly Borla, 4 mor

nitFromProjectedValue);

Resources

- https://nshipster.com/propertywrapper/
- SE-0258
- SE-0293

@wongzigii

@wongzigii