学習

playframework

Free unaffiliated eBook created from
Stack Overflow contributors.

#playframework

1
1: playframework 2
2
Examples 2
Play 1 2
2
2
2
Mac OS X 2
Linux 3
Windows 3
`sbt 3
Play 2.4.x / 2.5.x - WindowsJava 4
4
Play 2.5 4
CLI 5
6
2: Java - Hello World 7
7
Examples 7
7
7
7
Hello World "Hello World" 9
3: Java - JSON 11
11
Examples 11
JSON 11
/json 11
11
11

JSON.....11
.....12
.....12
.....12
.....12
.....12
JSONJava.....13
JSONJava.....13
JavaJSON.....13
JSONJSON.....13
JSON.....13
4: JSON - Scala.....15
.....15
Examples.....15
JSON.....15
JavaJSON.....16
JavaBodyParserJSON.....16
ScalaJSON.....16
.....17
/.....17
Json.....17
Json.....18
5: WSClientWeb.....19
.....19
Examples.....19
Scala.....19
6: IDE.....20
Examples.....20
IDEA.....20
.....20
.....20

Intellij 20
..... 20
Play IDEEclipse - JavaPlay 2.4,2.5 21
..... 21
eclipse IDE 21
PlayEclipse 22
eclipse IDE 22
Eclipse 23
Eclipse IDE 23
..... 23
EclipseScala 23
sbteclipse 24
..... 24
7: 25
Examples 25
..... 25
DDL 26
8: 27
Examples 27
- JavaPlay 2.4,2.5 27
..... 27
..... 27
..... 28
PowerMock 28
..... 29
JSON 29
..... 30
..... 30
9: - Java 31
Examples 31
Guice - 2.4,2.5 31

Play API **31**

..... **31**

@ImplementedBy 32

Play 32

Play 33

..... 34

10: - Scala **35**

..... 35

Examples 35

..... 35

Play 35

1 36

11: **37**

..... 37

Examples 37

..... 37

..... **38**

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: playframework

It is an unofficial and free playframework ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official playframework.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: playframeworkをいめる

このセクションでは、playframeworkのと、がそれをいたいをします。

また、プレイフレームワークのきなテーマについてもし、するトピックにリンクするがあります。 playframeworkのドキュメンテーションはしいので、それらのトピックのバージョンをするがあります。

Examples

Play 1のインストール

Playフレームワークをするには、Java 6がです。ソースからPlayをビルドしたいは、 Gitソースクライアントがソースコードをし、 Antをビルドするがあります。

のパスにJavaがあることをしてくださいするには`java --version`をしてください

Playでは、デフォルトのJavaまたはされているは**$ JAVA_HOME**パスでなJavaがされます。

コマンドラインユーティリティはPythonをします。したがって、どのUNIXシステムでもするはずですしかし、なくともPython 2.5がです。

バイナリパッケージからのインストール

に、インストールはのとおりです。

1. Javaをインストールします。
2. のPlayバイナリパッケージをダウンロードし、アーカイブをしてください。
3. システムパスに 'play'コマンドをし、であることをしてください。

Mac OS X

Javaがみまれているか、にインストールされるため、のステップをスキップできます。

1. のPlayバイナリパッケージをダウンロードし、 `/Applications` 。
2. `/etc/paths`をして、 `/etc/paths` Applications `/etc/paths` `/Applications/play-1.2.5`をし `/etc/paths` たとえば。

OS Xのはのとおりです。

1. HomeBrewをインストールする
2. `brew install playbrew install play`

Linux

JavaをインストールするにはSun-JDKまたはOpenJDKくのLinuxディストリビューションでデフォルトのJavaコマンドであるgcjではなくをしてください。

Windows

Javaをインストールするには、のJDKパッケージをダウンロードしてインストールするだけです。 Pythonランタイムがフレームワークにバンドルされているため、Pythonをにインストールするはありません。

`sbtをってインストールする

すでに`sbt`インストールされているは、 `activator`なしでのPlayプロジェクトをするがです。はのとおりです。

```
# create a new folder
mkdir myNewProject
# launch sbt
sbt
```

のがしたら、 `build.sbt`をしてのをします

```
name := """myProjectName"""

version := "1.0-SNAPSHOT"

offline := true

lazy val root = (project in file(".")).enablePlugins(PlayScala)
scalaVersion := "2.11.6"
# add required dependencies here .. below a list of dependencies I use
libraryDependencies ++= Seq(
  jdbc,
  cache,
  ws,
  filters,
  specs2 % Test,
  "com.github.nscala-time" %% "nscala-time" % "2.0.0",
  "javax.ws.rs" % "jsr311-api" % "1.0",
  "commons-io" % "commons-io" % "2.3",
  "org.asynchttpclient" % "async-http-client" % "2.0.4",
  cache
)


resolvers += "scalaz-bintray" at "http://dl.bintray.com/scalaz/releases"

resolvers ++= Seq("snapshots", "releases").map(Resolver.sonatypeRepo)

resolvers += "Typesafe Releases" at "http://repo.typesafe.com/typesafe/maven-releases/"
```

に、フォルダprojectをし、するPlayのバージョンをしてbuild.propertiesファイルをします

```
addSbtPlugin("com.typesafe.play" % "sbt-plugin" % "2.4.3")
```

それでおしまいプロジェクトがです。あなたはsbtできます。 sbtからactivatorとじコマンドにアクセスできます。

Play 2.4.x / 2.5.xのい - Windows、Java

インストール

ダウンロードとインストール

1. Java 8 - Oracleのサイトからするインストールをダウンロードします 。
2. Activator - www.playframework.com/downloadからzipファイルをダウンロードし、のPlayフォルダにファイルをします。

   ```
   c:\Play-2.4.2\activator-dist-1.3.5
   ```

3. sbt - www.scala-sbt.orgからダウンロードしてください。

をする

1. **JAVA_HOME** 、えば

   ```
   c:\Program Files\Java\jdk1.8.0_45
   ```

2. **PLAY_HOME** 、たとえば

   ```
   c:\Play-2.4.2\activator-dist-1.3.5;
   ```

3. えば**SBT_HOME**

   ```
   c:\Program Files (x86)\sbt\bin;
   ```

インストールされた3つのすべてのプログラムへのパスをパスにします。

```
%JAVA_HOME%\bin;%PLAY_HOME%;%SBT_HOME%;
```

Play 2.5のインストールの

Play 2.5.3の2.5のインストールにはさながあります。それをするには

1. ファイル*activator-dist-1.3.10 \ bin \ activator.bat*をし、55のに ""をします。なはのようにするがあります *set SBT_HOME =BIN_DIRECTORY*
2. アクティベータのルートディレクトリ*activator-dist-1.3.10*のにサブディレクトリ*conf*をします。
3. *conf*ディレクトリに*sbtconfig.txt*というののファイルをします。

CLIをしてしいアプリケーションをする

しいアプリケーションをするディレクトリから*cmd*をします。 CLIをしてしいアプリケーションをするのは、アプリケーションとテンプレートをCLIとしてすることです。

```
activator new my-play-app play-java
```

ちょうどすることがです

```
activator new
```

この、なテンプレートとアプリケーションをするようにめられます。

Play 2.4の、で*project / plugins.sbt*にしてください

```
// Use the Play sbt plugin for Play projects
addSbtPlugin("com.typesafe.play" % "sbt-plugin" % "2.4.x")
```

ここで2.4.xをするなバージョンできえてください。 Play 2.5では、このがにされます。

project / build.propertiesにな**sbt**バージョンがされていることをしてください。あなたのマシンにインストールされている**sbt**バージョンとするはずです。たとえば、Play2.4.xではのようになります。

```
sbt.version=0.13.8
```

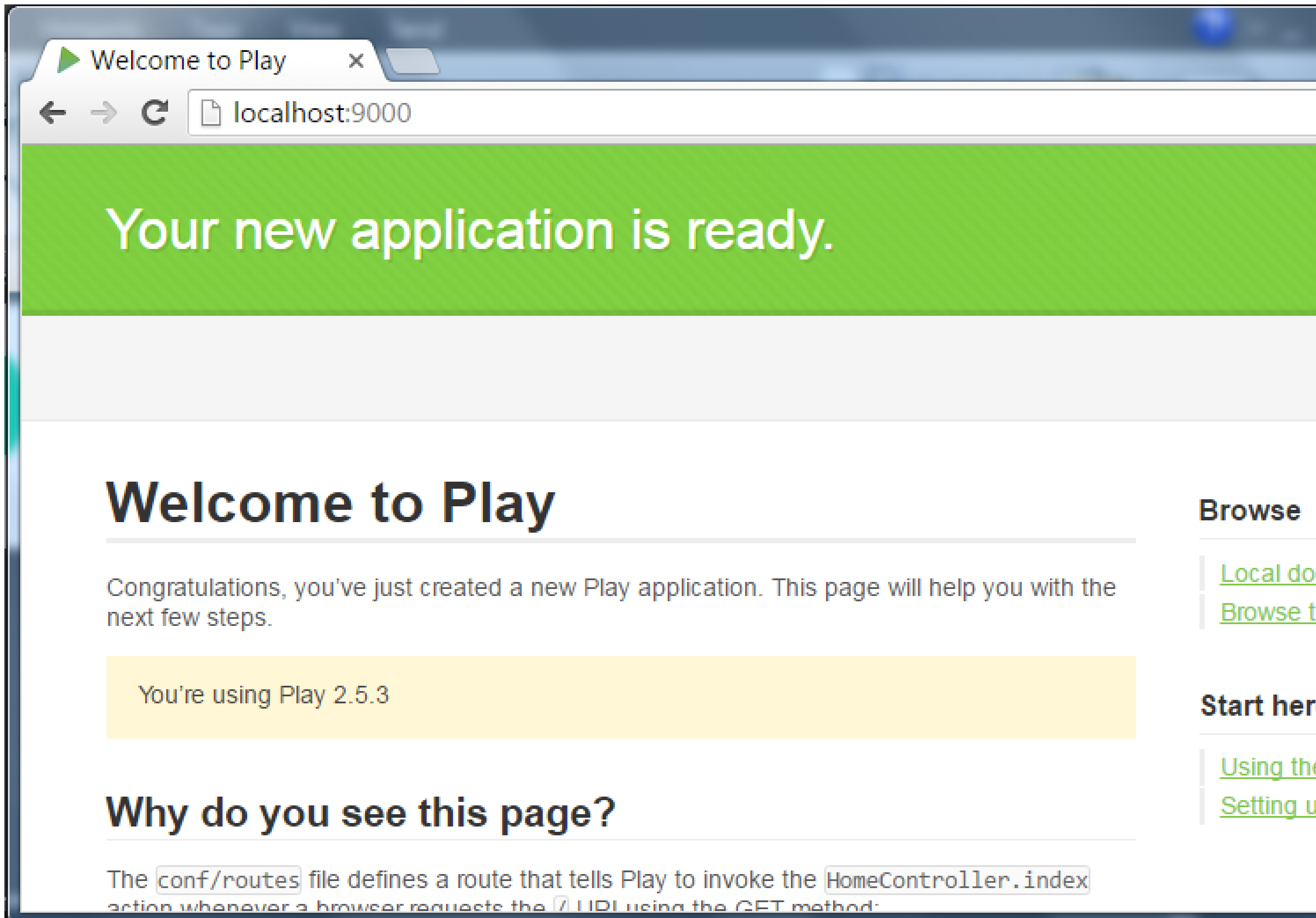
それで、しいアプリケーションがされるかもしれません

```
cd my-play-app
activator run
```

しばらくするとサーバーがし、のプロンプトがコンソールにされます。

```
[info] p.c.s.NettyServer - Listening for HTTP on /0:0:0:0:0:0:0:0:9000
(Server started, use Ctrl+D to stop and go back to the console...)
```

デフォルトでは、サーバーはポート9000をリッスンしています。ブラウザーからURL *http// localhost9000*でできます。のようなものがられます



のポートでアクティベータをする

デフォルトでは、アクチベータはhttpのポート9000またはhttpsの443のアプリケーションをします。なるポートhttpでアプリケーションをするには

```
activator "run 9005"
```

オンラインでplayframeworkをいめるをむ https://riptutorial.com/ja/playframework/topic/1052/playframeworkをいめる

2: Java - Hello World

- このチュートリアルでは、Linux / MacOSシステムでPlayをすることをにしています

Examples

のプロジェクトをする

しいプロジェクトをするには、のコマンドをします `HelloWorld`はプロジェクトの、 `play-java`はテンプレートです

```
$ ~/activator-1.3.10-minimal/bin/activator new HelloWorld play-java
```

このようながられるはずです

```
Fetching the latest list of templates...

OK, application "HelloWorld" is being created using the "play-java" template.

To run "HelloWorld" from the command line, "cd HelloWorld" then:
/home/YourUserName/HelloWorld/activator run

To run the test for "HelloWorld" from the command line, "cd HelloWorld" then:
/home/YourUserName/HelloWorld/activator test

To run the Activator UI for "HelloWorld" from the command line, "cd HelloWorld" then:
/home/YourUserName/HelloWorld/activator ui
```

> プロジェクトはのディレクトリにされますこのはホームフォルダです

すぐアプリケーションをするがいました

アクチベータをする

Play FrameworkのでののステップはActivatorをダウンロードすることです。 Activatorは、Play Frameworkアプリケーションの、、にされるツールです。

ActivatorはPlayダウンロードセクションからダウンロードできます ここではバージョン1.3.10をします

ファイルをダウンロードした、きみアクセスなディレクトリにをして、がいました

> このではActivatorがあなたのホームフォルダにされたとします

の

プロジェクトをしたとき、Activatorはアプリケーションのをえてくれました

```
To run "HelloWorld" from the command line, "cd HelloWorld" then:
    /home/YourUserName/HelloWorld/activator run
```

ここにはさなとしがあります。 `activator`ファイルは、プロジェクトのルートにはなく、`bin/activator`ます。また、のディレクトリをプロジェクトディレクトリにしたは、

```
bin/activator
```

Activatorは、プロジェクトをコンパイルしてするためになをダウンロードします。によっては、がかかることがあります。うまくいけば、プロンプトがされます

```
[HelloWorld] $
```

してたちのプロジェクトをすることができます`~run` これはたちのプロジェクトをし、をするためにアクティベーターをえてくれます。がある、なをコンパイルしてアプリケーションをします。Ctrl + DActivatorシェルにるまたはCtrl + DOSシェルにをすと、このプロセスをできます。

```
[HelloWorld] $ ~run
```

Playはよりくのをダウンロードします。このプロセスがしたら、あなたのアプリはすぐにえるようになります

```
-- (Running the application, auto-reloading is enabled) ---

[info] p.c.s.NettyServer - Listening for HTTP on /0:0:0:0:0:0:0:0:9000

(Server started, use Ctrl+D to stop and go back to the console...)
```

ブラウザでlocalhost9000にすると、Playフレームワークのページがされます

localhost:9000

Your new application is ready.

Welcome to Play

Congratulations, you've just created a new Play application. This page will help you with the next few steps.

You're using Play 2.5.4

おめでとう、あなたはあなたのアプリケーションでいくつかのをうができています

Hello Worldの "Hello World"

「Hello World」は、Hello Worldメッセージをしない、このにはふさわしくありません。だから1つをろう。

`app/controllers/HomeController.java`ファイルにのメソッドをします

```
public Result hello() {
    return ok("Hello world!");
}
```

`conf/routes`ファイルで、ファイルのにのをします。

```
GET     /hello                          controllers.HomeController.hello
```

あなたのをてみると、あなたがをえてアプリケーションをみんでいるに、Playがあなたのアプリケーションをコンパイルしていることにづくはずです

```
[info] Compiling 4 Scala sources and 1 Java source to
/home/YourUserName/HelloWorld/target/scala-2.11/classes...
[success] Compiled in 4s
```

localhost9000 / helloにすると 、ついにhello worldのメッセージがされます

```
Hello world!
```

オンラインでJava - Hello Worldをむ https://riptutorial.com/ja/playframework/topic/5887/java-----hello-world

3: Java - JSONの

ドキュメントをする https : //www.playframework.com/documentation/2.5.x/JavaJsonActions

Examples

JSONをでする

```
import play.libs.Json;


public JsonNode createJson() {
    // {"id": 33, "values": [3, 4, 5]}
    ObjectNode rootNode = Json.newObject();
    ArrayNode listNode = Json.newArray();

    long values[] = {3, 4, 5};
    for (long val: values) {
        listNode.add(val);
    }

    rootNode.put("id", 33);
    rootNode.set("values", listNode);
    return rootNode;
}
```

/ファイルからjsonをロードする

```
import play.libs.Json;
// (...)
```

パブリックフォルダからファイルをロードする

```
// Note: "app" is an play.Application instance
JsonNode node = Json.parse(app.resourceAsStream("public/myjson.json"));
```

からのロード

```
String myStr = "{\"name\": \"John Doe\"}";
JsonNode node = Json.parse(myStr);
```

JSONドキュメントをする

のでは、 `json`にはのデータをむJSONオブジェクトがまれています。

```
[
```

```
  {
    "name": "John Doe",
    "work": {
      "company": {
        "name": "ASDF INC",
        "country": "USA"
      },
      "cargo": "Programmer"
    },
    "tags": ["java", "jvm", "play"]
  },
  {
    "name": "Bob Doe",
    "work": {
      "company": {
        "name": "NOPE INC",
        "country": "AUSTRALIA"
      },
      "cargo": "SysAdmin"
    },
    "tags": ["puppet", "ssh", "networking"],
    "active": true
  }
]
```

のユーザーのをするでない

```
JsonNode node = json.get(0).get("name"); // --> "John Doe"
// This will throw a NullPointerException, because there is only two elements
JsonNode node = json.get(2).get("name"); // --> *crash*
```

ユーザーをするな

```
JsonNode node1 = json.at("/0/name"); // --> TextNode("John Doe")
JsonNode node2 = json.at("/2/name"); // --> MissingNode instance
if (! node2.isMissingNode()) {
    String name = node2.asText();
}
```

のユーザーがくをする

```
JsonNode node2 = json.at("/0/work/company/country"); // TextNode("USA")
```

すべてのを

```
List<JsonNode> d = json.findValues("country"); // List(TextNode("USA"), TextNode("AUSTRALIA"))
```

「アクティブ」をむすべてのユーザーをする

```
List<JsonNode> e = json.findParents("active"); // List(ObjectNode("Bob Doe"))
```

JSONとJavaオブジェクトの

デフォルトでは、JacksonPlay JSONのライブラリは、すべてのパブリックフィールドをじのjsonフィールドにマップしようとします。オブジェクトにgetter / setterがあるは、そのオブジェクトからをします。したがって、ISBNをするプライベートフィールドをつ`Book`クラスがあり、`getISBN/setISBN`というのメソッドを/している`getISBN/setISBN`、Jacksonは

- JavaからJSONにするときにフィールド "ISBN"をつJSONオブジェクトをする
- JSONオブジェクトに "ISBN"フィールドがあるは、`setISBN`メソッドをしてJavaオブジェクトのisbnフィールドをします。

JSONからJavaオブジェクトをする

```
public class Person {
    String id, name;
}

JsonNode node = Json.parse("{\"id\": \"3S2F\", \"name\", \"Salem\"}");
Person person = Json.fromJson(node, Person.class);
System.out.println("Hi " + person.name); // Hi Salem
```

JavaオブジェクトからJSONオブジェクトをする

```
// "person" is the object from the previous example
JsonNode personNode = Json.toJson(person)
```

JSONオブジェクトからJSONをする

```
// personNode comes from the previous example
String json = personNode.toString();
// or
String json = Json.stringify(json);
```

JSONプリント

```
System.out.println(personNode.toString());
/* Prints:
{"id":"3S2F","name":"Salem"}
*/

System.out.println(Json.prettyPrint(personNode));
/* Prints:
{
  "id" : "3S2F",
```

```
  "name" : "Salem"
}
*/
```

オンラインでJava - JSONのをむ https://riptutorial.com/ja/playframework/topic/6318/java-----jsonの

4: JSONの - Scala

ドキュメント パッケージドキュメント

play jsonパッケージをPlayからしてするには、

`"com.typesafe.play" % "play-json_2.11" % "2.5.3"`あなたの`build.sbt`、

- https://mvnrepository.com/artifact/com.typesafe.play/play-json_2.11
- sbtにPlay JSONライブラリをする

Examples

JSONをでする

JSONオブジェクトツリー `JsValue` をですることができます

```
import play.api.libs.json._

val json = JsObject(Map(
  "name" -> JsString("Jsony McJsonface"),
  "age" -> JsNumber(18),
  "hobbies" -> JsArray(Seq(
    JsString("Fishing"),
    JsString("Hunting"),
    JsString("Camping")
  ))
))
```

なをいくつかうことで、よりいのをすることもできます。

```
import play.api.libs.json._

val json = Json.obj(
  "name" -> "Jsony McJsonface",
  "age" -> 18,
  "hobbies" -> Seq(
    "Fishing",
    "Hunting",
    "Camping"
  )
)
```

JSONをするには

```
json.toString
// {"name":"Jsony McJsonface","age":18,"hobbies":["Fishing","Hunting","Camping"]}
Json.prettyPrint(json)
//  {
//    "name" : "Jsony McJsonface",
//    "age" : 18,
```

```
//    "hobbies" : [ "Fishing", "Hunting", "Camping" ]
//  }
```

JavaJSONリクエストをけれる

```
public Result sayHello() {
    JsonNode json = request().body().asJson();
    if(json == null) {
        return badRequest("Expecting Json data");
    } else {
        String name = json.findPath("name").textValue();
        if(name == null) {
            return badRequest("Missing parameter [name]");
        } else {
            return ok("Hello " + name);
        }
    }
}
```

JavaBodyParserでのJSONリクエストのけれ

```
@BodyParser.Of(BodyParser.Json.class)
public Result sayHello() {
    JsonNode json = request().body().asJson();
    String name = json.findPath("name").textValue();
    if(name == null) {
        return badRequest("Missing parameter [name]");
    } else {
        return ok("Hello " + name);
    }
}
```

ヒントこののは、がなものでない*Content-type*が`application/json`にされていても*JSON*がされていない、Playはに*HTTP*ステータスコード400ですることです。

ScalaJSONをでみむ

JSONがえられた

```
val str =
    """{
    |    "name" : "Jsony McJsonface",
    |    "age" : 18,
    |    "hobbies" : [ "Fishing", "Hunting", "Camping" ],
    |    "pet" : {
    |        "name" : "Doggy",
    |        "type" : "dog"
    |    }
    |}""".stripMargin
```

それをしてJsValueをし、JSONツリーをします

```
val json = Json.parse(str)
```

ツリーをしてのをする

```
(json \ "name").as[String]          // "Jsony McJsonface"
```

な

- `\`はJSONオブジェクトののキーにする
- `\\` JSONオブジェクトののキーのすべてのにし、ネストされたオブジェクトをにする
- のインデックスにするための`.apply(idx)` つまり`(idx)`
- `.as[T]`なサブタイプにキャストする
- `.asOpt[T]`なサブタイプへのキャストをみ、ったのはNoneをします。
- `.validate[T]` JSONをなサブタイプにキャストしようとします.JsSuccessまたはJsErrorをします。

```
(json \ "name").as[String]          // "Jsony McJsonface"
(json \ "pet" \ "name").as[String]  // "Doggy"
(json \\ "name").map(_.as[String])  // List("Jsony McJsonface", "Doggy")
(json \\ "type")(0).as[String]      // "dog"
(json \ "wrongkey").as[String]      // throws JsResultException
(json \ "age").as[Int]              // 18
(json \ "hobbies").as[Seq[String]]  // List("Fishing", "Hunting", "Camping")
(json \ "hobbies")(2).as[String]    // "Camping"
(json \ "age").asOpt[String]        // None
(json \ "age").validate[String]     // JsError containing some error detail
```

ケースクラスへの/からのなマッピング

に、JSONをうもなは、ケースクラスをJSONじフィールド、のタイプなどにマッピングすることです。

```
case class Person(
  name: String,
  age: Int,
  hobbies: Seq[String],
  pet: Pet
)

case class Pet(
  name: String,
  `type`: String
)

// these macros will define automatically the conversion to/from JSON
// based on the cases classes definition
implicit val petFormat = Json.format[Pet]
implicit val personFormat = Json.format[Person]
```

Jsonにする

```
val person = Person(
  "Jsony McJsonface",
  18,
  Seq("Fishing", "Hunting", "Camping"),
  Pet("Doggy", "dog")
)

Json.toJson(person).toString
// {"name":"Jsony
McJsonface","age":18,"hobbies":["Fishing","Hunting","Camping"],"pet":{"name":"Doggy","type":"dog"}}
```

Jsonからの

```
val str =
    """{
    |    "name" : "Jsony McJsonface",
    |    "age" : 18,
    |    "hobbies" : [ "Fishing", "Hunting", "Camping" ],
    |     "pet" : {
    |       "name" : "Doggy",
    |       "type" : "dog"
    |     }
    |}""".stripMargin


Json.parse(str).as[Person]
// Person(Jsony McJsonface,18,List(Fishing, Hunting, Camping),Pet(Doggy,dog))
```

オンラインでJSONの - Scalaをむ https://riptutorial.com/ja/playframework/topic/2983/jsonの----scala

5: WSClientをしたWebサービスの

へのリンク https : //www.playframework.com/documentation/2.5.x/ScalaWS

Examples

ないScala

HTTPは、WSClientクラスをしてされ、のクラスにパラメータとしてできます。

```
import javax.inject.Inject

import play.api.libs.ws.WSClient

import scala.concurrent.{ExecutionContext, Future}

class MyClass @Inject() (
  wsClient: WSClient
)(implicit ec: ExecutionContext){

  def doGetRequest(): Future[String] = {
    wsClient
      .url("http://www.google.com")
      .get()
      .map { response =>
      // Play won't check the response status,
      // you have to do it manually
      if ((200 to 299).contains(response.status)) {
        println("We got a good response")
        // response.body returns the raw string
        // response.json could be used if you know the response is JSON
        response.body
      } else
        throw new IllegalStateException(s"We received status ${response.status}")
    }
  }
}
```

オンラインでWSClientをしたWebサービスのをむ
https://riptutorial.com/ja/playframework/topic/2981/wsclientをしたwebサービスの

6: あなたのみのIDEをする

Examples

インテリジェントIDEA

1. Intellij IDEAインストールみコミュニティまたはアルティメット
2. IntelliJにインストールされたScalaプラグイン
3. たとえばActivator `activator new [nameoftheproject] play-scala)` でされたのPlayプロジェクトです。

プロジェクトをく

1. Open IntelliJ IDEA
2. メニュー—> `File` > `Open ...` >フォルダをクリックする[プロジェクト]> [`OK`]をクリックします。
3. ポップアップがき、いくつかのオプションがあります。デフォルトはほとんどのですが、きでないはでのにすることができます。 `OK`クリックします。
4. Intellij IDEAはしえてから、プロジェクトでするモジュールをするのポップアップをします。デフォルトでは`root`と`root-build`という2つのモジュールがされていなければなりません。もせずに[`OK`]をクリックします。
5. IntelliJがプロジェクトをきます。ファイルのをするには、IntelliJがのステータスバーにされているようにしえているに、ファイルをしてから、ににがっているはずです。

Intellijからアプリケーションをする

IDEをしてプロジェクトを/するだけでなく、 `sbt`コマンドラインをしてテストをコンパイル//するもいます。のは、Intellijからそれらをするがきです。デバッグモードをするはです。ステップ

1. メニューの`Run` > `Edit configurations...`
2. ポップアップで、の+をクリックします>リストで`Play 2 App`をします
3. にをけます[nameofyourproject]。デフォルトオプションのままにして、 `OK`ます。
4. `Run`メニューまたはUIのボタンから、このをして`Run`または`Debug`できるようになりました。`Run`は、コマンドラインから`sbt run`したかのように、アプリケーションをします。 `Debug`はじことをいますが、コードにブレークポイントをしてをし、がこっているのかをすることができます。

インポートオプション

これはプロジェクトのグローバルなオプションで、にで、その、メニュー—`Intellij IDEA` > `Preferences` > `Build, Execution, Deployment` > `Build tools` > `SBT` > `Project-level settings` > `Use auto-import Project-level settings`。

このオプションは、Scalaコードの`import`とはありません。これは、intellij IDEAが`build.sbt`ファイルをするときにうべきことをします。インポートがになっている、Intellij IDEAはちにしいビルドファイルをし、プロジェクトをにします。このはであり、ビルドファイルでしているときにIntellijをくするがあるため、すぐにになります。インポートがアクティブされている、`build.sbt`にしてbuild.sbtをしたことをでし、プロジェクトをしたいとします。ほとんどの、なポップアップがされ、そうするかどうかねられます。それのは、UIのSBTパネルにし、いいのをクリックしてにリフレッシュします。

Play IDEとしての**Eclipse - Java**、Play 2.4,2.5

き

Playには、さまざまなIDE-sのプラグインがいくつかあります。 **eclipse**プラグインは、コマンド*activator eclipse*をして、PlayアプリケーションをするEclipseプロジェクトにすることをにします 。 Eclipseプラグインは、プロジェクトごとにすることも、 **sbt**ユーザーごとにすることもできます。それはチームにしますが、どのアプローチをすべきですか。チームがeclipse IDEをしているは、プロジェクトレベルでプラグインをすることができます。 ScalaとJava 8 **luna**または**mars**をサポートするEclipseバージョンをダウンロードするがあります - http://scala-ide.org/download/sdk.htmlから。

プロジェクトごとに**eclipse IDE**をする

Playアプリケーションをeclipseにインポートするには

1. Eclipseプラグインを*project / plugins.sbt*にする

```
//Support Play in Eclipse
addSbtPlugin("com.typesafe.sbteclipse" % "sbteclipse-plugin" % "4.0.0")
```

2. eclipseコマンドのにコンパイルをするフラグを*build.sbt*にします。

```
EclipseKeys.preTasks := Seq(compile in Compile)
```

3. {user root} .sbt \ repositoriesファイルのユーザーリポジトリパスがしいであることをしてください。 *activator-launcher-local*プロパティと*activator-local*プロパティのなは、こののよう

になくとも3つのスラッシュをつがあります。

```
activator-local: file:////${activator.local.repository-C:/Play-2.5.3/activator-dist-1.3.10//repository},
[organization]/[module]/(scala_[scalaVersion]/)(sbt_[sbtVersion]/)[revision]/[type]s/[artifact](-[classifier]).[ext]
activator-launcher-local: file:////${activator.local.repository-${activator.home-${user.home}/.activator}/repository},
[organization]/[module]/(scala_[scalaVersion]/)(sbt_[sbtVersion]/)[revision]/[type]s/[artifact](-[classifier]).[ext]
```

4. アプリケーションをコンパイルする

```
activator compile
```

5. のようにして、しいアプリケーションのEclipseプロジェクトをします。

```
activator eclipse
```

これで、プロジェクトはのプロジェクトからワークスペースにeclipseにインポートするがいました。

PlayソースをEclipseにする

1. *build.sbt*にする

```
EclipseKeys.withSource := true
```

2. プロジェクトをコンパイルする

eclipse IDEをグローバルにする

sbtユーザーをします。

1. ユーザールートディレクトリのに*.sbt \ 0.13 \ plugins*というフォルダとファイル*plugins.sbt*をします。たとえば、Windowsユーザーの**asch**

```
c:\asch\.sbt\0.13\plugins\plugins.sbt
```

2. Eclipseプラグインを*plugins.sbt*にする

```
//Support Play in Eclipse
addSbtPlugin("com.typesafe.sbteclipse" % "sbteclipse-plugin" % "4.0.0")
```

3. ユーザの*.sbt*ディレクトリにファイル*sbteclipse.sbt*をします。たとえば、Windowsユーザーの**asch**

```
c:\asch\.sbt\0.13\sbteclipse.sbt
```

4. **activator eclipse**コマンドのにコンパイルをするフラグを*sbteclipse.sbt*にれます。

```
import com.typesafe.sbteclipse.plugin.EclipsePlugin.EclipseKeys
EclipseKeys.preTasks := Seq(compile in Compile)
```

5. の**Eclipse**キーをオプションでします。

Eclipseからのデバッグ

デバッグするには、デフォルトポート9999でアプリケーションをします。

```
activator -jvm-debug run
```

またはのポートをします。

```
activator -jvm-debug [port] run
```

では

1. プロジェクトをクリックし、[**Debug As]** 、[**Debug Configurations**]をします。
2. 「デバッグ」ダイアログで、「リモート**Java**アプリケーション」をクリックし、「 」をします。
3. ポートをするポートデフォルトのデバッグポートがされているは9999にして、[ ]をクリックします。

これからは、 デバッグをクリックしてのアプリケーションにすることができます。デバッグセッションをしても、サーバーはしません。

Eclipse IDE

1. Java81.8.0_91
2. EclipseネオンJavaScriptとWeb Developer
3. プレイフレームワーク2.5.4

EclipseにScalaをインストールする

1. Eclipseをする
2. `Help` < > `Eclipse Marketplace`
3. `Find Scala` を
4. Scala IDEをインストールする

セットアップsbteclipse

1. プロジェクトをきます.`\project\ plugins.sbt`
2. eclipseプロジェクトをするには`plugins.sbt`のコマンドをしてください

   addSbtPlugin "com.typesafe.sbteclipse" "sbteclipse-plugin" "4.0.0"

3. コマンドをき、 `cd C:\play\play-scala`プロジェクトをします。コマンドラインでのようにしてください

   アクティベーター·エクリプス

プロジェクトのインポート

1. メニューの`File` > `Import` in Eclipse
2. `Existing Projects into Workspace`
3. ルートディレクトリを

これで、プロジェクトはEclipse IDEでおよびできるになりました。

オンラインであなたのみのIDEをするをむ https://riptutorial.com/ja/playframework/topic/4437/あなたのみのideをする

7: スリック

Examples

らかなコード

`build.sbt`では、のものをめてくださいここにはMysqlとPostGreSQLがあります

```
"mysql" % "mysql-connector-java" % "5.1.20",
"org.postgresql" % "postgresql" % "9.3-1100-jdbc4",
"com.typesafe.slick" %% "slick" % "3.1.1",
"com.typesafe.play" %% "play-slick" % "1.1.1"
```

`application.conf`にのをします。

```
mydb.driverjava="slick.driver.MySQLDriver$"
mydb.driver="com.mysql.jdbc.Driver"
mydb.url="jdbc:mysql://hostaddress:3306/dbname?zeroDateTimeBehavior=convertToNull"
mydb.user="username"
mydb.password="password"
```

RDBMSにしないアーキテクチャをするには、のようなオブジェクトをします

```
package mypackage

import slick.driver.MySQLDriver
import slick.driver.PostgresDriver

object SlickDBDriver{
  val env = "something here"
  val driver = env match{
    case "postGreCondition" => PostgresDriver
    case _                  => MySQLDriver
  }
}
```

しいしいモデルをするとき

```
import mypackage.SlickDBDriver.driver.api._
import slick.lifted.{TableQuery, Tag}
import slick.model.ForeignKeyAction

case class MyModel(
  id: Option[Long],
  name: String
) extends Unique


class MyModelDB(tag: Tag) extends IndexedTable[MyModel](tag, "my_table"){
  def id                = column[Long]("id", O.PrimaryKey, O.AutoInc)
  def name              = column[String]("name")
```

```
  def * = (id.? , name) <> ((MyModel.apply _).tupled, MyModel.unapply _)
}

class MyModelCrud{
   import play.api.Play.current

   val dbConfig = DatabaseConfigProvider.get[JdbcProfile](Play.current)
   val db =  dbConfig.db

   val query = TableQuery[MyModelDB]

   // SELECT * FROM my_table;
   def list = db.run{query.result}
}
```

DDL

らかないのポイントはできるだけさなSQLコードをくことです。テーブルをしたら、データベースにテーブルをすることになります。

`val table = TableQuery[MyModel]`をしているは、のコマンドをしてテーブルSQLコード - DDLをできます。

```
import mypackage.SlickDBDriver.driver.api._
table.schema.createStatements
```

オンラインでスリックをむ https://riptutorial.com/ja/playframework/topic/4604/スリック

8: ユニットテスト

Examples

ユニットテスト - **Java**、Play 2.4,2.5

ヘルパーとのアプリケーション

クラスヘルパーはテストにくわれます。それは、Playアプリケーション、HTTPリクエストとレスポンス、セッション、クッキーテストになものをします。テストのコントローラは、Playアプリケーションのコンテキストでするがあります。 ヘルパーメソッド*fakeApplication*は、テストをするためのアプリケーションをします。 ヘルパーと*fakeApplication*をうために、テストクラスは*WithApplication*からしなければなりません 。

のヘルパー APIをするがあります。

```
Helpers.running(Application application, final Runnable block);
Helpers.fakeApplication();
```

ヘルパーのテストはのようになります

```
public class TestController extends WithApplication {
 @Test
 public void testSomething() {
     Helpers.running(Helpers.fakeApplication(), () -> {
         // put test stuff
         // put asserts
     });
 }
}
```

ヘルパーメソッドのインポートステートメントをすると、コードがよりコンパクトになります。

```
 import static play.test.Helpers.fakeApplication;
 import static play.test.Helpers.running;
 ...
 @Test
 public void testSomething() {
      running(fakeApplication(), () -> {
           // put test stuff
           // put asserts
       });
 }
```

}

コントローラのテスト

コントローラメソッドをびしてみましょう。コントローラメソッドは、ルートののURLにルーティングされたメソッドとしてバインドされています。ルーティングされたメソッドのびしはコントローラアクションとばれ、Javaタイプのびしをちます。Playは、アクションのいわゆるリバースルートをします。リバースルートへのコールは、なコールオブジェクトをします。このルーティングは、コントローラのテストにされます。

テストからコントローラアクションをびすには、のヘルパーAPIをするがあります。

```
Result result = Helpers.route(Helpers.fakeRequest(Call action));
```

コントローラテストの

1. ルート

```
GET /conference/:confId    controllers.ConferenceController.getConfId(confId: String)
POST /conference/:confId/participant
controllers.ConferenceController.addParticipant(confId:String)
```

2. されたルート

```
controllers.routes.ConferenceController.getConfId(conferenceId)
controllers.routes.ConferenceController.addParticipant(conferenceId)
```

3. メソッド*getConfId*は**GET**にバインドされており、にをけりません。これは、のようなテストでびすことができます。

```
Result result =
Helpers.route(Helpers.fakeRequest(controllers.routes.ConferenceController.getConfId(conferenceId)
```

4. メソッド*addParticipant*は**POST**にバインドされています。それは、のでをけることをしています。テストでのびしは、のようにうがあります。

```
ParticipantDetails inputData = DataSimulator.createParticipantDetails();
Call action = controllers.routes.ConferenceController.addParticipant(conferenceId);
Result result = route(Helpers.fakeRequest(action).bodyJson(Json.toJson(inputData));
```

PowerMockでのモッキング

モックをにするには、テストクラスにのようにをけるがあります。

```
@RunWith(PowerMockRunner.class)
@PowerMockIgnore({"javax.management.*", "javax.crypto.*"})
public class TestController extends WithApplication {
....
```

コントローラーラーアクションの

コントローラびしは*RequestBuilder*でされます

```
RequestBuilder fakeRequest = Helpers.fakeRequest(action);
```

のaddParticipantの、アクションはのようにされます

```
RequestBuilder mockActionRequest =
Helpers.fakeRequest(controllers.routes.ConferenceController.addParticipant(conferenceId));
```

コントローラメソッドをびすには

```
Result result = Helpers.route(mockActionRequest);
```

のテスト

```
@Test
public void testLoginOK() {
 running(fakeApplication(), () -> {
      ///*whatever mocking*/Mockito.when(...).thenReturn(...);
      RequestBuilder mockActionRequest = Helpers.fakeRequest(
           controllers.routes.LoginController.loginAdmin());
      Result result = route(mockActionRequest);
      assertEquals(OK, result.status());
 });
}
```

JSONでのアクションの

が*T*のオブジェクトであるとしましょう。アクションのモッキングは、いくつかのでうことができます。

オプション1

```
public static <T> RequestBuilder fakeRequestWithJson(T input, String method, String url) {
  JsonNode jsonNode = Json.toJson(input);
  RequestBuilder fakeRequest = Helpers.fakeRequest(method, url).bodyJson(jsonNode);
  System.out.println("Created fakeRequest="+fakeRequest +",
body="+fakeRequest.body().asJson());
  return fakeRequest;
}
```

オプション2

```
public static <T> RequestBuilder fakeActionRequestWithJson(Call action, T input) {
  JsonNode jsonNode = Json.toJson(input);
  RequestBuilder fakeRequest = Helpers.fakeRequest(action).bodyJson(jsonNode);
  System.out.println("Created fakeRequest="+fakeRequest +",
body="+fakeRequest.body().asJson());
  return fakeRequest;
}
```

ベースヘッダーによるアクションの

アクションの

```
public static final String BASIC_AUTH_VALUE = "dummy@com.com:12345";
public static RequestBuilder fakeActionRequestWithBaseAuthHeader(Call action) {
  String encoded = Base64.getEncoder().encodeToString(BASIC_AUTH_VALUE.getBytes());
  RequestBuilder fakeRequest =
Helpers.fakeRequest(action).header(Http.HeaderNames.AUTHORIZATION,
                                             "Basic " + encoded);
  System.out.println("Created fakeRequest="+fakeRequest.toString() );
  return fakeRequest;
}
```

セッションでのの

アクションの

```
public static final String FAKE_SESSION_ID = "12345";
public static RequestBuilder fakeActionRequestWithSession(Call action) {
  RequestBuilder fakeRequest = RequestBuilder fakeRequest =
Helpers.fakeRequest(action).session("sessionId", FAKE_SESSION_ID);
  System.out.println("Created fakeRequest="+fakeRequest.toString() );
  return fakeRequest;
}
```

Play *Session*クラスは、 *HashMap <String*、String>のなるです。それはなコードでされるかもしれません

```
public static Http.Session fakeSession() {
  return new Http.Session(new HashMap<String, String>());
}
```

オンラインでユニットテストをむ https://riptutorial.com/ja/playframework/topic/6192/ユニットテスト

9: - Java

Examples

Guiceとの - プレイ2.4,2.5

Guiceは、Playのデフォルトのさらなる**DI** フレームワークです。のフレームワークもにできますが、Guiceをすると、Playがベールのでをするため、がになります。

Play APIのインジェクション

Play 2.5からは、のバージョンではだったいくつかのAPI-sを**DI**でするがあります。これらは、たとえば、 *Configuration* 、 *JPAApi* 、 *CacheApi*などです。

Play API-sのメソッドは、Playクラスとカスタムクラスにしてにされるクラスとはなります。 にされるクラスのインジェクションは、な *@Inject*アノテーションをフィールドまたはコンストラクタのどちらかにくのとじくらいです。たとえば、プロパティーインジェクションをつコントローラーに*Configuration*をするには、のようにします。

```
@Inject
private Configuration configuration;
```

またはコンストラクタインジェクションをします。

```
private Configuration configuration;
@Inject
public MyController(Configuration configuration) {
    this.configuration = configuration;
}
```

DIのためにされたカスタムクラスのインジェクトは 、 *@Inject*アノテーションをってにインジェクトされたクラスのためにされるのとじようにうがあります。

DIのためにバインドされていないカスタムクラスからのは、 *Play.current*。 injectorをしてインジェクタをにびすことによってうがあります。たとえば、 をカスタムクラスにするには、のようなデータメンバーをします。

```
private Configuration configuration =
Play.current().injector().instanceOf(Configuration.class);
```

カスタムバインディング

カスタムバインディングは**@ImplementedBy**アノテーションで、または**Guice**モジュールをしてプログラムにうことができます。

@ImplementedByアノテーションによる

@ImplementedByアノテーションによるはもなです。のは、キャッシュのファサードをするサービスをしています。

1. このサービスは、のように*CacheProvider*インタフェースによってされます。

```
@ImplementedBy(RunTimeCacheProvider.class)
public interface CacheProvider {
   CacheApi getCache();
}
```

2. サービスはクラスRunTimeCacheProviderによってされます。

```
public class RunTimeCacheProvider implements CacheProvider {
   @Inject
   private CacheApi appCache;
   @Override
   public public CacheApi getCache() {
     return appCache;
   }
}
```

*appCache*データメンバーは、 *RunTimeCacheProvider*インスタンスのにされます。

3. キャッシュインスペクタは*@Inject*アノテーションをつコントローラのメンバとしてされ、コントローラからびされます

```
public class HomeController extends Controller {
  @Inject
  private CacheProvider cacheProvider;
  ...
  public Result getCacheData() {
        Object cacheData = cacheProvider.getCache().get("DEMO-KEY");
        return ok(String.format("Cache content:%s", cacheData));
  }
```

*@ImplementedBy*アノテーションをすると、バインディングがされます。のの*CacheProvider*は、に*RunTimeCacheProvider*をしてインスタンスされます。そのようなは、のをつインタフェースがあるにのみします。いくつかのをつインタフェースや、インタフェースをたないシングルトンとしてされたクラスではにちません。なところ、@ImplementedByはまれにしかされません。**Guice**モジュールとのプログラムバインディングをするがくなります。

デフォルトの**Play**モジュールによるインジェクションバインディング

デフォルトのPlayモジュールは、のようにされたルートプロジェクトディレクトリの*Module*というのクラスです。

```
import com.google.inject.AbstractModule;
public class Module extends AbstractModule {
  @Override
  protected void configure() {
      // bindings are here
  }
}
```

のスニペットはconfigureでのバインディングをしていますが、のバインディングはされます。

*CacheProvider*を*RunTimeCacheProvider*にプログラムでバインドするには

1. *cacheProvider*のから@*ImplementedBy*をします。

```
public interface CacheProvider {
  CacheApi getCache();
}
```

2. モジュールをのようにします。

```
public class Module extends AbstractModule {
  @Override
  protected void configure() {
    bind(CacheProvider.class).to(RunTimeCacheProvider.class);
  }
}
```

デフォルトの**Play**モジュールによるなバインディング

*RunTimeCacheProvider*は、のアプリケーションをした*JUnit*テストではうまくしませんユニットテストのトピックを。したがって、ユニット·テストには*CacheProvider*のさまざまながされます。バインディングは、にじてうがあります。

をてみましょう。

1. クラス*FakeCache*は、テストのにされる*CacheApi*のスタブをしますはそれほどくない - なるマップです。
2. *FakeCacheProvider*クラスは、テストのにする*CacheProvider*をしています。

```
public class FakeCacheProvider implements CacheProvider {
  private final CacheApi fakeCache = new FakeCache();
  @Override
  public CacheApi getCache() {
    return fakeCache;
  }
}
```

2. モジュールはのようにされています

```
public class Module extends AbstractModule {
  private final Environment environment;
  public Module(Environment environment, Configuration configuration) {
    this.environment = environment;
  }
  @Override
  protected void configure() {
     if (environment.isTest() ) {
      bind(CacheProvider.class).to(FakeCacheProvider.class);
    }
    else {
      bind(CacheProvider.class).to(RuntimeCacheProvider.class);
    }
  }
}
```

このは、にのみしています。モジュールのテストのバインディングは、アプリケーションとテストのにするため、ベストプラクティスではありません。テストのバインディングはテストでうべきであり、モジュールはテストのをするべきではありません。これをよりくするをてください...。

カスタムモジュールによるインジェクションバインディング

カスタムモジュールは、デフォルトのPlayモジュールとによくています。いは、どんなであっても、どんなパッケージにもしているということです。たとえば、モジュールOnStartupModuleはパッケージモジュールにします。

```
package modules;
import com.google.inject.AbstractModule;
public class OnStartupModule extends AbstractModule {
   @Override
   protected void configure() {
       ...
   }
}
```

カスタムモジュールは、Playによるびしのためににするがあります。モジュール*OnStartupModule*のは、を*application.conf*にするがあります 。

```
play.modules.enabled += "modules.OnStartupModule"
```

オンラインで - Javaをむ https://riptutorial.com/ja/playframework/topic/6060/----java

10: - Scala

- クラスMyClassUsingAnother @InjectmyOtherClassInjectedMyOtherClass{...}}
- @SingletonクラスMyClassThatShouldBeASingleton...

Examples

な

なシングルトンクラス

```
import javax.inject._
@Singleton
class BurgersRepository {
    // implementation goes here
}
```

のクラスで、のクラスへのアクセスがです。

```
import javax.inject._
class FastFoodService @Inject() (burgersRepository: BurgersRepository){
    // implementation goes here
    // burgersRepository can be used
}
```

にのコントローラをするコントローラ。 FastFoodServiceをシングルトンとしてマークしていないので、されるたびにしいインスタンスがされます。

```
import javax.inject._
import play.api.mvc._
@Singleton
class EatingController @Inject() (fastFoodService: FastFoodService) extends Controller {
    // implementation goes here
    // fastFoodService can be used
}
```

Playクラスをする

しばしば、フレームワークWSClientやConfigurationなどからクラスのインスタンスにアクセスするがあります。のクラスにそれらをすることができます

```
class ComplexService @Inject()(
  configuration: Configuration,
  wsClient: WSClient,
  applicationLifecycle: ApplicationLifecycle,
  cacheApi: CacheApi,
  actorSystem: ActorSystem,
  executionContext: ExecutionContext
  ) {
```

```
  // Implementation goes here
  // you can use all the injected classes :
  //
  // configuration to read your .conf files
  // wsClient to make HTTP requests
  // applicationLifecycle to register stuff to do when the app shutdowns
  // cacheApi to use a cache system
  // actorSystem to use AKKA
  // executionContext to work with Futures
}
```

ExecutionContextのように、にインポートされたがいやすくなるものもあります。コンストラクタの2のパラメータリストにそれらをするだけです

```
class ComplexService @Inject()(
  configuration: Configuration,
  wsClient: WSClient
  )(implicit executionContext: ExecutionContext) {
  // Implementation goes here
  // you can still use the injected classes
  // and executionContext is imported as an implicit argument for the whole class
}
```

1つのモジュールでカスタムバインディングをする

のなはアノテーションによってわれます。いくつかのクラスをインスタンスしてするをさらにするカスタムコードがです。このコードはModuleとばれています。

```
import com.google.inject.AbstractModule
// Play will automatically use any class called `Module` that is in the root package
class Module extends AbstractModule {

  override def configure() = {
    // Here you can put your customisation code.
    // The annotations are still used, but you can override or complete them.

    // Bind a class to a manual instantiation of it
    // i.e. the FunkService needs not to have any annotation, but can still
    // be injected in other classes
    bind(classOf[FunkService]).toInstance(new FunkService)

    // Bind an interface to a class implementing it
    // i.e. the DiscoService interface can be injected into another class
    // the DiscoServiceImplementation is the concrete class that will
    // be actually injected.
    bind(classOf[DiscoService]).to(classOf[DiscoServiceImplementation])

    // Bind a class to itself, but instantiates it when the application starts
    // Useful to executes code on startup
    bind(classOf[HouseMusicService]).asEagerSingleton()
  }

}
```

オンラインで - Scalaをむ https://riptutorial.com/ja/playframework/topic/3020/----scala

11: および

- アクチベータdist

Examples

ディストリビューションにディレクトリをする

ディレクトリ·`scripts`をパッケージにするには、のようにします。

1. フォルダスクリプトをプロジェクトにする
2. build.sbtのにのをします。

```
import NativePackagerHelper._
```

3. build.sbtで、しいディレクトリにマッピングをします

```
mappings in Universal ++= directory("scripts")
```

4. ***activator dist***をってパッケージをビルドします。 `target/universal/`しくされたアーカイブには、しいディレクトリがまれているはずです。

オンラインでおよびをむ https://riptutorial.com/ja/playframework/topic/6642/および

クレジット

S. No		Contributors
1	playframeworkをいめる	Abhinab Kanrar, Anton, asch, Community, implicitdef, James, John, robguinness
2	Java - Hello World	Salem
3	Java - JSONの	Salem
4	JSONの - Scala	Anton, asch, implicitdef, John, Salem
5	WSClientをしたWebサービスの	implicitdef, John, Salem
6	あなたのみのIDEをする	Alice, asch, implicitdef
7	スリック	John
8	ユニットテスト	asch
9	- Java	asch
10	- Scala	asch, implicitdef
11	および	JulienD