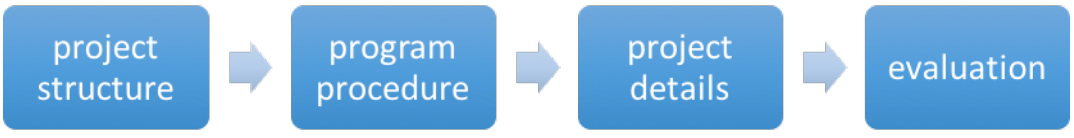
Table of Contents

Goal

Find an approach to do this kind of things.

Design

Thought



Plan

Tasks

module	task	description	time	status

Project structure	Vision	Project vision	0.5h	Done!
	Environment	Running environment		Done!
	Tool stack	Dev tool, test tool, deploy tool.		Done!
	Dependencies	Find all the dependents of the project, and know those dependencies' functions.	0.3 h	Done!
	Modules	Distinct modules and find the relationship of all modules.	1h	Done!
	Project interface	Interfaces of project, like API, UI.	1h	Done!
Program procedure	Core data structure	The core data structures used in the project.	1h	Done!
	Core algorithm	The core data algorithms used in the project.	2h	Done!
Project details	Knowledge	The knowledge you do not know	1h	Done!
	Style	Style of project: comment, naming, lint, closure. Slips, law.	1h	Done!
	Tricks	Programing tricks.	1h	Done!
	Other			
evaluation	Advantage	The advantages of this project	0.5h	Done!
	Disadvantage	The disadvantage of this project		
	Scenes	The scenes it suits and not suits		

Project structure

Vision

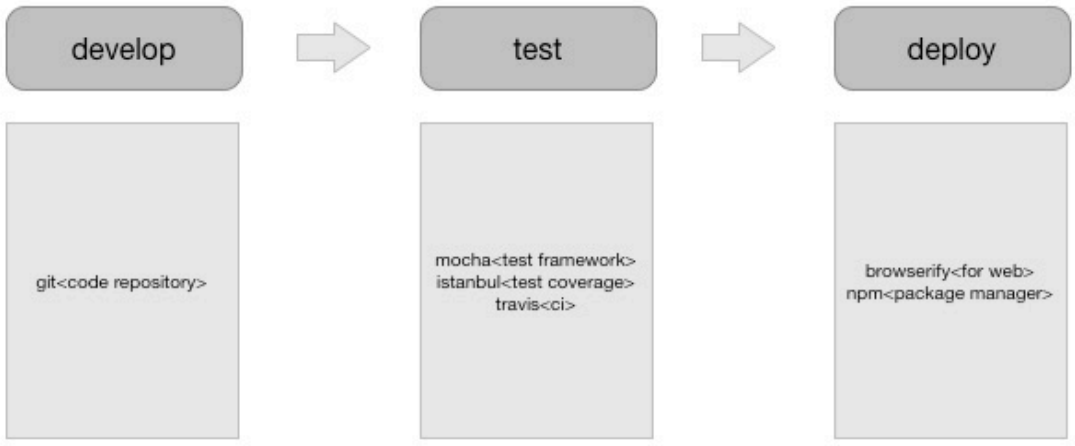
Generator based control flow goodness for nodejs and the browser, using promises, letting you write non-blocking code in a nice-ish way.

https://github.com/tj/co

Environment

iojs >= 1.0.0 node >= 0.12.0 ES5

Tool stack



Dependencies

None

Modules

Too simple to just have one module.

Project Interface

https://github.com/tj/co

interface	description	invoke	parameters		result
co	Returns a promise that resolves a generator, generator function, or any function that returns a generator.	co(fn*)	0	generator | generator function | any function returns a generator	Promise
co.wrap	Convert a generator into a regular function that returns a promise	co.wrap(fn*)	0	generator	Promise generator

Program procedure

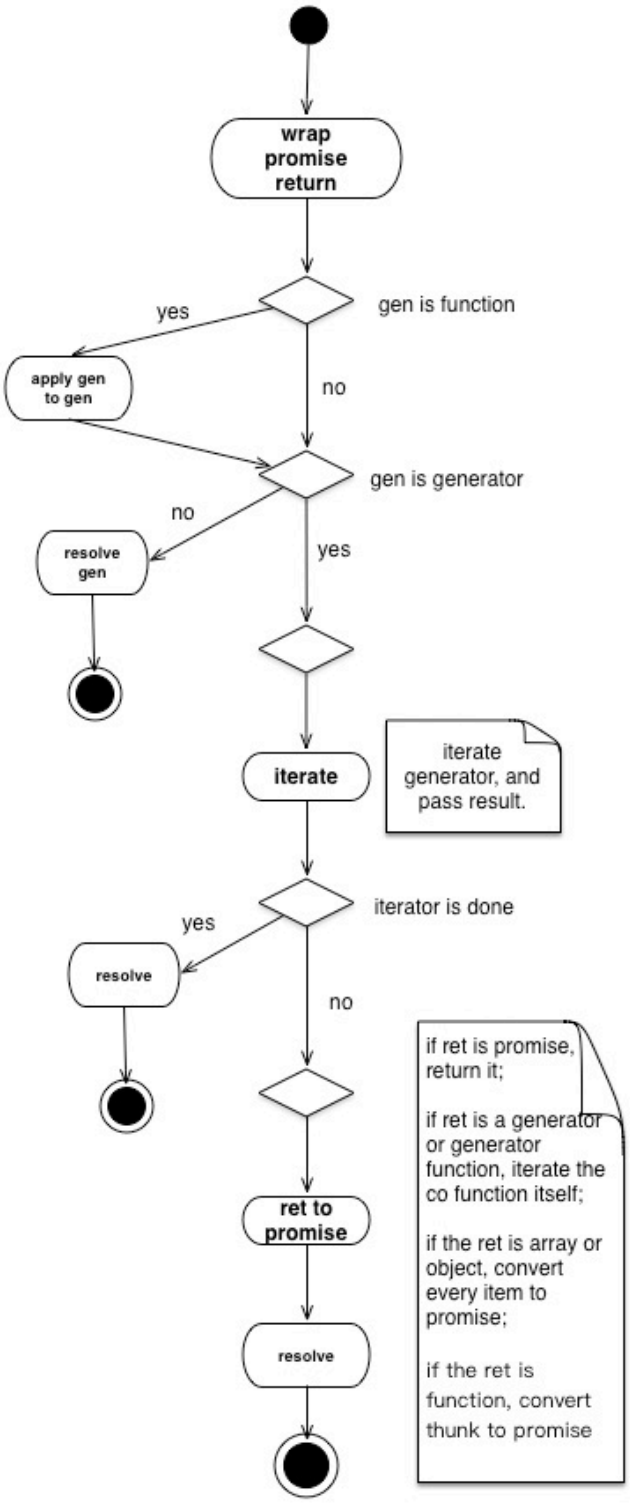
Core data structure

yieldable object

- promises
- thunks. https://github.com/thunks/thunks
- array(parallel)
- objects(parallel)
- generators
- generator functions

core algorithm

co function, execute the generator function or a generator


wrap promise return
gen is function
yes
apply gen to gen
no
gen is generator
no
resolve gen
yes
iterate
iterate generator, and pass result.
iterator is done
yes
resolve
no
if ret is promise, return it;
if ret is a generator or generator function, iterate the co function itself;
if the ret is array or object, convert every item to promise;
if the ret is function, convert thunk to promise
ret to promise
resolve

Project details

Knowledge

- thunk
- generator function
  constructor.name
  constructor.displayName
  isGenerator(constructor.prototype)
- plain object
  Object == val.constructor

Style

- comment

```
/**
 * Wrap the given generator `fn` into a
 * function that returns a promise.
 * This is a separate function so that
 * every `co()` call doesn't create a new,
 * unnecessary closure.
 *
 * @param {GeneratorFunction} fn
 * @return {Function}
 * @api public
 */

co.wrap = function (fn) {
  createPromise.__generatorFunction__ = fn;
  return createPromise;
  function createPromise() {
    return co.call(this, fn.apply(this, arguments));
  }
};
```

  type: Function, Object, …, Mixed, …

Tricks

- try-catch hack

evaluation

advantage

- very simple interfaces, easy to use

disadvantage or risk

- ES7 async await

scenes

- generator or generator function