EBook Gratis

APRENDIZAJE

ember.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#ember.js

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con ember.js

Observaciones

Esta sección proporciona una descripción general de qué es ember.js y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de ember.js, y vincular a los temas relacionados. Dado que la Documentación para ember.js es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Versiones

Versión actualizada

Versión	Fecha de lanzamiento
2.14.0 beta	2017-04-29
2.13.0	2017-04-29

Examples

Instalación o configuración

Comenzar con Ember es fácil. Los proyectos de Ember se crean y administran a través de nuestra herramienta de compilación de línea de comandos Ember CLI. Esta herramienta proporciona:

- Administración moderna de activos de aplicaciones (incluyendo concatenación, minificación y control de versiones).
- Generadores para ayudar a crear componentes, rutas y más.
- Un diseño de proyecto convencional, que facilita el acceso a las aplicaciones existentes de Ember.
- Soporte para JavaScript ES2015 / ES6 a través del proyecto Babel . Esto incluye soporte para módulos de JavaScript , que se utilizan en esta guía.
- Un completo arnés de prueba QUnit .
- La capacidad de consumir un creciente ecosistema de Ember Addons .

Dependencias

Node.js y npm

Ember CLI está construido con JavaScript y espera el tiempo de ejecución Node.js. También requiere dependencias buscadas a través de npm . npm está empaquetado con Node.js, por lo que si su computadora tiene Node.js instalado, está listo para comenzar.

Ember requiere Node.js 0.12 o superior y npm 2.7 o superior. Si no está seguro de si tiene Node.js o la versión correcta, ejecute esto en su línea de comando:

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

Si recibe un error de *"comando no encontrado"* o una versión desactualizada para el nodo:

- Los usuarios de Windows o Mac pueden descargar y ejecutar este instalador Node.js.
- Los usuarios de Mac a menudo prefieren instalar Node usando Homebrew . Después de instalar Homebrew, ejecute `brew install node` para instalar Node.js.
- Los usuarios de Linux pueden usar esta guía para la instalación de Node.js en Linux .

Si obtiene una versión desactualizada de npm, ejecute `npm install -g npm` .

Git

Ember requiere que Git gestione muchas de sus dependencias. Git viene con Mac OS X y la mayoría de las distribuciones de Linux. Los usuarios de Windows pueden descargar y ejecutar este instalador Git .

Vigilante (opcional)

En Mac y Linux, puede mejorar el rendimiento de visualización de archivos instalando Watchman .

PhantomJS (opcional)

Puede ejecutar sus pruebas desde la línea de comandos con PhantomJS, sin necesidad de abrir un navegador. Consulte las instrucciones de descarga de PhantomJS .

Instalación

Instala Ember usando npm:

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

Para verificar que su instalación fue exitosa, ejecute:

```
ember -v
```

Si se muestra un número de versión, estás listo para ir.

Creación de la aplicación

Ember CLI le permite usar una de las dos opciones para generar una nueva aplicación:

1. Cree una carpeta y ejecute `ember init` (genera la estructura de la aplicación y configura git y realiza su primer compromiso)
2. Ejecutar `ember new <app name>` (crea una carpeta con el nombre especificado, entra en ella y ejecuta `ember init`)

Una vez que se complete el proceso de generación, inicie un servidor de recarga en vivo dentro de la carpeta de la aplicación ejecutando:

```
ember server
```

o `ember s` para abreviar. * *Ta-da, ahora tienes una aplicación Ember en ejecución!* Documentos oficiales

Creando tu primera plantilla

Vamos a crear una nueva plantilla usando el comando `ember generate` .

```
ember generate template application
```

La plantilla de la `application` siempre está en pantalla cuando un usuario visita su aplicación. En el editor de su elección, abra `app/templates/application.hbs` y agregue el siguiente código:

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

{{outlet}}
```

Ahora debería ver el texto recién agregado en la página de bienvenida de su aplicación. También tenga en cuenta que Ember detecta automáticamente el nuevo archivo y vuelve a cargar la página por usted. Limpio, ¿verdad?

Implementar aplicación

Para implementar una aplicación Ember, simplemente transfiera la salida de la compilación de brasas a un servidor web. Esto se puede hacer con herramientas estándar de transferencia de archivos Unix como `rsync` o `scp` . También hay servicios que te permitirán desplegar fácilmente.

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

normalmente `ember build --environment=production` que hace más para preparar su código para la producción (gzip y minify code).

Cómo trabajar con complementos de JavaScript

Hay cuatro formas de trabajar con los complementos de JavaScript,

1. Complemento Ember
2. Importar complementos de JavaScript a nivel mundial
3. Consumir complementos de AMD con nombre
4. A través de `ember-browserify`

Asigne puertos de localhost (especialmente permisos / problemas de disponibilidad, ejecutando múltiples sitios de brasas simultáneamente)

Ocasionalmente, es útil asignar uno o más puertos manualmente en lugar de usar los valores predeterminados. Si lo hace, puede resolver problemas de disponibilidad / permisos de puerto o permitir la ejecución de más de una instancia de ember a la vez.

Para que ember-cli intente identificar y asignar un puerto disponible, use:

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

Ayuda por persona: "Pase 0 para elegir automáticamente un puerto disponible". (En una terminal, escriba ember help).

Para ejecutar más de un sitio de brasas al mismo tiempo, cada uno necesita su propio servidor y puertos de recarga en vivo. Un enfoque simple: en ventanas de Terminal separadas navegue a cada instancia y use lo siguiente para iniciarlas con sus propios puertos:

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

Si recibe un error sobre la disponibilidad o el permiso en cualquiera de estos casos, ingrese el siguiente script de Python en el indicador de su Terminal para identificar un puerto disponible:

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

Use los resultados para especificar los puertos que ahora sabe que estarán disponibles:

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

Lea Empezando con ember.js en línea: https://riptutorial.com/es/ember-js/topic/905/empezando-con-ember-js

Capítulo 2: Asistente de plantillas de formato de moneda

Observaciones

Más detalles disponibles en las guías Ember , de donde se tomó este ejemplo.

Compatible con Ember 2.2.0+ (2.11.0 era lo último en el momento de escribir este artículo)

Examples

Creando un nuevo ayudante

Utilice Ember CLI para generar un nuevo ayudante en su aplicación:

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

Luego edite `helpers/format-currency.js` para que contenga lo siguiente:

```
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);
```

Ahora puede usar `{{format-currency model.someNumericValue}}` en las plantillas.

Una prueba de unidad para el nuevo ayudante se crea automáticamente en `tests/unit/helpers/`

Lea Asistente de plantillas de formato de moneda en línea: https://riptutorial.com/es/ember-js/topic/6647/asistente-de-plantillas-de-formato-de-moneda

Capítulo 3: Ayudante de formato de fecha

Examples

Ayudante para un formato de fecha y hora limpio.

Cuando desee la fecha y hora actuales, puede hacer esto con la función de `Date` Javascript, pero devolverá el siguiente formato que no siempre es útil: `Wed Jun 07 2017 13:26:15 GMT+0200 (Romance (zomertijd))` .

Copie el siguiente código en `app/helpers/helpers.js` , y simplemente llame a `getCurrentDateAndFormat()` lugar de `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;
}
```

El ayudante extrae todos los valores de tiempo separados, agrega un 0 si el valor es inferior a 10 (para el formato y la legibilidad) y los vuelve a ensamblar en un formato más apropiado. En este caso: día, mes, año, horas y minutos.

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

se mostrarán: `07/06/2017 13h26`

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

se mostrarán: `06/07/2017 13h26`

Cambiar la posición del mes y la fecha, dependiendo de su región, es tan fácil como reemplazar `MM` con `dd` y viceversa, como se desprende del ejemplo anterior.

Lea Ayudante de formato de fecha en línea: https://riptutorial.com/es/ember-js/topic/10153/ayudante-de-formato-de-fecha

Capítulo 4: Cómo actualizar Ember, Ember Data y Ember CLI

Observaciones

- Para encontrar la última versión estable de **Ember** , haga clic aquí .
- Para encontrar la última versión estable de **Ember Data** , haga clic aquí .
- Para encontrar la última versión estable de **Ember CLI** , haga clic aquí .

Todos estos pasos fueron encontrados en la nota de lanzamiento de Ember cli .

Examples

Actualizando Ember

En este ejemplo, `2.13.2` es la última versión. Lo instalamos a través de `bower` , especificando la versión particular como `ember#2.13.2` e incluyendo el indicador de guardado para conservarlo en bower.json.

Al momento de escribir este post, la última versión es `2.13.2` . Desde la línea de comandos, en la raíz del directorio de su aplicación, ejecute:

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

Es posible que se le solicite que elija su versión de Ember. Si es así, ¡anteponga su respuesta con un `!` para asegurarse de que está guardado.

Actualizando Datos Ember

Dado que Ember Data es un complemento de la CLI de Ember, podemos instalarlo como cualquier otro complemento mediante la `ember install` .

Al momento de escribir este post, la última versión es `2.13.1` . Desde la línea de comandos, en la raíz del directorio de su aplicación, ejecute:

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

Actualizando Ember CLI

Ember CLI es un paquete normal de npm. Para actualizarlo tenemos que desinstalarlo y luego instalar la versión que queramos.

Al momento de escribir este post, la última versión es 2.13.2. Desde la línea de comandos ejecute:

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

Para verificar que se instaló la versión correcta ejecute:

```
ember -v
```

Luego actualiza tu proyecto. Esto borrará la memoria caché y actualizará la versión de la CLI de Ember en `package.json` . La última parte ejecutará el nuevo proyecto en su directorio de proyectos. Simplemente siga las indicaciones y revise todos los cambios.

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

referencias

Lea Cómo actualizar Ember, Ember Data y Ember CLI en línea: https://riptutorial.com/es/ember-js/topic/1722/como-actualizar-ember--ember-data-y-ember-cli

Capítulo 5: Cómo importar la biblioteca / plugin de JavaScript

Introducción

Abra el directorio de su proyecto ember.js. Encontrará un archivo llamado ember-cli-build.js. Puede instalar Sus bibliotecas o complementos usando bower, luego dirija la importación a la carpeta bower_components, pero si tiene un archivo que desea agregar, simplemente arrástrelo a la carpeta de Su proyecto y escriba la aplicación.import en ese archivo.

Sintaxis

- app.import ('ruta al archivo desde la carpeta del proyecto / archivo.js');

Examples

Ejemplo de archivo ember-cli-build.js

```
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();
};
```

Lea Cómo importar la biblioteca / plugin de JavaScript en línea: https://riptutorial.com/es/ember-js/topic/9239/como-importar-la-biblioteca---plugin-de-javascript

Capítulo 6: Componente - comunicación entre el componente hijo a padre.

Sintaxis

- `(yield` : le permite exportar elementos de un componente
- `(hash` : le permite exportar un hash o un objeto, ya que es necesario para llamar a componentes secundarios dentro del bloque principal. El requisito es que exista un `.` para que se cree el componente
- `(component` : crea el componente secundario que puede tomar cualquier cosa en el contexto principal. El componente puede ser procesado, ya que solo se llama cuando el usuario lo usa, así que agregue tantos atributos como sea necesario, y el usuario puede agregar el resto .
- `(action` : crea una acción basada en una función o una cadena que apunta a una función en el hash de `actions` del componente principal en este caso.

Observaciones

Para crear componentes que interactúan con un componente principal, los componentes compositivos son la mejor opción, aunque requieren Ember 2.3+.

Examples

Componentes compostables

Dentro de `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);
  }
});
```

Dentro de `usage.hbs`

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

Lea Componente - comunicación entre el componente hijo a padre. en línea: https://riptutorial.com/es/ember-js/topic/3066/componente---comunicacion-entre-el-componente-hijo-a-padre-

Capítulo 7: Depuración

Examples

Registro de EmberData

Los modelos de datos de brasas tienen un método toJSON que extrae los datos relevantes:

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

Este método utiliza el JSONSerializer para crear la representación JSON.

Si desea registrar los datos de una forma más específica de la aplicación, puede usar serializar :

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

que utiliza la estrategia de serialización que puede definir en el adaptador de la tienda para crear una representación JSON del modelo.

Todos los objetos en una aplicación Ember, incluidos los modelos de Datos Ember, heredan de Ember.CoreObject , que tiene un método toString que imprime esta representación:

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

Explicación:

- `app-name` es el nombre de tu aplicación
- `ember-type` es el tipo de brasa del objeto que está registrando (puede ser controlador, ruta, etc.)
- `object-name` es el nombre del objeto que está registrando (nombre de su modelo, controlador, ruta, etc.)
- id es una creación guId con Ember.guidFor o, por ejemplo, la id del modelo.

Puede sobrescribir este valor utilizando el método `toStringExtension` en su modelo particular.

Para un ejemplo de comparación, aquí es cómo podría verse el registro de un controlador de aplicación:

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

Ejecutando código de sólo depuración

Ember tiene un método global estático llamado runInDebug que puede ejecutar una función destinada a la depuración.

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

En una compilación de producción, este método se define como una función vacía (NOP). Los usos de este método en Ember se `ember.prod.js` compilación `ember.prod.js` .

Lea Depuración en línea: https://riptutorial.com/es/ember-js/topic/2320/depuracion

Capítulo 8: Inicialice Foundation o Bootstrap en ember-cli de manera adecuada

Introducción

Bootstrap: Creo que no es la forma correcta. La mejor manera, en mi opinión, es un complemento de ember-bootstrap.

ember-bootstrap usa las clases de Bootstrap CSS mientras reemplaza los comportamientos de los componentes Bootstrap implementa en bootstrap.js, como toggle, navbar, modal, etc., con componentes Ember nativos equivalentes, compatibles con clase CSS.

Fundación: hay un complemento llamado Ember CLI Foundation 6 SASS, también se instala mediante la línea de comandos.

Parámetros

Parámetro	Uso
Instalación de Ember	Descarga un nuevo paquete de extensión usando Ember
npm instalar	Descarga un nuevo paquete de extensión usando node.js
HABLAR CON DESCARO A	Lenguaje CSS necesario en fundacion
Ember-cli-build.js	Archivo con las importaciones de Ember, configuración, etc.
{{# bs-modal-simple}}	Creación de un nuevo bootstrap modal.
fade = fade	Establecer las animaciones modales.
{{# bs-button}}	Botón con un look cool de Bootstrap
{{# bs-form onSubmit (action = "Enviar")}}	Nuevo formulario con una acción después de enviar

Observaciones

Ambos complementos no son míos, pensé que sería bueno presentártelos, hay páginas github de complementos:

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

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

Puedes encontrar documentación allí.

Examples

Instalar ember-bootstrap con la versión predeterminada

```
ember install ember-bootstrap
```

Instalar ember-bootstrap con versión 4 y SASS - experimental

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

Instalar SASS y Fundación

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

Instalar dependencias de la Fundación.

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

Ember construir archivo con complementos de la Fundación

```
// 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();
};
```

Forma de muestra de Ember Bootstrap

```
{{#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}}
```

Lea Inicialice Foundation o Bootstrap en ember-cli de manera adecuada en línea: https://riptutorial.com/es/ember-js/topic/10176/inicialice-foundation-o-bootstrap-en-ember-cli-de-manera-adecuada

Capítulo 9: Pruebas

Introducción

Crear y mantener un conjunto de pruebas completo debe ser una prioridad para cada desarrollador. Las pruebas en Ember.js implican tratar con asincronía, Ember Run Loop y burlarse de su API. Es común que los desarrolladores de Ember.js tengan problemas al escribir pruebas. Sin embargo, hay algunos consejos que podrían ahorrarle tiempo y energía.

Examples

Esperando promesas en pruebas de manera elegante.

Puede hacer que la `function` pase al método `test()` `async` - luego puede usar la palabra clave `await` . Su prueba esperará hasta que las promesas se resuelvan y la prueba del código asíncrono sea más fácil y legible. En el siguiente ejemplo, la llamada que devuelve una Promesa es `changeset.validate()` . Tenga en cuenta que también se `set` llamada al `set` en `Ember.run` . La configuración de la cantidad tiene efectos asíncronos (observadores, propiedades calculadas) y, por lo tanto, necesitamos envolverla en `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'));
});
```

Lea Pruebas en línea: https://riptutorial.com/es/ember-js/topic/10722/pruebas

Capítulo 10: Tareas asíncronas en componentes

Observaciones

En `ember-concurrency` la configuración adicional de `error` es una `onerror` para evitar que las excepciones `onerror` se `onerror` en un `onerror` de Ember (ya que se debe manejar en la plantilla). Hay una solicitud de función para manejar esto mejor.

Examples

tarea de concurrencia

Un estándar alternativo de facto de la comunidad es un complemento llamado concurrencia de brasas que elimina gran parte de la confusión de la promesa.

Se puede instalar con el comando `ember install ember-concurrency` .

Pros

- Razonamiento intuitivo de código asíncrono complejo.
- Ofrece una API completa para la gestión de tareas.
- Puede ser cancelado
- Se puede usar directamente en un componente sin la necesidad de un objeto proxy.
- Destructuras promete dentro de la función de tarea.
- Puede usar los bloques JavaScript `try` / `catch` / `finally` para administrar la asignación asíncrona, las excepciones y la limpieza.
- Las tareas se cancelan `willDestroy` evento `willDestroy` , evitando errores al establecer valores en objetos destruidos (por ejemplo, después de un temporizador)

Contras

- No incorporado - requiere `ember install ember-concurrency`
- Utiliza funciones de generador que pueden confundir a los desarrolladores que solían prometer cadenas.

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()
});
```

Modelo

```
{{#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 viene con un ayudante incorporado que proporcionará propiedades computadas para el estado de una tarea asíncrona.

Pros

- Construido en - sin necesidad de un complemento.
- Se puede gestionar en el ciclo de vida de un componente.
- Proporciona propiedades de estado convenientes que pueden conducir la lógica de la plantilla.

Contras

- Debe estar envuelto en un objeto `Ember.Object` y no puede aplicarse directamente a un `Ember.Component` .
- Crea una desconexión entre la cadena de promesa original y la destrucción del valor del `content` .
- No es muy intuitivo y puede ser difícil de razonar.
- No se puede cancelar.

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);
    }
  }
});
```

Modelo

```
{{#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}}
```

Lea Tareas asíncronas en componentes en línea: https://riptutorial.com/es/ember-js/topic/1054/tareas-asincronas-en-componentes

Creditos

S. No	Capítulos	Contributors
1	Empezando con ember.js	Cameron, Community, Eric D. Johnson, Kenneth Larsen, locks, Nicos Karalis, ownsourcing dev training, Patsy Issa, Sid, Xinyang Li
2	Asistente de plantillas de formato de moneda	Alan Mabry, Andrius
3	Ayudante de formato de fecha	Daniel Kmak, Silvio Langereis
4	Cómo actualizar Ember, Ember Data y Ember CLI	Cameron, ddoria921, heat, locks
5	Cómo importar la biblioteca / plugin de JavaScript	Rafalsonn
6	Componente - comunicación entre el componente hijo a padre.	Cameron, knownasilya, locks, Xinyang Li
7	Depuración	nem035
8	Inicialice Foundation o Bootstrap en ember-cli de manera adecuada	Rafalsonn
9	Pruebas	Daniel Kmak
10	Tareas asíncronas en componentes	Sukima, user2708383, wdspkr