Introspecting Swift

dotSwift 2015, JP Simard, @simjp

Why?

Mantle

```
@interface GHIssue : MTLModel <MTLJSONSerializing>

@property GHUser *assignee;
@property NSDate *updatedAt;

@property NSString *title;
@property NSString *body;

@property NSDate *retrievedAt;

@end
```

FCModel & Realm

```
@interface Employee : RLMObject

@property NSString *name;
@property NSDate *startDate;
@property float salary;
@property BOOL fullTime;

@end
```

How?

The six ways to **introspect Swift**

1. Stick with compile-time types & constraints
2. Apply dynamic casting
3. Leverage Swift's **`MirrorType`**
4. Abuse Objective-C's runtime
5. Use private functions
6. Resort to inspecting memory layout

The six degrees of **evil**

1. Stick with compile-time types & constraints – 👍🏻
2. Apply dynamic casting – 👌🏻
3. Leverage Swift's **`MirrorType`** – 😕
4. Abuse Objective-C's runtime – 😬
5. Use private functions – 😨
6. Resort to inspecting memory layout – 😱

Compile-time
Types & Constraints

QueryKit

```
struct MyStruct {
  let intProp: Int

  struct Attributes {
    static let intProp = Attribute<Int>("intProp")
  }
}

// Usage
let intProp = MyStruct.Attributes.intProp
intProp > 0 // NSPredicate("intProp > 0"), typesafe
```

Argo

```
extension User: JSONDecodable {
  static func create(name: String)(email: String?)(role: Role)
                    (friends: [User]) -> User {
    return User(name: name, email: email, role: role, friends: friends)
  }

  static func decode(j: JSONValue) -> User? {
    return User.create
      <^> j <| "name"
      <*> j <|? "email" // Use ? for parsing optional values
      <*> j <| "role" // Custom types conforming to JSONDecodable work
      <*> j <|| "friends" // parse arrays of objects
  }
}
```

Dynamic Casting

as?

```swift
protocol XPCConvertible {}

extension Int64: XPCConvertible {}
extension String: XPCConvertible {}

func toXPC(object: XPCConvertible) -> xpc_object_t? {
  switch(object) {
    case let object as Int64:
      return xpc_int64_create(object)
    case let object as String:
      return xpc_string_create(object)
    default:
      fatalError("Unsupported type for object: \(object)")
      return nil
  }
}
```

Official Swift Reflection

```swift
/// The type returned by `reflect(x)`; supplies an API for runtime reflection on `x`
protocol MirrorType {
    /// The instance being reflected
    var value: Any { get }
    /// Identical to `value.dynamicType`
    var valueType: Any.Type { get }
    /// A unique identifier for `value` if it is a class instance; `nil` otherwise.
    var objectIdentifier: ObjectIdentifier? { get }
    /// The count of `value`\ 's logical children
    var count: Int { get }
    subscript (i: Int) -> (String, MirrorType) { get }
    /// A string description of `value`.
    var summary: String { get }
    /// A rich representation of `value` for an IDE, or `nil` if none is supplied.
    var quickLookObject: QuickLookObject? { get }
    /// How `value` should be presented in an IDE.
    var disposition: MirrorDisposition { get }
}
```

```swift
struct MyStruct {
    let stringProp: String
    let intProp: Int
}

let reflection = reflect(MyStruct(stringProp: "a", intProp: 1))

for i in 0..<reflection.count {
    let propertyName = reflection[i].0
    let value = reflection[i].1.value
    println("\(propertyName) = \(value)")
    if let value = value as? Int {
        println("int")
    } else if let value = value as? String {
        println("string")
    }
}

// stringProp = a
// string
// intProp = 1
// int
```

Objective-C Runtime

```swift
import Foundation

class objcSub: NSObject {
    let string: String?
    let int: Int?
}

var propCount: UInt32 = 0
let properties = clazz_copyPropertyList(objcSub.self, &propCount)

for i in 0..<Int(propCount) {
    let prop = properties[i]
    String.fromCString(property_getName(prop))! // => "string"
    String.fromCString(property_getAttributes(prop))! // => "T@,N,R,Vstring"
}
```

Using Private Functions

```
nm -a libswiftCore.dylib | grep "stdlib"

> ...
> __TFSs28_stdlib_getDemangledTypeNameU__FQ_SS
> ...
> _swift_stdlib_conformsToProtocol
> _swift_stdlib_demangleName
> _swift_stdlib_dynamicCastToExistential1
> _swift_stdlib_dynamicCastToExistential1Unconditional
> _swift_stdlib_getTypeName
> ...
```

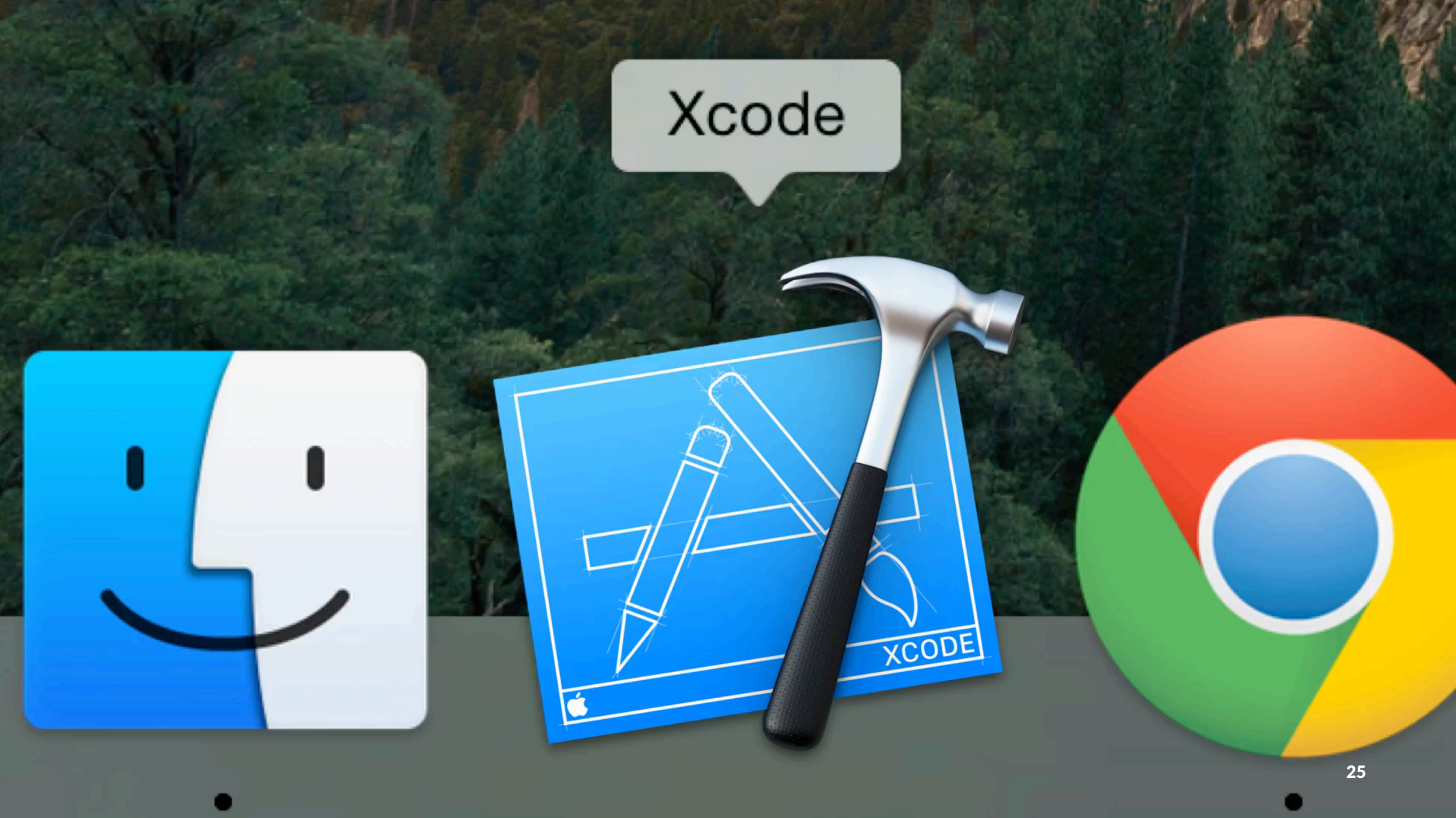
```swift
struct MyStruct {
    let stringProp: String
    let intProp: Int
}

let reflection = reflect(MyStruct(stringProp: "a", intProp: 1))

for i in 0..<reflection.count {
    let propertyName = reflection[i].0
    let value = reflection[i].1.value
    println("\(propertyName) = \(value)")
    println(_stdlib_getDemangledTypeName(value))
}

// stringProp = a
// Swift.String
// intProp = 1
// Swift.Int
```

Inspecting Memory Layout


Xcode
XCODE

```
struct _swift_data {
    unsigned long flags;
    const char *className;
    int fieldcount, flags2;
    const char *ivarNames;
    struct _swift_field **(*get_field_data)();
};

struct _swift_class {
    union {
        Class meta;
        unsigned long flags;
    };
    Class supr;
    void *buckets, *vtable, *pdata;
    int f1, f2; // added for Beta5
    int size, tos, mdsize, eight;
    struct _swift_data *swiftData;
    IMP dispatch[1];
};
```

```
class GenericClass<T> {}

class SimpleClass: NSObject {}

class ParentClass {
    let boolProp: Bool? // Optionals
    let intProp: Int    // Without default value
    var floatProp = 0 as Float // With default value
    var doubleProp = 0.0
    var stringProp = ""
    var simpleProp = SimpleClass()
    var genericProp = GenericClass<String>()
}
```

```
{
  "boolProp": "b",
  "intProp": "i",
  "floatProp": "f",
  "doubleProp": "d",
  "stringProp": "S",
  "simpleProp": "ModuleName.SimpleClass",
  "genericProp": "[Mangled GenericClass]"
}
```

Why?

RealmSwift

```
class Employee: Object {
  dynamic var name = "" // you can specify defaults
  dynamic var startDate = NSDate()
  dynamic var salary = 0.0
  dynamic var fullTime = true
}

class Company: Object {
  dynamic var name = ""
  dynamic var ceo: Employee? // optional
  let employees = List<Employee>()
}
```

The six ways to **introspect Swift**

1. Stick with compile-time types & constraints – 👍🏻
2. Apply dynamic casting – 👌🏻
3. Leverage Swift's **`MirrorType`** – 😕
4. Abuse Objective-C's runtime – 😬
5. Use private functions – 😨
6. Resort to inspecting memory layout – 😱

Links (1/2)

- This talk: *github.com/jpsim/talks*
- Mantle: *github.com/Mantle/Mantle*
- FCModel: *github.com/marcoarment/FCModel*
- Realm: *github.com/realm/realm-cocoa*
- QueryKit: *github.com/QueryKit/QueryKit*
- Argo: *github.com/thoughtbot/Argo*

Links (2/2)

- Dynamic Casting: *blog.segiddins.me*
- SwiftXPC: *github.com/jpsim/SwiftXPC*
- MirrorType Docs: *swiftdoc.org/protocol/MirrorType*
- Russ Bishop on horrible things: *russbishop.net*
- Injection for Xcode: *injectionforxcode.com*
- SwiftIvarTypeDetector: *github.com/jpsim/SwiftIvarTypeDetector*

Thank You!

`dotSwift().questions?.askThem!`

JP Simard, *@simjp*, *realm.io*