FREE eBook

LEARNING ember.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#ember.js

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ember-js

It is an unofficial and free ember.js ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ember.js.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with ember.js

Remarks

This section provides an overview of what ember.js is, and why a developer might want to use it.

It should also mention any large subjects within ember.js, and link out to the related topics. Since the Documentation for ember.js is new, you may need to create initial versions of those related topics.

Versions

Up-To-Date Release

Version	Release Date
2.14.0 beta	2017-04-29
2.13.0	2017-04-29

Examples

Installation or Setup

Getting started with Ember is easy. Ember projects are created and managed through our command line build tool Ember CLI. This tool provides:

- Modern application asset management (including concatenation, minification, and versioning).
- Generators to help create components, routes, and more.
- A conventional project layout, making existing Ember applications easy to approach.
- Support for ES2015/ES6 JavaScript via the Babel project. This includes support for JavaScript modules, which are used throughout this guide.
- A complete QUnit test harness.
- The ability to consume a growing ecosystem of Ember Addons.

Dependencies

Node.js and npm

Ember CLI is built with JavaScript, and expects the Node.js runtime. It also requires dependencies fetched via npm. npm is packaged with Node.js, so if your computer has Node.js installed you are ready to go.

Ember requires Node.js 0.12 or higher and npm 2.7 or higher. If you're not sure whether you have Node.js or the right version, run this on your command line:

```
node --version
npm --version
```

If you get a *"command not found"* error or an outdated version for Node:

- Windows or Mac users can download and run this Node.js installer.
- Mac users often prefer to install Node using Homebrew. After installing Homebrew, run `brew install node` to install Node.js.
- Linux users can use this guide for Node.js installation on Linux.

If you get an outdated version of npm, run `npm install -g npm`.

Git

Ember requires Git to manage many of its dependencies. Git comes with Mac OS X and most Linux distributions. Windows users can download and run this Git installer.

Watchman (optional)

On Mac and Linux, you can improve file watching performance by installing Watchman.

PhantomJS (optional)

You can run your tests from the command line with PhantomJS, without the need for a browser to be open. Consult the PhantomJS download instructions.

Installation

Install Ember using npm:

```
npm install -g ember-cli
```

To verify that your installation was successful, run:

```
ember -v
```

If a version number is shown, you're ready to go.

Creating App

Ember CLI allows you to use one of two options to generate a new app:

1. Create a folder and run `ember init` (generates application structure and sets up git and makes your first commit)

2. Run `ember new <app name>` (creates a folder with the specified name, steps into it and runs `ember init`)

Once the generation process is complete, boot a live-reload server within the app folder by running:

```
ember server
```

or `ember s` for short. **Ta-da, now you have a running Ember app!* Official Docs

Creating your first template

Let's create a new template using the `ember generate` command.

```
ember generate template application
```

The `application` template is always on screen when a user is visiting your application. In your editor of choice, open `app/templates/application.hbs` and add the following code:

```
<h2>My first Ember application</h2>

{{outlet}}
```

Now you should see the newly added text on the welcome page of your application. Also notice that Ember automatically detects the new file and reloads the page for you. Neat, right?

Deploy App

To deploy an Ember application simply transfer the output from ember build to a web server. This can be done with standard Unix file transfer tools such as `rsync` or `scp`. There are also services that will let you deploy easily.

```
ember build
scp -r dist/* myserver.com:/var/www/public/
```

normally we would use `ember build --environment=production` which does more to make your code ready for production (gzip and minify code).

How to work with JavaScript plugins

There are four ways to work with JavaScript plugins,

1. Ember add-on
2. Import JavaScript plugins globally
3. Consume named AMD plugins
4. Via `ember-browserify`

Assign localhost ports (esp. permissions/availability issues, running multiple

ember sites simultaneously)

Occasionally it's useful to assign one or more ports manually vs using the defaults. Doing so can solve port availability/permissions issues or accommodate running more than one ember instance at a time.

To have ember-cli attempt to identify and assign an available port, use:

```
ember serve --port 0
```

Per ember help: "Pass 0 to automatically pick an available port". (In a terminal, type ember help).

To run more than one ember site at the same time, each needs its own server and live-reload ports. A simple approach: in separate Terminal windows navigate to each instance and use the following to launch them with their own ports:

```
ember serve --port 0 --live-reload-port 0
```

If you get an error about availability or permission in any of these cases, enter the following python script at your Terminal prompt to identify an available port:

```
python -c 'import socket; s=socket.socket(); s.bind(("", 0)); print(s.getsockname()[1]);
s.close()'
```

Use the results to specify ports you now know to be available:

```
ember serve --port <known_port_1> --live-reload-port <known_port_2>
```

Read Getting started with ember.js online: https://riptutorial.com/ember-js/topic/905/getting-started-with-ember-js

Chapter 2: Asynchronous Tasks in Components

Remarks

in `ember-concurrency` the extra setting of `error` is a work around to prevent thrown exceptions from bubbling up to Ember's `onerror` (since it is meant to be handled in the template). There is a feature request to handle this better.

Examples

ember-concurrency task

An alternative community de facto standard is an addon called ember-concurrency that makes a lot of the promise confusion go away.

It can be installed with the command `ember install ember-concurrency`.

Pros

- Intuitive reasoning of complex asynchronous code.
- Offers a complete API for managing tasks.
- Can be canceled.
- Can be used directly in a component without the need of a proxy object.
- Destructures promises inside the task function.
- Can use JavaScript `try` / `catch` / `finally` blocks to manage asynchronous assignment, exceptions, and cleanup.
- Tasks are automagically cancelled on `willDestroy` event, avoiding errors setting values on destroyed objects (e.g. after a timer)

Cons

- Not builtin – requires `ember install ember-concurrency`
- Uses generator functions that can confuse developers used to promise chains.

JavaScript

```
import Ember from 'ember';
import { task, timeout } from 'ember-concurrency';

const { Component, set } = Ember;

export default Component.extend({
```

```
  myTask: task(function * () {
    set(this, 'error', null);
    try {
      yield timeout(2000);
      return 'Foobar';
    } catch (err) {
      set(this, 'error', error);
    }
  }).keepLatest()
});
```

Template

```
{{#if myTask.isIdle}}
  <button onclick={{perform myTask}}>
    Start Task
  </button>
{{else}}
  Loading&hellip;
{{/if}}

{{#if myTask.last.value}}
  Done. {{myTask.last.value}}
{{/if}}

{{#if error}}
  Something went wrong. {{error}}
{{/if}}
```

PromiseProxyMixin

Ember comes with a built in helper that will provide computed properties for the status of an asynchronous task.

Pros

- Built in – no need for an addon.
- Can be managed in the life cycle of a component.
- Provides convenient state properties that can drive the template logic.

Cons

- Must be wrapped in an `Ember.Object` and cannot be applied to an `Ember.Component` directly.
- Creates a disconnect between the original promise chain and the destructing of the `content` value.
- Is not very intuitive and can be difficult to reason with.
- Cannot be cancelled.

JavaScript

```
import Ember from 'ember';

const {
  Component, PromiseProxyMixin, get, set, computed,
  isPresent, run, RSVP: { Promise }
} = Ember;

const MyPromiseProxy = Ember.Object.extend(PromiseProxyMixin);

export default Component({
  myProxy: computed('promise', {
    get() {
      const promise = get(this, 'promise');
      return isPresent(promise) ? MyPromiseProxy.create({promise}) : null;
    }
  }),

  actions: {
    performTask() {
      const fakeTask = new Promise((resolve) => {
        run.later(resolve, 'Foobar', 2000);
      });
      set(this, 'promise', fakeTask);
    }
  }
});
```

Template

```
{{#if myProxy.isPending}}
  Loading&hellip;
{{else}}
  <button onclick={{action "performTask"}}>
    Start Task
  </button>
{{/if}}

{{#if myProxy.isFulfilled}}
  Done. {{myProxy.content}}
{{/if}}

{{#if myProxy.isRejected}}
  Something went wrong. {{myProxy.reason}}
{{/if}}
```

Read Asynchronous Tasks in Components online: https://riptutorial.com/ember-js/topic/1054/asynchronous-tasks-in-components

Chapter 3: Component - communication between child to parent component.

Syntax

- `(yield` -- Allows you to export items from a component
- `(hash` -- Allows you to export a hash or object, since this is required to call child components within the parent's block. The requirement is that there is a `.` for the component to be created
- `(component` -- Creates the child component which can take anything in the parent's context. The component can be curried, since it is only called when the user uses it, so add as many attributes as you need, and the user can add the rest.
- `(action` -- Creates an action based on a function or a string pointing to a function in the `actions` hash of the parent component in this case.

Remarks

For creating components that interact with a parent component, composable components are the best option, although they require Ember 2.3+.

Examples

Composable Components

Inside `parent-component.hbs`

```
{{yield (hash
   child=(
     component 'child-component'
     onaction=(action 'parentAction')
   )
)}}
```

Inside `parent-component.js`

```
export default Ember.Component.extend({
  actions: {
    // We pass this action to the child to call at it's discretion
    parentAction(childData) {
      alert('Data from child-component: ' + childData);
    }
  }
});
```

Inside `child-component.js`

```
export default Ember.Component.extend({
```

```
  // On click we call the action passed down to us from the parent
  click() {
    let data = this.get('data');
    this.get('onaction')(data);
  }
});
```

Inside `usage.hbs`

```
{{#parent-component as |ui|}}
  {{#each model as |item|}}
    {{ui.child data=item}}
  {{/each}}
{{/parent-component}}
```

Read Component - communication between child to parent component. online: https://riptutorial.com/ember-js/topic/3066/component---communication-between-child-to-parent-component-

Chapter 4: Currency formatting template helper

Remarks

More details available in Ember guides, where this example was taken from.

Compatible with Ember 2.2.0+ (2.11.0 was the latest at the time of writing)

Examples

Creating a new helper

Use Ember CLI to generate a new helper in your app:

```
ember generate helper format-currency
```

Then edit `helpers/format-currency.js` to contain the following:

```
import Ember from 'ember';

export function formatCurrency([value, ...rest]) {
  const dollars = Math.floor(value / 100);
  const cents = value % 100;
  const sign = '$';

  if (cents.toString().length === 1) { cents = '0' + cents; }
  return `${sign}${dollars}.${cents}`;
}

export default Ember.Helper.helper(formatCurrency);
```

Now you can use `{{format-currency model.someNumericValue}}` in templates.

A unit test for the new helper is automatically created in `tests/unit/helpers/`

Read Currency formatting template helper online: https://riptutorial.com/ember-js/topic/6647/currency-formatting-template-helper

Chapter 5: Date Format Helper

Examples

Helper for a clean date and hour format.

When you want the current date and time, you can do this with the Javascript function `Date`, but will return the following format which isn't always useful: `Wed Jun 07 2017 13:26:15 GMT+0200 (Romance (zomertijd))`.

Copy the following code into `app/helpers/helpers.js`, and simply call `getCurrentDateAndFormat()` instead of `new Date()`.

```
export function getCurrentDateAndFormat() {
    let today = new Date();
    let dd = today.getDate();
    let MM = today.getMonth()+1; //January is 0!
    let hh = today.getHours();
    let mm = today.getMinutes();
    let yyyy = today.getFullYear();

    if (dd<10) {
        dd= '0'+dd;
    }

    if (MM<10) {
        MM= '0'+MM;
    }

    if (hh<10) {
        hh= '0'+hh;
    }

    if (mm<10) {
        mm= '0'+mm;
    }

    today = dd+'/'+MM+'/'+yyyy+" "+hh+"h"+mm;

    return today;
}
```

The helper extracts all separate time values, adds a 0 if the value is below 10 (for format and readability) and reassembles them in a more fitting format. In this case: day, month, year, hours and minutes.

```
today = dd+'/'+MM+'/'+yyyy+" "+hh+"h"+mm;
```

will display: `07/06/2017 13h26`

```
today = MM+'/'+dd+'/'+yyyy+" "+hh+"h"+mm;
```

will display: `06/07/2017 13h26`

Changing month and date position, depending on your region, is as easy as replacing `MM` with `dd` and vice versa, as evident from above example.

Read Date Format Helper online: https://riptutorial.com/ember-js/topic/10153/date-format-helper

Chapter 6: Debugging

Examples

Logging EmberData

The ember data models have a toJSON method that extracts the relevant data:

```
console.log(model.toJSON());
```

This method uses the JSONSerializer to create the JSON representation.

If you want to log the data in a more app-specific way, you can use serialize:

```
model.serialize();
```

which uses the serialization strategy you can define in the store's adapter to create a JSON representation of the model.

All objects in an Ember app, including Ember Data models, inherit from Ember.CoreObject, which has a toString method that prints this representation:

```
<app-name@ember-type:object-name:id>
```

Explanation:

- `app-name` is the name of your application
- `ember-type` is the ember type of the object you are logging (can be controller, route etc.)
- `object-name` is the name of the object you are logging (name of your model, or controller, or route etc.)
- id is either a guId create with Ember.guidFor or, for example, the model's id.

You can overwrite this value using the method `toStringExtension` in your particular model.

For comparison example, here's how logging an application controller could look:

```
<my-awesome-app@controller:application::ember324>
```

Running debug-only code

Ember has a static global method called runInDebug which can run a function meant for debugging.

```
Ember.runInDebug(() => {
  // this code only runs in dev mode
});
```

In a production build, this method is defined as an empty function (NOP). Uses of this method in Ember itself are stripped from the `ember.prod.js` build.

Read Debugging online: https://riptutorial.com/ember-js/topic/2320/debugging

Chapter 7: How to import JavaScript library/plugin

Introduction

Open the directory of your ember.js project, You will find there a file named ember-cli-build.js. You can install Your libraries or plugins using bower, then point the import to the bower_components folder, but if you have a file You want to add, just drag them to the folder of Your project and write the app.import to that file.

Syntax

- app.import('path to file starting from project folder/file.js');

Examples

Example ember-cli-build.js file

```
var EmberApp = require('ember-cli/lib/broccoli/ember-app');

module.exports = function(defaults) {
  var app = new EmberApp(defaults, {
    // Add options here
    datatables: {
        core: true,
        style: 'bs',
        extensions: [
            { name: 'buttons', extensions: ['colVis','flash','html5','print'] },
            { name: 'responsive', style: 'bs' },
            'select'
        ],
        pdfmake: false,
        vfs_fonts: false,
        jszip: true
    }
  });
  //Imports:
  app.import('bower_components/js-cookie/src/js.cookie.js');
  app.import('bower_components/moment/min/moment.min.js');
  app.import('bower_components/crypto-js/crypto-js.js');
  // Use `app.import` to add additional libraries to the generated
  // output files.
  //
  // If you need to use different assets in different
  // environments, specify an object as the first parameter. That
  // object's keys should be the environment name and the values
  // should be the asset to use in that environment.
  //
  // If the library that you are including contains AMD or ES6
  // modules that you would like to import into your application
```

```
  // please specify an object with the list of modules as keys
  // along with the exports of each module as its value.

  return app.toTree();
};
```

Read How to import JavaScript library/plugin online: https://riptutorial.com/ember-js/topic/9239/how-to-import-javascript-library-plugin

Chapter 8: How to update Ember, Ember Data and Ember CLI

Remarks

- To find the latest stable version of **Ember**, click here.
- To find the latest stable version of **Ember Data**, click here.
- To find the latest stable version of **Ember CLI**, click here.

All these steps were found on Ember cli release note.

Examples

Updating Ember

In this example, `2.13.2` is the latest version. We install it via `bower`, specifying the particular version as `ember#2.13.2` and including the save flag to persist it to bower.json.

As of writing this post the latest version is `2.13.2`. From the command line, in the root of your app's directory, run:

```
bower install ember#2.13.2 --save
```

You may be prompted to choose your version of Ember. If you are, prepend your answer with a `!` to make sure it's saved.

Updating Ember Data

Since Ember Data is an Ember CLI add-on we can install it just like any other add-on by using `ember install`.

As of writing this post the latest version is `2.13.1`. From the command line, in the root of your app's directory, run:

```
ember install ember-data@2.13.1
```

Updating Ember CLI

Ember CLI is a normal npm package. To update it we have to uninstall it and then install the version we want.

As of writing this post the latest version is 2.13.2. From the command line run:

```
npm uninstall -g ember-cli
```

```
npm cache clean
bower cache clean
npm install -g ember-cli@2.13.2
```

To verify the proper version was installed run:

```
ember -v
```

Then update your project. This will clear out the cache and update the Ember CLI version in `package.json`. The last part will run the new project blueprint on your projects directory. Just follow the prompts and review all the changes.

```
rm -rf node_modules bower_components dist tmp
npm install --save-dev ember-cli@2.13.2
npm install
bower install
ember init
```

references

Read How to update Ember, Ember Data and Ember CLI online: https://riptutorial.com/ember-js/topic/1722/how-to-update-ember--ember-data-and-ember-cli

Chapter 9: Initialize Foundation or Bootstrap on ember-cli in a proper way

Introduction

Bootstrap : I think that's not proper way. The best way in my opinion is an ember-bootstrap addon.

ember-bootstrap uses the Bootstrap CSS classes while replacing the behaviors from the components Bootstrap implements in bootstrap.js, such as toggle, navbar, modal, etc., with equivalent, CSS class-compatible native Ember components.

Foundation: There is an addon called Ember CLI Foundation 6 SASS, it's also installed using command line.

Parameters

Parameter	Usage
Ember install	Download a new extension package using Ember
npm install	Download a new extension package using node.js
SASS	CSS language necessary in Foundation
Ember-cli-build.js	File with Ember imports, configuration, etc.
{{#bs-modal-simple}}	Creation of a new bootstrap modal
fade=fade	Set the modal animations
{{#bs-button}}	Button with a Bootstrap cool look
{{#bs-form onSubmit(action = "Submit")}}	New form with an action after submitting

Remarks

Both addons are not mine, I thought it will be nice to present them to You, there are github pages of addons:

Ember Bootstrap: https://github.com/kaliber5/ember-bootstrap

Ember foundation 6 https://github.com/acoustep/ember-cli-foundation-6-sass

You can find documentation there.

Examples

Install ember-bootstrap with default version

```
ember install ember-bootstrap
```

Instal ember-bootstrap with version 4 and SASS - experimental

```
ember generate ember-bootstrap --bootstrap-version=4 --preprocessor=sass
```

Install SASS and Foundation

```
npm install --save-dev ember-cli-sass
ember install ember-cli-foundation-6-sass
```

Install Foundation dependencies

```
ember g ember-cli-foundation-6-sass
```

Ember build file with Foundation addons

```
// ember-cli-build.js

/* global require, module */
var EmberApp = require('ember-cli/lib/broccoli/ember-app');

module.exports = function(defaults) {
  var app = new EmberApp(defaults, {
    // Add options here
    'ember-cli-foundation-6-sass': {
            'foundationJs': [
                'core',
                'util.box',
                'util.keyboard',
                'util.mediaQuery',
                'util.motion',
                'util.nest',
                'util.timerAndImageLoader',
                'util.touch',
                'util.triggers',
                'abide',
                'accordion',
                'accordionMenu',
                'drilldown',
                'dropdown',
                'dropdownMenu',
                'equalizer',
                'interchange',
                'magellan',
                'offcanvas',
                'orbit',
```

```
                'responsiveMenu',
                'responsiveToggle',
                'reveal',
                'slider',
                'sticky',
                'tabs',
                'toggler',
                'tooltip'
            ]
        },
    }
  });

  return app.toTree();
};
```

Ember Bootstrap sample form

```
{{#bs-form model=this onSubmit=(action "submit") as |form|}}
  {{#form.element label="Email" placeholder="Email" property="email" as |el|}}
    <div class="input-group">
      <input value={{el.value}} class="form-control" placeholder="Email" oninput={{action (mut
el.value) value="target.value"}} onchange={{action (mut el.value) value="target.value"}}
id={{el.id}} type="text">
      <span class="input-group-addon">@example.com</span>
    </div>
  {{/form.element}}
  {{bs-button defaultText="Submit" type="primary" buttonType="submit"}}
{{/bs-form}}
```

Read Initialize Foundation or Bootstrap on ember-cli in a proper way online: https://riptutorial.com/ember-js/topic/10176/initialize-foundation-or-bootstrap-on-ember-cli-in-a-proper-way

Chapter 10: Testing

Introduction

Creating and maintaining a comprehensive test suite should be a priority for each developer. Testing in Ember.js involves dealing with asynchrony, Ember Run Loop and mocking your API. It is common for Ember.js developers to struggle when writing tests. However, there are some tips which could save your time and energy.

Examples

Waiting for promises in tests in elegant way

You can make `function` passed to `test()` method `async` - then you can use `await` keyword. Your test will wait until Promises resolve and testing asynchronous code becomes easier and more readable. In the following example call that returns a Promise is `changeset.validate()`. Please notice also wrapping `set` call in `Ember.run`. Setting quantity has asynchronous effects (observers, computed properties) and thus we need to wrap it in `Ember.run`.

```
test('quantity validation: greater than 0', async function (assert) {
    assert.expect(3);

    const model = this.subject({
        quantity: 1
    });

    const changeset = createChangeset(model);

    await changeset.validate();

    assert.ok(!changeset.get('error.quantity'));

    Ember.run(() => {
        changeset.set('quantity', -1);
    });

    await changeset.validate();

    assert.equal(changeset.get('error.quantity.validation.length'), 1);
    assert.ok(!changeset.get('isValid'));
});
```

Read Testing online: https://riptutorial.com/ember-js/topic/10722/testing

Credits

S. No	Chapters	Contributors
1	Getting started with ember.js	Cameron, Community, Eric D. Johnson, Kenneth Larsen, locks, Nicos Karalis, ownsourcing dev training, Patsy Issa, Sid, Xinyang Li
2	Asynchronous Tasks in Components	Sukima, user2708383, wdspkr
3	Component - communication between child to parent component.	Cameron, knownasilya, locks, Xinyang Li
4	Currency formatting template helper	Alan Mabry, Andrius
5	Date Format Helper	Daniel Kmak, Silvio Langereis
6	Debugging	nem035
7	How to import JavaScript library/plugin	Rafalsonn
8	How to update Ember, Ember Data and Ember CLI	Cameron, ddoria921, heat, locks
9	Initialize Foundation or Bootstrap on ember-cli in a proper way	Rafalsonn
10	Testing	Daniel Kmak