IC leonidas / transparency

Transparency is a semantic template engine for the browser. It maps JSON objects to DOM elements by id, class and data-bind attributes. — Read more

http://leonidas.github.com/transparency/

transparency / 693 commits

Publish 0.9.14

pyykkis authored 23 days ago latest commit e449c0c2c4

browser	4 months ago	Fix paths to point 'dist/' folder. [pyykkis]
dist	23 days ago	Get rid of lodash, use trivial implementations instead. [pyykkis]
examples	23 days ago	Get rid of lodash, use trivial implementations instead. [pyykkis]
lib	23 days ago	Get rid of lodash, use trivial implementations instead. [pyykkis]
spec	2 months ago	Fix custom matcher config. [pyykkis]
src	2 months ago	Make transparency to work properly when added to project with browser... [pyykkis]
.gitignore	3 months ago	Don't ignore .js files. [pyykkis]
.travis.yml	2 months ago	Disable node.js 0.9 from travis build due to npm bugs [pyykkis]
FAQ.md	a year ago	Removed deprecated FAQ [miohtama]
Gruntfile.coffee	2 months ago	Add custom built lodash and remove underscore methods from helpers. [pyykkis]
Jamesfile.coffee	2 months ago	Make transparency to work properly when added to project with browser... [pyykkis]
LICENSE	2 years ago	Added MIT license [pyykkis]
PerformanceSpecRunner.html	3 months ago	Update to latest jquery. [pyykkis]
README.md	2 months ago	Update README.md [pyykkis]
Testling.html	2 months ago	Try self-made html for more robust testling results. [pyykkis]
component.json	23 days ago	Publish 0.9.14 [pyykkis]
index.js	2 months ago	Make transparency to work properly when added to project with browser... [pyykkis]
package.json	23 days ago	Publish 0.9.14 [pyykkis]
testem.json	4 months ago	Add testem config. [pyykkis]

README.md

Synopsis

Transparency is a client-side template engine which binds data to DOM. Just call `.render(data)`.

```
<div id="template">
  <span class="greeting"></span>
```

```
  <span data-bind="name"></span>
</div>
```

```
var hello = {
  greeting: 'Hello',
  name:     'world!'
};

$('#template').render(hello);
```

```
<div id="template">
  <span class="greeting">Hello</span>
  <span data-bind="name">world!</span>
</div>
```

build passing

Features

- **Semantic data binding** - No need for `<%=foo%>` or `{{foo}}` assignments
- **Collection rendering** - No need for hand-written loops
- **Valid HTML templates** - Write templates as a part of the HTML, in plain HTML
- **View logic in JavaScript** - No crippled micro-template language, just plain JavaScript functions

Transparency is compatible with IE9+, Chrome, Firefox, iOS, Android and other mobile browsers. Support for IE8 requires jQuery.

Community

- IRC: freenode/#transparency.js
- Google Groups: transparencyjs@googlegroups.com

Fiddle

Try Transparency with interactive examples.

Install

```
curl https://raw.github.com/leonidas/transparency/master/dist/transparency.min.js

# Or with Bower
npm install -g bower
bower install transparency
```

Require with script tags

```
<script src="js/jquery-1.7.1.min.js"></script>
<script src="js/transparency.min.js"></script>
```

or with AMD

```
require(['jquery', 'transparency'], function($, Transparency) {

  // Without jQuery
  Transparency.render(document.getElementById('template'), data);

  // With jQuery
  jQuery.fn.render = Transparency.jQueryPlugin;
  $('#template').render(data);
});
```

Node.js

`npm install transparency`

For server-side use, see examples/hello-server.

API documentation

Here are short and detailed examples how to use Transparency. For more elaborate examples, see User manual and FAQ. Feel free to add your own examples in the wiki, too!

Implementation details are explained in the annotated source code.

Binding values

Transparency binds values in-place. That means, you don't need to write separate template sections on your page. Instead, compose just a normal, static html page and start binding some dynamic values on it!

By default, Transparency binds JavaScript objects to a DOM element by `id`, `class`, `name` attribute and `data-bind` attribute. Default behavior can be changed by providing a custom matcher function, as explained in section Configuration. Values are escaped before rendering.

Template:

```
<div id="container">
  <div id="hello"></div>
  <div class="goodbye"></div>
  <input type="text" name="greeting" />
  <button class="hi-button" data-bind="hi-label"></button>
</div>
```

Javascript:

```
var hello = {
  hello:      'Hello',
  goodbye:    '<i>Goodbye!</i>',
  greeting:   'Howdy!',
  'hi-label': 'Terve!' // Finnish i18n
};

// with jQuery
$('#container').render(hello);

// ..or without
Transparency.render(document.getElementById('container'), hello);
```

Result:

```
<div class="container">
  <div id="hello">Hello</div>
  <div class="goodbye">lt;i&gt;Goodbye!&lt;/i&gt;</div>
  <input type="text" name="greeting" value="Howdy!" />
```

```
  <button class="hi-button" data-bind="hi-label">Terve!</button>
</div>
```

Rendering a list of models

Template:

```
<ul id="activities">
  <li class="activity"></li>
</ul>
```

Javascript:

```
var activities = [
  {activity: 'Jogging'},
  {activity: 'Gym'},
  {activity: 'Sky Diving'},
];

$('#activities').render(activities);

// or
Transparency.render(document.getElementById('activities'), activities);
```

Result:

```
<ul id="activities">
  <li class="activity">Jogging</li>
  <li class="activity">Gym</li>
  <li class="activity">Sky Diving</li>
</ul>
```

Rendering a list of plain values

With plain values, Transparency can't guess how you would like to bind the data to DOM, so a bit of help is needed. Directives are just for that.

Access to the plain values within the directives is provided through `this.value`. There's a whole lot more to say about the directives, but that's all we need for now. For the details, see section Directives.

Template:

```
<div>
  <div class="comments">
    <label>Comments:</label><span class="comment"></span>
  </div>
</div>
```

Javascript:

```
var comments, directives;

comments = ["That rules", "Great post!"];

// See section 'Directives' for the details
directives = {
  comment: {
    text: function() {
      return this.value;
    }
  }
};

$('.comments').render(comments, directives);
```

Result:

```
<div>
  <div class="comments">
    <label>Comments</label><span class="comment">That rules</span>
    <label>Comments</label><span class="comment">Great post!</span>
  </div>
</div>
```

Nested lists

Template:

```
<div class="container">
  <h1 class="title"></h1>
  <p class="post"></p>
  <div class="comments">
    <div class="comment">
      <span class="name"></span>
      <span class="text"></span>
    </div>
  </div>
</div>
```

Javascript:

```
var post = {
  title:    'Hello World',
  post:     'Hi there it is me',
  comments: [ {
      name: 'John',
      text: 'That rules'
    }, {
      name: 'Arnold',
      text: 'Great post!'
    }
  ]
};

$('.container').render(post);
```

Result:

```
<div class="container">
  <h1 class="title">Hello World</h1>
  <p class="post">Hi there it is me</p>
  <div class="comments">
    <div class="comment">
      <span class="name">John</span>
      <span class="text">That rules</span>
    </div>
    <div class="comment">
      <span class="name">Arnold</span>
      <span class="text">Great post!</span>
    </div>
  </div>
</div>
```

Nested objects

Template:

```
<div class="person">
  <div class="firstname"></div>
  <div class="lastname"></div>
```

```
  <div class="address">
    <div class="street"></div>
    <div class="zip"><span class="city"></span></div>
  </div>
</div>
```

Javascript:

```
var person = {
  firstname: 'John',
  lastname:  'Wayne',
  address: {
    street: '4th Street',
    city:   'San Francisco',
    zip:    '94199'
  }
};

$('.person').render(person);
```

Result:

```
<div class="container">
  <div class="firstname">John</div>
  <div class="lastname">Wayne</div>
  <div class="address">
    <div class="street">4th Street</div>
    <div class="zip">94199<span class="city">San Francisco</span></div>
  </div>
</div>
```

Directives

Directives are actions Transparency performs while rendering the templates. They can be used for setting element `text` or `html` content and attribute values, e.g., `class`, `src` or `href`.

Directives are plain javascript functions defined in a two-dimensional object literal, i.e.,

`directives[element][attribute] = function(params) {...}`

where `element` is value of `id`, `class`, `name` attribute or `data-bind` attribute of the target element. Similarly, `attribute` is the name of the target attribute.

When a directive function is executed, `this` is bound to the current model object. In addition, the directive function receives current element as `params.element`, current index as `params.index` and current value as `params.value`.

The return value of a directive function is assigned to the matching element attribute. The return value should be string, number or date.

Template:

```
<div class="person">
  <span class="name">My name is </span>
  <a class="email"></a>
</div>
```

Javascript:

```
var person, directives;

person = {
  firstname: 'Jasmine',
  lastname:  'Taylor',
  email:     'jasmine.tailor@example.com'
};
```

```
directives = {
  name: {
    text: function(params) {
      return params.value + this.firstname + " " + this.lastname;
    }
  },

  email: {
    href: function(params) {
      return "mailto:" + this.email;
    }
  }
};

$('.person').render(person, directives);
```

Result:

```
<div class="person">
  <span class="name">My name is Jasmine Taylor</span>
  <a class="email" href="mailto:jasmine.tailor@example.com">jasmine.tailor@example.com</a>
</div>
```

Nested directives

Template:

```
<div class="person">
  <span class="name"></span>
  <span class="email"></span>
  <div class="friends">
    <div class="friend">
      <span class="name"></span>
      <span class="email"></span>
    </div>
  </div>
</div>
```

Javascript:

```
person = {
  firstname:  'Jasmine',
  lastname:   'Taylor',
  email:      'jasmine.taylor@example.com',
  friends:    [ {
      firstname: 'John',
      lastname:  'Mayer',
      email:     'john.mayer@example.com'
    }, {
      firstname: 'Damien',
      lastname:  'Rice',
      email:     'damien.rice@example.com'
    }
  ]
};

nameDecorator = function() { "<b>" + this.firstname + " " + this.lastname + "</b>"; };

directives = {
  name: { html: nameDecorator },

  friends: {
    name: { html: nameDecorator }
  }
};
```

```
$('.person').render(person, directives);
```

Result:

```
<div class="person">
  <span class="name"><b>Jasmine Taylor</b></span>
  <span class="email">jasmine.taylor@example.com</span>
  <div class="friends">
    <div class="friend">
      <span class="name"><b>John Mayer</b></span>
      <span class="email">john.mayer@example.com</span>
    </div>
    <div class="friend">
      <span class="name"><b>Damien Rice</b></span>
      <span class="email">damien.rice@example.com</span>
    </div>
  </div>
</div>
```

Configuration

Transparency can be configured to use custom matcher for binding the values to DOM elements.

For example, one might want to bind only with `data-bind` attribute, but not with `class` or `id` attributes. Custom matcher function should take `key` and `element` as parameters and return `true` if the corresponding value should be bind to the given DOM element.

```
// The custom matcher gets Transparency `Element` wrapper with following properties
// element.el:         Raw DOM element
// element.name:       Lower-cased name of the DOM element, e.g., 'input'
// element.classNames: List of class names of the DOM element, e.g., ['person', 'selected']
Transparency.matcher = function(element, key) {
  return element.el.getAttribute('data-bind') == key;
};

// Values are bind only with 'data-bind' attribute
template.render({name: "Will Smith"});
```

Debugging templates, data and Transparency

http://leonidas.github.com/transparency/ is great place to fiddle around with your data and templates.

To enable debug mode, call `.render` with a `{debug: true}` config and open the javascript console.

```
$('container').render(data, {}, {debug: true});
```

Development environment

Install node.js 0.8.x and npm. Then, in the project folder

```
$ npm install grunt -g  # command-line build tool to enable TDD, auto-complation and minification
$ npm install           # Install the development dependencies
$ grunt                 # Compile, run tests, minify and start watching for modifications
```

The annotated source code should give a decent introduction.

Contributing

All the following are appreciated, in an asceding order of preference

1. A feature request or a bug report
2. Pull request with a failing unit test
3. Pull request with unit tests and corresponding implementation

In case the contribution is changing Transparency API, please create a ticket first in order to discuss and agree on design.