

Documentation

[Hummingbird Document...](#) / Service Lifecycle

Article

Service Lifecycle

Integration with Swift Service Lifecycle



Overview

To provide a mechanism to cleanly start and shutdown a Hummingbird application we have integrated with [Swift Service Lifecycle](#). This provides lifecycle management for service startup, shutdown and shutdown triggering by signals such as SIGINT and SIGTERM.

Service Lifecycle

To use Swift Service Lifecycle you have to conform the service you want managed to the protocol [Service](#). Internally this needs to call `withGracefulShutdownHandler` to handle graceful shutdown when we receive a shutdown signal.

```
struct MyService: Service {
    func run() async throws {
        withGracefulShutdownHandler {
            // run service
        } onGracefulShutdown {
            // shutdown service
        }
    }
}
```

Once you have this setup you can then include the service in a list of services added to a service group and have its lifecycle managed.

```
let serviceGroup = ServiceGroup(  
  configuration: .init(  
    services: [MyService(), MyOtherService()],  
    gracefulShutdownSignals: [.sigterm, .sigint]  
    logger: logger  
  )  
)  
try await serviceGroup.run()
```

Hummingbird Integration

Application conforms to `Service` and also provides a helper function that constructs the `ServiceGroup` including the application and then runs it.

```
let app = Application(router: router)  
try await app.runService()
```

All of the types that Hummingbird introduces that require some form of lifecycle management conform to `Service`. Application holds an internal `ServiceGroup` and any service you want managed can be added to the internal group using add Services(_:).

```
var app = Application(router: router)  
app.addServices(postgresClient, sessionStorage)  
try await app.runService()
```

Managing server startup

In some situations you might want some services to start up before you startup your HTTP server, for instance when doing a database migration. With Application you can add processes to run before starting up the server, but while other services are running using

`beforeServerStarts(perform:)`. You can call `beforeServerStarts` multiple times to add multiple processes to be run before we startup the server.

```
var app = Application(router: router)
app.addServices(dbClient)
app.beforeServerStarts {
    try await dbClient.migrate()
}
try await app.runService()
```

Read the Swift Service Lifecycle [documentation](#) to find out more.






See Also

Related Documentation

`struct Application`

Application type bringing together all the components of Hummingbird

Hummingbird Server

-  Router
The router directs requests to their handlers based on the contents of their path.
-  Request Decoding
Decoding of Requests with JSON content and other formats.
-  Response Encoding
Writing Responses using JSON and other formats.
-  Request Contexts
Controlling contextual data provided to middleware and route handlers
-  Middleware
Processing requests and responses outside of request handlers.



Error Handling

How to build errors for the server to return.



Logging, Metrics and Tracing

Considered the three pillars of observability, logging, metrics and tracing provide different ways of viewing how your application is working.



Result Builder Router

Building your router using a result builder.



Server protocol

Support for TLS and HTTP2 upgrades



Testing

Using the HummingbirdTesting framework to test your application



Persistent data

How to persist data between requests to your server.



Migrating to Hummingbird v2

Migration guide for converting Hummingbird v1 applications to Hummingbird v2
