1. Thế play là gì thì hôm nay ta sẽ đi hiểu nó nha?

Play là một khung ứng dụng web java và scala năng suất cao, tích hợp các thành phần và API để phát triển web hiện đại.

Play được phát triển bởi các nhà phát triển web để phát triển web bạn sẽ thấy kiến trúc Model-view-model của play quen thuộc và dễ học.

Play cung cấp cấp các mẫu lập trình ngắn gọn và chức năng. La một khung đầy đủ play bao gồm tất cả những thành phần bạn cần xây dựng ứng dụng web và dịch vụ **rest** chẳng hạn như máy chủ HTTP tích họp xử lý mẫu bảo vệ giả mạo yêu cầu trên nhiều trang we, cơ chê định tuyến mạnh mẽ.

Kiến trúc nhẹ không trang thái, thân thiện với web, play sử dụng luồng akka dưới vỏ bọc để cung cấp mức tài nguyên tối thiểu và dự đoán (cpu, memory, thread), nhờs mô hình phản ứng các ứng dung có thể mở rộng cả theo chiều ngang và chiều dọc.

Play không chủ quan về quyền truy cập cơ sở dữ liệu và tích hợp rất nhiều lớp ánh xạ(orm), hỗ trợ anorm, slick, jpa ngay lập tức.

Một ứng dụng Play chỉ bao gồm các tệp Play Jar để chạy bình thường, các tệp Jả này được xuất bản lên kho lưu trữ Maven do đó bạn có thể dung bất kỳ công cụ nào để xây dựng Java và Scala để xây dưng phương án Play.

Bố cục của ứng dụng Play được chuẩn hóa để giữ mọi thứ đơn giản nhất có thể. Sau khi biên dịch thành công đầu tiên, cấu trúc dự án trông như thế này:

app → Application sources

└ assets → Compiled asset sources

└ stylesheets → Typically LESS CSS sources

└ javascripts → Typically CoffeeScript sources

└ controllers → Application controllers

└ models → Application business layer

└ views → Templates

build.sbt → Application build script

conf → Configurations files and other non-compiled resources (on classpath)

└ application.conf → Main configuration file

└ routes → Routes definition

dist → Arbitrary files to be included in your projects distribution

public → Public assets

└ stylesheets → CSS files

└ javascripts → Javascript files

└ images → Image files

project → sbt configuration files

└ build.properties → Marker for sbt project

└ plugins.sbt → sbt plugins including the declaration for Play itself

lib → Unmanaged libraries dependencies

logs → Logs folder

└ application.log → Default log file

target → Generated stuff

└ resolution-cache → Info about dependencies

└ scala-2.13

└ api → Generated API docs

└ classes → Compiled class files

└ routes → Sources generated from routes

└ twirl → Sources generated from templates

└ universal → Application packaging

└ web → Compiled web assets

test → source folder for unit or functional tests

**app/thư mục**

Thư appmục chứa tất cả các tạo phẩm thực thi: mã nguồn Java và Scala, các mẫu và nguồn của nội dung đã biên dịch.

Có ba gói trong thư mục app, một gói cho mỗi thành phần của mẫu kiến ​​trúc MVC:

* app/controllers
* app/models
* app/views

Bạn có thể thêm các gói của riêng mình, ví dụ: app/servicesgói.

**Lưu ý** : trong Play, các tên gói controllers, models, và viewschỉ đơn giản là các quy ước có thể thay đổi nếu cần (chẳng hạn như thêm tiền tố cho mọi thứ bằng com.yourcompany).

Ngoài ra còn có một thư mục tùy chọn được gọi app/assetscho các nội dung được biên dịch như [nguồn LESS](http://lesscss.org/) và nguồn [CoffeeScript](https://coffeescript.org/) .

**public/thư mục**

Tài nguyên được lưu trữ trong thư mục publiclà tài sản tĩnh được phục vụ trực tiếp bởi máy chủ Web.

Thư mục này được chia thành ba thư mục con dành cho hình ảnh, biểu định kiểu CSS và tệp JavaScript. Bạn nên sắp xếp nội dung tĩnh của mình như thế này để giữ cho tất cả ứng dụng Play nhất quán.

Trong ứng dụng mới tạo, thư mục /publicđược ánh xạ tới /assetsđường dẫn URL, nhưng bạn có thể dễ dàng thay đổi điều đó hoặc thậm chí sử dụng một số thư mục cho nội dung tĩnh của mình.

**conf/thư mục**

Thư confmục chứa các tệp cấu hình của ứng dụng. Có hai tệp cấu hình chính:

* application.conflà [tập tin cấu hình](https://www.playframework.com/documentation/2.8.x/ConfigFile) chính cho ứng dụng
* routeslà tập tin định nghĩa của bộ định tuyến.

Nếu bạn cần thêm các tùy chọn cấu hình dành riêng cho ứng dụng của mình, bạn nên thêm nhiều tùy chọn hơn vào application.conftệp.

Nếu một thư viện cần một tệp cấu hình cụ thể, bạn nên cung cấp tệp đó trong thư mục conf.

**lib/thư mục**

Thư libmục là tùy chọn và chứa các phụ thuộc thư viện không được quản lý, tức là. tất cả các tệp JAR mà bạn muốn quản lý thủ công bên ngoài hệ thống xây dựng. Chỉ cần thả bất kỳ tệp JAR nào vào đây và chúng sẽ được thêm vào đường dẫn lớp ứng dụng của bạn.

**tập build.sbttin**

Các khai báo bản dựng chính của dự án của bạn thường được tìm thấy build.sbtở thư mục gốc của dự án.

**project/thư mục**

Thư projectmục chứa các định nghĩa bản dựng sbt:

* plugins.sbtđịnh nghĩa các plugin sbt được sử dụng bởi dự án này.
* build.propertieschứa phiên bản sbt để sử dụng để xây dựng ứng dụng của bạn.

**target/thư mục**

Thư targetmục chứa mọi thứ được tạo bởi hệ thống xây dựng. Có thể hữu ích khi biết những gì được tạo ra ở đây:

* classes/chứa tất cả các lớp đã biên dịch (từ cả hai nguồn Java và Scala).
* classes\_managed/chỉ chứa các lớp được quản lý bởi khung (chẳng hạn như các lớp được tạo bởi bộ định tuyến hoặc hệ thống mẫu). Có thể hữu ích khi thêm thư mục lớp này làm thư mục lớp bên ngoài trong dự án IDE của bạn.
* resource\_managed/chứa các tài nguyên được tạo, các tài sản được biên dịch điển hình như kết quả biên dịch LESS CSS và CoffeeScript.
* src\_managed/chứa các nguồn được tạo, chẳng hạn như các nguồn Scala được tạo bởi hệ thống mẫu.
* web/chứa nội dung được xử lý bởi [sbt-web,](https://github.com/sbt/sbt-web#sbt-web) chẳng hạn như nội dung từ thư mục app/assetsvà public.

**.gitignoretập tin điển hình**

Hệ thống kiểm soát phiên bản của bạn nên bỏ qua các thư mục đã tạo. Đây là .gitignoretệp điển hình cho ứng dụng Play:

logs

project/project

project/target

target

tmp

dist

.cache

RUNNING\_PID

**Bố cục sbt mặc định**

Bạn cũng có tùy chọn sử dụng bố cục mặc định được sử dụng bởi [sbt](https://www.scala-sbt.org/1.x/docs/Directories.html) và [Maven](https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html) . Để sử dụng bố cục này, bạn phải tắt plugin bố cục và thiết lập giám sát rõ ràng cho các mẫu xoay vòng:

lazy val root: Project = (project in file("."))

.enablePlugins(PlayScala)

// Use sbt default layout

.disablePlugins(PlayLayoutPlugin)

Thao tác này sẽ ngăn Play ghi đè bố cục sbt mặc định, giống như sau:

build.sbt → Application build script

src → Application sources

└ main → Compiled asset sources

└ java → Java sources

└ controllers → Java controllers

└ models → Java business layer

└ scala → Scala sources

└ controllers → Scala controllers

└ models → Scala business layer

└ resources → Configurations files and other non-compiled resources (on classpath)

└ application.conf → Main configuration file

└ routes → Routes definition

└ twirl

└ views → Templates

└ assets → Compiled asset sources

└ css → Typically LESS CSS sources

└ js → Typically CoffeeScript sources

└ public → Public assets

└ css → CSS files

└ js → Javascript files

└ images → Image files

└ test → Unit or functional tests

└ java → Java source folder for unit or functional tests

└ scala → Scala source folder for unit or functional tests

└ resources → Resource folder for unit or functional tests

└ universal → Arbitrary files to be included in your projects distribution

project → sbt configuration files

└ build.properties → Marker for sbt project

└ plugins.sbt → sbt plugins including the declaration for Play itself

lib → Unmanaged libraries dependencies

logs → Logs folder

└ application.log → Default log file

target → Generated stuff

└ scala-2.13

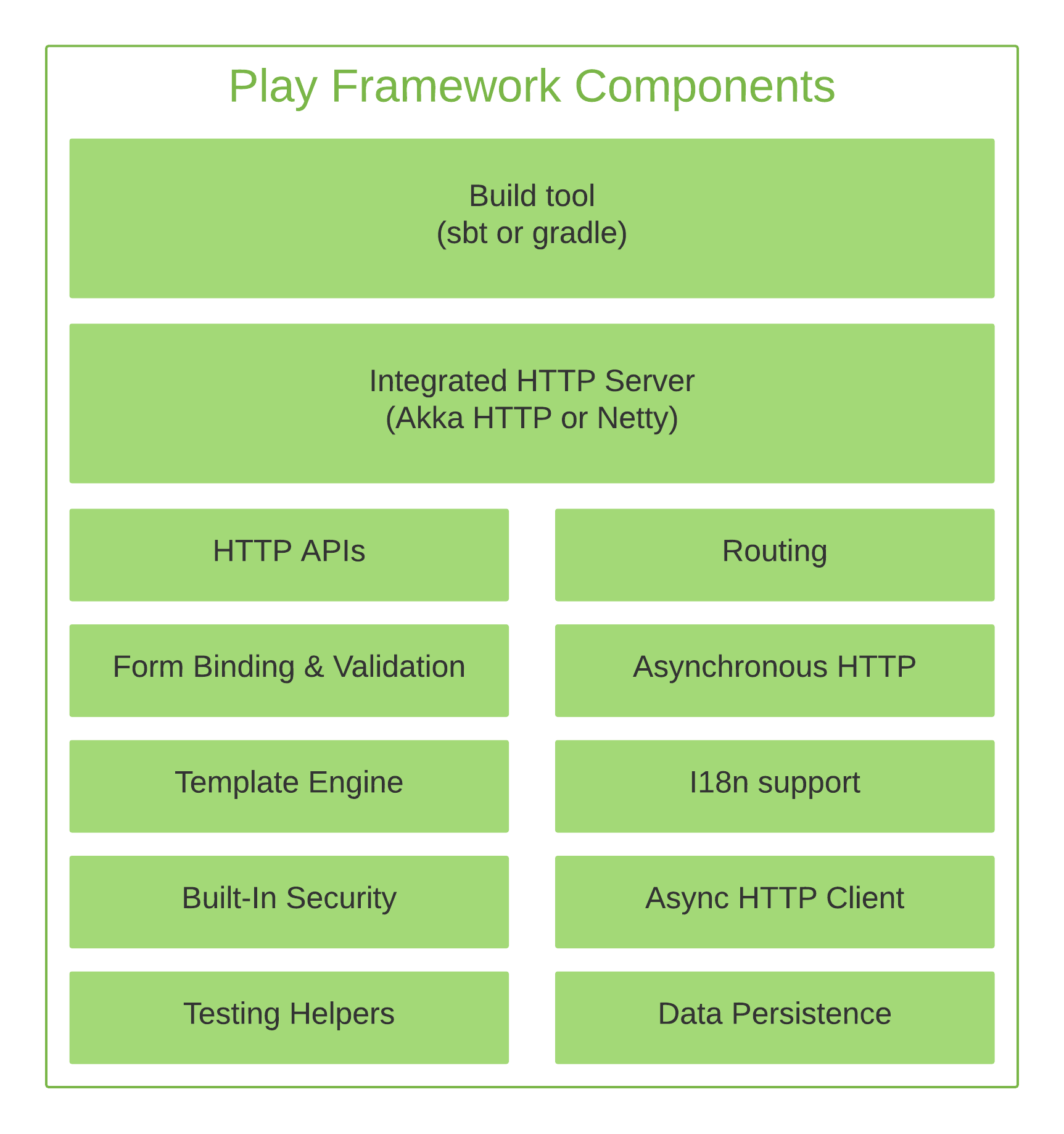
└ cache

└ classes → Compiled class files

└ classes\_managed → Managed class files (templates, ...)

└ resource\_managed → Managed resources (less, ...)

└ src\_managed → Generated sources (templates, ...)



## Truy cập cấu hình

Thông thường, bạn sẽ có được một Configđối tượng thông qua [Dependency Injection](https://www.playframework.com/documentation/2.8.x/JavaDependencyInjection) hoặc đơn giản bằng cách chuyển một thể hiện Configcủa thành phần của bạn:

/\*

\* Copyright (C) Lightbend Inc. <https://www.lightbend.com>

\*/

package controllers

import com.typesafe.config.Config;

import play.mvc.Controller;

import javax.inject.Inject;

public class MyController extends Controller {

private final Config config;

@Inject

public MyController(Config config) {

this.config = config;

}

}

## Hành động là gì?

Hầu hết các yêu cầu mà ứng dụng Play nhận được đều được xử lý bằng một hành động .

Một hành động về cơ bản là một phương thức Java xử lý các tham số yêu cầu và tạo ra kết quả để gửi tới máy khách.

public Result index(Http.Request request) {

return ok("Got request " + request + "!");

}

Một hành động trả về một play.mvc.Resultgiá trị, đại diện cho phản hồi HTTP để gửi tới máy khách web. Trong ví dụ này, okxây dựng phản hồi **200 OK** chứa nội dung phản hồi **văn bản/đơn giản** . Để biết thêm ví dụ về phản hồi HTTP, hãy xem [play.mvc.Resultscác phương thức](https://www.playframework.com/documentation/2.8.x/api/java/play/mvc/Results.html#method.summary) .

## Bộ điều khiển

Bộ điều khiển không gì khác hơn là một lớp mở rộng play.mvc.Controllernhóm một số phương thức hành động.

package controllers;

import play.\*;

import play.mvc.\*;

public class Application extends Controller {

public Result index() {

return ok("It works!");

}

}

Cú pháp đơn giản nhất để xác định một hành động là một phương thức không có tham số trả về một Resultgiá trị, như minh họa ở trên.

Một phương thức hành động cũng có thể có các tham số:

public Result index(String name) {

return ok("Hello " + name);

}

Các tham số này sẽ được giải quyết bằng phương thức Routervà sẽ chứa đầy các giá trị từ URL yêu cầu. Các giá trị tham số có thể được trích xuất từ ​​đường dẫn URL hoặc chuỗi truy vấn URL.

## Kết quả

Hãy bắt đầu với các kết quả đơn giản: một kết quả HTTP có mã trạng thái, một tập hợp các tiêu đề HTTP và phần thân sẽ được gửi tới máy khách web.

Các kết quả này được xác định bởi play.mvc.Resultvà play.mvc.Resultslớp này cung cấp một số trình trợ giúp để tạo ra các kết quả HTTP tiêu chuẩn, chẳng hạn như okphương pháp chúng ta đã sử dụng trong phần trước:

public Result index() {

return ok("Hello world!");

}

Dưới đây là một số ví dụ tạo ra các kết quả khác nhau:

Result ok = ok("Hello world!");

Result notFound = notFound();

Result pageNotFound = notFound("<h1>Page not found</h1>").as("text/html");

Result badRequest = badRequest(views.html.form.render(formWithErrors));

Result oops = internalServerError("Oops");

Result anyStatus = status(488, "Strange response type");

Tất cả những người trợ giúp này có thể được tìm thấy trong play.mvc.Resultslớp học.

## Chuyển hướng cũng là kết quả đơn giản

Chuyển hướng trình duyệt đến một URL mới chỉ là một loại kết quả đơn giản khác. Tuy nhiên, các loại kết quả này không có nội dung phản hồi.

Có một số người trợ giúp có sẵn để tạo kết quả chuyển hướng:

public Result index() {

return redirect("/user/home");

}

Mặc định là sử dụng một 303 SEE\_OTHERloại phản hồi, nhưng bạn cũng có thể chỉ định một mã trạng thái cụ thể hơn:

public Result index() {

return temporaryRedirect("/user/home");

}

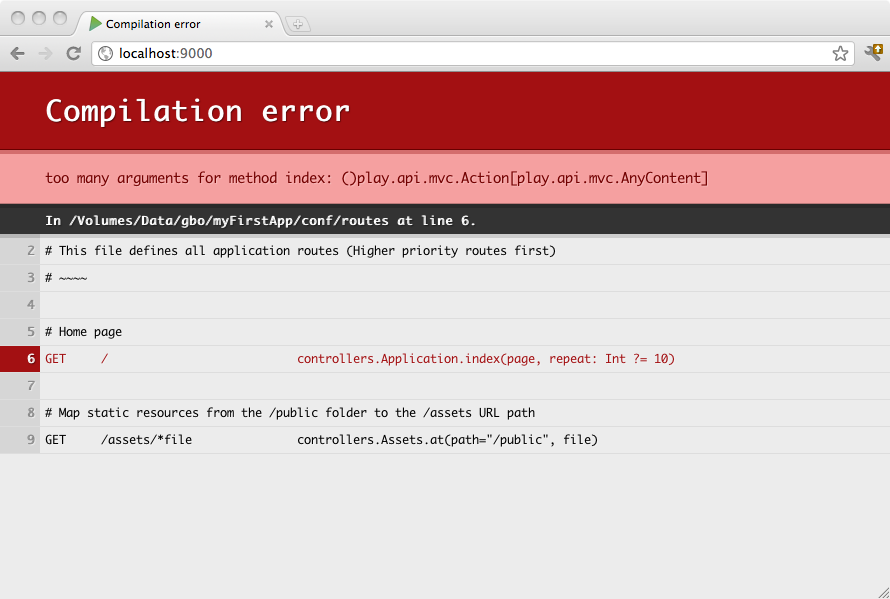
## Bộ định tuyến HTTP tích hợp

Bộ định tuyến là thành phần dịch từng yêu cầu HTTP đến thành lệnh gọi hành động (một phương thức công khai trong lớp trình điều khiển).

Một yêu cầu HTTP được xem như một sự kiện bởi khung MVC. Sự kiện này chứa hai thông tin chính:

* đường dẫn yêu cầu (chẳng hạn như /clients/1542, /photos/list), bao gồm cả chuỗi truy vấn.
* phương thức HTTP (GET, POST,…).

Các tuyến đường được xác định trong conf/routestệp, được biên dịch. Điều này có nghĩa là bạn sẽ thấy lỗi tuyến đường trực tiếp trong trình duyệt của mình:

[](https://www.playframework.com/documentation/2.8.x/resources/manual/working/javaGuide/main/http/images/routesError.png)

## Tiêm phụ thuộc

Trình tạo tuyến đường mặc định của Play tạo một lớp bộ định tuyến chấp nhận các thể hiện của bộ điều khiển trong một @Injecthàm tạo có chú thích. Điều đó có nghĩa là lớp phù hợp để sử dụng với phép nội xạ phụ thuộc và cũng có thể được khởi tạo thủ công bằng cách sử dụng hàm tạo.

Trước Play 2.7.0, Play đã hỗ trợ trình tạo tuyến tĩnh hỗ trợ xác định hành động dưới dạng staticphương thức. Điều đó không còn được hỗ trợ vì Play không còn phụ thuộc vào trạng thái tĩnh. Nếu bạn muốn sử dụng trạng thái tĩnh của riêng mình, bạn vẫn có thể làm như vậy trong bộ điều khiển bằng các phương thức cá thể.

## Cú pháp tập tin định tuyến

conf/routeslà tệp cấu hình được sử dụng bởi bộ định tuyến. Tệp này liệt kê tất cả các tuyến mà ứng dụng cần. Mỗi tuyến bao gồm một phương thức HTTP và mẫu URI được liên kết với lệnh gọi đến một phương thức hành động.

Hãy xem định nghĩa tuyến đường trông như thế nào:

GET /clients/:id controllers.Clients.show(id: Long)

**Lưu ý:** trong lệnh gọi hành động, loại tham số xuất hiện sau tên tham số, giống như trong Scala.

Mỗi tuyến bắt đầu bằng phương thức HTTP, theo sau là mẫu URI. Phần tử cuối cùng của route là định nghĩa cuộc gọi.

Bạn cũng có thể thêm nhận xét vào tệp tuyến đường, với #ký tự:

# Display a client.

GET /clients/:id controllers.Clients.show(id: Long)

Cũng có thể áp dụng các công cụ sửa đổi bằng cách đặt trước tuyến đường một dòng bắt đầu bằng a +. Điều này có thể thay đổi hành vi của một số thành phần Play. Một công cụ sửa đổi như vậy là công cụ sửa đổi “nocsrf” để bỏ qua [bộ lọc CSRF](https://www.playframework.com/documentation/2.8.x/JavaCsrf) :

+ nocsrf

POST /api/new controllers.Api.newThing()

## Phương thức HTTP

Phương thức HTTP có thể là bất kỳ phương thức hợp lệ nào được hỗ trợ bởi HTTP ( GET, PATCH, POST, PUT, DELETE, HEAD, OPTIONS).

## mẫu URI

Mẫu URI xác định đường dẫn yêu cầu của tuyến đường. Một số phần của đường dẫn yêu cầu có thể động.

### Đường dẫn tĩnh

Ví dụ: để đối sánh chính xác GET /clients/allcác yêu cầu đến, bạn có thể xác định tuyến đường này:

GET /clients/all controllers.Clients.list()

### bộ phận động

Nếu bạn muốn xác định một tuyến đường, chẳng hạn như truy xuất ứng dụng khách theo id, bạn cần thêm một phần động:

GET /clients/:id controllers.Clients.show(id: Long)

**Lưu ý:** Một mẫu URI có thể có nhiều phần động.

Chiến lược khớp mặc định cho phần động được xác định bởi cụm từ thông dụng [^/]+, nghĩa là bất kỳ phần động nào được xác định là :idsẽ khớp chính xác với một đoạn đường dẫn URI. Không giống như các loại mẫu khác, các đoạn đường dẫn được giải mã URI tự động trong tuyến đường, trước khi được chuyển đến bộ điều khiển của bạn và được mã hóa trong tuyến đường ngược lại.

### Các bộ phận động kéo dài một số /

Nếu bạn muốn một phần động nắm bắt nhiều hơn một đoạn đường dẫn URI, được phân tách bằng dấu gạch chéo lên, bạn có thể xác định một phần động bằng \*idcú pháp, còn được gọi là mẫu ký tự đại diện, sử dụng .\*biểu thức chính quy:

GET /files/\*name controllers.Application.download(name)

Ở đây, đối với một yêu cầu như thế GET /files/images/logo.png, phần nameđộng sẽ nắm bắt images/logo.pnggiá trị.

Lưu ý rằng các phần động bao trùm một số */*không được giải mã bởi bộ định tuyến hoặc được mã hóa bởi bộ định tuyến ngược lại . Bạn có trách nhiệm xác thực phân đoạn URI thô giống như đối với bất kỳ đầu vào nào của người dùng. Bộ định tuyến đảo ngược chỉ thực hiện nối chuỗi, vì vậy bạn cần đảm bảo đường dẫn kết quả hợp lệ và không chứa nhiều dấu gạch chéo ở đầu hoặc ký tự không phải ASCII chẳng hạn.

### Các phần động với các biểu thức chính quy tùy chỉnh

Bạn cũng có thể xác định biểu thức chính quy của riêng mình cho một phần động, sử dụng $id<regex>cú pháp:

GET /items/$id<[0-9]+> controllers.Items.show(id: Long)

Cũng giống như các tuyến ký tự đại diện, tham số không được giải mã bởi bộ định tuyến hoặc được mã hóa bởi bộ định tuyến ngược lại . Bạn chịu trách nhiệm xác thực thông tin đầu vào để đảm bảo thông tin đó hợp lý trong ngữ cảnh đó.

## Phương thức tạo lời kêu gọi hành động

Phần cuối cùng của định nghĩa tuyến đường là cuộc gọi. Phần này phải xác định một cuộc gọi hợp lệ đến một phương thức hành động.

Nếu phương thức không xác định bất kỳ tham số nào, chỉ cần đặt tên phương thức đủ điều kiện:

GET / controllers.Application.homePage()

Nếu phương thức hành động xác định tham số, thì các giá trị tham số tương ứng sẽ được tìm kiếm trong URI yêu cầu, được trích xuất từ ​​chính đường dẫn URI hoặc từ chuỗi truy vấn.

# Extract the page parameter from the path.

# i.e. http://myserver.com/index

GET /:page controllers.Application.show(page)

Hoặc:

# Extract the page parameter from the query string.

# i.e. http://myserver.com/?page=index

GET / controllers.Application.show(page)

Đây là showđịnh nghĩa phương thức tương ứng trong controllers.Applicationbộ điều khiển:

public Result show(String page) {

String content = Page.getContentOf(page);

return ok(content).as("text/html");

}

### các loại thông số

Đối với các tham số loại String, loại tham số là tùy chọn. Nếu bạn muốn Play chuyển đổi tham số đến thành một loại Scala cụ thể, bạn có thể thêm một loại rõ ràng:

GET /clients/:id controllers.Clients.show(id: Long)

Sau đó, sử dụng cùng loại cho tham số phương thức hành động tương ứng trong bộ điều khiển:

public Result show(Long id) {

Client client = clientService.findById(id);

return ok(views.html.Client.show(client));

}

**Lưu ý:** Các loại tham số được chỉ định bằng cú pháp hậu tố. Ngoài ra, các kiểu chung được chỉ định bằng cách sử dụng các []ký hiệu thay vì <>, như trong Java. Ví dụ: List[String]cùng loại với Java List<String>.

### Các tham số có giá trị cố định

Đôi khi, bạn sẽ muốn sử dụng một giá trị cố định cho một tham số:

# Extract the page parameter from the path, or fix the value for /

GET / controllers.Application.show(page = "home")

GET /:page controllers.Application.show(page)

### Các tham số có giá trị mặc định

Bạn cũng có thể cung cấp một giá trị mặc định sẽ được sử dụng nếu không tìm thấy giá trị nào trong yêu cầu đến:

# Pagination links, like /clients?page=3

GET /clients controllers.Clients.list(page: Int ?= 1)

### thông số tùy chọn

Bạn cũng có thể chỉ định một tham số tùy chọn không cần phải có trong tất cả các yêu cầu:

# The version parameter is optional. E.g. /api/list-all?version=3.0

GET /api/list-all controllers.Api.list(version ?= null)

# or

GET /api/list-all controllers.Api.listOpt(version: java.util.Optional[String])

### Liệt kê tham số

Bạn cũng có thể chỉ định các tham số danh sách cho các tham số chuỗi truy vấn lặp lại:

# The item parameter is a list.

# E.g. /api/list-items?item=red&item=new&item=slippers

GET /api/list-items controllers.Api.listItems(item: java.util.List[String])

# or

# E.g. /api/list-int-items?item=1&item=42

GET /api/list-int-items controllers.Api.listIntItems(item: java.util.List[Integer])

### Truyền yêu cầu hiện tại cho một phương thức hành động

Bạn cũng có thể chuyển yêu cầu hiện tại sang một phương thức hành động. Chỉ cần thêm nó như một tham số:

GET / controllers.Application.dashboard(request: Request)

Và phương pháp hành động tương ứng:

public Result dashboard(Http.Request request) {

return ok("Hello, your request path " + request.path());

}

Play sẽ tự động phát hiện một loại thông số tuyến đường Request(là thông số nhập cho play.mvc.Http.Request) và sẽ chuyển yêu cầu thực tế vào thông số của phương thức hành động tương ứng. Tất nhiên, bạn có thể trộn một Requesttham số với các tham số khác và việc tham số đó ở vị trí nào không quan trọng Request.

## Ưu tiên định tuyến

Nhiều tuyến đường có thể phù hợp với cùng một yêu cầu. Nếu có xung đột, tuyến đầu tiên (theo thứ tự khai báo) được sử dụng.

## Định tuyến ngược

Bộ định tuyến có thể được sử dụng để tạo URL từ bên trong lệnh gọi Java. Điều này cho phép tập trung tất cả các mẫu URI của bạn trong một tệp cấu hình duy nhất, vì vậy bạn có thể tự tin hơn khi tái cấu trúc ứng dụng của mình.

Đối với mỗi bộ điều khiển được sử dụng trong tệp tuyến đường, bộ định tuyến sẽ tạo một 'bộ điều khiển đảo ngược' trong routesgói, có cùng phương thức hành động, có cùng chữ ký, nhưng trả về play.mvc.Callthay vì tệp play.mvc.Result.

Định play.mvc.Callnghĩa một cuộc gọi HTTP và cung cấp cả phương thức HTTP và URI.

Ví dụ: nếu bạn tạo bộ điều khiển như:

package controllers;

import play.\*;

import play.mvc.\*;

public class Application extends Controller {

public Result hello(String name) {

return ok("Hello " + name + "!");

}

}

Và nếu bạn ánh xạ nó trong conf/routestệp:

# Hello action

GET /hello/:name controllers.Application.hello(name)

Sau đó, bạn có thể đảo ngược URL thành hellophương thức hành động bằng cách sử dụng controllers.routes.Applicationbộ điều khiển đảo ngược:

// Redirect to /hello/Bob

public Result index() {

return redirect(controllers.routes.Application.hello("Bob"));

}

**Lưu ý:** Có một routesgói phụ cho mỗi gói bộ điều khiển. Vì vậy, hành động controllers.Application.hellocó thể được đảo ngược thông qua controllers.routes.Application.hello(miễn là không có tuyến đường nào khác trước nó trong tệp tuyến đường khớp với đường dẫn đã tạo).

Phương thức hành động đảo ngược hoạt động khá đơn giản: nó lấy các tham số của bạn và thay thế chúng trở lại vào mẫu tuyến đường. Trong trường hợp đoạn đường dẫn ( :foo), giá trị được mã hóa trước khi thực hiện thay thế. Đối với các mẫu biểu thức chính quy và ký tự đại diện, chuỗi được thay thế ở dạng thô, vì giá trị có thể trải rộng trên nhiều phân đoạn. Đảm bảo rằng bạn thoát các thành phần đó như mong muốn khi chuyển chúng sang tuyến đường ngược lại và tránh chuyển thông tin đầu vào không hợp lệ của người dùng.

## tuyến đường tương đối

Có những trường hợp trả lại một tuyến đường tương đối thay vì tuyệt đối có thể hữu ích. Các tuyến đường được trả về play.mvc.Callluôn là tuyệt đối (chúng dẫn đầu bằng một /), điều này có thể dẫn đến sự cố khi các yêu cầu đối với ứng dụng web của bạn được viết lại bởi proxy HTTP, bộ cân bằng tải và cổng API. Một số ví dụ về việc sử dụng tuyến đường tương đối sẽ hữu ích bao gồm:

* Lưu trữ một ứng dụng đằng sau một cổng web có tiền tố tất cả các tuyến bằng thứ gì đó khác với thứ được định cấu hình trong conf/routestệp của bạn và bắt nguồn ứng dụng của bạn ở một tuyến mà nó không mong đợi.
* Khi hiển thị động các biểu định kiểu, bạn cần các liên kết nội dung phải tương đối vì chúng có thể sẽ được CDN phân phát từ các URL khác nhau.

Để có thể tạo một tuyến đường tương đối, bạn cần biết những gì để tạo tuyến đường mục tiêu so với (tuyến đường bắt đầu). Lộ trình bắt đầu có thể được lấy từ hiện tại RequestHeader. Do đó, để tạo tuyến đường tương đối, bạn bắt buộc phải chuyển tuyến đường hiện tại RequestHeaderhoặc tuyến đường bắt đầu làm Stringtham số.

Ví dụ: các điểm cuối của bộ điều khiển đã cho như:

package controllers;

import play.\*;

import play.mvc.\*;

public class Relative extends Controller {

public Result helloview(Http.Request request) {

ok(views.html.hello.render("Bob", request));

}

public Result hello(String name) {

return ok("Hello " + name + "!");

}

}

Và nếu bạn ánh xạ nó trong conf/routestệp:

GET /foo/bar/hello controllers.Relative.helloview(request: Request)

GET /hello/:name controllers.Relative.hello(name)

Sau đó, bạn có thể xác định các tuyến đường tương đối bằng cách sử dụng bộ định tuyến đảo ngược như trước đây và bao gồm một lệnh gọi bổ sung tới relativeTo(play.mvc.RequestHeader requestHeader):

@(name: String, request: Http.RequestHeader)

<h1>Hello @name</h1>

<a href="@routes.Relative.hello(name)">Absolute Link</a>

<a href="@routes.Relative.hello(name).relativeTo(request)">Relative Link</a>

**Lưu ý:** Thông số Http.Requestđược truyền từ bộ điều khiển được chuyển thành a Http.RequestHeadertrong tham số dạng xem.

Khi yêu cầu /foo/bar/helloHTML được tạo sẽ giống như vậy:

<!DOCTYPE html>

<html lang="en">

<head>

<title>Bob</title>

</head>

<body>

<a href="/hello/Bob">Absolute Link</a>

<a href="../../hello/Bob">Relative Link</a>

</body>

</html>

## Bộ điều khiển mặc định

Chơi bao gồm một [Defaultbộ điều khiển](https://www.playframework.com/documentation/2.8.x/api/scala/controllers/Default.html) cung cấp một số hành động hữu ích. Chúng có thể được gọi trực tiếp từ tệp tuyến đường:

# Redirects to https://www.playframework.com/ with 303 See Other

GET /about controllers.Default.redirect(to = "https://www.playframework.com/")

# Responds with 404 Not Found

GET /orders controllers.Default.notFound

# Responds with 500 Internal Server Error

GET /clients controllers.Default.error

# Responds with 501 Not Implemented

GET /posts controllers.Default.todo

Trong ví dụ này, GET /chuyển hướng đến một trang web bên ngoài, nhưng cũng có thể chuyển hướng đến một hành động khác (chẳng hạn như /poststrong ví dụ trên).

## Định tuyến nâng cao

Xem [Định tuyến DSL](https://www.playframework.com/documentation/2.8.x/JavaRoutingDsl) .

## Thay đổi mặc địnhContent-Type

Loại nội dung kết quả được tự động suy ra từ giá trị Java mà bạn chỉ định làm nội dung phản hồi.

Ví dụ:

Result textResult = ok("Hello World!");

Sẽ tự động đặt Content-Typetiêu đề thành text/plain, trong khi:

JsonNode json = Json.toJson(object);

Result jsonResult = ok(json);

sẽ đặt Content-Typetiêu đề thành application/json.

Điều này khá hữu ích, nhưng đôi khi bạn muốn thay đổi nó. Chỉ cần sử dụng as(newContentType)phương thức trên một kết quả để tạo một kết quả tương tự mới với một Content-Typetiêu đề khác:

Result htmlResult = ok("<h1>Hello World!</h1>").as("text/html");

hoặc thậm chí tốt hơn, sử dụng:

Result htmlResult = ok("<h1>Hello World!</h1>").as(MimeTypes.HTML);

## Thao tác tiêu đề HTTP

Bạn cũng có thể thêm (hoặc cập nhật) bất kỳ tiêu đề HTTP nào vào kết quả:

public Result index() {

return ok("<h1>Hello World!</h1>")

.as(MimeTypes.HTML)

.withHeader(CACHE\_CONTROL, "max-age=3600")

.withHeader(ETAG, "some-etag-calculated-value");

}

Lưu ý rằng việc đặt tiêu đề HTTP sẽ tự động loại bỏ giá trị trước đó nếu nó tồn tại trong kết quả ban đầu.

## Đặt và hủy cookie

Cookie chỉ là một dạng tiêu đề HTTP đặc biệt nhưng Play cung cấp một bộ công cụ trợ giúp để giúp việc này trở nên dễ dàng hơn.

Bạn có thể dễ dàng thêm Cookie vào phản hồi HTTP bằng cách sử dụng:

public Result index() {

return ok("<h1>Hello World!</h1>")

.as(MimeTypes.HTML)

.withCookies(Cookie.builder("theme", "blue").build());

}

Nếu bạn cần đặt thêm thông tin chi tiết, bao gồm đường dẫn, tên miền, thời hạn sử dụng, liệu nó có an toàn hay không và có nên đặt cờ chỉ HTTP hay không, bạn có thể thực hiện việc này bằng các phương thức quá tải:

public Result index() {

return ok("<h1>Hello World!</h1>")

.as(MimeTypes.HTML)

.withCookies(

Cookie.builder("theme", "blue")

.withMaxAge(Duration.ofSeconds(3600))

.withPath("/some/path")

.withDomain(".example.com")

.withSecure(false)

.withHttpOnly(true)

.withSameSite(Cookie.SameSite.STRICT)

.build());

}

Để loại bỏ một Cookie được lưu trữ trước đó trên trình duyệt web:

public Result index() {

return ok("<h1>Hello World!</h1>").as(MimeTypes.HTML).discardingCookie("theme");

}

Nếu bạn đặt đường dẫn hoặc miền khi đặt cookie, hãy đảm bảo rằng bạn đặt cùng một đường dẫn hoặc miền khi hủy cookie, vì trình duyệt sẽ chỉ hủy cookie nếu tên, đường dẫn và miền khớp nhau.

## Thay đổi bộ ký tự cho phản hồi HTTP dựa trên văn bản

Đối với phản hồi HTTP dựa trên văn bản, điều rất quan trọng là phải xử lý chính xác bộ ký tự. Phát xử lý điều đó cho bạn và sử dụng utf-8theo mặc định (xem [lý do sử dụng utf-8](https://www.w3.org/International/questions/qa-choosing-encodings#useunicode) ).

Bộ ký tự được sử dụng để chuyển đổi phản hồi văn bản thành các byte tương ứng để gửi qua ổ cắm mạng và để cập nhật Content-Typetiêu đề với ;charset=xxxphần mở rộng thích hợp.

Mã hóa có thể được chỉ định khi bạn đang tạo Resultgiá trị:

public Result index() {

return ok("<h1>Hello World!</h1>", "iso-8859-1")

.as("text/html; charset=iso-8859-1");

}

## Phạm vi kết quả

Play hỗ trợ một phần của [RFC 7233](https://tools.ietf.org/html/rfc7233) xác định cách hoạt động của các yêu cầu phạm vi và phản hồi một phần. Nó cho phép bạn cung cấp một tiêu đề 206 Partial Contentthỏa đáng nếu Rangecó trong yêu cầu. Nó cũng sẽ trả về a Accept-Ranges: bytescho giao hàng Result.

**Lưu ý:** Bên cạnh thực tế là một số phân tích cú pháp được thực hiện để xử lý nhiều phạm vi tốt hơn, multipart/byterangesvẫn chưa được hỗ trợ đầy đủ.

Kết quả phạm vi có thể được tạo cho tệp Source, InputStream, Tệp và Path. Xem [RangeResult](https://www.playframework.com/documentation/2.8.x/api/java/play/mvc/RangeResults.html)tài liệu API để biết tất cả các phương pháp có sẵn. Ví dụ:

public Result index(Http.Request request) {

String content = "This is the full content!";

InputStream input = getInputStream(content);

return RangeResults.ofStream(request, input, content.length());

}

Hoặc cho một Source:

public Result index(Http.Request request) {

String content = "This is the full content!";

Source<ByteString, NotUsed> source = sourceFrom(content);

return RangeResults.ofSource(

request, (long) content.length(), source, "file.txt", MimeTypes.TEXT);

}

Ví dụ: khi yêu cầu Rangekhông thỏa mãn, nếu phạm vi trong trường Rangetiêu đề của yêu cầu không trùng với phạm vi hiện tại của tài nguyên đã chọn, thì trạng thái HTTP 416(Phạm vi không thỏa mãn) sẽ được trả về.

Cũng có thể tìm kiếm trước một vị trí cụ thể của Sourcephạm vi để cung cấp kết quả hiệu quả hơn. Để làm điều đó, bạn có thể cung cấp một chức năng trong đó tìm kiếm trước xảy ra:

public Result index(Http.Request request) {

String content = "This is the full content!";

return RangeResults.ofSource(

request,

(long) content.length(),

offset ->

new RangeResults.SourceAndOffset(offset, sourceFrom(content).drop(offset)),

"file.txt",

MimeTypes.TEXT);

}

# Phạm vi phiên và Flash

## Nó khác như thế nào trong Play

Nếu bạn phải giữ dữ liệu trên nhiều yêu cầu HTTP, bạn có thể lưu chúng trong phạm vi Phiên hoặc Flash. Dữ liệu được lưu trữ trong Phiên có sẵn trong toàn bộ phiên người dùng và dữ liệu được lưu trữ trong phạm vi flash chỉ có sẵn cho yêu cầu tiếp theo.

## Làm việc với Cookie

Điều quan trọng là phải hiểu rằng dữ liệu Phiên và Flash không được lưu trữ trong máy chủ mà được thêm vào mỗi Yêu cầu HTTP tiếp theo, sử dụng cookie HTTP.

Vì Phiên và Flash được triển khai bằng cách sử dụng cookie nên có một số ý nghĩa quan trọng.

* Kích thước dữ liệu rất hạn chế (tối đa 4 KB).
* Bạn chỉ có thể lưu trữ các giá trị chuỗi, mặc dù bạn có thể tuần tự hóa JSON thành cookie.
* Thông tin trong cookie hiển thị với trình duyệt và do đó có thể làm lộ dữ liệu nhạy cảm.
* Thông tin cookie là bất biến đối với yêu cầu ban đầu và chỉ có sẵn cho các yêu cầu tiếp theo.

Điểm cuối cùng có thể là một nguồn gây nhầm lẫn. Khi bạn sửa đổi cookie, bạn đang cung cấp thông tin cho phản hồi và Play phải phân tích cú pháp lần nữa để xem giá trị được cập nhật. Nếu bạn muốn đảm bảo thông tin phiên là hiện tại thì bạn phải luôn ghép nối sửa đổi phiên với Chuyển hướng.

Cookie phiên được ký bằng khóa bí mật để khách hàng không thể sửa đổi dữ liệu cookie (hoặc nó sẽ bị vô hiệu).

Phiên Play không nhằm mục đích sử dụng làm bộ đệm. Nếu cần lưu vào bộ nhớ cache một số dữ liệu liên quan đến một phiên cụ thể, bạn có thể sử dụng cơ chế bộ nhớ đệm tích hợp của Play và sử dụng phiên để lưu trữ một ID duy nhất nhằm liên kết dữ liệu đã lưu trong bộ nhớ cache với một người dùng cụ thể.

## Cấu hình phiên

Tên mặc định cho cookie là PLAY\_SESSION. Điều này có thể được thay đổi bằng cách cấu hình khóa play.http.session.cookieNametrong application.conf.

Nếu tên của cookie bị thay đổi, cookie trước đó có thể bị hủy bằng cách sử dụng các phương pháp tương tự được đề cập trong [Cài đặt và hủy cookie](https://www.playframework.com/documentation/2.8.x/JavaResponse) .

Vui lòng xem [Định cấu hình cookie phiên](https://www.playframework.com/documentation/2.8.x/SettingsSession) để biết thêm thông tin về cách định cấu hình tham số cookie phiên trong application.conf.

### Thời gian chờ / Hết hạn phiên

Theo mặc định, không có thời gian chờ kỹ thuật cho Phiên. Nó hết hạn khi người dùng đóng trình duyệt web. Nếu bạn cần thời gian chờ chức năng cho một ứng dụng cụ thể, bạn đặt tuổi tối đa của cookie phiên bằng cách định cấu hình khóa play.http.session.maxAgetrong application.confvà điều này cũng sẽ được đặt thành play.http.session.jwt.expiresAftercùng một giá trị. Thuộc maxAgetính sẽ xóa cookie khỏi trình duyệt và expxác nhận quyền sở hữu JWT sẽ được đặt trong cookie và sẽ làm cho cookie không hợp lệ sau khoảng thời gian nhất định. Vui lòng xem [Định cấu hình cookie phiên](https://www.playframework.com/documentation/2.8.x/SettingsSession) để biết thêm thông tin.

## Lưu trữ dữ liệu vào Session

Vì Phiên chỉ là một Cookie, nó cũng chỉ là một tiêu đề HTTP, nhưng Play cung cấp một phương thức trợ giúp để lưu trữ giá trị phiên:

public Result login(Http.Request request) {

return redirect("/home")

.addingToSession(request, "connected", "user@gmail.com");

}

Theo cách tương tự, bạn có thể xóa bất kỳ giá trị nào khỏi phiên đến:

public Result logout(Http.Request request) {

return redirect("/home").removingFromSession(request, "connected");

}

## Đọc giá trị Phiên

Bạn có thể truy xuất Phiên đến từ yêu cầu HTTP:

public Result index(Http.Request request) {

return request

.session()

.get("connected")

.map(user -> ok("Hello " + user))

.orElseGet(() -> unauthorized("Oops, you are not connected"));

}

## Loại bỏ toàn bộ phiên

Nếu bạn muốn loại bỏ toàn bộ phiên, có một thao tác đặc biệt:

public Result logout() {

return redirect("/home").withNewSession();

}

## Phạm vi flash

Phạm vi Flash hoạt động chính xác như Phiên, nhưng có một điểm khác biệt:

* dữ liệu chỉ được lưu giữ cho một yêu cầu

**Quan trọng:** Chỉ nên sử dụng phạm vi flash để truyền thông báo thành công/lỗi trên các ứng dụng đơn giản không phải Ajax. Vì dữ liệu chỉ được lưu giữ cho yêu cầu tiếp theo và vì không có gì đảm bảo để đảm bảo thứ tự yêu cầu trong một ứng dụng Web phức tạp, nên phạm vi Flash phải tuân theo các điều kiện cạnh tranh.

Vì vậy, ví dụ: sau khi lưu một mục, bạn có thể muốn chuyển hướng người dùng trở lại trang chỉ mục và bạn có thể muốn hiển thị một thông báo trên trang chỉ mục cho biết rằng việc lưu đã thành công. Trong hành động lưu, bạn sẽ thêm thông báo thành công vào phạm vi flash:

public Result save() {

return redirect("/home").flashing("success", "The item has been created");

}

Sau đó, trong hành động lập chỉ mục, bạn có thể kiểm tra xem thông báo thành công có tồn tại trong phạm vi flash hay không và nếu có, hãy hiển thị thông báo đó:

public Result index(Http.Request request) {

return ok(request.flash().get("success").orElse("Welcome!"));

}

Một giá trị flash cũng tự động có sẵn trong các mẫu Twirl. Ví dụ:

@flash.get("success").orElse("Welcome!")