Documentation Language: Swift

Hummingbird / Application

Structure

Application

Application type bringing together all the components of Hummingbird

```
struct Application<Responder> where Responder : HTTPResponder, Responder.Con
FromSource, Responder.Context.Source == ApplicationRequestContextSource
```

Overview

`Application` is a concrete implementation of `ApplicationProtocol`. It provides the glue between your router and the HTTP server.

```
// create router
let router = Router()
router.get("hello") { _,_ in
    return "hello"
}
// create application
let app = Application(
    router: router,
    server: .http1()    // This is the default value
)
// run application
try await app.runService()
```

Generic Type

`Application` is a generic type, if you want to pass it around it is easier to use the opaque type `some ApplicationProtocol` than work out its exact parameters types.

```
func buildApplication() -> some ApplicationProtocol {
    let router = Router()
    router.get("hello") { _,_ in
        return "hello"
    }
    // create application
    let app = Application(router: router)
}
```

Services

`Application` has its own `ServiceGroup` which is used to manage the lifecycle of all the services it creates. You can add your own services to this group to have them managed as well.

```
var app = Application(router: router)
app.addServices(postgresClient, jobQueueHandler)
```

Check out swift-service-lifecycle for more details on service lifecycle management.

Topics

Initializers

`init(responder: Responder, server: HTTPServerBuilder, configuration: ApplicationConfiguration, services: [Service], onServerRunning: (Channel) async -> Void, eventLoopGroupProvider: EventLoopGroupProvider, logger: Logger?)`

Initialize new Application

```
init<ResponderBuilder>(router: ResponderBuilder, server: HTTPServer
Builder, configuration: ApplicationConfiguration, services: [Service
], onServerRunning: (Channel) async -> Void, eventLoopGroupProvider:
EventLoopGroupProvider, logger: Logger?)
```

Initialize new Application

Instance Properties

```
var configuration: ApplicationConfiguration
```

Configuration

```
let eventLoopGroup: EventLoopGroup
```

event loop group used by application

```
var logger: Logger
```

Logger

```
var processesRunBeforeServerStart: [() async throws -> Void]
```

Processes to be run before server is started

```
let responder: Responder
```

routes requests to responders based on URI

```
let server: HTTPServerBuilder
```

Server channel setup

```
var services: [any Service]
```

services attached to the application.

Instance Methods

```
func addServices(any Service...)
```

Add service to be managed by application ServiceGroup

```
func beforeServerStarts(perform: () async throws -> Void)
```

Add a process to run before we kick off the server service

```
func buildResponder() async throws -> Responder
```

```
func onServerRunning(Channel) async
```

This is called once the server is running and we have an active Channel

Default Implementations

- ApplicationProtocol Implementations
- CustomStringConvertible Implementations

Relationships

Conforms To

```
ApplicationProtocol
ServiceLifecycle.Service
Swift.Copyable
Swift.CustomStringConvertible
Swift.Sendable
```

See Also

Application

```
protocol ApplicationProtocol
```

Application protocol bringing together all the components of Hummingbird

```
struct ApplicationConfiguration
```

Application configuration

```
enum EventLoopGroupProvider
```

Where should the application get its EventLoopGroup from