Ember.js Components

•••

Sebastián Fernández
Jorge Schellman

Tabla de Contenidos

1. Componentes en Ember 2.x

Definiendo un Componente (¡ siempre con “-” !)

```
$ ember g component blog-post
installing component
  create app/components/blog-post.js
  create app/templates/components/blog-post.hbs
installing component-test
  create tests/integration/components/blog-post-test.js
```

```
{{! app/templates/components/blog-post.hbs }}
<article class="blog-post">
  <h1>{{title}}</h1>
  <p>{{body}}</p>
</article>
```

```
// app/components/blog-post.js
import Ember from 'ember';

export default Ember.Component.extend({
});
```

Pasando propiedades en el template:

```
{{! app/templates/index.hbs }}
{{#each model as |post|}}
  {{blog-post title=post.title body=post.body}}
{{/each}}
```

Cargando modelo en el route handler:

```
// app/routes/index.js
import Ember from 'ember';

export default Ember.Route.extend({
  model() {
    return this.get('store').findAll('post');
  }
});
```

HTML renderizado:

```
<article class="blog-post">
  <h1>Un posteo</h1>
  <p>Un párrafo (qué original..)</p>
</article>
```

Renderizando Componentes Dinámicamente

{{blog-post}} ⟷ {{component 'blog-post'}}

```
{{! app/templates/components/post-style-1.hbs }}
<article class="blog-post-1">
  <h1>{{post.title}}</h1>
  <p>{{post.body}}</p>
</article>
```

```
{{! app/templates/components/post-style-2.hbs }}
<article class="blog-post-2">
  <h1>{{post.title}}</h1>
  <p>Hola {{post.author}}!</p>
</article>
```

```
{{! app/templates/index.hbs }}
{{#each model as |post|}}
  {{component post.componentName post=post}}
{{/each}}
```

```
// app/routes/index.js
import Ember from 'ember';

export default Ember.Route.extend({
  model() {
    return this.get('store').findAll('post');
  }
});
```

Parámetros Posicionales

Podemos pasar parámetros por posición:

```
{{! app/templates/index.hbs }}
{{#each model as |post|}}
  {{blog-post post.title post.body}}
{{/each}}
```

```
// app/components/blog-post.js
import Ember from 'ember';

const BlogPostComponent = Ember.Component.extend({});

BlogPostComponent.reopenClass({
  positionalParams: ['title', 'body']
});

export default BlogPostComponent;
```

```
// app/components/blog-post.js
import Ember from 'ember';

const BlogPostComponent = Ember.Component.extend({
  title: Ember.computed('params.[]', function(){
    return this.get('params')[0];
  }),
  body: Ember.computed('params.[]', function(){
    return this.get('params')[1];
  })
});

BlogPostComponent.reopenClass({
  positionalParams: 'params'
});

export default BlogPostComponent;
```

Customizando el Elemento de un Componente: tagName

Por defecto los componentes están dentro de ...

Para usar otro tag: propiedad `tagName`

```
// app/components/nav-bar.js
export default Ember.Component.extend({
 tagName: 'nav'
});
```

```
{{! app/templates/components/nav-bar.hbs }}
<ul>
  <li>{{#link-to "home"}}Home{{/link-to}}</li>
  <li>{{#link-to "about"}}About{{/link-to}}</li>
</ul>
```

Customizando el Elemento de un Componente: classNames & classNameBindings

Clase por defecto: “ember-view”
Para añadir clases: `classNames`

```
// app/components/btn-link.js
export default Ember.Component.extend({
 classNames: ['btn', 'btn-xs']
});
```

```
<div class="ember-view btn btn-xs">
..
</div>
```

Es posible determinar clases según propiedades del componente con `classNameBindings`

```
export default Ember.Component.extend({
  classNameBindings: ['isEnabled:enabled:disabled'],
  isEnabled: true //false
});
```

```
<div class="ember-view enabled"> <!-- disabled -->
..
</div>
```

Customizando el Elemento de un Componente: attributeBindings

```
export default Ember.Component.extend({
  tagName: 'a',
  attributeBindings: ['href'],
  href: 'http://emberjs.com'
});
```

```
export default Ember.Component.extend({
  tagName: 'a',
  attributeBindings: ['customHref:href'],
  customHref: 'http://emberjs.com'
});
```

```
<a href="http://emberjs.com">
..
</a>
```

Envolviendo Contenido en un Componente: {{yield}}

```
{{! app/templates/components/blog-post.hbs }}
<h1>{{title}}</h1>
<div class="body">{{yield}}</div>
```

```
{{! app/templates/index.hbs }}
{{#blog-post title=title}}
  <p class="author">por {{author}}</p>
  {{body}}
{{/blog-post}}
```

```
post: {
  title: "Mi Título",
  author: "Jorge",
  body: "Presentando Ember
Components"
}
```

```
<h1>Mi Título</h1>
<div class="body">
  <p class="author>por Jorge</p>
  Presentando Ember Components
</div>
```

Envolviendo Contenido en un Componente: {{yield}}

Pueden retornar output para ser usado en un bloque

```
{{! app/templates/components/item-info.hbs }}
{{yield model.title model.body model.author}}
```

```
{{! app/templates/components/item-info.hbs }}
{{#if hasBlock}}
  {{yield model.title}}
  {{yield model.body}}
  {{yield model.author}}
{{else}}
  <h1>{{model.title}}</h1>
  <p class="author">Authored by {{model.author}}</p>
  <p>{{model.body}}</p>
{{/if}}
```

```
{{! app/templates/index.hbs }}
{{#item-info model=post as |title body author|}}
  <h2>{{title}}</h2>
  <p class="author">by {{author}}</p>
  <div class="post-body">{{body}}</p>
{{/item-info}}
```

Envolviendo Contenido en un Componente: {{yield}}

Yielding Components: Compartiendo contenido mediante componentes contextuales

```
{{! app/templates/components/alert-box.hbs }}
<div class="alert-box">
  {{yield (hash
    close-button=(component 'alert-box-button' onclick=(action 'close'))
  )}}
</div>
```

```
{{! app/templates/index.hbs }}
{{#alert-box as |box|}}
  Danger, Will Robinson!
  <div style="float:right">
    {{#box.close-button}}
      It's just a plain old meteorite.
    {{/box.close-button}}
  </div>
{{/alert-box}}
```

- Contenido del componente compartido sólo con el contexto del bloque del componente padre.
  - A diferencia de simplemente anidar componentes → todos los contextos iguales, incluso con el del template del llamador del componente.

Manipulando Eventos

Podemos incluir event handlers en los componentes:

```
export default Ember.Component.extend({
  doubleClick() {
    alert("DoubleClickableComponent was clicked!");
  }
});
```

Para habilitar bubbling a componentes padres:

```
export default Ember.Component.extend({
  doubleClick() {
   alert("DoubleClickableComponent was clicked!");
    return true;
  }
});
```

(Touch Events) touchStart, touchMove, touchEnd, touchCancel
(Keyboard Events) keyDown, keyUp, keyPress
(Mouse Events) mouseDown, mouseUp, contextMenu, click, doubleClick, mouseMove, focusIn, focusOut, mouseEnter, mouseLeave
(Form Events) submit, change, focusIn, focusOut, input
(HTML5 drag and drop events) dragStart, drag, dragEnter, dragLeave, dragOver, dragEnd, y drop

Manipulando Eventos Mandando Acciones

Pasamos acción “didDrop” (definida en el route handler o controller)

```
{{! app/templates/index.hbs }}
{{drop-target action=(action "didDrop")}}
```

```
{{! app/components/drop-target.js }}
export default Ember.Component.extend({
  attributeBindings: ['draggable'],
  draggable: 'true',

  // podemos utilizar el objeto de evento si definimos
  // el event handler en el componente:
  drop(event) {
    let id = event.dataTransfer.getData('text/data');
    // en caso de evento ‘drop’ se llama a la acción
    // pasada en el template con argumento id:
    this.get('action')(id);
  }
});
```

(Touch Events) touchStart, touchMove, touchEnd, touchCancel
(Keyboard Events) keyDown, keyUp, keyPress
(Mouse Events) mouseDown, mouseUp, contextMenu, click, doubleClick, mouseMove, focusIn, focusOut, mouseEnter, mouseLeave
(Form Events) submit, change, focusIn, focusOut, input
(HTML5 drag and drop events) dragStart, drag, dragEnter, dragLeave, dragOver, dragEnd, y drop

Manipulando Eventos Mandando Acciones

Pasamos acción "`signUp`" a un inline event handler en el template del componente

```
{{! app/templates/components/sign-btn.hbs }}
<button onclick={{action 'signUp'}}>Sign Up</button>
```

```
{{! app/components/sign-btn.js }}
export default Ember.Component.extend({
  actions: {
    signuUp(event){
      // podemos utilizar el objeto evento del
      // browser
    }
  }
});
```

Comportamiento por defecto:

```
{{! app/templates/components/sign-btn.hbs }}
<button {{action 'signUp'}}>Sign Up</button>
```

```
{{! app/components/sign-btn.js }}
export default Ember.Component.extend({
  actions: {
    signuUp(){
      // en el template no se recibe el objeto
      // evento, por lo que en la definición de
      // esta acción no se puede utilizar
    }
  }
});
```

El Ciclo de Vida de un Componente*:

En render inicial

1. init
2. didRecieveAttrs
3. willRender
4. **didInsertElement**
5. didRender

En re-render

1. **didUpdateAttrs**
2. didRecieveAttrs
3. willUpdate
4. willRender
5. didUpdate
6. didRender

En destrucción del componente

1. willDestroyElement
2. willClearRender
3. didDestroyElement

*Ver aplicación para más detalles

2. Ember 1 → Ember 2

Cambios

Adición del nuevo motor de renderizado: Glimmer Engine.

- Soporte para paréntesis angulares <> para ember components.
- Significó que varios add-ons de terceros tuvieron que reescribirse para volver a ser compatibles..
- Mientras no se usen add-ons de terceros, Glimmer es compatible con el antiguo.

Se pretende seguir el data flow de React: one way binding por defecto

The new data flow model, including one-way data flow by default, separation of attributes from component state, opt-in mutable bindings, and callback-style actions are all opt-in through the use of another big new feature: angle-bracket components.

```
{{!-- title is a mutable two-way binding --}}
{{my-component title=model.name}}

{{!-- title is just an (immutable) value --}}
<my-component title={{model.name}} />
```

Cambios

- No se remueven los componentes regulares con llaves para mantener una forma fácil de 2-way binding.
- Se eliminaron cosas como los metalmorf tags
- Se añadieron los componentes dinámicos.
- Se agregó fastboot, que es parte de renderizado en servidor

3. Web Components y {Ember, Angular, React}.Components

Ember Components y Web Components

Ember Components

- Scope aislado. Sólo conoce el contexto entregado. Cuerpo del componente está en el mismo scope que el del template en el que se llamó.
- Aplicaciones Ember usables en cualquier browser moderno.
- Sólo funcionan con Ember.js
- Built-in data-binding
- Lifecycle hooks: didInsertElement, didUpdateAttrs, willClearRender, willDestroyElement, etc
- Posibilidad de crear componentes que heredan de otros componentes.
- Modificar tagName, className y atributos del tag.
- Distribución de componentes Ember con addons de Ember CLI se sigue desarrollando.

Web Components

- Custom elements soportado no en todos los browsers.
- Posibilidad de definir nuevos elementos HTML/DOM.
- Crear elementos que extienden de otros elementos.
- Extiende API de elementos DOM existentes.

Integración de Web Components con Ember, Angular y React

Ember: posible, no sencilo.

Angular: posible. Más facilidades que con Ember.

React: prácticamente imposible.

Enlaces

- https://guides.emberjs.com/v2.8.0/components/defining-a-component/ (Guía oficial)
- http://emberjs.com/blog/2016/01/15/ember-2-3-released.html#toc_contextual-components (Componentes contextuales)
- http://emberwatch.com/recipes/helpers_and_components (Cookbook)
- http://yoember.com/ (Buen tutorial ✓)
- http://cbateman.com/blog/web-components-in-angular-ember-and-react/ (Comparación de componentes)
- http://emberjs.com/blog/2015/05/10/run-up-to-two-oh.html (Transición a Ember 2)