Akka-HTTPで作るAPIサーバ

2018-11-10 Scala Kansai Summit - kamijin-fanta

諸注意

- スライド公開します
- 撮影NG (イベントのレギュレーション)
- Twitter: #scala_ks #s1

自己紹介

Scala関西Summit運営

- Github: kamijin-fanta
- Twitter: kamijin_fanta

Akka HTTP

- HTTP Server
  - HTMLテンプレート・フォーム等を使用したWebサイト
  - クライアント・バックエンドが分離したAPIサーバ
- HTTP Client
  - スクレイピング
  - マイクロサービスの他のエンドポイントの呼び出し
- HTTP Serverについて主に紹介

Akka HTTP is...

- Webフレームワーク
  - 例
    - Play, Rails, Django, Laravel
  - 機能
    - ルーティング・テンプレート・データベース・セッション・認証・テスト

Akka HTTP is HTTP Library

- Akka HTTPはWebフレームワークではない
  - 有: ルーティング・テスト
  - 無: テンプレート・データベース・セッション・認証

Why Akka HTTP

- 何故機能の少ないAkka HTTPを選択するのか
- Webフレームワークが適切でないユースケースが存在
  - 大きなビジネスロジックが存在し、APIが付加的なものである
  - RDB(MySQL等)ではなくKVS(Dynamo等)を使用する
  - 認証に独自のSSOを使用する
  - 気に入ったライブラリ(twirl, circe等)を使用したい
  - フレームワークのアップグレードコストが無視できない
- プログラミングの考え方
  - 他のWebフレームワーク: HTTPのサービスを記述する
  - Akka HTTP: アプリケーションを記述し、HTTPと統合する

Beginner friendly

- Akka HTTPは2つの側面が有る
  - 大規模サービス構築に向く
    - DB・セッション管理なども1から作れる
    - クリーンアーキテクチャ等の設計手法と親和性が高い
    - 単機能なので破壊的変更が少なく、アップグレードコストが低い
  - 単機能で覚える機能が少ない
    - Scalaの習得後のステップで扱うのに適している

Recommended for...

- こんな人におすすめ
  - フレームワークが負担になってきた
    - アップグレード・独自の認証・スケーリング
  - シンプルな機能の組み合わせでHTTPサーバを作りたい
    - 単機能なディレクティブを組み合わせる
  - Scalaの言語をある程度覚え、何を作ろうか迷っている
    - ライブラリ特有の機能等の覚えるべきことが少ない

Understand code

Install

- Requirement
  - sbt
  - JDK

```
// build.sbt
libraryDependencies ++= Seq(
  "com.typesafe.akka" %% "akka-http"   % "10.1.5",
  "com.typesafe.akka" %% "akka-stream" % "2.5.12",
)
```

Basic Server

```
object HttpServerUseHttpApp {
  object WebServer extends HttpApp {
    override def routes: Route =
      path("hello") {
        get {
          val entity = HttpEntity(
            ContentTypes.`text/html(UTF-8)`,
            "<h1>hello akka-http</h1>"
          )
          complete(entity)
        }
      }
  }

  def main(args: Array[String]): Unit =
    WebServer.startServer("localhost", 8080)
}
```

```
$ sbt
> run
```

```
$ curl localhost:8080/hello
<h1>hello akka-http</h1>
```

Model, Types

- リクエスト・レスポンスに対応する型が定義されている
  - Request, Response, Uri, Header(Accept, Cookie) etc...
- Scalaのパターンマッチングを行うことができる
  - 例:Reqに含まれるAcceptヘッダからUEで許可されているMIMEタイプを抽出
- 多数の型が定義されているので、一部だけ紹介

Methods

- `HttpMethods.CONNECT`
- `HttpMethods.DELETE`
- `HttpMethods.GET`
- `HttpMethods.HEAD`
- `HttpMethods.OPTIONS`
- `HttpMethods.PATCH`
- `HttpMethods.POST`
- `HttpMethods.PUT`
- `HttpMethods.TRACE`

Uri / Query

```
assert(
  Uri("https://akka.io:443/try-akka/") ===
    Uri.from(scheme = "https", host = "akka.io", port = 443, path = "/try-akka/")
)

assert(Query("key1=value1&key2=Foo&key2=Bar").get("key1") === Some("value1"))

assert(Query("key=Foo&key=Bar").getAll("key") === List("Bar", "Foo"))
```

Headers

```
RawHeader("x-custom-header", "value")
headers.Host("example.com")
headers.`Access-Control-Allow-Origin`(HttpOrigin("https://example.com"))
```

Routing DSL

```
object RoutingBasic extends HttpApp {
  def main(args: Array[String]): Unit =
    startServer("localhost", 8080)

  override def routes: Route =
    pathPrefix("hello") {
      get {
        complete("get hello")
      } ~ post {
        complete("post hello")
      }
    } ~ ((get | post) & path("user" / Segment)) { userName =>
      complete(s"UserName: $userName")
    }
}
```

- Directive(pathPrefix, get, post, complete, path)を組み合わせてRouteを作る
- 入れ子にすることや、 `~` `&` `|` で連結することが出来る

What is Directive?

```
// 1. 内部のルートに委譲または、拒否しフィルタリングを行う
filter(args...) {
  ???
}

// 2. 値を抽出し、内部のルートに渡す
extract(args...) { variable =>
  ???
}

// 3. コンテンツを返す
complate(???)
```

2つ以上の性質を組み合わせたディレクティブも存在

Composing Route

```
override def routes: Route =
  pathPrefix("hello") {
    get {
      complete("get hello")
    } ~ post {
      complete("post hello")
    }
  } ~ ((get | post) & path("user" / Segment)) { userName =>
    complete(s"UserName: $userName")
  }
```

- `~`: ルートで拒否されたときに、次のRouteへ処理を移す
- `&`: 両方のディレクティブの条件を満たす必要がある
- `|`: どちらかのディレクティブの条件を満たす必要がある

```scala
val getOrPostUser = (get | post) & path("user" / Segment)

override def routes: Route =
  pathPrefix("hello") {
    get {
      complete("get hello")
    } ~ post {
      complete("post hello")
    }
  } ~ getOrPostUser { userName =>
    complete(s"UserName: $userName")
  }
```

Test Kit

```
libraryDependencies ++= Seq(
  "com.typesafe.akka" %% "akka-http-testkit" % "10.1.3" % Test,
  "org.scalatest" %% "scalatest" % "3.0.5" % Test
)
```

```
class SimpleHttpServerSpec extends FunSpec with ScalatestRouteTest {
  it("basic test") {
    val route = path("hello") & get {
        complete(HttpEntity(ContentTypes.`text/html(UTF-8)`,
          "<h1>Say hello to akka-http</h1>"))
      }
    Get("/hello") ~> route ~> check {
      assert(status === StatusCodes.OK)
      assert(responseAs[String] === "<h1>Say hello to akka-http</h1>")
      assert(contentType === ContentTypes.`text/html(UTF-8)`)
    }
  }
}
```

- `Get("/hello")` はHttpRequestを作るためのショートハンド
  - `Get` `Post` `Put` `Patch` `Delete` `Options` `Head` もある

```
object HttpRequest {
  def apply(
    method:   HttpMethod                    = HttpMethods.GET,
    uri:      Uri                           = Uri./,
    headers:  immutable.Seq[HttpHeader] = Nil,
    entity:   RequestEntity                 = HttpEntity.Empty,
    protocol: HttpProtocol                  = HttpProtocols.`HTTP/1.1`) = new HttpRequest(me
}
```

Easy to test

- DirectiveはRequestから値の抽出・フィルタリングを行う純粋関数→テストしやすい
- 任意のHttpRequestをRouteに投げた結果を検証する仕組みがTestKitで用意されている
  - 簡易なテストを楽に書くためのショートハンドも存在

Rejection

```
  override def routes: Route =
    path("hello") {
      get {
        complete("get hello")
      } ~ post {
        complete("post hello")
      }
    }
```

```
$ curl -X DELETE localhost:8080/hello

HTTP method not allowed, supported methods: GET, POST
```

- 適切なエラーをクライアントに返すための仕組み

Provide Rejections

```
  override def routes: Route =
    path("foo") {
      reject
    } ~ path("bar") {
      reject(MissingQueryParamRejection("rejection reason"))
    }
```

```
final case class MissingQueryParamRejection(parameterName: String)
  extends jserver.MissingQueryParamRejection with Rejection

trait Rejection
```

```
$ curl localhost:8080/bar
Request is missing required query parameter 'rejection reason'
```

Handle Rejection

- 標準ではデフォルトのRejectionHandlerが使用されている
- 上書きor拡張することで、カスタムのエラーメッセージを返すことが可能
  - `Route.seal` でラップし、implicit valを注入する

```scala
implicit def myRejectionHandler: RejectionHandler =  // point 1
  RejectionHandler.newBuilder()
    .handle {
      case reject: MissingQueryParamRejection =>
        complete(
          StatusCodes.BadRequest,
          s"required query parameter [${reject.parameterName}]")
    }.result()

override def routes: Route = Route.seal(internalRoutes)  // point 2

def internalRoutes: Route =
  path("bar") {
    reject(MissingQueryParamRejection("rejection reason"))
  }
```

```scala
final case class MissingQueryParamRejection(parameterName: String)
  extends jserver.MissingQueryParamRejection with Rejection
```

- エラーを型を使って表現することが出来る
- RejectionHandlerで、エラーメッセージを容易にカスタマイズ可能
  - JSON/HTML/XMLでのエラー対応等

Marshall

- Marshal
  - オブジェクトをシリアライズする仕組み
  - 文字列・バイト列などに変換するMarshallerを定義する

```scala
import io.circe.generic.auto._
import io.circe.syntax._

case class User(name: String, age: Int)

implicit val userMarshaller: ToEntityMarshaller[User] = Marshaller.opaque { user =>
  HttpEntity(ContentTypes.`application/json`, user.asJson.noSpaces)
}

val route: Route = get {
  complete(User("mika", 20))
}

Get("/") ~> route ~> check {
  assert(contentType === ContentTypes.`application/json`)
  assert(responseAs[String] === """{"name":"mika","age":20}""")
}
```

Examples

- https://github.com/kamijin-fanta/akka-http-2018/tree/master/src/main/scala/examples
- https://github.com/kamijin-fanta/akka-http-2018/tree/master/src/test/scala/examples

Summary

- Route DSL
- Model
- Directive
- Rejection
- Marshall
- Test Kit

Akka HTTPはこれらのパーツを組み合わせ、
シンプルなHTTPサーバ構築をサポートするライブラリ

技術書典５新刊

kinyou_benkyokai vol.1

こんな人にオススメ！

- 重厚なHTTPフレームワークが負担で、薄いライブラリでAPIを作りたい
- 開発しているAPIサーバに手軽なリクエストテストを行いたい
- C#がどんな中間言語で動いているかを理解したい
- OSPF,BGP,RIPでもないルーティングプロトコルを学びたい

頒布情報

頒布場所 技術書典5　2018年10月8日　き１２

ダウンロードカードをお持ちの方はこちら

書籍＋電子版	電子版
700 yen	500 yen
pixiv Booth	pixiv Booth
技術書典5 き１２	技術書典5 き１２
	ダウンロードサイト

https://kinyoubenkyokai.github.io/book/techbook05/

ありがとうございました

Akka-HTTPで作るAPIサーバ

2018-11-10 Scala Kansai Summit - kamijin-fanta