Javascript API文檔產生工具  
 JSDuck

# 目錄索引

[目錄索引 2](#_Toc480554436)

[前言 3](#_Toc480554437)

[JSDuck簡介 4](#_Toc480554438)

[其他工具比較 5](#_Toc480554439)

[安裝 8](#_Toc480554440)

[第一步：下載 RubyInstaller 8](#_Toc480554441)

[第二步：安裝 DevKit 8](#_Toc480554442)

[第三步：使用gem安裝 JSDuck 9](#_Toc480554443)

[JSDuck CMD指令 10](#_Toc480554444)

[JSDuck 讀取配置檔 13](#_Toc480554445)

[路徑配置： 13](#_Toc480554446)

[設置conf.json 13](#_Toc480554447)

[支援標籤 15](#_Toc480554448)

[常用標籤 16](#_Toc480554449)

[基礎 17](#_Toc480554450)

[@author 17](#_Toc480554451)

[@param 17](#_Toc480554452)

[@return (別名：@returns) 20](#_Toc480554453)

[@method 23](#_Toc480554454)

[進階 25](#_Toc480554455)

[@class 25](#_Toc480554456)

[@example 27](#_Toc480554457)

[@member 28](#_Toc480554458)

[@abstract 29](#_Toc480554459)

[@extends 30](#_Toc480554460)

[@property 30](#_Toc480554461)

[總結 32](#_Toc480554462)

[參考資料 33](#_Toc480554463)

# 前言

大多數時候我們都關注於應用層面的代碼可維護性，如：OO、模塊化、MVC，編碼規範、可擴展和復用性，但這都是屬於設計層面需要考慮的事情，可維護性還應包含另一個方面，也是很多coder容易忽略的地方，就是將自己寫的程式以文檔的形式沉澱起來，對自己工作有一個結構化的組織，也可以幫助到他人。

試想一下如下情況：  
1、團隊中加入了新的同學，這時他可能需要快速的將目前項目中現有程式有一個系統的瞭解，如：某個公共工具模塊的用途，某個通用組件的介面，程式之間的依賴性，這時他只有去看源碼和註釋了。  
2、團隊中有同學離開，但他寫的程式在此之前已經深入到項目的各個層面，最瞭解和能快速修改維護這些程式的人可能只有他，之後在反覆運算時遇上其環節，其他接手的同學只能望眼欲穿了。  
3、自己寫了一個不錯的可復用組件，打算推廣到團隊中，這時他人需要使用自己的組件時不得不去看源碼和註釋了。

這時候如果每個人都對自己程式有一個文檔化的闡釋，便可對上述問題做出很大的緩解，通常程式的文檔化需求只存在於大型項目或是擁有成熟體系的框架之中，對於前端們來講其實大多數時候都想用文檔來展示和沉澱自己的知識成果的，可一直缺乏一套行之有效快速一致性的解決方案，導致在大家談到程式文檔化的時候都因為時間關係，繁瑣程度就望而卻步了。

長期以來前端開發們都缺乏一個像樣的文檔化工具，雖然在這之前出現過 YUI DOC、JSDoc ，但這些工具始終難於將開發者從複雜的配置中解脫出來，從而使開發者很快便將它們遺忘。

如果有對sencha的 Extjs和 Sencha Touch 開發框架有過瞭解的同學想必都對為其定製的API文檔印象深刻，富應用形態的展現方式和樹節點的命名空間管理，  避免了 YUI DOC 和 jQeury API 那樣沉長頁面帶來的繁瑣，而文檔中附加的學習的範例也能幫助初學者儘快上手，生成這樣強大的 API 文檔使用的是jsduck，它不僅在使用和配置上非常簡單，文檔的 UI 和對話模式也是目前對於開發者來說是最友好的。

# JSDuck簡介

jsduck 是 Sencha眾多開源項目中的一個，它是使用 [ruby](http://www.ruby-lang.org/zh_cn/) 編寫的 javascript API 文檔生成器。

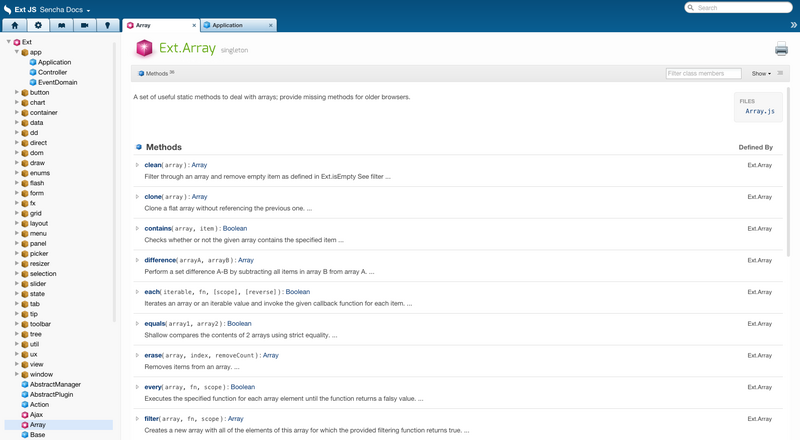
Jsduck強力功能點如下：

* 樹形類別命名空間組織
* 類別的子父關係的層次體系展示
* 成員與事件和配置項快速索引
* 可穿插著色代碼範例演示和編輯範例代碼
* 類別成員源碼實現部分快速導航

JSDuck也針對自家產品Extjs有極大的支援，在JSDuck github官網上你可以看到範例上有許多JSDuck針對Extjs框架的支援，透過自動偵測的機制，讓代碼能自動被歸類或很輕易的就能產生清楚的文檔結構。

但即使你不是使用extJS進行開發，JSDuck仍然是一個非常強大的工具，一方面它的語法非常靈活，描述支援Markdown語法，上手難度遠遠低於JSDoc 3；另一方面它生成的文檔可讀性堪比YUIDoc，同時支持源碼的對照，學習起來也非常的容易。

下面是JSDuck生成文檔的介面：



JSDuck在構造方法參數、屬性、方法都非常清晰的列在文檔之中。它的link也非常的好用，能夠準確的定位到不同模組中的內容。JSDuck同樣提供了全域搜索的方法，而且還在你敲下關鍵字的同時給你相關的提示。除了這些以外，JSDuck還提供了對於依賴(dependency)、以及查看源碼(View source)的方法。

如果硬要說JSDuck有什麼不好的地方，估計就是它把一切都Handle太完美了以致於沒有給你提供什麼可以自己定制的餘地。

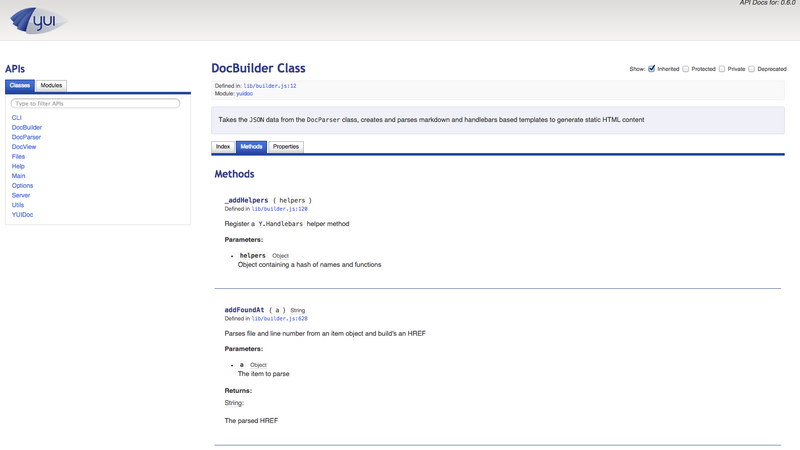
# 其他工具比較

可以快速流覽下面的介紹來快速的瞭解這些工具特性。

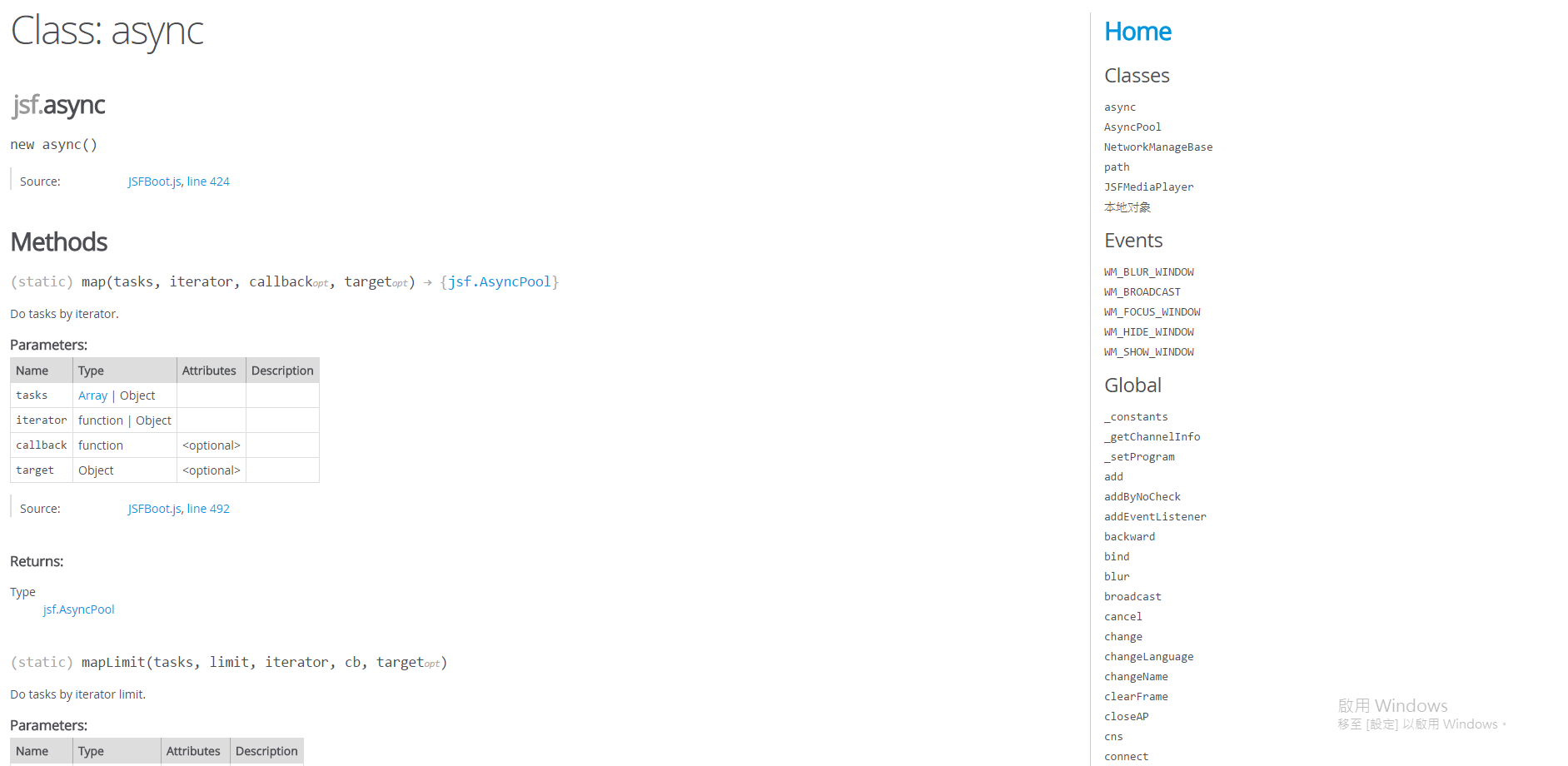
**YUIDoc**是YUI的附屬項目。YUI團隊希望這個文檔工具不僅僅可以支援Javascript，而是更多的指令碼語言，因此它並不考慮實際的代碼實施細節，而只是對注釋部分進行了掃描。從好的一面來說，如果你同時使用了Ruby/PHP/Python等等，YUI或許是一個比較適合的工具。從反面來說，因為它沒有考慮實施細節，你需要額外的變數聲明、同時生成的文檔有可能會和實際的實施細節不吻合。

YUIDoc提供了非常清晰的文檔格式。不僅物件內部的內容非常清晰，而且相互引用的link也工作的非常好。同時YUIDoc提供了全域搜索的功能，你可以容易根據關鍵字找到對應的內容。

由於YUIDoc是Yahoo下屬YUI專案的一個部分，它並不像JSDoc 3提供了那麼多可定制的功能。能夠修改的大概就只有Logo，基本的CSS樣式而已。



**JSDoc** 的前身是JSDoc Toolkit。它會對代碼的具體實施進行掃描，因此你如果不按照符合JSDoc的注釋語法來進行注釋的話，可能生成的文檔一個字也沒有。雖然它的學習曲線很高，但是一旦掌握了之後，由於它提供了完整的模版開發，事件觸發等等介面，其靈活性也是不容小覷的。



JSDoc預設的樣式非常非常的糟糕，不過JSDoc 3在自己的介紹頁面裡推薦了一個協力廠商開發的模版系統“Docstrap”。這個模版雖然比JSDoc 3的默認模版好上很多，但是與YUIDoc和JSDuck生成的內容相比就差強人意了。其中的link的錨點也會偶爾不能正常工作。此外，它並不支援全域對於變數的搜索，你可以在Docstrap Github的issue中找到他們相關的開發計畫。

JSDoc 3屬於完全開源的項目，因此如果你等不及社區的更新，你完全可以自己對JSDoc進行深度的開發。JSDoc對外提供的開發介面有3個：

* 模版Template
* 事件Event  
  其中最有意思的我認為是事件功能。JSDoc幾乎對載入檔、分析注釋和生成文檔的每一步都提供了事件的hook：parseBegin, fileBegin, beforeParse, jsdocCommentFound……通過這些事件，你甚至可以進一步定義自己需要的comment tag和解析規則。
* 外掛程式Plugins

更加詳細的內容則可以在[JSDoc的使用說明](http://usejsdoc.org/)上找到詳細的講解。

**Dox**是一個非常羽量級和可以定制化的工具，它使用和JSDoc 3相容的注釋語法標準，並將其轉化成JSON格式。因此在呈現方式上具有比JSDoc 3更強的可定制性，當然也意味著你初期的麻煩會比較多。

**Docco**是一種行間注釋的方式。比起API注釋，它更適合於注釋代碼的實施邏輯，以便於後期程式師能夠快速捕捉到原作者在此處的實施意圖。應該說是非常適合開源專案、多個作者共同維護的一個文檔工具。比如最經典的[Backbone Annotated Source](http://backbonejs.org/docs/backbone.html)就是使用Docco進行注釋的。

最後選擇了JSDuck作為文檔生成的工具。考慮到畢竟寫代碼才是我們的主要工作，注釋和文檔越簡單能夠表達意思越好用。但是選擇使用哪種工具還需要根據使用場景來具體考慮，沒有哪一個工具是最好用的工具，只有當下最適合你或你團隊的工具。

# 安裝

To install JSDuck through gem, you first need Ruby.

## 第一步：下載 RubyInstaller

一開始先去 [RubyInstaller](http://www.google.com/url?q=http%3A%2F%2Frubyinstaller.org%2Fdownloads%2F&sa=D&sntz=1&usg=AFQjCNErUhsL9fOlow5sN5Jl6BzPVAsh-Q) 官網下載 windows 專門的安裝應用程式。

* 執行安裝程式。  
  (除了一直按 Next 之外，記得安裝途中把 **Add Ruby executables to your PATH** 勾選起來。)
* 安裝完後，在 cmd 上測試**ruby -v**
* 成功安裝會出現 ruby XXX 版本號

恭喜 Ruby 已經成功安裝在你的電腦了。

You also need to install the **Development Kit** to be able to build native extensions - download it from the same site, then follow the [installation instructions](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2Foneclick%2Frubyinstaller%2Fwiki%2FDevelopment-Kit&sa=D&sntz=1&usg=AFQjCNEyAXyMhwLpXOF1kixl269U6UggXg) (be careful when using the self-extracting exe - if you don't give it a new directory name, it will unpack all files to the current directory, which you probably don't want).

## 第二步：安裝 DevKit

為了安裝擴充，必須要有個輔助 toolkit ，這就是 [DevKit](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2Foneclick%2Frubyinstaller%2Fwiki%2Fdevelopment-kit&sa=D&sntz=1&usg=AFQjCNEaqAp0kTpJTyqJeTB3qVxyPFEDLA)。

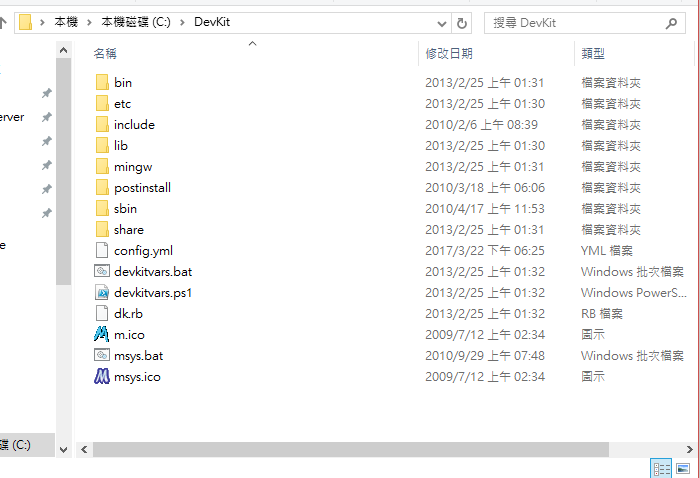
一樣先在 [RubyInstaller](http://www.google.com/url?q=http%3A%2F%2Frubyinstaller.org%2Fdownloads%2F&sa=D&sntz=1&usg=AFQjCNErUhsL9fOlow5sN5Jl6BzPVAsh-Q) ，左側偏下的地方下載符合你剛剛安裝 Ruby 版本的 DevKit 。

* 解壓縮到你認得的地方 (像我是放在 C:\DevKit )。
* 然後開啟 cmd 移到剛剛解壓縮的地方，下指令

> ruby dk.rb init   
(這是為了偵測你安裝了哪些 Ruby ，然後會產生 config.yml 這個檔，你可以修改這個檔選擇你要用的 Ruby 版本。如果沒有偵測到 Ruby 的話，可以直接加上 Ruby 安裝路徑，例如： C:\Ruby23-x64)

* 確定好 config.yml 之後，下指令 ruby dk.rb install

這時 DevKit就成功安裝完成了。



## 第三步：使用gem安裝 JSDuck

using the standard gem to install JSDuck:

> gem install jsduck

成功安裝完畢之後可以使用 jsuck –version 來看確認安裝的版本



那麼JSDuck已成功安裝在你的電腦了。

# JSDuck CMD指令

使用下方指令查詢jsduck指令支援的參數與使用方式：

> jsduck --help

C:\Users\chenc\Desktop\JSDuck>jsduck --help

Usage: jsduck [options] files/dirs...

For example:

# Documentation for builtin JavaScript classes like Array and String

jsduck --output output/dir --builtin-classes

# Documentation for your own JavaScript

jsduck --output output/dir input-file.js some/input/dir

The main options:

-o, --output=PATH Directory to write all this documentation.

--export=full/examples Exports docs in JSON.

--builtin-classes Includes docs for JavaScript builtins.

--seo Enables SEO-friendly print version.

--config=PATH Loads config options from JSON file.

--encoding=NAME Input encoding (defaults to UTF-8).

--exclude=PATH Exclude input file or directory.

Customizing output:

--title=TEXT Custom title text for the documentation.

--footer=TEXT Custom footer text for the documentation.

--head-html=HTML HTML for the <head> section of index.html.

--body-html=HTML HTML for the <body> section of index.html.

--css=CSS Extra CSS rules to include to the page.

--message=HTML (Warning) message to show prominently.

--welcome=PATH File with content for welcome page.

--guides=PATH JSON file describing the guides.

--videos=PATH JSON file describing the videos.

--examples=PATH JSON file describing the examples.

--categories=PATH JSON file defining categories for classes.

--no-source Turns off the output of source files.

--images=PATH Path for images referenced by {@img} tag.

--tests Creates page for testing inline examples.

--import=VERSION:PATH Imports docs generating @since & @new.

--new-since=VERSION Since when to label items with @new tag.

--comments-url=URL Address of comments server.

--comments-domain=STRING A name identifying the subset of comments.

--search-url=URL Address of guides search server.

--search-domain=STRING A name identifying the subset to search from.

Tweaking:

--tags=PATH Path to custom tag implementations.

--ignore-global Turns off the creation of 'global' class.

--external=Foo,Bar,Baz Declares list of external classes.

--[no-]ext4-events Forces Ext4 options param on events.

--examples-base-url=URL Base URL for examples with relative URL-s.

--link=TPL HTML template for replacing {@link}.

--img=TPL HTML template for replacing {@img}.

--eg-iframe=PATH HTML file used to display inline examples.

--ext-namespaces=Ext,Foo Additional Ext JS namespaces to recognize.

--touch-examples-ui Turns on phone/tablet UI for examples.

--ignore-html=TAG1,TAG2 Ignore a particular unclosed HTML tag.

Performance:

-p, --processes=COUNT The number of parallel processes to use.

--[no-]cache Turns parser cache on/off (EXPERIMENTAL).

--cache-dir=PATH Directory where to cache the parsed source.

Debugging:

-v, --verbose Turns on excessive logging.

--warnings=+A,-B,+C Turns warnings selectively on/off.

--warnings-exit-nonzero Exits with non-zero exit code on warnings.

--[no-]color Turn on/off colorized terminal output.

--pretty-json Turns on pretty-printing of JSON.

--template=PATH Dir containing the UI template files.

--template-links Creates symlinks to UI template files.

-d, --debug Same as --template=template --template-links.

--extjs-path=PATH Path for main ExtJS JavaScript file.

--local-storage-db=NAME Prefix for LocalStorage database names.

-h, --help[=--some-option] Use --help=--option for help on option.

--version Prints JSDuck version

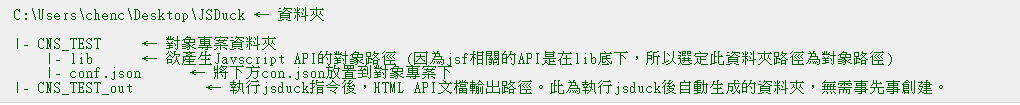
# JSDuck 讀取配置檔

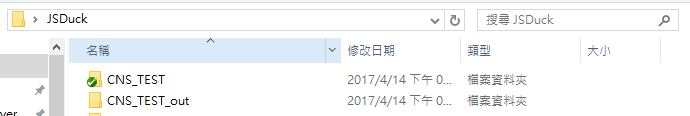
除了上面的指另外，當然我們也可以很簡單的透過jsduck --config來讀取設置好的conf.json配置檔，來避免需要在cmd上輸入過多的參數。

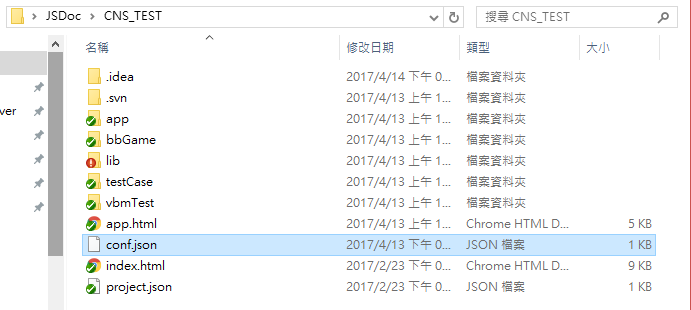
$ jsduck --config=conf.json

下面我們先用實際的例子來進行教學，讓jsduck可以透過讀取conf.json來輸出我們的HTML API文檔。

## 路徑配置：



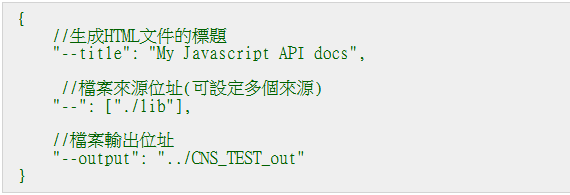




1. CNS\_TEST - 對象專案資料夾
2. lib - 欲產生Javscript API的對象路徑 (因為jsf相關的API是在lib底下，所以選定此資料夾路徑為對象路徑)
3. conf.json - 提供jsduck讀取的配置檔。

## 設置conf.json

建立一個conf.json檔，並將該檔案放置到對象專案底下：

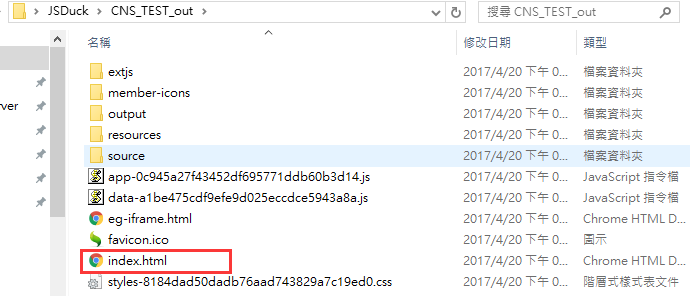


※補充說明： ./ 為當前目錄，../ 為上一層目錄。

接著在JSDuck資料夾路徑下輸入此指令

C:\Users\chenc\Desktop\JSDuck> jsduck --config=.\CNS\_TEST\conf.json --verbose

等到執行完畢後，之後就可以在CNS\_TEST\_out資料夾底下看到生成出來的HTML API文檔。(index.html)



※補充：上面的路徑配置與conf.json可以依照個人需求進行修改。

在執行指令過程中他會列出所有parsing的對象，並一一地將它們生成HTML的頁面，如果輸入命令後卻沒有成功產生API文檔頁面的話，可以透過CMD介面上的錯誤訊息去排查來源，排除後再嘗試重新輸入指令即可。

到這邊讀者應該能很輕易的駕馭這JSDuck這個工具了，接下來我們將針對JSDuck的標籤進行解說，畢竟要生成可讀性高的文檔，我們必須先學習如何撰寫正確的JSDuck標籤和註解方式。

# 支援標籤

下面是所有JSDuck內建支援的標籤，有部分的標籤已不建議使用，有部分標籤則是為了支援ext-doc，

* [@abstract](https://github.com/senchalabs/jsduck/wiki/%40abstract) //定義一個class或member為抽象類型
* [@accessor](https://github.com/senchalabs/jsduck/wiki/%40accessor)
* [@alias](https://github.com/senchalabs/jsduck/wiki/%40alias)
* [@alternateClassName](https://github.com/senchalabs/jsduck/wiki/%40alternateClassName)
* [@aside](https://github.com/senchalabs/jsduck/wiki/%40aside)
* [@author](https://github.com/senchalabs/jsduck/wiki/%40author) //代碼作者
* [@cfg](https://github.com/senchalabs/jsduck/wiki/%40cfg) //為構造器的配置項，並在其後跟隨“{className}”，再跟隨參數名。
* [@chainable](https://github.com/senchalabs/jsduck/wiki/%40chainable) //如果回傳值為this，則使用chainable的標籤
* [@class](https://github.com/senchalabs/jsduck/wiki/%40class) 標記一個類別
* [~~@constructor~~](https://github.com/senchalabs/jsduck/wiki/%40constructor) // \*\* 已棄用，為了相容ext-doc，新的代碼建議不使用。
* [@deprecated](https://github.com/senchalabs/jsduck/wiki/%40deprecated) 標記此成員或類別已被棄用，在升級過渡的時候需相容之前的API時特別有用
* [@docauthor](https://github.com/senchalabs/jsduck/wiki/%40docauthor) 基本上跟@author的作用相同，標記誰是這個類別或成員檔的作者。
* [@enum](https://github.com/senchalabs/jsduck/wiki/%40enum) 被標記的對象會以類似類別的方式呈現在文檔的左側。且其所有列舉值都會以特性來呈現。
* [@event](https://github.com/senchalabs/jsduck/wiki/%40event) 標記一個事件，隨後通常會跟隨@param標籤作為事件的參數說明。
* [@evented](https://github.com/senchalabs/jsduck/wiki/%40evented)
* [@example](https://github.com/senchalabs/jsduck/wiki/Inline-examples) 給類別或者方法添加一個代碼範例，jsduck不僅會給代碼著色，還能給代碼生成一個代碼編輯器，編輯代碼後可即時預覽，使用@example需要四個空格的縮進。
* [@experimental](https://github.com/senchalabs/jsduck/wiki/%40experimental) 標記此成員或類別為實驗性質，未來可能會移除。
* [@extends](https://github.com/senchalabs/jsduck/wiki/%40extends) 標記一個類繼承自另一個類，生成後會有一個類型繼承體系陳列在文檔視圖的右側。
* [@fires](https://github.com/senchalabs/jsduck/wiki/%40fires) (in 5.x beta) 用來標駐該函式會觸發的事件
* [@ftype](https://github.com/senchalabs/jsduck/wiki/%40ftype)
* [@hide](https://github.com/senchalabs/jsduck/wiki/%40hide)
* [@ignore](https://github.com/senchalabs/jsduck/wiki/%40ignore) 忽略一個類別或成員，如果用在@class上，所有成員都會被移除不會顯示在文件上。
* [@inheritable](https://github.com/senchalabs/jsduck/wiki/%40inheritable) 與@static標籤一起使用，用來標示被子類別繼承的靜態成員。
* [@inheritdoc](https://github.com/senchalabs/jsduck/wiki/%40inheritdoc)
* [@localdoc](https://github.com/senchalabs/jsduck/wiki/%40localdoc) (in 5.x beta)
* [@markdown](https://github.com/senchalabs/jsduck/wiki/%40markdown) 此標籤是為了向相容所添加，但基本上可以完全忽略不使用。
* [@member](https://github.com/senchalabs/jsduck/wiki/%40member) 標註此成員是屬於哪個類別
* [@method](https://github.com/senchalabs/jsduck/wiki/%40method) 標記一個對象為函式
* [@mixins](https://github.com/senchalabs/jsduck/wiki/%40mixins)
* [@new](https://github.com/senchalabs/jsduck/wiki/%40new) 用來標記一個類別或成員是一近期新增加的，文檔上會顯示一顆星號的標記。
* [@override](https://github.com/senchalabs/jsduck/wiki/%40override)
* [@param](https://github.com/senchalabs/jsduck/wiki/%40param) 與@cfg類似，給一個函數成員標記其所需的參數類型和描述
* [@preventable](https://github.com/senchalabs/jsduck/wiki/%40preventable)
* [@private](https://github.com/senchalabs/jsduck/wiki/%40private) 將成員標記成私有，雖然也有@public但如果不特殊標明即為公有。
* [@property](https://github.com/senchalabs/jsduck/wiki/%40property) 標註特性的名稱、類型、預設值
* [@protected](https://github.com/senchalabs/jsduck/wiki/%40protected) 將成員標記成受保護的。
* [@ptype](https://github.com/senchalabs/jsduck/wiki/%40ptype)
* [@readonly](https://github.com/senchalabs/jsduck/wiki/%40readonly) 標註一個特性為只可讀取，不應該被覆值。
* [@removed](https://github.com/senchalabs/jsduck/wiki/%40removed) 標註一個類別或成員已被移除，並說明是從某哪個版本之後被移除。
* [@requires](https://github.com/senchalabs/jsduck/wiki/%40requires) 用來標住對象會用到哪些其他類別。
* [@return](https://github.com/senchalabs/jsduck/wiki/%40return) 與@cfg類似，標記一個函數成員調用過後的返回類型。
* [@scss mixin](https://github.com/senchalabs/jsduck/wiki/%40scss-mixin)
* [@since](https://github.com/senchalabs/jsduck/wiki/%40since) 標記一個類別或成員從哪個版本之後開始提供
* [@singleton](https://github.com/senchalabs/jsduck/wiki/%40singleton)
* [@static](https://github.com/senchalabs/jsduck/wiki/%40static) 將成員標記成靜態的，靜態成員也會在文檔中進行分類展示。
* [@template](https://github.com/senchalabs/jsduck/wiki/%40template)
* [@throws](https://github.com/senchalabs/jsduck/wiki/%40throws) 用來描述一個函數可能會拋出的例外
* [~~@type~~](https://github.com/senchalabs/jsduck/wiki/%40type) // \*\* 已棄用，為了相容ext-doc，新的代碼建議不使用，請改用@property。
* [@uses](https://github.com/senchalabs/jsduck/wiki/%40uses) 看起來跟@requires是差不多的...
* [@var](https://github.com/senchalabs/jsduck/wiki/%40var) 用來標註SCSS參數的名稱、類型、預設值
* [@xtype](https://github.com/senchalabs/jsduck/wiki/%40xtype)

內置標籤

* [{@link}](https://github.com/senchalabs/jsduck/wiki/Guide#wiki-cross-references) 在文檔注釋中標記某個類或類成員的錨點。
* [{@img}](https://github.com/senchalabs/jsduck/wiki/Guide#wiki-images) 在文檔注釋中連結一張圖片，讓文檔變得更具可讀性。
* [{@video}](https://github.com/senchalabs/jsduck/wiki/Guide#wiki-videos)

Various tags use {TypeDefinitions}, the syntax for these is described here:

* [Type Definitions](https://github.com/senchalabs/jsduck/wiki/Type-Definitions)

# 常用標籤

由於JSDuck在設計上，本身很多的標籤是對於自家ExtJS框架的支援，且有部分的標籤是為了相容之前的ext-doc的標籤，所以相對來說標籤的種類會有很多種，但一般情況下，我們只會使用較常見的標籤，所以這邊針對比較常見又重要的標籤進行教學與範例演示。此外，你可以直接在 <https://github.com/senchalabs/jsduck/wiki/Guide>頁面上快速地瞭解簡易的用法。

## 基礎

### @author

@author標籤用來標註類別或成員的撰寫者。

在預設情況下，JSDuck會自動隱藏author的訊息。但如果你希望它顯示出來，必須自己另外撰寫客製化的@author tag來覆寫它(但必須額外撰寫ruby程式)。

**語法：**

@author Some name...

**例子：**

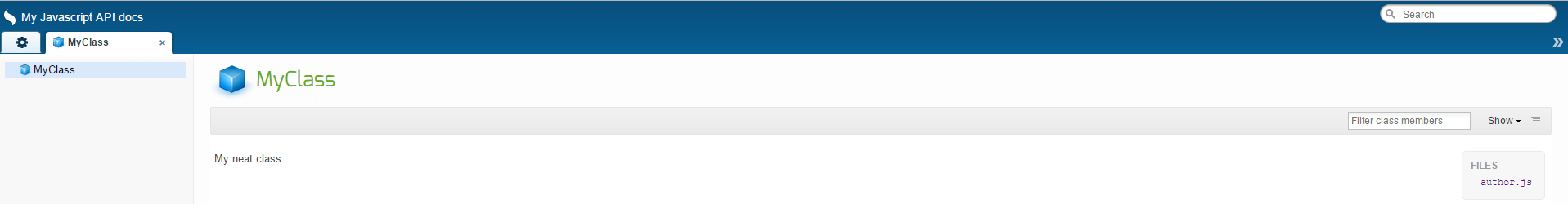
/\*\*

\* @class MyClass

\* My neat class.

\* @author John Doe <john@example.com>

\*/



### @param

@param標籤用來標示@method或@events的參數。

它除了可以針對參數的Type作定義，也可以標註那些參數是選擇性添加的，且可定義它的預設值[name=”default-value”]。

**語法：**

@param name

Some description...

@param {Type} name

Some description...

@param {Type} [name]

Some description...

@param {Type} [name="default-value"]

Some description...

@param {Type} name.subproperty

Some description...

**例子：**

/\*\*

\* Joins array of strings into one.

\* @param {String[]} pieces The list of strings to concatenate.

\* @param {Array} [glue=", "] Separator to inject between strings.

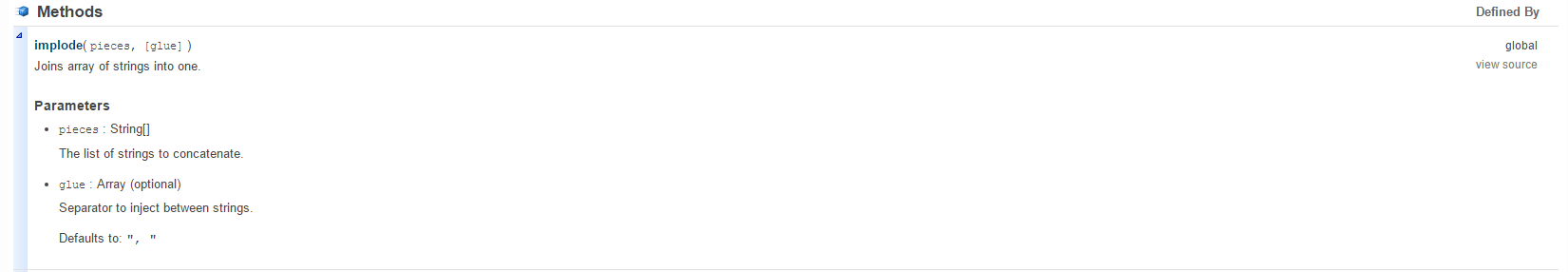
\*/

function implode(pieces, glue) {

glue = glue || ", ";

return pieces.join(glue);

}



如果參數本身還帶有特性的話，我們可以直接在@parm後立即定義它們。

/\*\*

\* @param user A user record

\* @param user.name The name of the user.

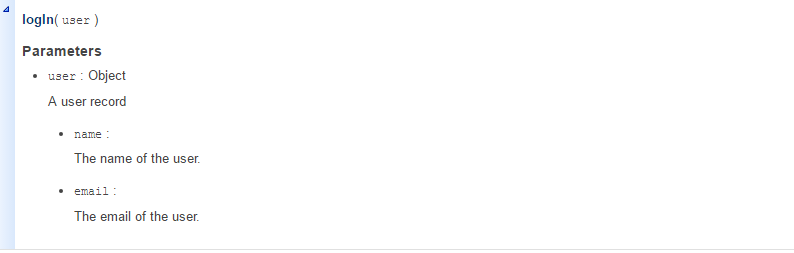
\* @param user.email The email of the user.

\*/

function logIn(user) {

doLogIn(user.name, user.email);

}



如果參數是一個函式，你可以透過撰寫類似物件特性的方法來完成。(**return**特性在文檔中算是回傳值的一個保留字)

/\*\*

\* Filters elements in array with given function.

\* @param {Function} fn The callback function.

\* @param {Mixed} fn.value Value of the element in iteration.

\* @param {Number} fn.index Index of the element in iteration.

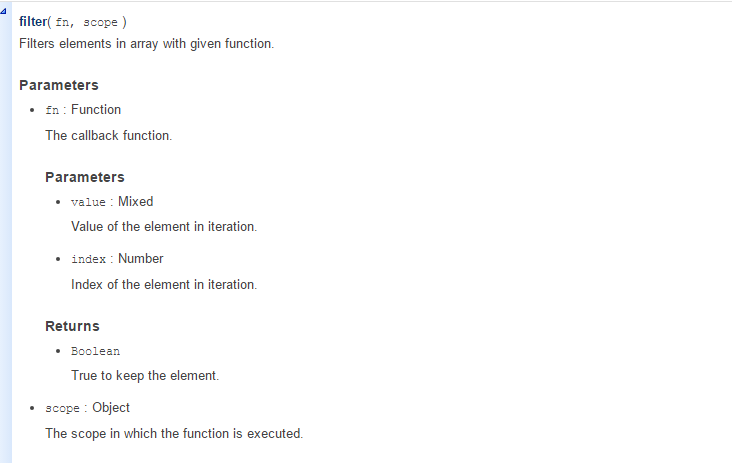
\* @param {Boolean} fn.**return** True to keep the element.

\* @param {Object} scope The scope in which the function is executed.

\*/

function filter(fn, scope) {

}



### @return (別名：@returns)

@return 標籤用來標註一個method的回傳值。

**語法：**

@return {Type}

Some description...

@return {Type} return.subproperty

Some description...

**例子：**

如果你要回傳物件時，請參照已下的撰寫方式

/\*\*

\* @return {Object} A user record

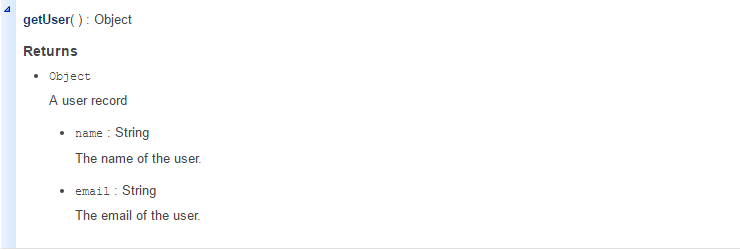
\* @return {String} return.name The name of the user.

\* @return {String} return.email The email of the user.

\*/

function getUser() {

}



如果你是要回傳this的話，你必須改使用@chainable。

/\*\*

\* @class MyComponent

\*/

var MyComponent = {

/\*\*

\* Shows the component

\* @chainable

\*/

showComponet: function() {

return this.setVisible(true);

},

/\*\*

\* Highlights the component temporarily with yellow color.

\* @chainable

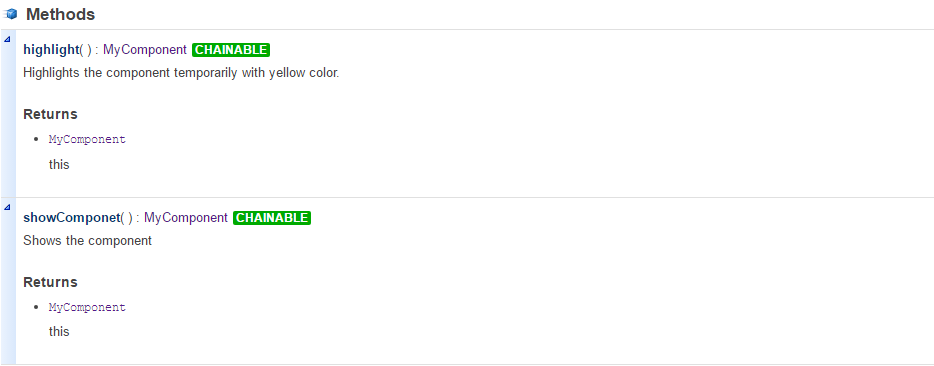
\*/

highlight: function() {

return this.setBackground("yellow").wait(1000).clear();

}

}



如果你要回傳 fucntion時，所回傳的function的參數與回傳值也可以被文檔化。

/\*\*

\* @return {Function} A predicate function that

\* determines is number is odd or even.

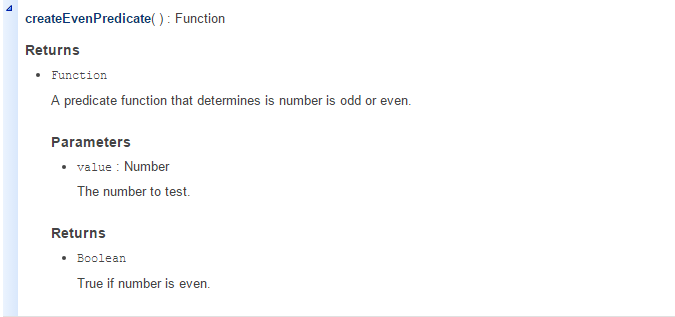
\* @return {Number} return.value The number to test.

\* @return {Boolean} return.return True if number is even.

\*/

function createEvenPredicate() {

}



※這邊return與@param用法類似但容易混淆，可以回去@param比較看看，或者實際使用JSDuck執行看看不同寫法所產生出來的API文檔是否有差異。

### @method

@method標籤用來標註方法的名稱。

當然你也不添加此標籤讓JSDuck本身的自動偵測機制讓它自動識別並產生。

**語法：**

@method

Documentation for the method.

@method name

Documentation for the method.

**例子：**

/\*\*

\* @method area

\* Returns area of a circle

\* @param {Number} r Radius of the circle.

\* @return {Number} The area

\*/

下面的撰寫方式與上面的撰寫方式，文檔所產生出來的結果。

/\*\*

\* Returns area of a circle

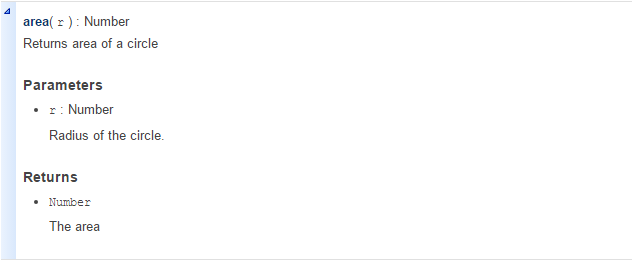
\* @param {Number} r Radius of the circle.

\* @return {Number} The area

\*/

function area(r) {

}



## 進階

### @class

@class 標籤用來標註類別，透過此標籤來標註類別，也可以指定一個類別名稱。

**語法：**

@class [<ClassName>]

**例子：**

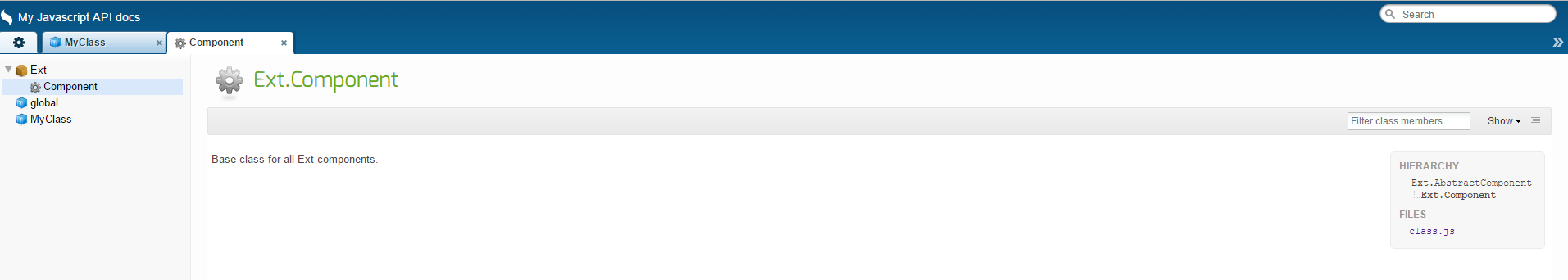
/\*\*

\* @class Ext.Component

\* @extends Ext.AbstractComponent

\* Base class for all Ext components.

\*/



另外，JSDuck 本身有提供自動偵測的機制，針對Ext.define關鍵字(Extjs專案)會自動識別，此外，function宣告若使以大寫開頭，則也會自動被歸類成一個類別，這邊就不詳細討論。

/\*\*

\* A simple class.

\*/

function Component() {

}



透過@class標籤我們可以讓JSDuck將對象識別為一個Class對象。

/\*\*

\* @class

\* I'm a class now.

\*/

Ext.supports = {};

**@constructor tag inside class**

This use is deprecated.(已棄用)，See [@constructor](https://github.com/senchalabs/jsduck/wiki/@constructor) for details.

**@cfg tags inside class**

用來支援之前ext-doc的標籤語法。

**下面列出幾個用在類別內的標籤(可暫時跳過，不常使用)：**

* [@alternateClassName](https://github.com/senchalabs/jsduck/wiki/@alternateClassName)
* [@mixins](https://github.com/senchalabs/jsduck/wiki/@mixins)
* [@requires](https://github.com/senchalabs/jsduck/wiki/@requires)
* [@uses](https://github.com/senchalabs/jsduck/wiki/@uses)
* [@singleton](https://github.com/senchalabs/jsduck/wiki/@singleton)
* [@aside](https://github.com/senchalabs/jsduck/wiki/@aside)
* [@alias](https://github.com/senchalabs/jsduck/wiki/@alias)
* [@ftype](https://github.com/senchalabs/jsduck/wiki/@ftype)
* [@ptype](https://github.com/senchalabs/jsduck/wiki/@ptype)
* [@xtype](https://github.com/senchalabs/jsduck/wiki/@xtype)
* [@override](https://github.com/senchalabs/jsduck/wiki/@override)

Also the following tags work in class context:

* [@private](https://github.com/senchalabs/jsduck/wiki/@private)
* [@ignore](https://github.com/senchalabs/jsduck/wiki/@ignore)
* [@deprecated](https://github.com/senchalabs/jsduck/wiki/@deprecated)
* [@removed](https://github.com/senchalabs/jsduck/wiki/@removed)
* [@author](https://github.com/senchalabs/jsduck/wiki/@author)
* [@docauthor](https://github.com/senchalabs/jsduck/wiki/@docauthor)
* [@inheritdoc](https://github.com/senchalabs/jsduck/wiki/@inheritdoc)
* [@abstract](https://github.com/senchalabs/jsduck/wiki/@abstract)

**關於類別中的靜態成員：**

所有在類別中定義的成員，基本上預設都會被歸類為實體(instance)成員，但類別中的方法和特性有時候也可以是靜態成員，我們可以透過添加**@stati**c標籤來讓類別中的成員被標註為靜態。

**類別中的成員的狀態：**

* [@private](https://github.com/senchalabs/jsduck/wiki/@private) - member which may only be accessed from the the class itself.
* [@protected](https://github.com/senchalabs/jsduck/wiki/@protected) - member which may only be accessed from the class itself and its subclasses.
* [@abstract](https://github.com/senchalabs/jsduck/wiki/@abstract) - an implementation for the member must be provided in subclasses.
* [@deprecated](https://github.com/senchalabs/jsduck/wiki/@deprecated) - member is supported for backwards compatibility and may be removed in future versions.
* [@removed](https://github.com/senchalabs/jsduck/wiki/@removed) - member existed in earlier version but has been removed since. Useful for giving directions to users upgrading from older version.
* [@hide](https://github.com/senchalabs/jsduck/wiki/@hide) - hides a member of a parent class. Useful if child class doesn't support everything parent supports. (One should generally keep the Liskov substitution principle in mind though.)
* [@ignore](https://github.com/senchalabs/jsduck/wiki/@ignore) - removes the member completely from documentation. As if the doc-comment never was there.

### @example

@example 標籤讓你可以在註解當中標註一塊用來演示代碼如何使用的區塊。

**語法：**

@example ClassName

**例子：**

/\*\*

\* @class example

\* See the example:

\*

\* @example

\* Ext.create('Ext.Button', {

\* text: 'Click me',

\* renderTo: Ext.getBody()

\* });

\*/



### @member

@member 標籤用來標註該成員是屬於哪個類別。

**語法：**

@member ClassName

**例子：**

/\*\*

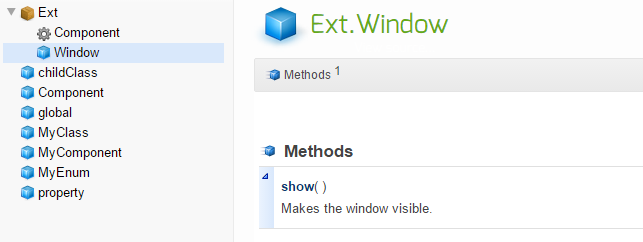
\* @member Ext.Window

\* Makes the window visible.

\*/

function show() {

}



### @abstract

@abstract 標籤用來標註抽象的類別或成員，且其成員必須被子類別所實作。

**語法：**

@abstract

**例子：**

/\*\*

\* Shows the component

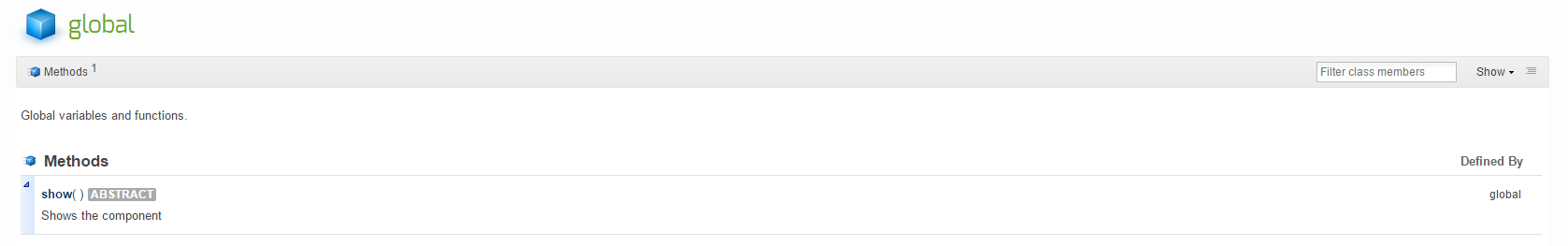
\* @abstract

\*/

function show(){

alert("not implemented");

}



### @extends

@extends 標籤用來標註這個類別是繼承自哪個父類別。

**語法：**

@extends ParentClassName

**例子：**

/\*\*

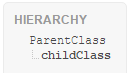
\* @class childClass

\* @extends ParentClass

\*/

function childClass (){

}



### @property

@property 標籤用來標註特性的名稱、類型、預設值。

**語法：**

@property

Documentation for the property.

@property name

Documentation for the property.

@property {Type} name

Documentation for the property.

@property {Type} [name="default value"]

Documentation for the property.

@property {Type} name.subproperty

Documentation for the subproperty.

**例子：**

這邊的特性名稱是hidden；類型為Boolean；預設值為false。

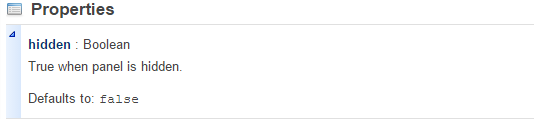
/\*\*

\* @property {Boolean} [hidden=false]

\* True when panel is hidden.

\*/

Ext.Panel.prototype.hidden = false;



如果你是要標註物件的特性，其實用法與@param的語法相同，範例如下：

var property = {

/\*\*

\* @property user A user record

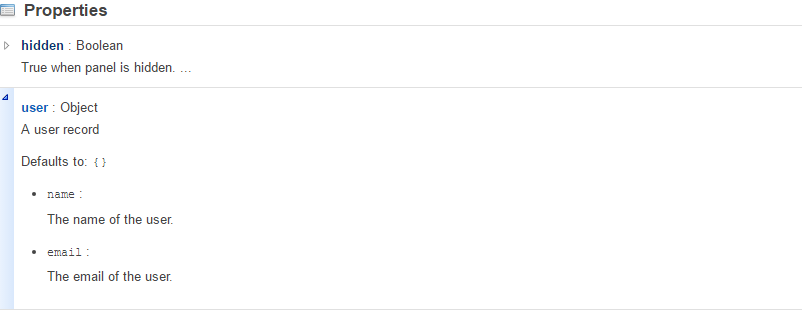
\* @property user.name The name of the user.

\* @property user.email The email of the user.

\*/

user: {}

}



# 總結

JSDuck官網中提供了許多標籤，但這邊不建議讀者一一的去把標籤的使用方法和內容全部瞭解完，而是建議從常用的標籤開始著手，熟悉了常用的標籤後直接嘗試將標籤加入專案當中，並起直接使用JSDuck工具將文檔產生出來，最後再根據生產出來的內容作調整或修正。或者可以根據項目上的需求與團隊成員的共識來訂定出一致的註解撰寫風格，後續有多餘的時間可以在瞭解其他的標籤用法來讓文檔的內容更豐富。

文檔化你的項目不僅可以讓前端工程師們將自己寫的注釋變更具有價值，也可以為項目後期維護帶來巨大便捷，在協同作戰環境下起著至關重要的作用。文檔化的最終目的還是服務團隊中的每一個成員，只要團隊中的成員能夠快速從文檔中瞭解目前項目中現有程式碼，那麼它就是一個好的文檔，一個好的工具。

# 參考資料

1. JSDuck github wiki <https://github.com/senchalabs/jsduck/wiki>
2. <https://zack7498.gitbooks.io/jsduck/content/>
3. <http://blog.720ui.com/2015/js_auto_01_doc/>
4. Javascript自動化文檔工具：YUI Doc, JSDoc 3, JSDuck等比較<https://segmentfault.com/a/1190000002579067>