Introducing Scalatra

By Ivan Porto Carrero, Ross A. Baker, Dave Hrycyszyn, Stefan Ollinger, and Jared Armstrong, authors of *Scalatra in Action*

Scalatra is an HTTP micro-framework, written in the up-and-coming new language, Scala. Scalatra is used by LinkedIn, the BBC, the Guardian newspaper, games website IGN, social identity provider JanRain, the Wordnik online dictionary, and the British government's new flagship website. It's fast, battle-tested, and in production. This article introduces you to the simplicity of Scalatra micro-framework.

Add a small configuration file, download, and run one command, and you've got a running web application. There's a lot more to Scalatra, but the following simple code expresses its essence. You define a route, which is a combination of an HTTP verb `(get)`, and a URL route matcher `(/hello)`, which executes a block of code on the server and returns the result (in this case, the string `Hello world!`).

Listing 1 Scalatra in Action

```
package com.example.yourapp
import org.scalatra._

class HelloWorld extends ScalatraFilter {
  get("/hello") {
    "Hello there!"
  }
}
```

If you put this code up on http://yourwebsite.org, you could type http://yourwebsite.org/hello into your browser and see the text "Hello world!" in return.

And yet...it's so small. What do all of these big organizations see in it? The interesting thing is, there isn't a single answer to that question because Scalatra is a micro-framework.

Micro-frameworks vs. full-stack frameworks

A micro-framework is a very small framework or way of organizing your code, with only minimal helper functions built in. Micro-frameworks are extremely modular, allowing you to pick and choose the libraries you want and to easily extend things as necessary.

Most web programmers are used to full-sized frameworks. In general, these try to solve all the problems you'll commonly encounter and to do so within the confines of a fairly unified codebase. Despite their obvious differences, Spring, Symfony, Ruby on Rails, Django, and Play frameworks all share something: they make the assumption that you won't often need (or even want) to step out of the environments they provide. Not on most days, anyway.

Micro-frameworks are different. Instead of trying to provide for your every need, they try to do a few things well. In Scalatra's case, you get a great way of expressing HTTP actions, a lot of optional modules for common external tasks, and an easy way of bolting on additional libraries to accomplish whatever it is you want.

The micro-framework approach is a candid admission by the Scalatra authors that they don't have a clue what you're going to build. It'll probably be related to HTTP, but your choices beyond that—of data stores, template libraries, testing frameworks, strategies for asynchronous requests and responses, server push, message queuing systems, and API documentation—are up to you. Maybe you don't need half of the things on that list. Maybe you need all of them, but you want to choose each library yourself.

This lean way of doing things allows you to build up exactly the right capabilities for the job at hand and keeps the amount of extra code in your project to a minimum. If you're building a full-sized web application, like a social network app or an e-commerce application, you can use Scalatra as a full model-view-controller stack. Conversely, if you need to build out a small, high-performance system, such as an API that routes incoming request data to other systems or a blazing fast OAuth2 server, you can import only what's needed to do so.

Those big organizations are using Scalatra because they can each use it to solve their specific problems in the way which makes the most sense to them.

What's Scalatra good at?

Scalatra can replace a full-stack web development framework for most tasks. Mobile app development has exploded over the past four years. At the same time, single-page in-browser development frameworks such as backbone.js, ember.js, and angular.js are rapidly gaining in popularity.

All of this means that there are a lot of clients that still make heavy use of web technologies, but aren't the traditional browser clients we've seen for the past twenty years. Scalatra is a perfect fit for these. It's easy to install, lightweight, and fast. It lets you design and build out high-performance web APIs very quickly, and it has special tools integrated with it to produce beautiful, functional, and correct API documentation.

Another major area of internet innovation recently is in the realm of server-push technologies. Scalatra incorporates advanced constructs for event-driven programming, allowing you to easily push information into your users' browsers—so they see constantly updated information without having to refresh the page. This technology has the potential to turn the web upside down.

Lastly, in some cases, people are using full-sized frameworks for the bulk of their web development functionality, and they bust out Scalatra for specific, high-performance actions. This means that it's relatively easy to try it out in an isolated part of your system, for example to solve performance hotspots, and see if you like it.

Hello Scalatra!

Before we can move on, though, you'll need to have a running installation of Scalatra. Once you're all set up, let's generate, build and run the traditional HelloWorld example.

Generating a new project

Run the following code in your terminal:

```
g8 scalatra/scalatra-sbt
```

This will check out a pre-built application skeleton for you (from Github) and ask you some questions about your application. Hit Enter to accept the defaults.

Listing 2 Generating a project

```
$ g8 scalatra/scalatra-sbt
      > organization [com.example]:
      > package [com.example.app]:
      > name [scalatra-sbt-prototype]:
      > servlet_name [MyScalatraServlet]:
      > scala_version [2.9.1]:
      > version [0.1.0-SNAPSHOT
```

Giter8 will take your answers to these questions and write them into the build.sbt file for your application, so you can go change them in that file later on. Table 1 shows a basic rundown on what the questions mean. Once you've answered the questions, your answers will be applied to the giter8 templates and the project skeleton will be saved out onto your local system.

Table 1 g8 questions

g8 question	What it means
organization	Used for publishing. Should be the reverse of a domain name you control. If you don't own a domain,

	com.github.username is a popular choice.
package	All Scala code belongs in a package. The Scala Style Guide recommends that your packages start with your organization. This convention is used across multiple JVM languages and gives your project a globally unique namespace.
name	The name of your project. g8 will generate a project into a folder of this name, and the artifacts you publish will be based on this name.
servlet_name	The name of your servlet. This might be something like BlogServlet or just Blog.
scala_version	The version of Scala your project is built with. When in doubt, use the default.
version	The version number of your project. This is entirely up to you, but we like Semantic Versioning.

Once you've answered the questions, your answers will be applied to the giter8 templates and the project skeleton will be saved out onto your local system.

Check for giter8 templates to speed you up

There are quite a few different Scalatra Giter8 templates, for various purposes. If you're looking to find out how to integrate Scalatra with some other library, do a bit of searching on GithHub, and you may be pleasantly surprised.

Downloading dependencies and building the app

Do a to enter the top-level cd scalatra-sbt-prototype directory of your new application, and type `sbt`. This will take a little while, especially the first time. SBT looks at the file build.sbt and downloads Scala, a Scala compiler, Scalatra, and a small set of dependencies of the Scalatra application. That one three-letter command gives you a full web development environment!

Once sbt has finished downloading everything, you'll get an sbt prompt, which looks like this: `'>'`.

Starting the Hello, World application

Let's start the HelloWorld application:

```
container:start
```

That will compile the application and start a web server running on http://localhost:8080. When you see some output like this, you'll know it's running:

```
[info] Started SelectChannelConnector@0.0.0.0:8080
```

Go ahead and visit it in your browser. The first request will be slow, as the application sets itself up for the first time; subsequent requests will get faster and faster as the JVM optimizes code paths for your machine.

We can now take a quick look at the application code. Open up the file scalatra-sbt-prototype/src/main/scala/com/example/app/MyScalatraServlet.scala. You'll see the following:

Listing 3 Your first Scalatra application

```
          package com.example.app
import org.scalatra._
import scalate.ScalateSupport
class MyScalatraServlet extends ScalatraServlet with ScalateSupport {
  get("/") {
    <html>
      <body>
        <h1>Hello, world!</h1>
        Say <a href="hello-scalate">hello to Scalate</a>.
      </body>
    </html>
  }
```

```
  notFound {
    // remove content type in case it was set through an action
    contentType = null
    // Try to render a ScalateTemplate if no route matched
    findTemplate(requestPath) map { path =&gt;
      contentType = "text/html"
      layoutTemplate(path)
    } orElse serveStaticResource() getOrElse resourceNotFound()
  }
}
```

This is a very simple Scalatra app, containing a single HTTP route and some code to deal with 404 Not Found errors.

Making changes and seeing them in your browser

Let's change the code so that the response is exactly the same as the original "Hello, world!" example which started off this article.

Listing 4 Changing the output from an XML literal to a string

```
get("/") {
  "Hello world!"
}
```

Save the file and refresh your browser. The example hasn't changed. Scala needs to recompile the program in order to make your changes appear. Doing this manually every time you made a change to your source code would be pretty awful, so there's a handy way to make your changes appear automatically.

Back at the SBT console, type this:

```
~ ;copy-resources;aux-compile
```

From this point on, every time you change your code, SBT will automatically recompile to make your changes visible. Hit Refresh, and you should see your changes.

Lastly, let's change the URL matcher. Change this:

Listing 5 Changing the url matcher

```
// change this:
get("/") {
"Hello world!"
}
// to this:
get("/hello") {
  "Hello world!"
}
```

Visit it in your browser, at http://localhost:8080/hello. You're done!

Summary

You have seen your first Scalatra code examples and found out about some specific areas where Scalatra shines. Lastly, you've built and run a Hello World application using Scalatra.

Here are some other Manning titles you might be interested in:

Functional Programming in Scala
Paul Chiusano and Rúnar Bjarnason

Scala in Depth
Joshua D. Suereth

Play for Scala
Peter Hilton, Erik Bakker, and Francisco Canedo

Last updated: January 25, 2013