Server Tuning

Contents

While Luminus aims to provide reasonable defaults, you will likely want to tune your server configuration to fit the needs of your specific application.

Application Middleware

Luminus includes a number of useful wrappers in its default middleware stack. These include ring-defaults, session middleware, ring-middleware-format, and ring-webjars.

While such middleware is useful in most situations, it will also incur a performance cost. In cases where you wish to optimize for performance, only wrap the routes with the necessary middleware. For example, if a route doesn't server static resources, then you would not need to wrap it with the webjars middleware.

Immutant Configuration

Luminus defaults to using Immutant as the default server. This server provides a number of options that are unique to it.

Immutant allows setting the number of worker and IO threads using the `:worker-threads` and the `:io-threads` keys respectively. For example, we could update the default configuration as follows:

```
(mount/defstate http-server
  :start
  (http/start
    (-> env
        (assoc
```

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

```
          :handler handler/app
          :worker-threads 200
          :io-threads (* 2 (.availableProcessors (Runtime/getRuntime))
        (update :port #(or (-> env :options :port) %))))
  :stop
  (http/stop http-server))
```

Note that the `env` is used to supply the base configuration, meaning that any initialization parameters can also be supplied using the environment variables at runtime.

Another feature provided by Immutant is the ability to chain multiple independent handlers together. Luminus provides the `wrap-handler` helper function for this purpose:

```
(mount/defstate http-server
                :start
                (->
                   (http/start
                     (-> env
                        (assoc :handler handler/app)
                        (update :port #(or (-> env :options :port) %))
                   (http/wrap-handler io-routes {:path "/io" :dispatch
                :stop
                (http/stop http-server))
```

The additional handler uses a `:path` prefix as a context. All the routes served by this handler will be prefixed with the supplied path. Each handler can have its own middleware stack that's independent of other handlers in the application.

Immutant uses separate thread pools for managing the IO and the worker threads. The `:dispatch?` flag is used to decide whether the request should be dispatched by the IO thread to a separate worker thread. Since dispatching the request to a worker carries overhead, it may be more performant to handle some requests, such as hardcoded text responses, directly in the IO thread.

Context paths

Set the value of `:handler-path` key to customize the global path for the application (default is `/`). In the example below the value is gotten from a custom environment variable `:my-path`.

```
(mount/defstate http-server
    :start
    (http/start
      (-> env
        (assoc :handler (handler/app))
        (update :port #(or (-> env :options :port) %))
        (update :handler-path #(or (-> env :my-path) %))))
    :stop
    (http/stop http-server))
```

You can also supply the `:app-context` key in the environment that's used by the `wrap-context` wrapper in the `middleware` namespace. It will populate the `*app-context*` variable in the `layout` namespace. That can be used to populate the context on the page for the client to use.
