Sessions and Cookies

Contents

Sessions

Luminus defaults to using in-memory sessions.

When using the Immutant server the sessions are backed by the servlet session provided by the wrap-session middleware.

The session middleware is initialized in the `<app>.middleware` namespace by the `wrap-base` function. Session timeout is specified in seconds and defaults to 30 minutes of inactivity.

```
(defn wrap-base [handler]
  (-> handler
      wrap-dev
      wrap-formats
      wrap-webjars
      (wrap-defaults
        (-> site-defaults
            (assoc-in [:security :anti-forgery] false)
            (dissoc :session)))
      wrap-flash
      wrap-session
      wrap-context
      wrap-internal-error))
```

Otherwise, sessions are backed by the ring-ttl-session library. It provides a session store that stores the data in-memory with a time-to-live (TTL).

We can easily swap the default memory store for a different one, such as a cookie store. Below, we explicitly specify the

`ring.middleware.session.cookie/cookie-store` with the name `example-app-session` as our session store:


```
(wrap-defaults
  (-> site-defaults
      (assoc-in [:security :anti-forgery] false)
      (assoc-in [:session :store] (cookie-store))
      (assoc-in [:session :cookie-name] "example-app-sessions")))
```

We can also specify the maximum age for our session cookies using the `:max-age` key:

```
(wrap-defaults
  (-> site-defaults
      (assoc-in [:security :anti-forgery] false)
      (assoc-in [:session :store] (cookie-store))
      (assoc-in [:session :cookie-attrs] {:max-age 10})))
```

When using cookie store it is also important to specify a secret key (16 characters) for cookie encryption. Otherwise a random one will be generated each time application is started and sessions created before will be lost.

```
(wrap-defaults
  (-> site-defaults
      (assoc-in [:security :anti-forgery] false)
      (assoc-in [:session :store] (cookie-store {:key "BuD3KgdAXhDHrJX
      (assoc-in [:session :cookie-name] "example-app-sessions")))
```

You may also wish to take a look at Redis for your session store. Creating Redis sessions is easy thanks to Carmine. You would simply need to define a connection and use the `taoensso.carmine.ring/carmine-store` with it:

```
(def redis-conn {:pool {<opts>} :spec {<opts>}})
```

```
(wrap-defaults
  (-> site-defaults
      (assoc-in [:security :anti-forgery] false)
      (assoc-in [:session :store] (carmine-store redis-conn))))
```

For further information, please see the official API documentation.

Accessing the session

Ring tracks sessions using the request map and the current session will be found under the `:session` key. Below we have a simple example of interaction with the session.

```
(ns myapp.home
  (:require [compojure.core :refer [defroutes GET]]
            [ring.util.response :refer [response]]))
```

```
(defn set-user! [id {session :session}]
  (-> (response (str "User set to: " id))
      (assoc :session (assoc session :user id))
      (assoc :headers {"Content-Type" "text/plain"})))

(defn remove-user! [{session :session}]
  (-> (response "User removed")
      (assoc :session (dissoc session :user))
      (assoc :headers {"Content-Type" "text/plain"})))

(defn clear-session! []
  (-> (response "Session cleared")
      (dissoc :session)
      (assoc :headers {"Content-Type" "text/plain"})))

(defroutes app-routes
  (GET "/login/:id" [id :as req] (set-user! id req))
  (GET "/remove" req (remove-user! req))
  (GET "/logout" req (clear-session!)))
```

Note that the the default `<app>.layout/render` function does not allow setting the session. The function is intended to render the page and this should not be conflated with any controller actions. In a scenario where you wish to set the session and render a page a redirect is the recommended approach.

Flash sessions

Flash sessions have a lifespan of a single request, these can be accessed using the `:flash` key instead of the `:session` key used for regular sessions.

Cookies

Cookies are found under the `:cookies` key of the request, eg:

```
{:cookies {"username" {:value "Bob"}}}
```

Conversely, to set a cookie on the response we simply update the response map with the desired cookie value:

```
(-> "response with a cookie" response (assoc-in [:cookies "username" :
```

Cookies can contain the following additional attributes in addition to the `:value` key:

:domain - restrict the cookie to a specific domain

:path - restrict the cookie to a specific path

:secure - restrict the cookie to HTTPS URLs if true

:http-only - restrict the cookie to HTTP if true (not accessible via e.g. JavaScript)
:max-age - the number of seconds until the cookie expires
:expires - a specific date and time the cookie expires

Luminus framework is released under the MIT License - Copyright © 2019