better templates from the ground up with Node.js

Garann Means : @garannm
you need templates
you need a template strategy
assumption: html
doesn’t belong in your js
how come:

- it has to be escaped
- it's less modular
- string or array concatenation is easy to mess up
- it makes it difficult to actually see your HTML source
but you want to avoid manipulating the dom
how come:

- poor performance
- fragile code
- depends on hooks
- depends on hierarchy
- again, HTML and JS are tightly coupled
you can’t avoid the dom

- when wiring up events
- when accepting user input
- when reflecting changes
so save up your dom allowance
templates are useless without data
it can come from anywhere

- proper data models
- state settings in object literals
- transformed user input
- notifications via XHR or web sockets to display
- even first render from the server
assumption: all data needs transforming
for instance:

- making things plural
- currency and date formats
- user names and pronouns
- subtotaling and other arithmetic
- differences for boolean states
- language translation
that stuff is not application logic

it’s part of the display
assumption: JS doesn’t belong in your HTML
getting around that

- helpers and filters in templates
- classes and data attributes for behaviors
- even better: modular templates
- add helpers to data properties before render
- add behaviors to elements post-render
simple template

<h2>{{=it.title}}</h2>
<h3>{{=it.artist.name}}</h3>
<img src="img/{{=it.id}}.jpg" alt="{{=it.title}}" />
<ul class="info">
  <li>{{=priceFormat(it.price)}}</li>
  {{~it.info :info_item}}
  <li>{{=info_item}}</li>
  {{~}}
</ul>
uses

- rendering some data from an XHR on the client
- rendering a whole “page” or state in a SPA
- rendering a page or view on the server
saved and loaded

- has no meta info about how it's used - it's up to the app to know

- in node: have to be sure i/o is complete before page is delivered

- on client: use XHR (on init) or AMD (in view module)

- might be a partial

- needs to be registered on both client and server
an object that feeds it

{
  id: 101,
  title: "Las Dos Fridas",
  artist: {
    name: "Frida Kahlo"
  },
  price: 59.99,
  info: [
    '11" x 14"',
    "acid-free paper",
    "suitable for matting",
    "limited edition"
  ]
}
$.when(
  $.get( "tmpl/detail.dot", function( tmpl ) {
    detailTpl = doT.template( tmpl );
  }, "text" )
).then( init );
function loadDetail( id ) {
    if ( id.data ) {
        id = id.data.id;
    }
    $.get( "detail/" + id, function( info ) {
        $( "div.detail" ).html( detailTmpl( info ) );
    });
}
some data transformation

function priceFormat( price ) { 
    price = price % 1 == 0 ? price + " .00" : price;
    return "$" + price;
}
that’s pretty easy
on the server

```javascript
// using Express and Consolidate
app.get("/product/:id", function( req, res ) {
  res.render("detail", products[ req.params.id ]);
});
```
problem!

- no globals in Node
- ReferenceError: priceFormat is not defined
- helpers need to be registered
templates increase in difficulty pretty quick
framework templates

- back to easy, right?
- templates may provide data binding
- templates may provide behaviors
- if you want to do something differently, you’ll need to think about all the parts
templates glue the whole display together
node.js can help
manual labor
template "builds"

- loaded as text from an element or file
- run through a compile function
- stored as a function
- calling function with data produces hydrated markup
precompiling

- no need to fetch template text
- compiled function is just JS
- can be concatenated onto the rest of your JS
- can be treated as a module if you provide a wrapper
what does that have to do with node?
it’s just easier

- get your template engine from npm
- template logic is in JS
- objects and dot notation work like JS
- truthy and falsey work like JS
- wrappers and helpers are in the same language
node is excellent for building js
but why let the client have all the fun
shared templates save you work
you don't need node

zillions of flavors of Mustache

parsers for lots of other popular template engines

client-side versions of some server-side templating frameworks
but it makes sense

templates themselves are easy

template architecture is hard

you want to share more than just the template
twice the app with half the material
not just single page apps

- providing fully rendered pages from the server
- sharing states
- sharing data transformation and helpers
- not using two different template languages, which will drive you mad
partials are your best frenemy
using a partial

<h2>Modern Art</h2>
<div id="results">
{{~it.results :r}}
{{#def.result_detail}}
{{~}}
</div>
the partial itself

```html
<div class="result" data-id="{{=r.id}}">
  <img src="img/{{=r.id}}.jpg" alt="{{=r.title}}" />
  <h3>{{=r.title}}</h3>
  <span class="artist-name">
    {{=r.artist.name}}
  </span>
  <span class="price">
    {{=r.price}}
  </span>
</div>
```
the good

can share chrome around a variety of displays

templates can be as granular as you want
don’t need giant conditional blocks
re-render only the pieces of the page that change
the bad

- need to be available and registered before containing templates
- have to be compiled before solo use
- a template isn’t usually a module
- so it can’t define dependencies
$.when(  
  $.get( "tmpl/result_detail.dot", function( tmpl ) {  
    partials.result_detail = tmpl;
  }, "text" )
).then(  
  $.get( "tmpl/detail.dot", function( tmpl ) {  
    detailTmpl = 
      doT.compile( tmpl, partials );
  }, "text" )
).then( init );
on the server

// throw Express easy rendering out the window
// can't use external partials with Consolidate
// have to write it from scratch
// so.. looks basically the same as the client
we can make this work better.
what do we want?

- composable templates
- modules that manage their dependencies
- a place to map helpers and behaviors to properties and elements
- template code abstracted out of application logic
- something that works on both the client and server
how do we get it?
we write it ourselves*

// Tmpl.js
define( [ "require", "doT" ],
  function ( require, doT ) {

    function Tmpl() {

      return this;
    }

    return Tmpl;
  });
should be simple to use

// results.js
define([ "Tmpl", "resultDetail" ],
  function( Tmpl, resultDetail ) {

    this = new Tmpl();

    this.name = "results";
    this.el = "#contents";
    this.partials = { detail: resultDetail };

    this.init();
    return this;

});
and to use recursively

```javascript
// result_detail.js
define( [ "Tmpl", "utils" ], function( Tmpl, utils ) {

    this = new Tmpl();
    this.name = "result_detail";
    this.config = { varname: "r" };
    this.el = "#results";
    this.helpers = { "price": utils.priceFormat };

    this.init();
    return this;
});
```
do ugly stuff on init

```javascript
this.init = function() {
  this.tmpl_txt = require( "text!tmpl/" + this.name );

  var def = {};
  if ( this.partials ) {
    for ( var p in this.partials ) {
      var partial = this.partials[p];
      def[ partial.name ] = partial.tmpl_txt;
    }
  }

  this.tmpl_fn =
    doT.template( this.tmpl_txt, this.config, def );
};
```
but meta info stays with the template
app only knows it exists

// some_view.js
define(["results"], function(results) {

function blahBlahBlah() {
...
results.render({
  search_text: "Modern art",
  results: resultsArrayFromXHR
});
...
}
});
function getHTML( fn, data ) {
    if ( this.helpers ) {
        data = this.transform( data );
    }
    if ( data.length ) {
        var html = "";
        data.forEach( function( d ) {
            html += fn( d );
        });
    }
    return html.length ? html : fn( data );
}

this.render = function( data, $el ) {
    $el = $el || $( this.el );
    return $el.html( getHTML( this.tmpl_fn, data ) );
};
and data transformation

this.transform = function( data ) {
  if ( data.length ) {
    data = data.map( this.transform );
  } else {
    this.helpers.forEach( function( propName ) {
      data[ propName ] = this.helpers[ propName ].call(
        this, data[ propName ]
      );
    });
  }
  return data;
}
this.append = function( data, $el ) {
    $el = $el || $( this.el );
    return $el.append( getHTML( this.tmpl_fn, data ) );
};

this.prepend = function( data, $el ) {
    $el = $el || $( this.el );
    return $el.prepend( getHTML( this.tmpl_fn, data ) );
};

this.serve = function( data ) {
    return getHTML( this.tmpl_fn, data );
};
so we can also do

```javascript
function aNewDetailAppeared() {
  ...
  resultspartials.detail.append(
    {
      id: 103,
      title: "Madame Pompadour",
      artist: {
        name: "Amedeo Modigliani"
      },
      price: 29.99
    }
  )
  ...
}
```
or on the server

```javascript
var resultsTmpl = requirejs( "results" );
...

app.get( "/results/:search", function( req, res ) {
    var data = {
        search_text: req.params.search,
        results: getSearchResults( req.params.search)
    };
    res.send( resultsTmpl.serve( data ) );
});
```
✓ composable templates
✓ modules that manage their dependencies
✓ a place to map helpers and behaviors to properties and elements
✓ template code abstracted out of application logic
✓ something that works on both the client and server
and we can take it further

- multiple templates for multiple states
- data transformation can create new properties
- a custom config that changes all the delimiters to ⚒ can be baked in
- templates can wire up plugins and widget events
a good foundation:

- using templates is a good start
- a template is more than just some HTML
- letting templates manage their own meta information keeps your app code clean
- using Node keeps templates and app code consistent
- might as well build it in from the start
that’s a solid day’s work

thanks!

@garannm / garann.com / garann@gmail.com