www.autoscout24.com

Scala School – Filters & Action Composition

02.02.2016 Matthew Lloyd, Alexey Gravanov

What are Actions?

The standard way of handling requests in Play

```
val echo = Action { request =>
  Ok("Got request [" + request + "]")
}
```

- They accept a `Request[A]`
- They return a `Result`
- Asynchronous by default (they return in a `Future[Result]`)
- Utilise BodyParsers to parse request bodies.

Custom Actions
Timing Action

A Basic Timing Action

We can do more with Actions by creating custom actions

- Let's make a simple action that will calculate how long our service takes to respond.
- We need to be able to wrap an existing `Action`.
- Calculate the time it's taken.
- And modify it's response by adding a `Response-Time` Header

A Basic Timing Action - Implementation

```
case class TimingAction[A](action: Action[A]) extends Action[A] {

  def apply(request: Request[A]): Future[Result] = {
    val startTime = System.currentTimeMillis

    action(request).map { result =>
      val endTime = System.currentTimeMillis
      val requestTime = endTime - startTime

      result.withHeaders("Request-Time" -> requestTime.toString)
    }
  }

  override def parser: BodyParser[A] = action.parser
}
```

A Basic Timing Action

What's this doing?

- Wrap the existing action in the constructor.
- Implement the `apply` method.
- Call the wrapped action.
- Map the `Future[Result]` and add the Header.
- Ensure we carry across any existing BodyParsers.

A Basic Timing Action – Usage

```
def veryLongAction =
  TimingAction {
    Action {
      val result = doSomethingComplicated()
      Ok(s"Got $result")
    }
  }
```

Custom Actions
Login Action

A Login Action

- Let's make a simple action that will accept a `UserID` and `Pass`.
- Lookup the User from the ID.
- Compare the password.
- And either allow access to the resource or return `Forbidden`.

A Login Action – Implementation

```
case class LoginAction(userId: Int, pass: String) extends ActionBuilder[Request] {
  def invokeBlock[A](request: Request[A],
                     block: (Request[A]) => Future[Result]): Future[Result] = {
    User.get(userId) match {
      case Some(userAccount) if pass == userAccount.pass => block(request)
      case _ => Future(Results.Forbidden)
    }
  }
}
```

A Login Action - Usage

```
def authorised(userId: Int, pass: String) =
  LoginAction(userId, pass) { req =>
    Ok("Super Secret Page")
  }
```

Custom Actions
Advanced Login Action

An Advanced Login Action

- We want to be able to access our `User` Data once logged in.
- We need to pass the `User` data through the `Request`.
- Let's use a `WrappedRequest` to include `User` Data.

An Advanced Login Action – Custom Request

```
case class UserWrappedRequest[A](user: User,
                                 request: Request[A]) extends WrappedRequest[A](request)
```

An Advanced Login Action - Implementation

```
case class AdminLoginAction(userId: Int, pass: String)
  extends ActionBuilder[UserWrappedRequest] {

  def invokeBlock[A](request: Request[A],
                     block: (UserWrappedRequest[A]) => Future[Result]): Future[Result] = {
    User.get(userId) match {
      case Some(userAccount) if userAccount.pass == pass =>
        block(UserWrappedRequest(userAccount, request))
      case _ =>
        Future(Results.Forbidden)
    }
  }

}
```

An Advanced Login Action - Usage

```
def authorised(userId: Int, pass: String) =
  AdminLoginAction(userId, pass) { req =>
    Ok(s"Super Secret Page that only ${req.user.name} can see")
  }
```

Custom Actions
Putting it all together

Putting it all together

- `ActionBuilder` allows us to easily compose multiple Actions.
- You can build up Actions to perform specific tasks.
- Then compose them together when using them.

Putting **it all together – Action Composition**

```
def authorised(userId: Int, pass: String) =
  (TimingAction() andThen AdminLoginAction(userId, pass)) { req =>
    Ok(s"Super Secret Page that only ${req.user.name} can see")
  }
```

Filters & Actions
Filters

What are Filters?

A way of applying logic to all requests.

- Filters will apply to all requests.
- They can be used similarly to Action Composition.
- Good for Timing, Logging, Gzip etc...

A Timing Filter – Implementation

```
class TimingFilter extends Filter {

  def apply(nextFilter: RequestHeader => Future[Result])
           (requestHeader: RequestHeader): Future[Result] = {

    val startTime = System.currentTimeMillis

    nextFilter(requestHeader).map { result =>

      val endTime = System.currentTimeMillis
      val requestTime = endTime - startTime

      result.withHeaders("Request-Time" -> requestTime.toString)
    }
  }
}
```

A Basic HTTP Auth Filter – Implementation

```
class HTTPBasicAuthFilter extends Filter {

  def decodeBasicAuth(auth: String): Option[(String, String)] = ???

  def apply(nextFilter: (RequestHeader) => Future[Result])
           (requestHeader: RequestHeader): Future[Result] = {

    requestHeader.headers.get("authorization").map { basicAuth =>
      decodeBasicAuth(basicAuth) match {
        case Some((user, pass)) if User.authenticate(user, pass) =>
          nextFilter(requestHeader)
        case _ =>
          Future.successful(Results.Unauthorized)
      }
    }.getOrElse(Future.successful(Results.Unauthorized))

  }
}
```

Filters - Usage

```
class AS24Filters @Inject() (
  timingFilter: TimingFilter,
  authFilter: HTTPBasicAuthFilter
) extends HttpFilters {
  val filters = Seq(timingFilter, authFilter)
}
```

```
play.http.filters=filters.AS24Filters
```

Anything else to know?

- Filters are executed in a chain one by one.
- Filters can modify both requests and results.
- Filters are executed AFTER routing but BEFORE actions.
- This means they have access to the routing info from the Request header.

When to use Filters vs Action Composition?

Filters

- Only to be used for global cross cutting concerns.
- When you need to affect ALL routes at once.

Action Composition

- For specific concerns for certain endpoints.
- When you need access to Route Parameters.
- Should only really be used for sub set of endpoints.