Response types

Contents

Responses

Ring responses are generated using the ring-http-response library. The library provides a number of helpers for producing responses with their respective HTTP Status codes.

For example, the `ring.util.http-response/ok` helper is used to generate a response with the status `200`. The following code will produce a valid response map with the content set as its `:body` key.

```
(ok {:foo "bar"})

;;result of calling response
{:status  200
 :headers {}
 :body    {:foo "bar"}}
```

The response body can be one of a string, a sequence, a file, or an input stream. The body must correspond appropriately with the response's status code.

A string, it will be sent back to the client as is. For a sequence, a string representing each element is sent to the client. Finally, if the response is a file or an input stream, then the server sends its contents to the client.

Response encoding

Build Tool: lein ▾

Topics

Your First Application
REPL Driven Developme
Application Profiles
HTML Templating
Static Assets
ClojureScript
Routing
RESTful Services
Request types
- **Response types**

Websockets
Middleware
Sessions and Cookies
Input Validation
Security
Component Lifecycle
Database Access
Database Migrations
Logging
Internationalization
Testing
Server Tuning
Environment Variables
Deployment
Useful Libraries
Sample Applications
Upgrading
Clojure Resources

Books

By default, the muuntaja middleware library is used to infer the response type when a route returns a map containing the `:body` key:

```
(GET "/json" [] {:body {:foo "bar"}})
```

The middleware is found in the `<app-name>.middleware` namespace of your application. The middleware function is called `wrap-formats`:

```
(defn wrap-formats [handler]
  (let [wrapped (-> handler wrap-params wrap-format)]
    (fn [request]
      ;; disable wrap-formats for websockets
      ;; since they're not compatible with this middleware
      ((if (:websocket? request) handler wrapped) request))))
```

Muuntaja will use the `Content-Type` header to infer the content of the request, and the `Accept` header to infer the response format.

ADDING CUSTOM ENCODERS WITH MUUNTAJA

Some types might not have default encoders available and cannot be serialized automatically. For example, if you're using clj-time it would be nice to have the DateTime objects it produces serialize automatically.

Here's an example of how we can add a custom serializer for `org.joda.time.DateTime` objects generated by cljs-time using transit.

First, we'll need to require the namespaces for working with clj-time and transit:

```
(ns <app>.middleware
  (:require
   ...
   [cognitect.transit :as transit]
   [clj-time.coerce :as coerce]
   [muuntaja.core :as m]
   [muuntaja.format.transit :as formats]))
```

Next, we'll add a transit writer for the `org.joda.time.DateTime` type:

```
(def joda-time-writer
  (transit/write-handler
    (constantly "m")
    (fn [v] (-> ^org.joda.time.ReadableInstant v .getMillis))
    (fn [v] (-> ^org.joda.time.ReadableInstant v .getMillis .toString)
```

Finally, we'll update the default options passed to `wrap-format` to let it know that we'd like to use `joda-time-writer` for this type:

```
(def wrap-format-options
  (update
    m/default-options
    :formats
    merge
    {"application/transit+json"
     {:decoder [(partial formats/make-transit-decoder :json)]
```

```
      :encoder [#(formats/make-transit-encoder
                  :json
                  (merge
                    %
                    {:handlers {org.joda.time.DateTime joda-time-writ

(defn wrap-formats [handler]
  (let [wrapper (-> handler wrap-params (wrap-format wrap-format-optio
    (fn [request]
      ;; disable wrap-formats for websockets
      ;; since they're not compatible with this middleware
      ((if (:websocket? request) handler wrapped) request))))
```

The dates produced by clj-time will now be automatically serialized when returned as part of the response.

Setting headers

Setting additional response headers is done by calling `ring.util.http-response/header`, and passing it a map of HTTP headers. Note that the keys **must** be strings.

```
(-> "hello world" response (header "x-csrf" "csrf"))
```

Setting content type

You can set a custom response type by using the `ring.util.http-response/content-type` function, eg:

```
(GET "/project" []
  (-> (clojure.java.io/input-stream "report.pdf")
      ok
      (content-type "application/pdf")))
```

Setting custom status

Setting a custom status is accomplished by passing the content to the `ring.util.http-response/status` function:

```
(GET "/missing-page" []
  (-> "your page could not be found"
      ok
      (status 404)))
```

Redirects

Redirects are handled by the `ring.util.http-response/found` function. The function will set a `302` redirect status on the response.

```
(GET "/old-location" []
  (found "/new-location"))
```

Please refer to the ring-http-response to see other available helpers.
