how to **rewrite** your JS app *ten times**

Garann Means 🌸  @garannm

* at least
the hardest problem in app development
worries in starting

- how to split things up
- where does the app start?
- what utilities will I rely on?
- OMG do I have any idea what I’m doing?
coding is easy.

planning is hard.
we tend to overprioritize the easy one.
biggest worry

Is it possible that I am going to end up rewriting all of this?
different set of worries
rewriting what already exists

what’s the reason for the rewrite?
what if I accidentally regress something?
are there any yaks I can shave?
OMG do I have any idea what I’m doing?
ok. rewrites are inevitable.
good rewrites

- a new technology comes along
- your business expands
- you upgrade your infrastructure
- plain old incremental refactors
- existing app is like EMERGENCY STATUS broken
bad rewrites

you’ve got 20% of the app written and you realize it’s all wrong
bad rewrites require starting over [again].
bad rewrites are not inevitable.
<ul id="bad-rewrites-i-have-done" class="non-comprehensive">
“let’s use [framework]!”
we love [framework]!

- lots of posts about it on HN
- someone on your team used it once at a hackathon
- it’s MVC, you’ve already got some models!
function ThisIsGoingSoWell( opts ) {
    this.init = function() {};
    this.url = "cool/api/path";
}
Frmwrk.extend( ThisIsGoingSoWell, Model );
function ActuallyDoStuff() {
    // TODO: state objects?
    // TODO: event handlers?
    // TODO: idk this isn’t in the docs??
};
love is fleeting

- solves easy problems
- provides common abstractions
- doesn’t help you organize your app
the framework should be the last choice, not the first.
"nevermind, we’ll write our own!"
it’s gonna be artisanal, y’all

- suited to your app specifically
- you know the code inside and out
- no bloat from features you don’t need
( function MyFrmwrk( window ) {
    this.init = function() {};
    this.url = function() {};
    window.Frmwrk = this;
})( window );
Congratulations!

- you’re rewriting your favorite framework
- with just enough difference that other devs don’t understand it
- and turns out your favorite framework isn’t so easy to write
only write **one** application at a time.
#3

“we can’t define an API without any code!”
do I look like a fortune teller!

- don’t want to outline an API too early
- just more places to change it
- how can you optimize that YOU CAN’T
ViewOne.render = function() {
    // get data, populate template, transform inline

};

ViewTwo.render = function() {
    // get transformed data, populate template

};

ViewThree.render = function() {
    // get populated template from server

};
ViewOne.render = function() {
  // get data, populate template, transform inline
  $.get( "/viewOne" , function( data ) {
    container.html( tmpl( data ) );
  });
};
ViewTwo.render = function() {
    // get transformed data, populate template
    $.get(
        "/viewTwo",
        { language: "en" },
        function( data ) {
            container.html( tmpl( data ) );
        });
};
ViewThree.render = function() {
    // get populated template from server
    $.get(
        "/viewThree",
        { language: "en", subtmpl: "error" },
        function( html ) {
            container.html( html );
        });
};
where the $!&#% is this getting set?!

逻辑不在一个一致的地方是很难找到的，

并且很难重用，

并且使得很难有一个可预测的API。

不可预测的APIs会生出不可预测的架构。
specify APIs thoroughly and early.
“good point, if it’s not JS we’ll do it in PHP”
**super-ultra-optimized**

- static stuff rendered by server
- dynamic stuff by client
- server folks speak their language, JS folks speak theirs
oh wait.

this partial template right here?

we need it on the client, too

no big, just copy it in JS

(six months pass)

arrrrgh why didn’t we just write these templates in the same language
plan the stuff **outside** your JavaScript as well.
#5

“this DOM doesn’t work with our webapp”
Polly-who?

- there are a lot of inconsistencies (still)
- that’s a lot of extra code
- standard implementations may not be quite right
- let’s just write our own DOM!
// _so classic_
( function FileUploader() {
  // we’ll just make a little file selector
  // and style it real pretty
  // and add some cool progress events
  // and the Flash movie that runs it all
})();
// oh.
(function FileUploadHandler() {
  // expose all that Flash stuff to JS
  // let the widget be used in a larger form
})(());
// srsly?
(function FileUploaderDegrade() {
    // fall back to normal upload if no Flash
    // check for XHR2, add progress stuff
    // map DOM API so it matches FileUploader
    // let the widget be used in a larger form
})();
we used to have to do this; we don’t now

- someone at [browser co, inc] is working on improving this stuff
- someone else made a polyfill
- someone else fancied it up and made a jQuery plugin
- if you’re not in the widget business, stop
only write one application at a time.
#6
“XHR? more like HTML amirite”
we’re fetching HTML anyway

- let’s put all our stuff in it!
- object setting? data attribute
- i18n? data attribute
- entire app state??? DATA ATTRIBUTES
MyApp.init = function() {
    // look, no options!
    // we’ll just select the necessary nodes..
    // and then find their attributes..
    // parse those to data types..
};
MyApp.onSomethingChanged = function() {
    // we’ll just select the necessary nodes..
    // and then set their attributes..
    // and update the server, too..
    // wat, 503?!
    // we’ll just select the necessary nodes..
    // and then unset their attributes..
};
the DOM is not a data store

- it’s expensive and messy
- it violates DRY from the jump
- it doesn’t provide objects you can work with
data deserves a real home.
#7

“there’s a plugin/module for that!”
the wheel is perfect

and there are so many wheels to choose from!

a tried and true solution for everything

partway there still beats the starting line
function veryImportant( item ) {
    var $item = $( item );
    $item.transformIntoMagicWidget();
    // NOW we can actually use it
    $item.on( "pluginEvent", superImportantFn );
}
function superImportantFn( e ) {
    var $t = $( this ),
        important = $t.pluginVal();
    $t.addClass( "specialState" );
    $t.removeClass( "pluginState" );
    $t.find( ".aChild" ).append( coolStuff );
    $t.find( ".someBtn" ).prop( "disabled" );
    // $t.find( "ugh this is a nightmare" );
}
$.fn.transformIntoMagicWidget = function() {
    return this.each(function() {
        // MODIFIED: grm 11-10-2013
        // added all that state and CSS stuff
        ...;
    });
};
you’re writing a whole new plugin
based on something you don’t control
that might get abandoned
and getting minimal value from the original

oh hey.
don’t use external tools that almost do something.
#8

“MVC means never having to say require”
but this framework doesn’t use AMD

- single point of entry = one place to manage dependencies
- it’s already clear what the code is doing
- loading stuff is abstracted out anyway
- the internet says you don’t have to worry about speed if you don’t have any JPGs
BigController = function() {
  BigView.renderEverything();
};
Uncaught Error: Object #<Object> has no method ‘renderEverything’
BigController = function( refresh ) {
  if ( refresh ) {
    BigView.setData( data );
  }
  BigView.renderEverything();
};
Uncaught ReferenceError: data is not defined
BigController = function( refresh ) {
    if ( refresh ) {
        if ( data ) {
            BigView.setData( data );
        } else {
            $.get( "/newData", function( d ) {
                BigView.setData( d );
            });
        }
    }
    BigView.renderEverything();
};
Uncaught Error: u should use a dependency loader
lack of modularity
courage messes

- relying on globals or sub-globals is fragile
- lots of tests to see whether things are available
- whole chain has to be reexamined for each edge case
- can’t abstract out loading
assume everything will be a module.
#9

“it works with [terrible thing you use now]!”
we’re limiting our risk!

- less to rewrite
- more time to improve
- makes use of all that hard work we did on the original
/* TODO: delete me i am being replaced
function doStuff() {
    var output = TerribleThing.use();
    Module.transform( output );
    OtherModule.render( output );
    CoolModule.observe( output );
}
*/
function doStuff() {
    var output = TerribleThing.use();
    Frmwrk.transform( output );
    Frmwrk.TrriblView.render( output );
    Frmwrk.Events.observe( output );
}

you saved 4 LoC!

- rewriting around existing code tends to give you.. the same code
- if a tool is bad or niche, no framework on earth can save you from it
- trying to avoid change limits your ability to improve
- which leaves you itching to.. rewrite
throw out **assumptions**

about how things will work.
#10

“we found our solution; what’s the problem?”
this thing looks great!

- super modern
- fast benchmarks

- incredibly flexible, rich tooling, great community, thorough docs, brilliant API, scales to infinity, tiny download, hipster syntax, cool colors, foot massages, free unicorns, and a trip to space!
but does it solve your problem?

- easy to gloss over if you haven’t defined your problem
- no i mean like broken it down
- no into more than three bullet points
- taking the time to define problems avoids irrelevant solutions
to proof of concept you have to know what to prove.
</ul>
no more.
we don’t want any of that.
none of those are uncommon.
it’s hard to avoid rewriting completely..
the goal is not to do it more than once.
rewrite planning

- can you devote the people?
- do you know the problems the rewrite should solve?
- do you have tests to avoid regressions?
- most important: can you commit?
#0 way to rewrite a lot: cold feet.
you have to commit

don’t expect it to happen overnight

don’t expect to iterate on atomic pieces

don’t fail fast (you did that your current app)

don’t start coding until you have a plan
throw out everything you know about RAD.
a plan you can’t commit to isn’t one.
good plans keep you from getting to ten rewrites.
thanks!

@garannm  ♫ garann@gmail.com  ♫ garann.com