

Clojure core.async Reference

<https://clojure.github.io/core.async/>

processes/threads

All return a channel immediately to the calling thread. When execution completes, the value of the last expression in `<body>` (or return value of `<f>`) will be put on the returned channel.

<code>(go <body>)</code>	asynchronously execute <code><body></code>
<code>(go-loop [<bindings>] <body>)</code>	equivalent to <code>(go (loop [<bindings>] <body>))</code>
<code>(thread <body>)</code>	run <code><body></code> in new thread
<code>(thread-call <f>)</code>	call function <code><f></code> in a new thread

creating channels

<code>(chan)</code>	create an unbuffered channel
<code>(chan <n>)</code>	create a channel with buffer size <code><n></code>
<code>(chan <buf>)</code>	create a channel with a given buffer (see below)

taking a value from a channel

Taking operations will return/pass `nil` if the channel is closed.

<code>(<! <c>)</code>	take from <code><c></code> in a go block; parks until value available
<code>(<!! <c>)</code>	take from <code><c></code> out of go block; blocks until value available
<code>(take! <c> <f>)</code>	take from <code><c></code> asynchronously; pass value to <code><f></code>

putting a value on a channel

Putting operations will return/pass `true` if port is not closed. You cannot put `nil` on a channel.

<code>(>! <c> <v>)</code>	put <code><v></code> on <code><c></code> in a go block; parks until buffer space free
<code>(>!! <c> <v>)</code>	put <code><v></code> on <code><c></code> out of go block; blocks until buffer space free
<code>(put! <c> <v> <f>)</code>	put <code><v></code> on <code><c></code> asynchronously; pass true to <code><f></code> if <code><c></code> is not closed

creating buffers

<code>(buffer <n>)</code>	create a fixed buffer of size <code><n></code>
<code>(dropping-buffer <n>)</code>	create a buffer of size <code><n></code> that drops new value when full
<code>(sliding-buffer <n>)</code>	create a buffer of size <code><n></code> that drops oldest value when full

alternative operators

<code>(alt! [<port1> ... <portn>])</code>	operate on 1 available port in go block; parks until available
<code>(alt! <ports> <option value>)</code>	like above but with options
when <code><port></code> is <code><c></code> <code>[<c> <v>]</code>	operation is <code>(<! <c>)</code> <code>(>! <c> <v>)</code>
when <code><option value></code> is <code>:default <v></code> <code>:priority true</code>	returns <code><v></code> immediately if nothing available if 2 or more available ports, resolve tie with order in <code><ports></code> vector
<code>(alt!! [<port1> ... <portn>])</code>	operate on 1 available <code><port></code> out of go block; blocks until available
when <code><port></code> is <code><c></code> <code>[<c> <v>]</code>	operation is <code>(<!! <c>)</code> <code>(>!! <c> <v>)</code>