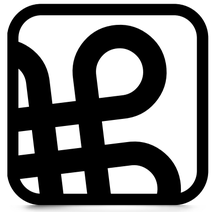


Macoun

Swift Reflection für Value Types

Benedikt Terhechte

@terhechte www.appventure.me

Ablauf

- Die Reflection API: Was kann sie?
- Fallbeispiel: Structs zu Core Data
- Optionals & Arrays unterstützen
- Generisch Typen unterstützen
- Geschwindigkeit
- Fazit

Die Reflection API

Mirror

```
Mirror(reflecting subject: Any)
```

- Erzeugt einen `struct Mirror` für das gegebene Subject.
- Mirror bietet Analyse-Methoden
- Any = Allgemeinster Typ (Value Type, Tuple, Class, Struct, etc)
- Swift-Intern (Default) oder `CustomReflectable Protokoll`

Mirror.displayStyle Property

- Gibt den Subjekt-Typ (Optional) zurück

```
public enum DisplayStyle {
    case Struct
    case Class
    case Enum
    case Tuple
    case Optional
    case Collection
    case Dictionary
    case Set
}
```

Mirror.children Property

- Etwaige Kind-Elemente (bei Arrays, Structs, Tuples, etc)

```
typealias Mirror.Child =
(label: String?, value: Any)

// Array: a.append(“Cup”)
// -> (label: nil, value: “Cup”)

// Struct: a.itemName = “Car”
// -> (label: “itemName”, value: “Car”)
```

Mutterklassen-Mirror

- Bei Klassen kann auch die Mutterklasse inspiziert werden

```
public func superclassMirror() -> Mirror?
```

CustomReflectable Protokoll

- Selbst einen Mirror erzeugen der das Subject beschreibt

CustomReflectable Protokoll

- Selbst einen Mirror erzeugen der das Subject beschreibt

```
 struct Mirror {
  init<T>(_ subject: T,
      children: DictionaryLiteral<String, Any>,
      displayStyle: Mirror.DisplayStyle?,
      ancestorRepresentation: Mirror.AncestorRepresentation)
}
```

CustomReflectable Protokoll

- Selbst einen Mirror erzeugen der das Subject beschreibt

```
public init<T>(_ subject: T,
      children: DictionaryLiteral<String, Any>,
      displayStyle: Mirror.DisplayStyle?,
      ancestorRepresentation: Mirror.AncestorRepresentation)
```

CustomReflectable Protokoll

```
public enum AncestorRepresentation {
        /// Generate a default mirror for all ancestor classes.
        case Generated
        /// Use the ancestor's implementation of `customMirror()`
        case Customized(() -> Mirror)
        /// Suppress the representation of all ancestor classes.
        case Suppressed
}
```

Mirror.descendant

- KVC-Ähnliche Lookups in Swift: á la `valueForKeyPath`:

```
struct Episode { let name: String }
struct Channel { let episodes: [Episode] }
struct Base { let channel: Channel? }

let example = Base(channel: Channel(episodes: [Episode(name: "Ep1"),
Episode(name: "Ep2")]))

let episode = Mirror(reflecting: example).
   descendant("channel", "Some", "episodes", 0, "name")
```

Mirror.descendant

- Base.channel (Optional)

```
struct Episode { let name: String }
struct Channel { let episodes: [Episode] }
struct Base { let channel: Channel? }

let example = Base(channel: Channel(episodes: [Episode(name: "Ep1"),
Episode(name: "Ep2")]))

let episode = Mirror(reflecting: example).
   descendant("channel", "Some", "episodes", 0, "name")
```

Mirror.descendant

- Optional.Some

```
 struct Episode { let name: String }
struct Channel { let episodes: [Episode] }
struct Base { let channel: Channel? }

let example = Base(channel: Channel(episodes: [Episode(name: "Ep1"),
Episode(name: "Ep2")]))

let episode = Mirror(reflecting: example).
   descendant("channel", "Some", "episodes", 0, "name")
```

Mirror.descendant

- Channel.episodes (Array)

```
struct Episode { let name: String }
struct Channel { let episodes: [Episode] }
struct Base { let channel: Channel? }

let example = Base(channel: Channel(episodes: [Episode(name: "Ep1"),
Episode(name: "Ep2")]))

let episode = Mirror(reflecting: example).
   descendant("channel", "Some", "episodes", 0, "name")
```

Mirror.descendant

- Array[0]

```
struct Episode { let name: String }
struct Channel { let episodes: [Episode] }
struct Base { let channel: Channel? }

let example = Base(channel: Channel(episodes: [Episode(name: "Ep1"),
Episode(name: "Ep2")]))

let episode = Mirror(reflecting: example).
   descendant("channel", "Some", "episodes", 0, "name")
```

Mirror.descendant

- Episode.name

```
struct Episode { let name: String }
struct Channel { let episodes: [Episode] }
struct Base { let channel: Channel? }

let example = Base(channel: Channel(episodes: [Episode(name: "Ep1"),
Episode(name: "Ep2")]))

let episode = Mirror(reflecting: example).
   descendant("channel", "Some", "episodes", 0, “name")
// episode = “Ep1”
```

Mirror.descendant

- Swift Dokumentation sagt: Langsam!
- Für Playground gedacht
- Eher nicht für Production Code

Was kann man mit Reflection machen

- Objekte Inspizieren (Inklusive Mutterklassen)
- Objekte Serialisieren
- CustomReflectable: Anpassen wie etwas im Playground angezeigt wird
- Was nicht geht ist das Schreiben von Werten

Fallbeispiel:
Structs zu Core Data

Structs zu Core Data

- Value Types nutzen
- Auch in der NSObject-Welt
- Swift Structs generalisiert in Core Data Objekte konvertieren

Szenario

- Neuestes Startup: “**Books Bunny**”
- Künstliche Intelligenz für Bookmarks
- Automatisch relevante besuchte Seiten bookmarken
- Viele Besuche (Value Types)
- Wenige Bookmarks (Bequem in Core Data)

Basis Bookmark Struct

```
struct Bookmark {
    let title: String
    let pagerank: Int
    let url: String
    let created: NSDate
}
```

Basis Protokoll

Basis Protokoll

```
protocol StructDecoder {
    static var EntityName: String { get }
    func toCoreData(context: NSManagedObjectContext)
         throws -> NSManagedObject
}
```

Protokoll-Implementierung

```
struct Bookmark: StructDecoder {
    static var EntityName = "Bookmark"
    let title: String
    let pagerank: Int
    let url: String
    let created: NSDate
}
```

Demo

Wrap Up: Wir haben

- Ein allgemeines Protokoll “`StructDecoder`" angelegt
- `toCoreData` Methode als Protocoll Extension angelegt
- Darin, mit Reflection über ein Struct iteriert

Bookmark Erweiterung

- Optionale Titel
- Keywords

Optionals

```
struct Bookmark: StructDecoder {
    static var EntityName = "Bookmark"
    let title: String?
    let pagerank: Int
    let url: String
    let created: NSDate
}
```

Arrays

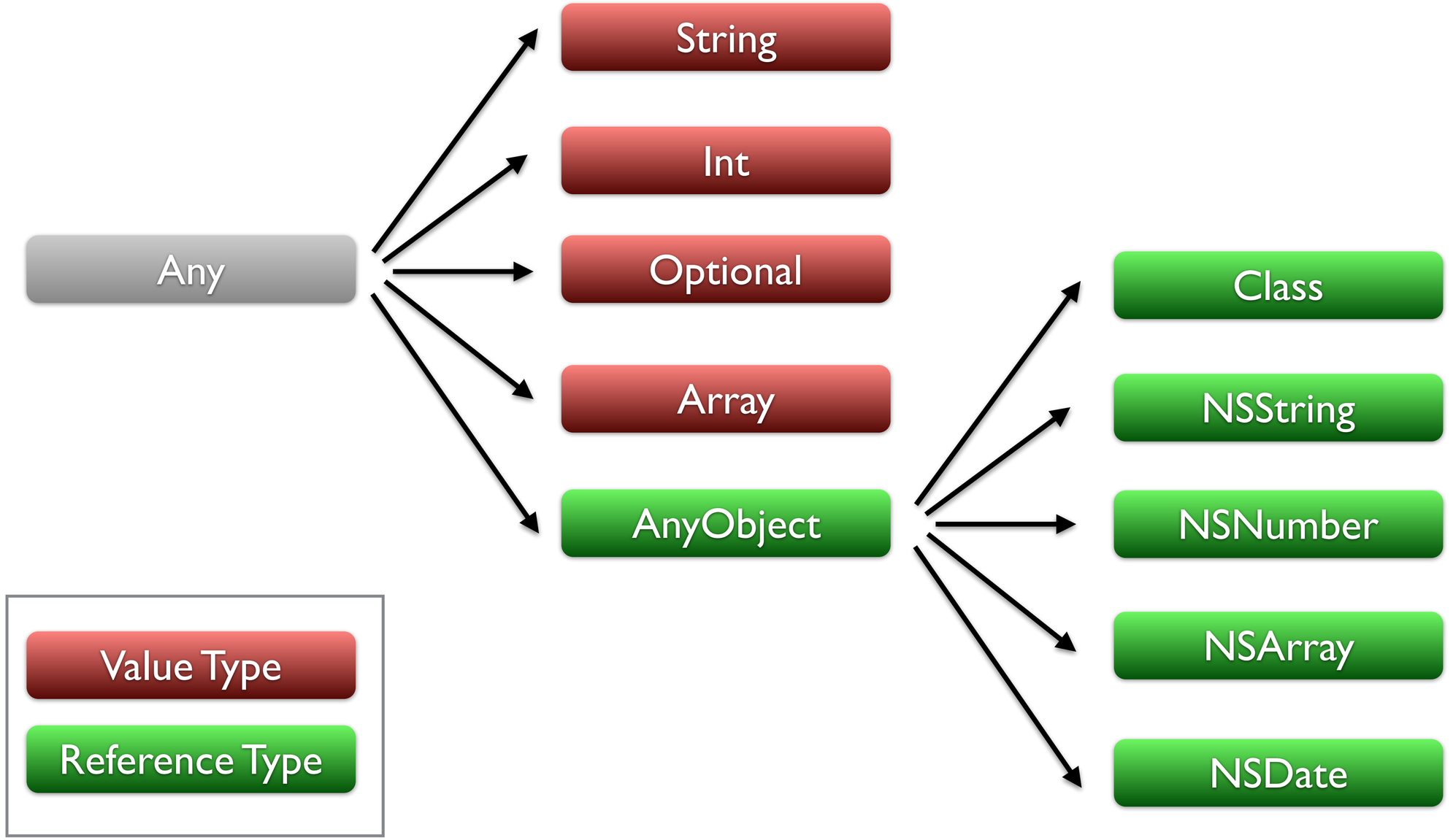
- Jedes Bookmark soll eine Liste an Keywords haben
- Core Data Relation (Ordered Set)

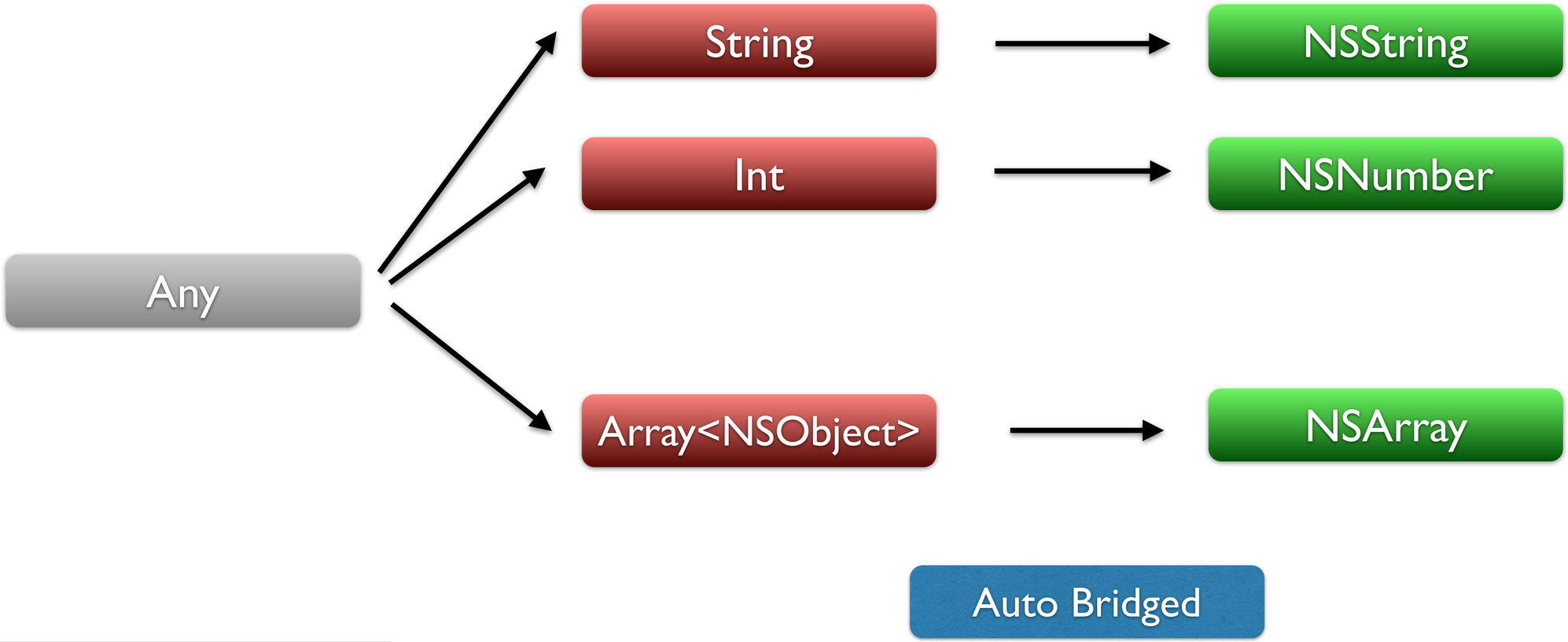
```
struct Bookmark: StructDecoder {
   static var EntityName = “Bookmark"
   …
   let keywords: [Keyword]
}

struct Keyword: StructDecoder {
    static var EntityName = "Keyword"
    let name: String
}
```

NSManagedObject

```
public func setValue(value: AnyObject?, forKey key: String)
```


Any
String
Int
Optional
Array
AnyObject
Class
NSString
NSNumber
NSArray
NSDate
Value Type
Reference Type


String
NSString
Int
NSNumber
Any
Array<NSObject>
NSArray
Auto Bridged
Value Type
Reference Type

Problem 1

- Optionals sind Value Types ohne NSObject alternative
- Nicht ‘AnyObject’
- Können nicht einfach so mit ‘setValueForKey’ genutzt werden.

Problem 2

- Arrays sind auch nicht “AnyObject”
- Ausser Arrays mit NSObject-Typen, diese werden konvertiert
- Wir haben aber ein Array mit Structs

Demo

Wrap Up: Wir haben

- Einen weiteren Mirror erzeugt um Optionals & Arrays zu identifizieren
- Per Pattern Matching die verschiedenen Fälle unterschieden
- Die Inhalte des Arrays rekursiv konvertiert

Weitere Typen unterstützen

- Generischer Weg zur Typ-Konvertierung
- Beispiel: Ein `enum`, dass eine Bookmark-Gruppe definiert

Generisch Typen unterstützen

```
enum BookmarkGroup: String {
    case Business = "Business"
    case News = "News"
}

struct Bookmark: StructDecoder {
    static var EntityName = "Bookmark"
    ...
    let group: BookmarkGroup
}
```

Demo

Wrap Up: Wir haben

- Ein neues Protokoll “`StructDecodable`” erstellt
- Methode `decodeInto`
- Den Matching Code an das Protokoll angepasst
- Mittels Protocoll Extensions unsere Typen konform gemacht

Erweiterung

- Damit lassen sich auch komplexe Fälle abdecken

```
struct Bookmark: StructDecoder {
    static var EntityName = "Bookmark"
    let group: [BookmarkGroup]
}

extension CollectionType where Self.Generator.Element :
StructDecodable {
    func decodeInto(object: NSManagedObject, withKey key: String) {
        self.first?.decodeInto(object, withKey: key)
    }
}
```

Geschwindigkeit

- Wie schnell / langsam ist die Reflection?

Geschwindigkeit

- Wie schnell / langsam ist die Reflection?
- 2000 NSManagedObject anlegen: 0,062 Sekunden
- 2000 Bookmarks anlegen / konvertieren: 0,207 Sekunden
- Fast 3,5x langsamer

Geschwindigkeit

- Was, wenn man `CustomReflectable` implementiert?

```
func customMirror() -> Mirror {
    let children = DictionaryLiteral<String, Any>(dictionaryLiteral:
     ("title", self.title), ("pagerank", self.pagerank),
     ("url", self.url), ("created", self.created),
     ("keywords", self.keywords), ("group", self.group))

      return Mirror.init(Bookmark.self, children: children,
          displayStyle: Mirror.DisplayStyle.Struct,
          ancestorRepresentation:.Suppressed)
}
```

Geschwindigkeit

- Wie schnell / langsam ist die Reflection?
- 2000 NSManagedObject anlegen: 0,062 Sekunden
- 2000 Bookmarks anlegen / konvertieren: 0,207 Sekunden
- **2000 Bookmarks anlegen / konvertieren: 0,203 Sekunden**
- Fast 3,5x langsamer

Fazit

Begrenzte Möglichkeiten

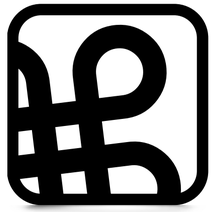
- Nur Lese-Zugriff
- Für Schreib-Zugriffe auf NSObject-Basierte Typen (@objc) und `setValueForKey` ausweichen (wenn möglich)
- Fast 4x langsamer als direkte Objekt-Nutzung

Mehr Info

- Vollständige Implementierung:
- CoreValue (https://github.com/terhechte/CoreValue)

Fragen?

Vielen Dank



Macoun