CÓDIGO QUE GENERA CÓDIGO

para hacernos la vida más fácil

GUIDO MARUCCI BLAS

Co-founder at Wolox

- @guidomb
- guidomb
- https://guidomb.blog/

AGENDA

- ¿Qué es metaprogramming?
- Introducción a Sourcery
- Sourcery, virtual views y algoritmo de diff

¿QUÉ ES METAPROGRAMMING?

CÓDIGO QUE GENERA CÓDIGO

Metaprogramming is a programming technique in which computer programs have the ability to treat programs as their data. It means that a program can be designed to read, generate, analyse or transform other programs, and even modify itself while running.

- Trata a otros programas como datos de entrada o salida
- Pueden diseñarse para leer, generar, analizar o transformar otros programas
- Se puede modificar a si mismo mientras corre

HAY DISTINTAS FORMAS DE HACER METAPROGRAMACIÓN

- En tiempo de compilación (C, Rust, Scala)
- En tiempo de ejecución (Java, Ruby)

SOURCERY

- Sourcery is a code generator for Swift language, built on top of Apple's own SourceKit.
- https://github.com/krzysztofzablocki/Sourcery

OBJETIVO

Poder listar todos los posibles casos de un enum y saber cuantos casos tiene un enum.

```
protocol AutoCases { }
```

```
{% for enum in types.implementing.AutoCases|enum %}
extension {{ enum.name }} {
  static let count: Int = {{ enum.cases.count }}
  {% if not enum.hasAssociatedValues %}
  static let allCases: [{{ enum.name }}] = [
  {% for case in enum.cases %}  .{{ case.name }}{% if not forloop.last %},{% endif %}
  {% endfor %}]
  {% endif %}
}
{% endfor %}
```

```
{% for enum in types.implementing.AutoCases|enum %}
extension {{ enum.name }} {
  static let count: Int = {{ enum.cases.count }}
  {% if not enum.hasAssociatedValues %}
  static let allCases: [{{ enum.name }}] = [
  {% for case in enum.cases %}  .{{ case.name }}{% if not forloop.last %},{% endif %}
  {% endfor %}]
  {% endif %}
}
{% endfor %}
```

```
{% for enum in types.implementing.AutoCases|enum %}
extension {{ enum.name }} {
  static let count: Int = {{ enum.cases.count }}
  {% if not enum.hasAssociatedValues %}
  static let allCases: [{{ enum.name }}] = [
  {% for case in enum.cases %}  .{{ case.name }}{% if not forloop.last %},{% endif %}
  {% endfor %}]
  {% endif %}
}
{% endfor %}
```

```
{% for enum in types.implementing.AutoCases|enum %}
extension {{ enum.name }} {
  static let count: Int = {{ enum.cases.count }}
  {% if not enum.hasAssociatedValues %}
  static let allCases: [{{ enum.name }}] = [
  {% for case in enum.cases %}  .{{ case.name }}{% if not forloop.last %},{% endif %}
  {% endfor %}]
  {% endif %}
}
{% endfor %}
```

```
{% for enum in types.implementing.AutoCases|enum %}
extension {{ enum.name }} {
  static let count: Int = {{ enum.cases.count }}
  {% if not enum.hasAssociatedValues %}
  static let allCases: [{{ enum.name }}] = [
  {% for case in enum.cases %}  .{{ case.name }}{% if not forloop.last %},{% endif %}
  {% endfor %}]
  {% endif %}
}
{% endfor %}
```

```
{% for enum in types.implementing.AutoCases|enum %}
extension {{ enum.name }} {
  static let count: Int = {{ enum.cases.count }}
  {% if not enum.hasAssociatedValues %}
  static let allCases: [{{ enum.name }}] = [
  {% for case in enum.cases %}  .{{ case.name }}{% if not forloop.last %},{% endif %}
  {% endfor %}]
  {% endif %}
}
{% endfor %}
```

```
{% for enum in types.implementing.AutoCases|enum %}
extension {{ enum.name }} {
  static let count: Int = {{ enum.cases.count }}
  {% if not enum.hasAssociatedValues %}
  static let allCases: [{{ enum.name }}] = [
  {% for case in enum.cases %}  .{{ case.name }}{% if not forloop.last %},{% endif %}
  {% endfor %}]
  {% endif %}
}
{% endfor %}
```

```
enum HTTPMethod: AutoCases {

    case post
    case get
    case head
    case put
    case delete

}
```

```
brew install sourcery
cd MyProject
sourcery --templates ./Templates \
         --output ./Sources/AutoGenerated \
         --sources ./Sources
```

```swift
extension HTTPMethod {

    static let count: Int = 5

    static let allCases: [HTTPMethod] = [
        .post, .get, .head, .put, .delete
    ]

}
```

OBJETIVO

Convertir automaticamente las `CodingKeys` de un enconder de snake case a camel case

(No es más necesario a partir de Swift 4.1 gracias a `.keyDecodingStrategy`)*

```
"user": {
  "login": "baxterthehacker",
  "id": 6752317,
  "avatar_url": "https://avatars...",
  "gravatar_id": "",
  "url": "https://api.github.com/users/baxterthehacker",
  "gists_url": "https://api.github.com/users/...",
  "starred_url": "https://api.github.com/...",
  "organizations_url": "https://api.github.com/...",
  "repos_url": "https://api.github.com/...",
  "type": "User",
  "site_admin": false
}
```

```
public struct GitHubUser: Decodable, AutoCodingKeys {

    public let login: String
    public let id: UInt
    public let avatarUrl: URL
    public let gravatarId: String
    public let url: URL
    public let gistsUrl: URL
    public let starredUrl: URL
    public let organizationsUrl: URL
    public let reposUrl: URL
    public let type: String
    public let siteAdmin: Bool

}
```

```
public struct GitHubUser: Decodable, AutoCodingKeys {

    public let login: String
    public let id: UInt
    public let avatarUrl: URL
    public let gravatarId: String
    public let url: URL
    public let gistsUrl: URL
    public let starredUrl: URL
    public let organizationsUrl: URL
    public let reposUrl: URL
    public let type: String
    public let siteAdmin: Bool

}
```

```
{% for type in types.implementing.AutoCodingKeys %}
extension {{ type.name }} {

    enum CodingKeys : String, CodingKey {

    {% for instanceVariable in type.instanceVariables %}
        case {{ instanceVariable.name }} = "{{ instanceVariable.name|camelToSnakeCase }}"
    {% endfor %}

    }

}
{% endfor %}
```

```
{% for type in types.implementing.AutoCodingKeys %}
extension {{ type.name }} {

    enum CodingKeys : String, CodingKey {

    {% for instanceVariable in type.instanceVariables %}
        case {{ instanceVariable.name }} = "{{ instanceVariable.name|camelToSnakeCase }}"
    {% endfor %}

    }

}
{% endfor %}
```

```
{% for type in types.implementing.AutoCodingKeys %}
extension {{ type.name }} {

    enum CodingKeys : String, CodingKey {

    {% for instanceVariable in type.instanceVariables %}
        case {{ instanceVariable.name }} = "{{ instanceVariable.name|camelToSnakeCase }}"
    {% endfor %}

    }

}
{% endfor %}
```

```
{% for type in types.implementing.AutoCodingKeys %}
extension {{ type.name }} {

    enum CodingKeys : String, CodingKey {

    {% for instanceVariable in type.instanceVariables %}
        case {{ instanceVariable.name }} = "{{ instanceVariable.name|camelToSnakeCase }}"
    {% endfor %}

    }

}
{% endfor %}
```

```
{% for type in types.implementing.AutoCodingKeys %}
extension {{ type.name }} {

    enum CodingKeys : String, CodingKey {

    {% for instanceVariable in type.instanceVariables %}
        case {{ instanceVariable.name }} = "{{ instanceVariable.name|camelToSnakeCase }}"
    {% endfor %}

    }

}
{% endfor %}
```

```
extension GitHubUser {

    enum CodingKeys : String, CodingKey {

        case login = "login"
        case id = "id"
        case avatarUrl = "avatar_url"
        case gravatarId = "gravatar_id"
        case url = "url"
        case gistsUrl = "gists_url"
        case starredUrl = "starred_url"
        case reposUrl = "repos_url"
        case type = "type"
        case siteAdmin = "site_admin"

    }

}
```

```
extension GitHubUser {

    enum CodingKeys : String, CodingKey {

        case login = "login"
        case id = "id"
        case avatarUrl = "avatar_url"
        case gravatarId = "gravatar_id"
        case url = "url"
        case gistsUrl = "gists_url"
        case starredUrl = "starred_url"
        case reposUrl = "repos_url"
        case type = "type"
        case siteAdmin = "site_admin"

    }

}
```

SOURCERY

- Evitamos trabajo repetitivo
- Evitamos bugs debido a olvidos o problemas de sincronización entre la "fuente de verdad" y el código
- No incurrimos en mayores riesgos al incluir Sourcery como dependencia, termina generando código Swift.

One more thing...

PORTAL

A (potentially) cross-platform, unidirectional data flow framework to build applications using a declarative and immutable UI API.

PORTAL

- github.com/guidomb/Portal
- Implementación de la arquitectura de Elm (o React/Redux) en Swift
  - Incluye virtual views, diffing algorithm y manejo de estado

RENDERING PIPELINE

1. La aplicación genera una jerarquía virtual de vistas para el estado actual de la aplicación
2. Portal compara la vieja jerarquía virtual de vistas con la recientemente generada
3. Portal aplica los cambios solo a los nodos que fueron actualizados (objetos UIView)

1. La aplicación genera una jerarquía virtual de vistas para el estado actual de la aplicación
2. Portal compara la vieja jerarquía virtual de vistas con la recientemente generada
3. Portal aplica los cambios solo a los nodos que fueron actualizados (objetos UIView)

1. La aplicación genera una jerarquía virtual de vistas para el estado actual de la aplicación
2. Portal compara la vieja jerarquía virtual de vistas con la recientemente generada
3. Portal aplica los cambios solo a los nodos que fueron actualizados (objetos UIView)

```
let component: Component<Message> = container(
  children: [
    label(
      text: "Hello Portal!",
      style: labelStyleSheet() { base, label in
          base.backgroundColor = .white
          label.textColor = .red
          label.textSize = 12
      },
      layout: layout() {
          $0.flex = flex() {
              $0.grow = .one
          }
          $0.justifyContent = .flexEnd
      }
    )
    button(
      properties: properties() {
          $0.text = "Tap to like!"
          $0.onTap = .like
      }
    )
    button(
      properties: properties() {
          $0.text = "Tap to go to detail screen"
          $0.onTap = .goToDetailScreen
      }
    )
  ]
)
```

```
button(
    properties: properties() {
        $0.text = "Tap to go to detail screen"
        $0.onTap = .goToDetailScreen
    }
)
```

```
label(
    text: "Hello Portal!",
    style: labelStyleSheet() { base, label in
        base.backgroundColor = .white
        label.textColor = .red
        label.textSize = 12
    },
    layout: layout() {
        $0.flex = flex() {
            $0.grow = .one
        }
        $0.justifyContent = .flexEnd
    }
)
```

```
label(
    text: "Hello Portal!",
    style: labelStyleSheet() { base, label in
        base.backgroundColor = .white
        label.textColor = .red
        label.textSize = 12
    },
    layout: layout() {
        $0.flex = flex() {
            $0.grow = .one
        }
        $0.justifyContent = .flexEnd
    }
)
```

```
label(
    text: "Hello Portal!",
    style: labelStyleSheet() { base, label in
        base.backgroundColor = .white
        label.textColor = .red
        label.textSize = 12
    },
    layout: layout() {
        $0.flex = flex() {
            $0.grow = .one
        }
        $0.justifyContent = .flexEnd
    }
)
```

```
label(
    text: "Hello Portal!",
    style: labelStyleSheet() { base, label in
        base.backgroundColor = .white
        label.textColor = .red
        label.textSize = 12
    },
    layout: layout() {
        $0.flex = flex() {
            $0.grow = .one
        }
        $0.justifyContent = .flexEnd
    }
)
```

```
public indirect enum Component<MessageType> {

    case button(ButtonProperties<MessageType>, StyleSheet<ButtonStyleSheet>, Layout)
    case label(LabelProperties, StyleSheet<LabelStyleSheet>, Layout)
    case mapView(MapProperties, StyleSheet<EmptyStyleSheet>, Layout)
    case imageView(Image, StyleSheet<EmptyStyleSheet>, Layout)
    case container([Component<MessageType>], StyleSheet<EmptyStyleSheet>, Layout)
    case table(TableProperties<MessageType>, StyleSheet<TableStyleSheet>, Layout)
    case collection(CollectionProperties<MessageType>, StyleSheet<CollectionStyleSheet>, Layout)
    case carousel(CarouselProperties<MessageType>, StyleSheet<EmptyStyleSheet>, Layout)
    case touchable(gesture: Gesture<MessageType>, child: Component<MessageType>)
    case segmented(ZipList<SegmentProperties<MessageType>>, StyleSheet<SegmentedStyleSheet>, Layout)
    case progress(ProgressCounter, StyleSheet<ProgressStyleSheet>, Layout)
    case textField(TextFieldProperties<MessageType>, StyleSheet<TextFieldStyleSheet>, Layout)
    case custom(CustomComponent, StyleSheet<EmptyStyleSheet>, Layout)
    case spinner(StyleSheet<SpinnerStyleSheet>, Layout)
    case textView(TextViewProperties, StyleSheet<TextViewStyleSheet>, Layout)
    case toggle(ToggleProperties<MessageType>, StyleSheet<ToggleStyleSheet>, Layout)

}
```

```
diff(old: Component, new: Component) → ChangeSet
```

```
apply(changeSet: ChangeSet, to: UIView)
```

```
Component<MessageType>.button(
    ButtonProperties<MessageType>,
    StyleSheet<ButtonStyleSheet>,
    Layout
)
```

```
Component<MessageType>.button(
    ButtonProperties<MessageType>,
    StyleSheet<ButtonStyleSheet>,
    Layout
)
```

```
public struct ButtonProperties<MessageType>: AutoPropertyDiffable {

    public var text: String?
    public var isActive: Bool
    public var icon: Image?
    // sourcery: skipDiff
    public var onTap: MessageType?

    public init(
        text: String? = .none,
        isActive: Bool = false,
        icon: Image? = .none,
        onTap: MessageType? = .none) {
        self.text = text
        self.isActive = isActive
        self.icon = icon
        self.onTap = onTap
    }

}
```

```
public struct ButtonProperties<MessageType>: AutoPropertyDiffable {

    public var text: String?
    public var isActive: Bool
    public var icon: Image?
    // sourcery: skipDiff
    public var onTap: MessageType?

    public init(
        text: String? = .none,
        isActive: Bool = false,
        icon: Image? = .none,
        onTap: MessageType? = .none) {
        self.text = text
        self.isActive = isActive
        self.icon = icon
        self.onTap = onTap
    }

}
```

AutoPropertyDiffable.stencil

```
{% for type in types.implementing.AutoPropertyDiffable %}
// MARK: - {{ type.name }} AutoPropertyDiffable
{% if type.accessLevel == "public" %}public {% endif %}extension {{ type.name }} {

    {% if type.accessLevel == "public" %}public {% endif %}enum Property {

        {% for instanceVariable in type.instanceVariables|publicGet %}
        {% if not instanceVariable.annotations.ignoreInChangeSet %}
        {% if instanceVariable.type.based.AutoPropertyDiffable %}
        case {{ instanceVariable.name }}([{{ instanceVariable.type.name }}.Property]{% if instanceVariable.isOptional %}?{% endif %})
        {% else %}
        case {{ instanceVariable.name }}({{ instanceVariable.typeName }})
        {% endif %}
        {% endif %}
        {% endfor %}

    }

    {% if type.accessLevel == "public" %}public {% endif %}var fullChangeSet: [{{ type.name }}.Property] {
        return [
        {% for instanceVariable in type.instanceVariables|publicGet %}
        {% if not instanceVariable.annotations.ignoreInChangeSet %}
        {% if instanceVariable.type.based.AutoPropertyDiffable %}
            .{{ instanceVariable.name }}(self.{{ instanceVariable.name }}){% if instanceVariable.isOptional %}?{% endif %}.fullChangeSet),
        {% else %}
            .{{ instanceVariable.name }}(self.{{ instanceVariable.name }}),
        {% endif %}
        {% endif %}
        {% endfor %}
        ]
    }

    {% if type.accessLevel == "public" %}public {% endif %}func changeSet(for {{ type.name|lowerFirst }}: {{ type.name }}) -> [{{ type.name }}.Property] {
        var changeSet: [{{ type.name }}.Property] = []
        {% for instanceVariable in type.instanceVariables|publicGet %}
        {% if not instanceVariable.annotations.ignoreInChangeSet %}
        {% if instanceVariable.annotations.skipDiff %}
          {% if instanceVariable.type.based.AutoPropertyDiffable %}
        changeSet.append(.{{ instanceVariable.name }}({{ type.name|lowerFirst }}.{{ instanceVariable.name }}.fullChangeSet))
          {% else %}
        changeSet.append(.{{ instanceVariable.name }}({{ type.name|lowerFirst }}.{{ instanceVariable.name }}))
          {% endif %}
        {% else %}
          {% if instanceVariable.type.based.AutoPropertyDiffable %}
            {% if instanceVariable.isOptional %}
        switch (self.{{ instanceVariable.name }}, {{ type.name|lowerFirst }}.{{ instanceVariable.name }}) {
        case (.some(let old), .some(let new)):
            let {{ instanceVariable.name }}ChangeSet = old.changeSet(for: new)
            if !{{ instanceVariable.name }}ChangeSet.isEmpty {
                changeSet.append(.{{ instanceVariable.name }}({{ instanceVariable.name }}ChangeSet))
            }
        case (.none, .some(let new)):
            changeSet.append(.{{ instanceVariable.name }}(new.fullChangeSet))
        case (.some(_), .none):
            changeSet.append(.{{ instanceVariable.name }}(.none))
        case (.none, .none):
            break
        }
            {% else %}
        let {{ instanceVariable.name}}ChangeSet = self.{{ instanceVariable.name }}.changeSet(for: {{ type.name|lowerFirst }}.{{ instanceVariable.name }})
        if !{{ instanceVariable.name}}ChangeSet.isEmpty {
            changeSet.append(.{{ instanceVariable.name }}({{ instanceVariable.name}}ChangeSet))
        }
            {% endif %}
          {% else %}
        if self.{{ instanceVariable.name }} != {{ type.name|lowerFirst }}.{{ instanceVariable.name }} {
            changeSet.append(.{{ instanceVariable.name }}({{ type.name|lowerFirst }}.{{ instanceVariable.name }}))
        }
          {% endif %}
        {% endif%}
        {% endif%}
        {% endfor %}
        return changeSet
    }

}

{% endfor %}
```

```swift
public extension ButtonProperties {

    public enum Property {

        case text(String?)
        case isActive(Bool)
        case icon(Image?)
        case onTap(MessageType?)

    }

    public var fullChangeSet: [ButtonProperties.Property] {
        return [
            .text(self.text),
            .isActive(self.isActive),
            .icon(self.icon),
            .onTap(self.onTap),
        ]
    }

    public func changeSet(for buttonProperties: ButtonProperties) -> [ButtonProperties.Property] {
        var changeSet: [ButtonProperties.Property] = []
        if self.text != buttonProperties.text {
            changeSet.append(.text(buttonProperties.text))
        }
        if self.isActive != buttonProperties.isActive {
            changeSet.append(.isActive(buttonProperties.isActive))
        }
        if self.icon != buttonProperties.icon {
            changeSet.append(.icon(buttonProperties.icon))
        }
        changeSet.append(.onTap(buttonProperties.onTap))
        return changeSet
    }

}
```

```
public extension ButtonProperties {

    public enum Property {

        case text(String?)
        case isActive(Bool)
        case icon(Image?)
        case onTap(MessageType?)

    }

    // ...

}
```

```
public extension ButtonProperties {

    // ...

    public var fullChangeSet: [ButtonProperties.Property] {
        return [
            .text(self.text),
            .isActive(self.isActive),
            .icon(self.icon),
            .onTap(self.onTap),
        ]
    }

    // ...

}
```

```swift
public extension ButtonProperties {

    // ...

    public func changeSet(for buttonProperties: ButtonProperties) -> [ButtonProperties.Property] {
        var changeSet: [ButtonProperties.Property] = []
        if self.text != buttonProperties.text {
            changeSet.append(.text(buttonProperties.text))
        }
        if self.isActive != buttonProperties.isActive {
            changeSet.append(.isActive(buttonProperties.isActive))
        }
        if self.icon != buttonProperties.icon {
            changeSet.append(.icon(buttonProperties.icon))
        }
        changeSet.append(.onTap(buttonProperties.onTap))
        return changeSet
    }

}
```

```swift
public extension ButtonProperties {

    // ...

    public func changeSet(for buttonProperties: ButtonProperties) -> [ButtonProperties.Property] {
        var changeSet: [ButtonProperties.Property] = []
        if self.text != buttonProperties.text {
            changeSet.append(.text(buttonProperties.text))
        }
        if self.isActive != buttonProperties.isActive {
            changeSet.append(.isActive(buttonProperties.isActive))
        }
        if self.icon != buttonProperties.icon {
            changeSet.append(.icon(buttonProperties.icon))
        }
        changeSet.append(.onTap(buttonProperties.onTap))
        return changeSet
    }

}
```

```swift
public extension ButtonProperties {

    // ...

    public func changeSet(for buttonProperties: ButtonProperties) -> [ButtonProperties.Property] {
        var changeSet: [ButtonProperties.Property] = []
        if self.text != buttonProperties.text {
            changeSet.append(.text(buttonProperties.text))
        }
        if self.isActive != buttonProperties.isActive {
            changeSet.append(.isActive(buttonProperties.isActive))
        }
        if self.icon != buttonProperties.icon {
            changeSet.append(.icon(buttonProperties.icon))
        }
        changeSet.append(.onTap(buttonProperties.onTap))
        return changeSet
    }

}
```

```
fileprivate extension UIButton {

    fileprivate func apply<MessageType>(changeSet: [ButtonProperties<MessageType>.Property]) {
        for property in changeSet {
            switch property {

            case .text(let text):
                self.setTitle(text, for: .normal)

            case .icon(let maybeIcon):
                if let icon = maybeIcon {
                    self.load(image: icon) { self.setImage($0, for: .normal) }
                } else {
                    self.setImage(.none, for: .normal)
                }

            case .isActive(let isActive):
                self.isSelected = isActive

            case .onTap(let onTap):
                if let message = onTap {
                    _ = self.on(event: .touchUpInside, dispatch: message)
                } else {
                    let _: MessageDispatcher<MessageType>? = self.stopDispatchingMessages(for: .touchUpInside)
                }
            }
        }
    }
}
```

```
case .text(let text):
    self.setTitle(text, for: .normal)

case .icon(let maybeIcon):
    if let icon = maybeIcon {
        self.load(image: icon) { self.setImage($0, for: .normal) }
    } else {
        self.setImage(.none, for: .normal)
    }
```

`AutoPropertyDiffable.generated.swift`

1074 lineas

Soucery y metaprogramming permiten generar código para ahorrar tiempo, evitar problemas de mantenibilidad y dejarnos invertir tiempo en tareas que agregan más valor

¿PREGUNTAS?

Soucery y metaprogramming permiten generar código para ahorrar tiempo, evitar problemas de mantenibilidad y dejarnos invertir tiempo en tareas que agregan más valor