Правила хорошего тона при написании плагина на jQuery

Я написал уйму плагинов на jQuery. Если посмотреть код всех плагинов, сортируя их по дате публикации на github, то можно проследить эволюцию кода. Ни в одном из этих плагинов не соблюдены все рекомендации, которые будут описаны ниже. Все что будет описано, лишь мой личный опыт, накопленный от проекта к проекту.
Писать расширения на jQuery довольно просто, но если хотите узнать как написать их так, чтобы потом их было просто поддерживать и расширять, добро пожаловать под кат.

Далее пойдет некий свод рекомендаций. Предполагается, что читатель знаком с разработкой плагинов на jQuery. Но я все же приведу пример jQuery плагина, структуру которого обычно рекомендуют в соответствующих статьях.

```
(function ($) {
    $.fn.tooltips = function (opt) {
        var options = $.extend(true, {
            position: 'top'
        }, opt);
        return this.each(function () {
            $(this).after('<div class="xd_tooltips ' + options.position + '">' +
$(this).data('title') + '</div>');
        });
    };
}(jQuery));
```

Это весь функционал нашего плагина. Он добавляет после каждого элемента выборки, **div.xd_tooltips** новый элемент. Это все, что он делает.

Все операции надо делать в отдельном классе

Задумав серьезный плагин, который будет расти, не нужно всю логику засовывать в анонимную функцию в метод each, как в примере. Создайте отдельную функцию конструктор (класс). Затем для каждого элемента из выборки, создавайте отдельный экземпляр этого класса, и сохраняйте ссылку на него в data исходного элемента. При повторном вызове плагина проверяем *data*. Это решает проблему повторной инициализации.

```
(function ($) {
    var Tooltips = function (elm) {
        $(elm).after('<div class="xd_tooltips">' + $(elm).data('title') + '</div>');
    };
    $.fn.tooltips = function (opt) {
```

```
        var options = $.extend(true, {
            position: 'top'
        }, opt);
        return this.each(function () {
            if (!$(this).data('tooltips')) {
                $(this).data('tooltips', new Tooltips(this, options));
            }
        });
    };
}(jQuery));
```

Все общие операции *hide,show,destroy* делайте методами класса

Когда плагин будет большим, и вы захотите использовать часть его функционала в другом плагине/коде, то будет очень удобно если у класса будут доступны открытые методы. Методы можно делать через прототипы, однако тут есть минус — нельзя использовать приватные переменные. Лучшим вариантом будет использование локальных функций, которые затем вернутся в хеше.

Примерно так:

```
// лишний код, и нужно следить за this
var PluginName = function () {
    this.init = function () {
        // инициализация
    };
};
// нормально, но нет локальных приватных переменных
// и также есть проблема с this
var PluginName = function () {};
PluginName.prototype.init = function () {
    // инициализация
};
// идеально
var PluginName = function () {
    var self,
        init = function () {
            // инициализация
        };
    self = {
        init: init
    };
    return self;
};
```

Почему третий вариант лучший?
Во первых — теперь методы внутри класса, можно использовать без префикса *this*. *this* — в неумелых руках это зло. В JS он может быть чем угодно, и очень рекомендую использовать его как можно реже.

Во вторых, когда плагин будет большим, вы решите пропустить его через *uglify*. Тогда в последнем варианте, все упоминания init (кроме ключа

хеша *self.init*) будут заменены на однобуквенную переменную, которая при частом использовании метода, прилично уменьшит код.

В третьих, вам доступны приватные методы и переменные, которые вы объявите в верхнем *var*. Это очень удобно. Что-то вроде *private* методов во «взрослых» ЯП.

Из последнего правила нужно выжать еще одно:

Все инициализацию объекта также надо держать в отдельном методе init

Не делайте по возможности ничего в моменте создания экземпляра класса. Это просто опыт. Ваш плагин рано или поздно вырастет, и вам захочется использовать его код из другого места, не обязательно из jQuery. К примеру у вас в коде будет крутой парсер, и вы захотите использовать его вне всяких DOM элементов.

Тогда:

```
var tooltips = new Tooltips();
tooltips.parse(somedata);
```

Не будет ничего создавать в дереве DOM, а сделает ровно то, что нужно.

Делайте для каждого экземпляра класса отдельный экземпляр настроек

В первом примере мы сделали одну глобальную *options* переменную. А в scope jQuery могло попасть несколько объектов. Чувствуете чем это грозит? JS так работает, что если один из элементов, потом будет менять эти опции, то они изменятся у всех элементов. Это не всегда верно. Поэтому *options* подаем вторым элементом в конструктор нашего класса.

Из всего вышеизложенного, плагин будет выглядеть так:

```
(function ($) {
    var Tooltips = function (elm, options) {
        var self,
            init = function () {
                $(elm).after('<div class="xd_tooltips ' + options.position + '">' +
$(elm).data('title') + '</div>');
            };
        self = {
            init: init
        };
        return self;
    };
    $.fn.tooltips = function (opt) {
```

```
        return this.each(function () {
            var tooltip;
            if (!$(this).data('tooltips')) {
                tooltip = new Tooltips(this, $.extend(true, {
                    position: 'top'
                }, opt));
                tooltip.init();
                $(this).data('tooltips', tooltip);
            }
        });
    };
}(jQuery));
```

Настройки по умолчанию должны быть видны глобально

В примере выше, мы соединяем настройки opt и настройки по умолчанию в виде хеша {}. Это не верно — всегда найдется пользователь, которому такие настройки не подойдут, а вызывать каждый раз плагин со своими настройками будет накладно. Конечно, он может залезть в код плагина и исправить настройки на свои, но тогда он потеряет обновление плагина. Поэтому настройки по умолчанию должны быть видны глобально, и их можно было изменить также глобально.

Для нашего плагина можно использовать это так:

```
$.fn.tooltips = function (opt) {
    return this.each(function () {
        var tooltip;
        if (!$(this).data('tooltips')) {
            tooltip = new Tooltips(this, $.fn.tooltips.defaultOptions, opt);
            tooltip.init();
            $(this).data('tooltips', tooltip);
        }
    });
};
$.fn.tooltips.defaultOptions = {
    position: 'top'
};
```

Тогда любой разработчик сможет у себя в скрипте объявить:

```
$.fn.tooltips.defaultOptions.position = 'left';
```

А не делать это при каждой инициализации плагина.

Не всегда возвращайте *this*

Во всех примерах выше, код инициализации сводился к тому, что мы сразу же возвращали *this.each*. На самом деле почти все методы (плагины) jQuery возвращают *this*.

Поэтому возможны такие вещи как:

```
$('.tooltips')
    .css('position', 'absolute')
    .show()
    .append('<div>1</div>');
```

Это очень удобно. Но к примеру метод *val*, если нет параметров, возвращает значение элемента. Так и у нас, надо предусмотреть способ, по которому, наш плагин мог бы возвращать какие-то значения.

```
$.fn.tooltips = function (opt, opt2) {
    var result = this;
    this.each(function () {
        var tooltip;
        if (!$(this).data('tooltips')) {
            tooltip = new Tooltips(this, $.fn.tooltips.defaultOptions, opt);
            tooltip.init();
            $(this).data('tooltips', tooltip);
        } else {
            tooltip = $(this).data('tooltips');
        }
        if ($.type(opt) === 'string' && tooltip[opt] !== undefined &&
$.isFunction(tooltip[opt])) {
            result = tooltip[opt](opt2);
        }
    });
    return result;
};
```

Обратите внимание, мы добавили второй параметр *opt2*. Он нам будет нужен именно для случаев вызова каких-то методов.
К примеру, для плагинов ввода/вывода актуально изменить исходное *value*.

```
$('input[type=date]').datetimepicker();
$('input[type=date]').datetimepicker('setValue', '12.02.2016');
console.log($('input[type=date]').datetimepicker('getValue'));
```

Используйте 'use strict'

Нет серьезно, вы еще не используете? JS прощает слишком много, и отсюда вырастает огромное количество ошибок. Использование этой директивы убережет вас хотя бы от глобальных переменных.

```
(function ($) {
        'use strict';
        //...
} (jQuery));
```

Кто-то скажет, что можно объявить *'use strict'* в самом начале файла, но я не рекомендую этого делать. Когда проект вырастет, и его будут использовать другие плагины. А создатели тех плагинов не использовали 'use script'. Когда *grunt/gulp/npm* соберет все пакеты в один *build* файл, то вас будет ждать неприятный сюрприз.

Очень рекомендую еще *JSLint* для валидации кода. Я много лет использую *notepad++* для разработки, и в нем нет подсветки ошибок. Может это не актуально для IDE, но мне JSLint позволяет писать более качественный код.

Используйте свой внутренний reset CSS

В начале своего CSS файла, обнуляйте весь CSS для всего плагина. Т.е. нужен какой-то главный класс, под которым будут все остальные. При использовании Less, это придаст дополнительную читабельность:

```
.jodit {
  font-family: Helvetica, Arial, Verdana, Tahoma, sans-serif;
  &, & *{
    box-sizing: border-box;
    padding:0;
    margin: 0;
  }
}
```

Используйте префиксы в названиях CSS классов

CSS в браузерах вещь глобальная. А предугадать в каком окружении будет ваш плагин, невозможно. Поэтому все, даже малозначительные классы, пишите с префиксом. Я очень часто видел, как вспомогательные классы *disable,active,hover*, полностью ломают внешний вид плагина.

Используйте современные способы сборки проекта

В комментариях верно подсказали, что ни один современный JS проект уже не живет без обновлений, автоматических сборщиков, и отслеживания зависимостей.
В коде мы наивно полагали, что jQuery к проекту уже подключен. А что если это не так. Все сломается.

```
;(function (factory) {
        if (typeof define === 'function' && define.amd) {
                // AMD
                define(['jquery'], factory);
        } else if (typeof exports === 'object') {
                // Node/CommonJS для Browserify
                module.exports = factory;
        } else {
                // Используя глобальные переменные браузера
                factory(jQuery);
        }
}(function ($) {
    'use script';
    // код плагина
}));
```

Такой скрипт по умолчанию в браузере отработает точно также как и раньше, однако при использовании Browserify и ему подобных систем, автоматически подтянет все нужные зависимости. И тут может быть не только jQuery. Любая другая библиотека.

Регистрируйте свой проект в bower и npm

Серьезные ребята не будут использовать ваш плагин, если он не будет иметь возможность обновления. Качать плагин архивом с github это уже вчерашний день. Сейчас, достаточно зарегистрироваться на npmjs.com
Затем в корне своего проекта выполнить

```
npm init
```

И следовать инструкциям. В корне проекта появится *package.json*
После чего команда

```
npm publish ./
```

Из того же места, опубликует плагин в npm.

И конечные пользователи смогут устанавливать его себе так:

```
npm install tooltips
```

Выполнять *npm publish ./* надо будет при каждой новой версии. Это не очень то удобно, а до этого еще нужно набрать кучу «git» команд.

Для себя я автоматизировал этот процесс через *package.json*. Добавил в поле *scripts* , такие команды:

```
"version": "1.0.0",
"scripts": {
    "github": "git add --all  && git commit -m \"New version %npm_package_version%\" &&
git tag %npm_package_version% && git push --tags origin HEAD:master && npm publish",
  },
```

А далее просто запускаю:

```
npm run github
```

Оно добавляет все файлы в *git*, делает *commit*, создает тег текущей версии, заливает все это на *github*, а потом еще и обновляет версию на *npmjs*.
Версию оно берет из *package.json*, и ее перед каждым вызовом нужно обновлять. Но можно и это автоматизировать, просто добавить в начало, перед *git add*:

```
npm version patch
```

Я работаю по windows, поэтому переменные *package.json* используются так — *%npm_package_version%*, в других ОС нужно использовать *$npm_package_version* .
Для bower ситуация почти аналогичная. Только нигде регистрироваться не надо.
Если у вас еще не установлен bower.

```
npm install -g bower
```

Затем

```
bower init
```

В корне проекта.

Когда будет создан bower.json, опубликуйте все на github, или другой хостинг.

После этого можно регистрировать пакет:

```
bower register jodit https://github.com/xdan/jodit.git
```

На этом пока все, пишите свои советы в комментариях. Спасибо за внимание!

Теги:

- javascript
- jquery
- plugins

Хабы:

- JavaScript
- jQuery