

Документирование JS кода

Автор: [Старшинов Алексей](#)

Оглавление

Введение	3
Рекомендации	4
Документирование классов.....	4
Документирование констант.....	4
Документирование переменных и полей класса	4
Документирование функций и методов класса	5
Общие рекомендации	5
Список допустимых тегов и их синтаксис.....	5
@class.....	5
@extends	5
@param.....	5
@link	6
@see.....	6
@private.....	6
@field	6
@type	6
@return.....	7
@author	7
Особенности JSDoc и JsDoc Toolkit	7
Общие недостатки.....	7
Недостатки JSDoc.....	8
Недостатки JsDoc Toolkit	8

Введение

В отличие от JavaDoc, JSDoc не существует официально и на него нет единого стандарта. Набор тегов используемый для документирования функций, классов и т.д., варьируется от одного сборщика документации к другому.

Пока выбор встал между JSDoc (<http://sourceforge.net/projects/jsdoc/>) и JsDoc Toolkit (<http://www.jsdoctoolkit.org/>). Первый написан на Perl и требует его наличия (под Windows – ActivePerl), второй – на Java и требует наличия JRE. По значимому набору тегов и удобству работы с программой отличий практически не имеют. Оба имеют те или иные «особенности», и в зависимости от их «злости» должен будет выбран первый или второй сборщик.

JsDoc кажется заброшенным проектом, в Bug Lists и Feature Requests записи очень старые, последний релиз датируется мартом 2007 года. Последний релиз JsDoc Toolkit датируется 5 августа 2008 года.

Рекомендации

Документирование классов

1. Класс должен иметь как можно более полное описание, по возможности, включая примеры использования.
2. Должен присутствовать тег `@author`. Если авторов несколько, все они должны быть перечислены в нескольких тегах `@author`.
3. Использование тега `@version` не приветствуется.
4. Класс должен описываться одним из следующих способов:

a) `function ClassName () {...тело конструктора, список полей и методов...}`

b¹⁾ `var ClassName = function() {...тело конструктора, список полей и методов...}`

5. Если класс наследуется от другого, то необходимо указывать предка тегом `@extends`.

Документирование констант

1. Константы должны быть написаны заглавными буквами. Если название состоит из нескольких слов, они разделяются знаком подчеркивания.
2. Должен присутствовать тег `@constant`²
3. Должен присутствовать тег `@type`
4. Если константа используется какой-то одной или ограниченным набором функций, то должна быть указана одна или несколько ссылок на них посредством тега `@see` или `@link`.
5. Константа должна иметь описание. Описание может быть сокращено до минимального, если сделана ссылка на использующий константу метод и в его описании будет указано назначение констант. (В случае со списком констант это рекомендуемый способ описания).

Документирование переменных и полей класса

1. Должен быть указан тип переменной посредством префикса к имени. Список допустимых префиксов: `s` (String), `i` или `n` (Number), `a` (Array), `o` (Object), `b` (Boolean), `m` (mixed), `f` (function).
2. Тип *обязательно* должен быть указан посредством тега `@type`.
3. Если переменная приватная, надо указать тег `@private`.
4. Т.к. в некоторых случаях сборщики некорректно выделяют поля (например, когда полю присвоена функция), рекомендуется указывать тег `@field`.
5. Сборщики документации почему то не различают поля, если им не сделано присваивания, поэтому рекомендуется при описании любых полей делать присвоение значения по умолчанию.

¹ JSDoc 1.10.2 не поддерживает такой способ

² Поддерживается только JsDoc Toolkit

Документирование функций и методов класса

1. Как можно более полно описывать действия производимые методом.
2. В случае частного метода, использовать тег `@private`
3. Если метод имеет параметры, обязательно использовать теги `@param` для их описания. В этих тегах обязательно указывать тип аргумента.
4. Если метод возвращает значение, использовать тег `@return` с указанием типа возвращаемого значения.

Общие рекомендации

1. Использовать перекрестные ссылки в описаниях посредством тегов `@see` и `@link`
2. Для выделения констант использовать теги `<tt>`. Например, `<tt>true</tt>`. Аналогично для `false` и `null`.
3. Можно использовать и другие HTML-теги, если это делает описание более удобным для чтения. Но надо учитывать степень ухудшения читаемости документации в самих исходных файлах.
4. Последовательность тегов в общем случае такая:
 - Описание (возможно, включая теги `@link`)
 - Пустая строка
 - Специальные указательные теги (в такой последовательности): `@private`, `@class`, `@field`.
 - Указания типов и описания параметров (в такой последовательности): `@type`, `@param` (в порядке следования аргументов), `@return`
 - Дополнительные ссылки и автор(ы): `@author`, `@see`.
5. Не использовать `@author` везде кроме описания класса. В том числе для методов.

Список допустимых тегов и их синтаксис

`@class`

`@class` без параметров

Служит для указания класса.

`@extends`

`@extends` родительский класс

Служит для указания родительского класса.

`@param`

`@param {type} name description`

Типе – это указание типа. Обязательно в фигурных скобках. Допустимые типы (по аналогии с префиксами переменных): `String`, `Number`, `Array`, `Object`, `Bool`, `function` (с прописной буквы, т.к. не является классом). Также можно указывать любые другие имена классов, например,

Control. Если тип является смешанным, допускается использовать слово `mixed`, но если список поддерживаемых типов невелик, то следует указать все через вертикальную черту:

```
@param {String|Number} someParam description
```

Если передается массив какого-то определенного типа, то к этому типу надо добавить угловые скобки. Например, для массива строк тип будет `String[]`.

Если функция принимает сложный объект, то можно описать его поля³, например:

```
/**
 * @param userInfo Information about the user.
 * @param userInfo.name The name of the user.
 * @param userInfo.email The email of the user.
 */
function logIn(userInfo) {
  doLogIn(userInfo.name, userInfo.email);
}
```

Имя аргумента должно совпадать с именем аргумента. Если аргумент является необязательным он может быть взят в квадратные скобки⁴. При этом может быть указано значение по умолчанию.

Пример:

```
@param {String} [accessLevel="author"] The user accessLevel is optional with default value "author".
```

@link

@link ClassName#memberName

Служит для указания на (другой) класс и его метод/поле. Поле отделяется от имени класса символом `#`. JsDoc Toolkit поддерживает более широкий синтаксис ссылок (<http://www.jsdoctoolkit.org/wiki/?page=namepath>).

@see

@see ClassName#memberName

Аналогично @link.

@private

@private *без параметров*

Указывает на то, что класс, его метод или поле являются приватными.

@field

@field *без параметров*

Принудительно указывает на то, что *это* – поле.

@type

@type type

³ Поддерживается только JsDoc Toolkit

⁴ Поддерживается только JsDoc Toolkit

Type –тип переменной. См. список допустимых типов в описанию к тегу @param. Указывает без фигурных скобок.

@return

@return {type} description

Type – тип переменной. См. список допустимых типов в описанию к тегу @param. Обязан быть указан⁵.

@author

@author author name

Указывает автора. Имя автора – простой текст, так что допустимо использовать и теги, например, для создания ссылки на домашнюю страницу или электронный адрес.

```
@author <a href="mailto:micmath@gmail.com">Michael Mathews</a>
```

Особенности JSDoc и JsDoc Toolkit

Общие недостатки

1. Не понимают приватных полей, если отсутствует присваивание. В следующем коде поле someField вообще не попадет в документацию.

```
function MyClass()
{
  /**
   * Description for field.
   *
   * @private
   * @field
   * @type String
   */
  var someField;
}
```

Решение: надо обязательно делать присваивание какого-либо значения, даже null или undefined.

Второе решение (только для JsDoc Toolkit): принудительно заставить сборщик документации вставить имя, используя тег @name:

```
/** @constructor */
function MyClass()
{
  /**
   * Description for field.
   * @name MyClass#someField
   * @private
   * @field
   * @type String
   */
}
```

⁵ Не проверял, но в багнотах к JsDoc была просьба убрать обязательное указание типа в @return.

```
    */
    var someField;

}
```

Недостатки JSDoc

1. Не понимает тега `@see` в описании полей. Например в следующем коде:

```
/**
 * List of event listeners. The first index - an event name
 * (<tt>String</tt>), the 2nd - an array of that event listeners.
 *
 * @private
 * @type Array[]
 * @see Control#addEventListener
 * @see Control#removeEventListener
 */
var events = new Array();
```

Обе ссылки `@see` не попадут в документацию. Теги `@link` при этом исправно работают.

2. Не понимает класса заданного в виде: `var MyClass = function () {...}`

3. Не всегда корректно находит классы, даже задаваемые «нормально» (`function MyClass() {...}`). Необходимо принудительно использовать тег `@class`.

Недостатки JsDoc Toolkit

1. Плохой внешний вид документации. Альтернатив пока не существует, но шаблоны можно менять, надо лишь научиться.

2. Большое количество тегов. Есть все необходимые, но и много «лишних». Надо ограничивать себя в их использовании.