Structure

Just enough Clojure ✓

Clojure State & Concurrency

ClojureScript & core.async

Structure

Just enough Clojure ✓

Clojure State & Concurrency

ClojureScript & core.async

State and Concurrency

State is Messy

Most OO language
state is all over the place

State and Concurrency

State is Messy

Most OO language
state is all over the place
complexity + concurrency =
DISASTER

State and Concurrency

Clojure to the rescue!

functional style +
immutable data structures =
concurrency

State and Concurrency

Atoms

store state that is *independent*

State and Concurrency

Atoms

store state that is *independent*

```
(def who-atom (atom :caterpillar))
```

State and Concurrency

Atoms

need to dereference it with @

```
(def who-atom (atom :caterpillar))

who-atom
;; -> #<Atom@e6df69d: :caterpillar>

@who-atom
;; -> :caterpillar
```

State and Concurrency

Atoms

change the value with reset!

```
(reset! who-atom :chrysalis)
;; -> :chrysalis

@who-atom
;; -> :chrysalis
```

State and Concurrency

Atoms

change the value with swap!

```
(defn change [state]
  (case state
    :caterpillar :chrysalis
    :chrysalis :butterfly
    :butterfly))
```

State and Concurrency

Atoms

change the value with swap!

```
(swap! who-atom change)
;; -> :chrysalis

@who-atom
;; -> :chrysalis
```

State and Concurrency

Atoms

How does this work?

- swap! reads the value of atom
- applies the function
- compares the value again to make sure another thread hasn't changed it
- sets the value to the result

State and Concurrency

Atoms

How does this work?

- swap! reads the value of atom
- applies the function
- compares the value again to make sure another thread hasn't changed it
- sets the value to the result

might be retries – so beware side effects

State and Concurrency

Atoms

operation is atomic

other threads do not see inconsistent vals

dereferencing atom does not block

either sees value before or after swap

State and Concurrency

Atoms

change the value with swap!

```
(def counter (atom 0))
@counter
;; -> 0

(dotimes [_ 5] (swap! counter inc))
@counter
;; -> 5
```

State and Concurrency

Atoms

MORE THREADS

```
(def counter (atom 0))
@counter
;; -> 0

(let [n 5]
  (future (dotimes [_ n] (swap! counter inc)))
  (future (dotimes [_ n] (swap! counter inc)))
  (future (dotimes [_ n] (swap! counter inc))))
@counter
;; -> 15
```

State and Concurrency

Atoms

MORE THREADS

```
(def counter (atom 0))
@counter
;; -> 0

(let [n 5]
  (future (dotimes [_ n] (swap! counter inc)))
  (future (dotimes [_ n] (swap! counter inc)))
  (future (dotimes [_ n] (swap! counter inc))))
@counter
;; -> 15
```

State and Concurrency

Atoms

MORE THREADS

```
(def counter (atom 0))
@counter
;; -> 0

(let [n 5]
  (future (dotimes [_ n] (swap! counter inc)))
  (future (dotimes [_ n] (swap! counter inc)))
  (future (dotimes [_ n] (swap! counter inc))))
@counter
;; -> 15
```

Run in another thread

State and Concurrency

Atoms

What about side effects?

```
(def counter (atom 0))

(defn inc-print [val]
  (println val)
  (inc val))

(swap! counter inc-print)
;; 0
;; -> 1
```

State and Concurrency

Atoms

What about side effects?

```
(def counter (atom 0))
(let [n 2]
(future (dotimes [_ n] (swap! counter inc-print)))
(future (dotimes [_ n] (swap! counter inc-print)))
(future (dotimes [_ n] (swap! counter inc-print))))
```

State and Concurrency

Atoms

What about side effects?

```
;; 0
;; 1
;; 2
;; 2
;; 3
;; 4
;; 5
@counter
;; -> 6
```

State and Concurrency

Atoms

What about side effects?

```
;; 0
;; 1
;; 2
;; 2
;; 3
;; 4
;; 5
@counter
;; -> 6
```

Swap! Retrying

State and Concurrency

Atoms

Independent & Synchronous Changes

State and Concurrency

Atoms

Independent & Synchronous Changes

What about Coordinated Changes?

State and Concurrency

Refs

Allow for Coordinated Shared State

Have to change value in transaction

Software Transactional Memory (STM)

State and Concurrency

Refs

Atomic: error with one ref – none updated

Consistent: optional validator before committing

Isolated: every transaction has isolated view of world

State and Concurrency

Refs

defining

```
(def alice-height (ref 3))
(def right-hand-bites (ref 10))

@alice-height
;; -> 3
@right-hand-bites
;; -> 10
```

State and Concurrency

Refs

updating with alter and dosync

```
(defn eat-from-right-hand []
  (when (pos? @right-hand-bites)
    (alter right-hand-bites dec)
    (alter alice-height #(+ % 24))))
```

State and Concurrency

Refs

updating with alter and dosync

```
(eat-from-right-hand)
;; -> IllegalStateException No
transaction running
```

need to do it in an transaction

State and Concurrency

Refs

updating with alter and dosync

```
(dosync (eat-from-right-hand))
;; -> 27
```

need to do it in an transaction

State and Concurrency

Refs

updating with alter and dosync

```
(let [n 2]
  (future (dotimes [_ n] (dosync (eat-from-right-hand))))
  (future (dotimes [_ n] (dosync (eat-from-right-hand))))
  (future (dotimes [_ n] (dosync (eat-from-right-hand))))

@alice-height
;; -> 147

@right-hand-bites
;; -> 4
```

State and Concurrency

Refs

resetting value with ref-set

```
(def alice-height (ref 3))

(dosync (ref-set alice-height 5))

@alice-height
;; -> 5
```

State and Concurrency

Agents

Independent and asynchronous changes

State and Concurrency

Agents

defining

```
(def who-agent (agent :caterpillar))

@who-agent
;; -> :caterpillar
```

State and Concurrency

Agents

send

```
(defn change [state]
  (case state
    :caterpillar :chrysalis
    :chrysalis :butterfly
    :butterfly))

(send who-agent change)
;; -> #<Agent@31c89c8b>

@who-agent
;; -> :chrysalis
```

State and Concurrency

Agents

send

- Send dispatches the action to the agent
- Gets processed by thread in thread pool
- Only processes one action at a time
- Like a pipeline
- send returns immediately

State and Concurrency

Agents

send-off

- Use send-off if you might block on I/O
- send uses fixed thread pool (good for cpu bound operations)
- send-off uses expandable thread pool in case of i/o blocking

State and Concurrency

Summary

Type	Communication	Coordination
Atom	Synchronous	Uncoordinated
Ref	Synchronous	Coordinated
Agent	Asynchronous	Uncoordinated

State and Concurrency

Wait there is more!

`pmap`

like map but parallel (with futures)

```
(pmap inc [1 2 3 4])
;; -> (2 3 4 5)
```

State and Concurrency

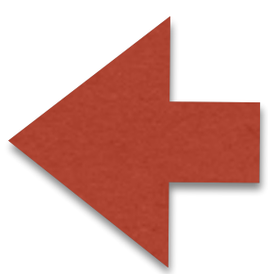
Wait there is more!

`promise`

set only once with `deliver`

```
(def my-promise (promise))

@my-promise
```



Will Block

State and Concurrency

Wait there is more!

`promise`

set only once with `deliver`

```
(def my-promise (promise))

(realized? my-promise)
;; -> false
```

State and Concurrency

Wait there is more!

`promise`

set only once with `deliver`

```
(deliver my-promise "cake")

(realized? my-promise)
;; -> true

@my-promise
;; -> "cake"
```

State and Concurrency

Wait there is more!

`delay`

evaluates when deref then cache

```
(def my-delay (delay (println "Cake is tasty") "cake"))

@my-delay
;; prints "Cake is tasty"
;; -> "cake"

@my-delay
;; -> "cake"
```

Exercises!

Summary

- cd oscon-solve-concurrency
- cd clojure-intro
- lein test-refresh
- open intro2_state_concurrency_test.clj
- uncomment tests to get started

Structure

Just enough Clojure ✓

Clojure State & Concurrency ✓

ClojureScript & core.async

core.async

Library

Async & Concurrent Communication using Channels

core.async

clojure / core.async

Watch 149 | Star 948

Facilities for async programming and communication in Clojure

364 commits | 14 branches | 16 releases | 13 contributors

branch: master | core.async / +

fix bad test, missing comparison value in =

puredanger authored 6 days ago — latest commit 27117a2bd5

doc	initial commit	2 years ago
examples	Clarify printing in walkthrough	2 years ago
script	remove maven debug flags	2 years ago
src	fix bad test, missing comparison value in =	6 days ago
.gitignore	update cljsbuild, update builds, clean targets & ignores	5 months ago
CONTRIBUTING.md	Add github CONTRIBUTING file	2 years ago
README.md	Update latest version in README	10 months ago
VERSION_TEMPLATE	add VERSION_TEMPLATE text file	2 years ago
epl.html	initial commit	2 years ago
pom.template.xml	update to tools.analyzer.jvm 0.6.6	3 months ago
project.clj	stop using old hand rolled runner, dogfood clj.test async support	3 months ago

Code

Pull req

Wiki

Pulse

Graphs

SSH clone URL

git@github

You can clone

or Subversion.

Clone

Dow

core.async

Basics

Creating a channel

```
(def tea-channel (async/chan))
```

core.async

Basics

Putting things on a channel synchronously

Blocking put: >!!

Blocking get:<!!

```
(def tea-channel (async/chan))
```

core.async

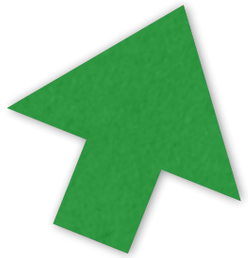
Basics

Putting things on a channel synchronously

Blocking put: >!!

Blocking get:<!!

```
(def tea-channel (async/chan))
```



unbuffered

core.async

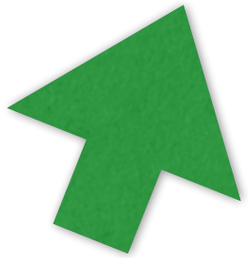
Basics

Putting things on a channel synchronously

Blocking put: >!!

Blocking get:<!!

```
(def tea-channel (async/chan 10))
```



buffered

core.async

Basics

Putting things on a channel synchronously

```
(async/>!! tea-channel :cup-of-tea)
;; -> true
```

core.async

Basics

Getting things off a channel synchronously

```
(async/<!! tea-channel)
;; -> :cup-of-tea
```

core.async

Basics

Closing a channel with close!

```
(async/>!! tea-channel :cup-of-tea-2)
;; -> true
(async/>!! tea-channel :cup-of-tea-3)
;; -> true
(async/>!! tea-channel :cup-of-tea-4)
;; -> true

(async/close! tea-channel)
;; -> nil
```

core.async

Basics

Closing a channel with close!

```
(async/>!! tea-channel :cup-of-tea-5)
;; -> false
```

Cannot put items on a closed channel

core.async

Basics

Closing a channel with close!

```
(async/<!! tea-channel)
;; -> :cup-of-tea-2
(async/<!! tea-channel)
;; -> :cup-of-tea-3
(async/<!! tea-channel)
;; -> :cup-of-tea-4
```

Can get items off of a closed channel

core.async

Basics

Closing a channel with close!

```
(async/<!! tea-channel)
;; -> nil
```

Get nil if channel is “drained” of values

core.async

Basics

Nil is special

```
(async/>!! tea-channel nil)
;; IllegalArgumentException Can't put
nil on channel
```

Cannot put nil on a channel

core.async

Basics

Putting things on/off a channel asynchronously

async put: `>!`

async get: `<!`

both need to be in a `go` block

core.async

Basics

Putting things on/off a channel asynchronously

```
(let [tea-channel (async/chan)]
  (async/go (async/>! tea-channel :cup-of-tea-1))
  (async/go
    (println "Thanks for the “
      (async/<! tea-   channel))))

;; Will print to stdout:
;; Thanks for the :cup-of-tea-1
```

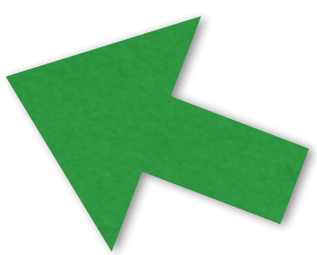
core.async

Basics

using a go-loop to continuously execute

```
(def tea-channel (async/chan 10))

(async/go-loop []
  (println "Thanks for the "
    (async/<! tea-channel))
  (recur))
```



keep going

core.async

Basics

using a go-loop to continuously execute

```
(async/>!! tea-channel :hot-cup-of-tea)
;; Will print to stdout:
;; Thanks for the :hot-cup-of-tea

(async/>!! tea-channel :tea-with-sugar)
;; Will print to stdout:
;; Thanks for the :tea-with-sugar

(async/>!! tea-channel :tea-with-milk)
;; Will print to stdout:
;; Thanks for the :tea-with-milk
```

core.async

How does this go-loop work?

- go block has special pool of threads
- take from channel doesn't block but pauses execution
- go-loop takes value from channel when available, then recur, and wait for the next value

core.async

multiple channels

alts!

wait for input across many channels

core.async

multiple channels

alts!

```
(def tea-channel (async/chan 10))
(def milk-channel (async/chan 10))
(def sugar-channel (async/chan 10))
```

core.async

multiple channels

alts!

```
(async/go-loop []
  (let [[v ch]
    (async/alts! [tea-channel
                  milk-channel
                  sugar-channel])]
    (println "Got " v " from " ch)
    (recur)))
```

core.async

multiple channels

alts!

```
(async/>!! sugar-channel :sugar)
;; Will print to stdout:
;;Got :sugar from #<ManyToManyChannel@2555e95>

(async/>!! milk-channel :milk)
;; Will print to stdout:
;; Got :milk from ManyToManyChannel@1a1850e5

(async/>!! tea-channel :tea)
;; Will print to stdout:
;; Got :tea from #ManyToManyChannel@130f42ba>
```

core.async

multiple channels

alts!

go blocks are lightweight

can have lots of channels

poll endpoints
while not blocking main processing

Exercises!

Summary

- `cd oscon-solve-concurrency`
- `cd async-playground`
- `lein test-refresh`
- `open src/async_playground/core.clj`
- follow along comments and experiment

Structure

Just enough Clojure ✓

Clojure State & Currency ✓

ClojureScript & core.async ✓

ClojureScript

Clojure for the Browser

Uses Google's Closure Compiler

Can use all of Google Closure libs

Google Closure compiler is really smart

Can use any JavaScript lib

ClojureScript

Clojure for the Browser

Subset of Clojure

there are atoms – but no refs, agents

JavaScript interop rather than Java

Only integer and floating point numbers

There is core.async!

ClojureScript

ClojureScript REPL

```
ClojureScript:cljs.user> (+ 1 1)
;; => 2
```

ClojureScript

ClojureScript REPL

```
(js/Date)
;; ->"Sun Oct 26 2014 11:27:20 GMT-0400 (EDT)"
```

ClojureScript

ClojureScript REPL

```
(first [1 2 3 4])
;; ->1
```

ClojureScript

ClojureScript REPL

```
(def x (atom 0))
;; -> #<Atom: 0>

(swap! x inc)
;; -> 1
```

ClojureScript

Figwheel

This repository Search

Pull requests Issues Gist

bhauman / **lein-figwheel**

Watch 42 St

Leiningen plugin that pushes ClojureScript code changes to the client

584 commits **7** branches **11** releases **29** contributors

branch: **master** **lein-figwheel** / +

update README and change log

bhauman authored 9 days ago latest commit 6b6fd443a2

doc	updating docs a bit	a year ago
example	bump and deploy 0.3.7	9 days ago
plugin	bump and deploy 0.3.7	9 days ago
sidecar	bump and deploy 0.3.7	9 days ago
support	bump and deploy 0.3.7	9 days ago
.gitignore	make cider optional	28 days ago
CHANGES.md	update README and change log	9 days ago
LICENSE	initial commit	a year ago

ClojureScript

Figwheel

Super feedback for ClojureScript development

Autoreloading of the browser

Push on save!

ClojureScript

Playground Example

Playground

Checkout your developer console.

cd clojurescript-async-demo

lein fig wheel

browser http://localhost:3449/

ClojureScript

Up to you – experiment!

Playground

Checkout your developer console.

make another dot

color dot by odd even

make the dots move