Closures

peter kos | WT | 2/7/23

Async recap

Closures 101

Closures 506

Async recap

Lots of different ways to do async **safely**

Lots of different ways to do async **safely**

literally `async`

closures

actors

using Rust

Closures 101

```
let networkRequest =  ...

networkRequest.perform { (response) in
    print("Response: \(response)")
}
```

```
let networkRequest =  ...

networkRequest.perform { (response) in
    print("Response: \(response)")
}
```

```
let networkRequest =  ...


networkRequest.perform(completion: { (response) in
    print("Response: \(response)")
})
```

```
func perform(completion: () → (String)) {
    // …
}

networkRequest.perform(completion: { (response) in
    print("Response: \(response)")
})
```

```
func perform(completion: () → (String)) {
    // …
}
```

Paramater that accepts a **function**

Paramater that accepts a **function**

“ ... drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list.”

Paramater that accepts a **function**

Removing currying `func` declaration syntax

- Proposal: SE-0002
- Author: Joe Groff
- Status: **Implemented (Swift 3)**
- Implementation: apple/swift@983a674

Introduction

Curried function declaration syntax `func foo(x: Int)(y: Int)` is of limited usefulness and creates a lot of language and implementation complexity. We should remove it.

Common closure patterns

Syntactic sugar (and cinnamon)

Syntactic sugar (and cinnamon)

```
request.perform(completion: { response in … })
request.perform { response in … }
request.perform { _ in … }
```

Result

Result

```
func getUsers(completion: () → Result<UserDTO, NetworkError>) {
    //  ...
}
```

```
func getUsers(completion: () → Result<UserDTO, NetworkError>) {
    //  ...
}

 request.getUsers { result in
     switch result {
         case .success(let userDTO):
             //  ...
         case .error(let error):
             //  ...
     }
 }
```

with love, from Rust

```
request.getUsers { result in
    switch result {
        case .success(let userDTO):
            // ...
        case .error(let error):
            // ...
    }
}
```

with love, from Rust

```
let userData = match request {
    Ok(userDTO) ⇒ userDTO,
    Err(Error) ⇒ nil
}
```

```
request.getUsers { result in
    switch result {
        case .success(let userDTO):
            // ...
        case .error(let error):
            // ...
    }
}
```

```
func getUsers(completion: () → Result<UserDTO, NetworkError>) {
    //  ...
}

 request.getUsers { result in
     switch result {
         case .success(let userDTO):
             //  ...
         case .error(let error):
             //  ...
     }
 }
```

weak self

weak self

```
request.perform { [weak self] in
    guard let self = self else { return }
    //  ...
}
```

weak self


```
request.perform { [weak self] in
    guard let self else { return }
    //  ...
}
```

Closures 506

1. Weak vs. unowned
2. Capture lists
3. @escaping
4. @autoclosure

weak ~= optional

unowned ~= implicitly unwrapped optional

Capture lists

Capture lists

```
request.perform { [weak self] response in ... }
```

Capture lists

```
request.perform { [weak self] response in ... }
request.perform { [weak userArray] response in ... }
```

Capture lists

```
request.perform { [weak self] response in ... }
request.perform { [weak userArray] response in ... }
request.perform { [weak userArray, settings] response in ... }
```

@escaping

When the closure is called

by something else

after the function returns

@escaping

```
func getSlackUsers(completion: @escaping (Result< ... >) → Void) {




}
```

@escaping

```
func getSlackUsers(completion: @escaping (Result< ... >) → Void) {

    // Some setup
    let request =  ...
    URLSession.dataTask(with: request) { result in
        //  ...
        completion(result)
    }
}
```

@autoclosure

Auto-wraps a normal func arg to a closure

1. Weak vs. unowned
2. Capture lists
3. @escaping
4. @autoclosure

Closures

peter kos | WT | 2/7/23