Joyride + SCI

Small Clojure Interpreter

```
$ clj
Clojure 1.11.0
user=> (require '[sci.core :as sci])
nil
user=> (sci/eval-string "(+ 1 2 3)")
6
```

```
$ clj -M:cljs -m cljs.main -re node
ClojureScript 1.11.54
cljs.user=> (require '[sci.core :as sci])
nil
cljs.user=> (sci/eval-string "(+ 1 2 3)")
6
```

Small Clojure Interpreter

NEW USAGES OF CLJS THROUGH SCI

Nbb (Node.js)

Scittle (browser)

Joyride (VSCode)

4ever-clojure

Clerk viewers

Maria.cloud (soon)

More...

Sci: adding libs

```
(ns my-namespace
  (:require [sci.core :as sci]))

(defn my-function [x] (inc x))

(def my-sci-namespace
  {'my-function my-function})

(def sci-opts
  {:namespaces
    {'my-namespace my-sci-namespace}})

(sci/eval-string
  "(require '[my-namespace :as mns])
   (mns/my-function 1)"
  sci-opts)
;; ⇒ 2
```

Sci.configs

A collection of ready to be used SCI configs:

https://github.com/babashka/sci.configs

```
(ns example
  (:require [sci.configs.funcool.promesa :as promesa-config]
            [sci.core :as sci]))

(def sci-ctx
  (sci/init
    {:namespaces
      {'promesa.core promesa-config/promesa-namespace
       'promesa.protocols promesa-config/promesa-protocols-namespace}}))

(→ (sci/eval-string* sci-ctx
      "(require '[promesa.core :as p])
       (p/delay 1000 :hello)")
    (.then prn))
;; ⇒ :hello
```

Sci: JS defaults

```
(ns my.sci-env
  (:require [sci.core :as sci]))

(def ctx (sci/init
           {:classes {'js js/globalThis
                      'Math js/Math
                      :allow :all}}))

(sci/enable-unrestricted-access!) ;; alter-var-root + set!-ing vars without binding
(sci/alter-var-root sci/print-fn (constantly *print-fn*))
(sci/alter-var-root sci/print-err-fn (constantly *print-err-fn*))

(sci/eval-string* ctx "(prn (Math/sin 1337))")
;; ⇒ -0.9683343651587963
;; ⇒ nil
```

Loading workspace and user scripts

master joyride / examples / .joyride / scripts / clojuredocs.cljs

PEZ Make clojuredocs example use REPL first then fallback on clojure.core... ...

1 contributor

46 lines (42 sloc) | 1.81 KB

```
(ns clojuredocs
  (:require ["ext://betterthantomorrow.calva$v0" :refer [ranges repl]]
            ["vscode" :as vscode]
            [clojure.string :as string]
            [clojure.edn :as edn]
            [joyride.core :as joyride]
            [promesa.core :as p]))

```

Loading workspace and user scripts

```
(require '[clojuredocs :as docs])
```

```
(defn ns->path [namespace]
  (-> (str namespace)
      (munge)
      (str/replace  "." "/")
      (str ".cljs")))

(defn source-script-by-ns [namespace]
  (let [ns-path (ns->path namespace)
        path-if-exists (fn [search-path]
                         (let [file-path (path/join search-path ns-path)]
                           (when (fs/existsSync file-path)
                             file-path)))
        ;; workspace first, then user - the and is a nil check for no workspace
        path-to-load (first (keep #(and % (path-if-exists %))
                                  [(conf/workspace-abs-scripts-path) (conf/user-abs-scripts-path)]))]
    (when path-to-load
      {:file ns-path
       :source (str (fs/readFileSync path-to-load))})))
```

Sci: loading external libs

```
:load-fn (fn [{:keys [ns libname opts]}]
           (cond
             (symbol? libname)
             (source-script-by-ns libname)
             (string? libname) ;; node built-in or npm library
             (cond
               (= "vscode" libname)
               (do (sci/add-class! @!ctx 'vscode vscode)
                   (sci/add-import! @!ctx ns 'vscode (:as opts))
                   {:handled true})

               (active-extension? libname)
               (let [module (extension-module libname)
                     munged-ns (symbol (munge libname))
                     refer (:refer opts)]
                 (sci/add-class! @!ctx munged-ns module)
                 (sci/add-import! @!ctx ns munged-ns (:as opts))
                 (when refer
                   (doseq [sym refer]
                     (let [prop (gobject/get module sym)
                           sub-sym (symbol (str munged-ns "$" sym))]
                       (sci/add-class! @!ctx sub-sym prop)
                       (sci/add-import! @!ctx ns sub-sym sym))))
                 {:handled true})

               :else
               (let [mod (require* @sci/file libname)
                     ns-sym (symbol libname)]
                 (sci/add-class! @!ctx ns-sym mod)
                 (sci/add-import! @!ctx ns ns-sym
                                  (or (:as opts)
                                      ns-sym))
                 {:handled true}))))}))
```

NPM libaries

```
(defn require* [from-script lib]
  (let [req (module/createRequire (path/resolve (or from-script "./script.cljs")))
        resolved (.resolve req lib)]
    (js/require resolved)))
```

```
  :else
  (let [mod (require* @sci/file libname)
        ns-sym (symbol libname)]
    (sci/add-class! @!ctx ns-sym mod)
    (sci/add-import! @!ctx ns ns-sym
                     (or (:as opts)
                         ns-sym))
    {:handled true})))})))
```

NPM libaries

Get Started | npm.cljs M

.joyride > scripts > npm.cljs > {} npm

```
(ns npm
  (:require ["moment" :as moment])) => nil

(.format (moment) "dddd") => "Tuesday"

(require '["axios" :as axios]
         '[promesa.core :as p])

(def result (atom nil))
(->
 (axios.get "https://clojure.org")
 (p/then #(reset! result %))) => #<Promise[~]>

@result => #js {:status 200, :statusText "OK", :headers #object[Axiosh
```

Require extension APIs

```
(ns z-joylib.calva-api
  (:require ...
            ["ext://betterthantomorrow.calva$v0" :as calva :refer [repl ranges]])
            ...)

(def current-form-text (second (ranges.currentForm)))
```

```
(active-extension? libname)
(let [module (extension-module libname)
      munged-ns (symbol (munge libname))
      refer (:refer opts)]
  (sci/add-class! @!ctx munged-ns module)
  (sci/add-import! @!ctx ns munged-ns (:as opts))
  (when refer
    (doseq [sym refer]
      (let [prop (gobject/get module sym)
            sub-sym (symbol (str munged-ns "$" sym))]
        (sci/add-class! @!ctx sub-sym prop)
        (sci/add-import! @!ctx ns sub-sym sym))))
  {:handled true})
```

nREPL

```clojure
(defn do-handle-eval [{:keys [ns code _sci-ctx-atom _load-file? file] :as request} send-fn] 2 references
  (sci/with-bindings
    {sci/ns ns
     sci/print-length @sci/print-length
     sci/print-newline true
     sci/file (or file @sci/file)}
    ;; we alter-var-root this because the print-fn may go out of scope in case
    ;; of returned delays
    (sci/alter-var-root sci/print-fn (constantly
                                      (fn [s]
                                        (send-fn request {"out" s}))))
    (try (let [v (jsci/eval-string code)]
           (sci/alter-var-root sci/*3 (constantly @sci/*2))
           (sci/alter-var-root sci/*2 (constantly @sci/*1))
           (sci/alter-var-root sci/*1 (constantly v))
           (send-fn request {"value" (format-value (:nrepl.middleware.print/print request)
                                                   (:nrepl.middleware.print/options request)
                                                   v)
                             "ns" (str @sci/ns)})
           (send-fn request {"status" ["done"]}))
         (catch :default e
           (sci/alter-var-root sci/*e (constantly e))
           (let [data (ex-data e)]
             (when-let [message (or (:message data) (.-message e))]
               (send-fn request {"err" (str message "\n")}))
             (send-fn request {"ex" (str e)
                               "ns" (str @sci/ns)
                               "status" ["done"]}))))))
```

Built-in

Implemented fully in CLJS

Calva: start joyride REPL