jQuery Plugin Patterns

Seminar JavaScript Patterns and Anti-Patterns

Christoph Heidelmann
Agon Lohaj

Content

- Introduction
  - What is jQuery
  - Why plugin patterns?
- jQuery Plugin Patterns
- Refactoring
- Conclusion

What is jQuery?

- a JavaScript library
- released 2006 at BarCamp NYC by John Resig
- provides apis for:
  - document traversal and manipulation
  - event handling
  - animation
  - Ajax
- cross browser support

Why jQuery?

- Important impact on web development history
- most popular library in use today (1)
- huge community
- large amount of available plugins

http://plugins.jquery.com/

1) http://w3techs.com/technologies/overview/javascript_library/all

Why Plugin Patterns

- Better code reuse
- Better extensibility
- Better organization
- compatibility between libraries
- Similiar API
- Maintain complexity when working with several components

jQuery Plugin Patterns

- Basic
- Extend
- Lightweight
- Namespaced Pattern
- Prototypal Inheritance
- Best options
- Custom Events

jQuery Plugin Patterns (continued)

- ‘Highly Configurable’ Mutable
- UI Widget Factory
- UI Widget Factory “bridge”
- UI Widget Factory for jQuery Mobile
- UI Widget + RequireJS module
- Universal Module Definition Pattern

The basic plugin

```
;(function ( $, window, document, undefined ) {
    $.fn.myPluginName = function(options) {…};
})( jQuery, window, document );

$('element').myPluginName(options)

$('element').plugin2(options).myPluginName(options)
```

The extend version

```
$.fn.extend( {
    pluginname: function( options ) {…}
});
```

for defining a large amount of functions and properties

Preventing from multiple initializations

```
$.fn.myPluginName = function ( options ) {
    return this.each(function () {
     if (!$.data(this, 'plugin_' + pluginName))
     {
          $.data(this, 'plugin_' + pluginName,
                            new Plugin( this, options ));
    }}
});

$("div").divPlugin()
```

The plugin constructor

```
function Plugin( element, options ) {
  this.el = element;
  this.$el = $(element);
  …
  this.init();
};
```

Best Options

approach to implement options into the plugin

```
$.fn.pluginName.options = { ... };


$.fn.pluginName = function ( options ) {
   this.options = $.extend( {}, $.fn.pluginName.options,
                  options );
   ...
};
```

Prototype the plugin

```
Plugin.prototype = {

    init: function() {...},

    destroy: function() {...},

    publicMethod: function() {...},

    _privateMethod: function() {...}

}
```

Namespace the plugin

```
if (!$.myNamespace) {
     $.myNamespace = {};
};
$.myNamespace.myPluginName = function() {   ... }
$.fn.mynamespace_myPluginName = function() { ... }
```

Object-To-DOM Bridge Pattern

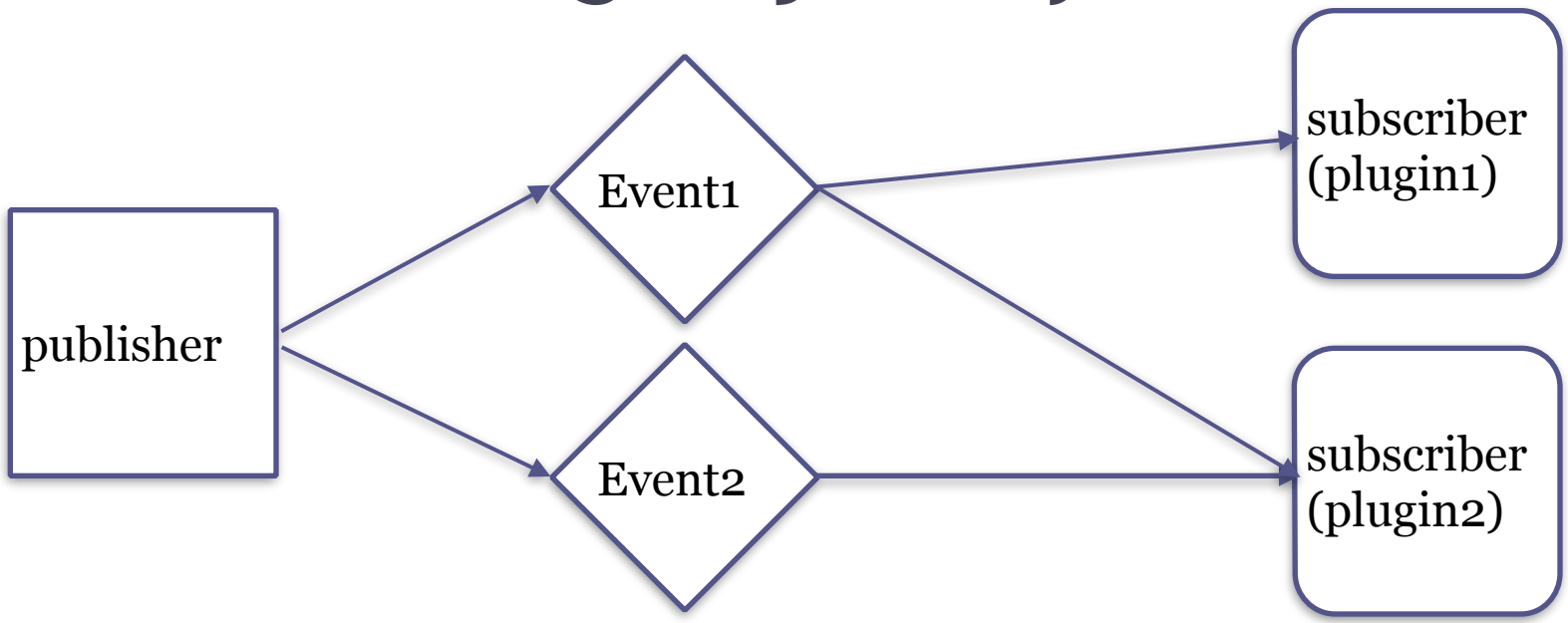
```
var myObject = {
    init: function( options, elem ) { … },
    options: { … },
    _build: function() { … }
};

$.data(this, 'plugin_' + pluginName,
Object.create(object).init(options, this ));

$.plugin("myplugin", myObject);

$.myplugin();
```

Event binding in jQuery




```
$(".plugins").trigger("myEventStart",["param1","param2"]);

this.$element.bind( "myEventStart", function( e ) { … });
```

‘Highly Configurable’ mutable

Develop Plugin by using objects defined in prototypes

```
var Plugin = function( elem, options ){ … };
// the plugin prototype
Plugin.prototype = {
    defaults: { … }, init: function() { … }, sampleMethod: function() { …
}
};
Plugin.defaults = Plugin.prototype.defaults;
$.fn.plugin = function(options) {
    return this.each(function() {
        new Plugin(this, options).init();
    });
};
```

UI Widget Factory

Complex, stateful plugins based on OOP

```
;(function ( $, window, document, undefined ) {
    // define your widget under a namespace of your choice

    // with additional parameters e.g.
    // $.widget( "namespace.widgetname", (optional) - an
    // existing widget prototype to inherit from, an object
    // literal to become the widget's prototype );

    $.widget( "namespace.widgetname" , {
            options: { … },
            _create: function { … },
            destroy: function { … }
    });
});
```

UI Widget Factory “bridge”

Middle layer between $.widget and jQuery API

```
// a "widgetName" object constructor
var widgetName = function( options, element ){ … }
// the "widgetName" prototype
widgetName.prototype = {
    _create: function(){ … }, _init: function(){ … },
    option: function( key, value ){ … }, publicFunction: function(){ … },
    _privateFunction: function(){ … }
};

// usage:
// $.widget.bridge("foo", widgetName);
```

UI Widget Factory for mobile (+ Require JS)

Mobile widgets (and wrapping them inside RequireJS modules)

```
//ao.myWidget.js file:
define("ao.myWidget", ["jquery", "text!templates/asset.html", "jquery-
ui.custom.min","underscore"], function ($, assetHtml) {
    // define your widget under a namespace of your choice
    $.widget("ao.widget", $.mobile.widget, {
        options: {},
        _create: function () {
            var template = _.template(assetHTML);
        }, destroy: function () { … },
        _setOption: function ( key, value ) { … }
    };
});
```

Refactoring

jQuery loader plugin

Three methods:

$.loader()

```
<div id=„background“></div>
<div id=„loader">
    Loading ...
</div>
```

$.loader(„setContent“)

changes content of #loader

$.loader(„close“)

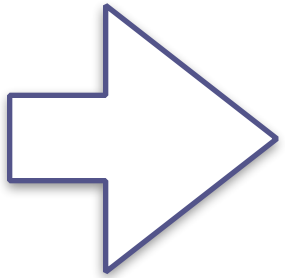
removes elements from DOM

Problems of the plugin

- multiple initialisations -> multiple loader elements with same id
- „setContent“ and „close“ effects every element
- the content is set via a string -> not flexible
- not chainable

Make the plugin depending on an element

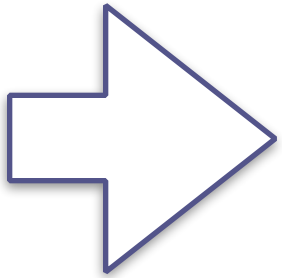
```
$.loader()
$.loader("setContent")
$.loader("close")
```



```
$("#loader").loader()
$("#loader").loader("show")
$("#loader").loader("hide")
```

Prototype the plugin

```
$.loader = function(option){
    switch(option)
    {
    case 'close':
    case 'setContent':
    default:
    }
}
```



```
Loader.prototype = {
    init : function(){…},
    show : function(){…},
    close : function(){…},
    destroy : function(){…}
};
```

Preventing from multiple initializations

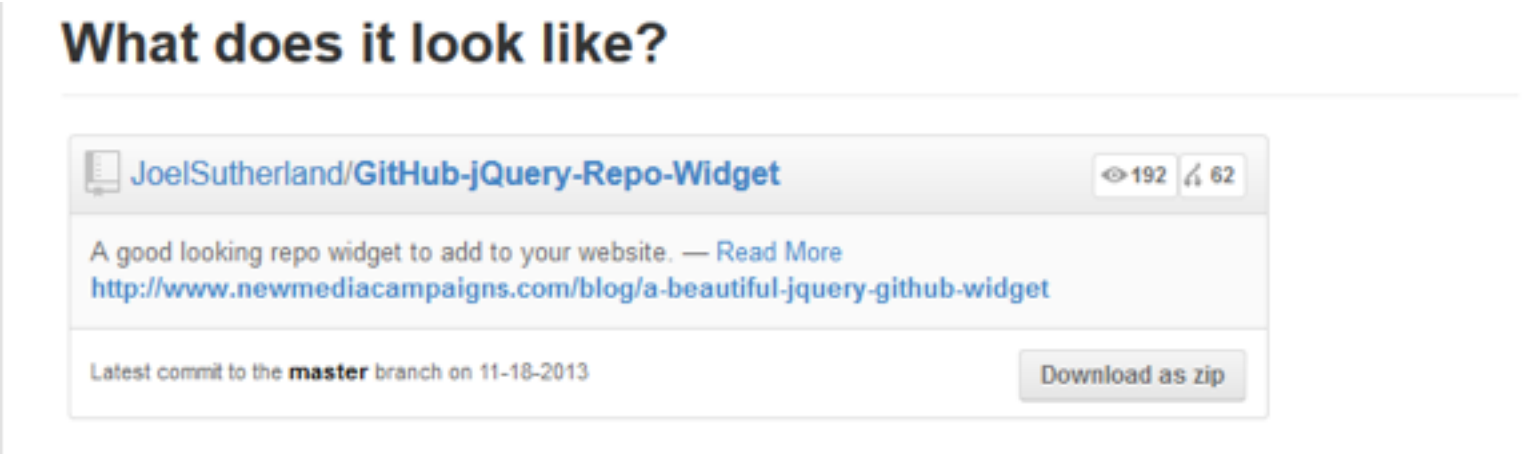
```
$.fn.loader=function(method_options){
   return this.each(function () {
   if(!$.data(this,'plugin_loader'))
         {
         $.data(this,'plugin_loader',
               new Loader(this,method_options));
         }
   } else {
```

Forwarding the method string to the plugin

```
if ( plugin[method_options] ) {
   return plugin[ method_options ]();
} else {
    if(method_options){
       $.error( 'Method not exists');
    }
 }
```

GitHub Repo Widget

A beautiful widget that displays the status of your GitHub repo.



https://github.com/JoelSutherland/GitHub-jQuery-Repo-Widget

GitHub Repo Widget (snippet)

```
;jQuery(document).ready(function($){
        $('.github-widget').each(function(){
                $container = this;
                $widget = $("...some html...");
                $widget.appendTo($container);
                ...
                // more code for initialisation
        }
}
```

GitHub Repo Widget (analysis)

Pros:

- Include the script and your good to go
- Good UI

Cons:

- More of a script rather than a widget
- Dependent on CSS classes and not on html elements
- Poorly organized

Refactoring (Widget factory + Custom Events)

```
$.widget( "custom.githubWidget" , {
        _create: function () {
                var self = this;
                self.element.addClass("github-box repo");
                $widget = $("...some html code...");
                self.element.append($widget);
                self.element.bind( "refresh", function(e){
                        if(self.options.repo != null){// get data }

                });
        }, ...
});
var widget = $(“#ID").githubWidget();
```

Refactoring (Bridge + ‘Highly configurable’ mutable)

```
var githubWidget = function( options, element ){
        ...
        this.metadata = $( element ).data( 'plugin-options' );
        this.init();
}
githubWidget.prototype = {
        init: function(){
                $(this.element).addClass("github-box repo");
                this.config = $.extend({}, this.defaults, this.options,this.metadata);
        },
        option: function( key, value ){ ... }
};
$.widget.bridge("bridgeWidget", githubWidget);
var widget = $(“#ID").bridgeWidget();
```

Conclusion

- Mature Library
- Powerful and feature enriched JavaScripting library
- Through Plugin Patterns
  - Ensure better reusability and extensibility
  - Make the code more organized and easy to read
  - Offer premium usage for other developers

References

- The jQuery loader plugin:

https://github.com/mimiz/jquery-loader-plugin/blob/master/jquery.loader.js

- Github repo widget

https://github.com/JoelSutherland/GitHub-jQuery-Repo-Widget

- Patterns resources

http://addyosmani.com/resources/essentialjsdesignpatterns/book/#jquerypluginpatterns

http://shichuan.github.io/javascript-patterns/#design-patterns

- The refactoring repository

https://github.com/AgonLohaj/seminar-patterns-and-anti-patterns

```
if ( typeof Object.create !== "function" ) {
    Object.create = function (o) {
        function F() {}
        F.prototype = o;
        return new F();
    };
}
```