
Table of Contents

Greasemonkey	1.1
1.1 Greasemonkey 是什么？	1.2
1.2 安装 Greasemonkey	1.3
2.2. 用元数据描述您的用户脚本	1.4
6.1 存取持久数据	1.5
3. API 编程接口	1.6

Greasemonkey

深入浅出 Greasemonkey

版权 © 2005 Mark Pilgrim[sebug paper](#)

这本书、及其样例代码和视频文件都是自由软件。在“GNU 通用公共许可证(自由软件基金会)(版本2以及更新版本)”许可下，您可以随意的再分发和/或修改它们。我们发行这本书、及其样例代码和视频文件，希望它能对您有所帮助。但是我们并没有提供任何担保！请查阅

GNU 通用公共许可证

获取更多细节。

目录

- 1. 开始
 - 1.1. Greasemonkey 是什么？
 - 1.2. 安装 Greasemonkey
 - 1.3. 安装用户脚本
 - 1.4. 管理用户脚本
- 2. 您的第一个用户脚本
 - 2.1. Hello World
 - 2.2. 用元数据描述您的用户脚本
 - 2.3. 编写用户脚本代码
 - 2.4. 修改用户脚本
- 3. 调试用户脚本
 - 3.1. 用错误控制台追踪错误
 - 3.2. 用 `GM_log` 记日志
 - 3.3. 用 DOM 查看器查看元素
 - 3.4. 用 Javascript Shell 计算表达式
 - 3.5. 其他调试工具

- 4. 公共模式
 - 4.1. 在域名以及它所有子域名上执行用户脚本
 - 4.2. 测试 Greasemonkey 函数是否有效
 - 4.3. 测试页面中是否有

Illegal HTML tag removed : HTML

元素

- 4.4. 操作所有

Illegal HTML tag removed : HTML

元素

- 4.5. 操作特定

Illegal HTML tag removed : HTML

元素的所有实例

- 4.6. 操作所有有特定属性的元素
- 4.7. 在元素前插入内容
- 4.8. 在元素后插入内容
- 4.9. 删除元素
- 4.10. 替换元素为新内容
- 4.11. 快速插入复杂的

Illegal HTML tag removed : HTML

- 4.12. 在没有服务器的情况下添加图片
- 4.13. 添加 CSS 样式
- 4.14. 获取元素样式
- 4.15. 设置元素样式
- 4.16. 处理已渲染的页面
- 4.17. 匹配大小写无关的属性值
- 4.18. 获取当前域名
- 4.19. 改写链接

- 4.20. 重定向页面
- 4.21. 俘获用户点击
- 4.22. 覆盖内建的 Javascript 方法
- 4.23. 解析

Illegal HTML tag removed : XML

- 5. 实例教学
 - 5.1. 案例：GMail Secure
 - 5.2. 案例：Bloglines Autoload
 - 5.3. 案例：Ain't It Readable
 - 5.4. 案例：Offsite Blank
 - 5.5. 案例：Dumb Quotes
 - 5.6. 案例：Frownies
 - 5.7. 案例：Zoom Textarea
 - 5.8. 案例：Access Bar
- 6. 高级话题
 - 6.1. 存取持久数据
 - 6.2. 在菜单栏上添加菜单项
 - 6.3. 整合其他网站的数据
 - 6.4. 把您的用户脚本编译为扩展
- Greasemonkey API 参考
 - GM_log
 - 记录日志到错误控制台
 - GM_getValue
 - 读取脚本专用的配置值
 - GM_setValue
 - 设置脚本专用的配置值
 - GM_registerMenuCommand
 - 在 用户脚本命令 (C) 子菜单中添加菜单项

- GM_xmlHttpRequest
 - 进行任意的 HTTP 请求
- GM_openInTab
 - 在新标签中打开指定的 URL
- GM_addStyle
 - 给页面添加 CSS 样式
- “参考资料”链接清单
- 技巧清单
- 实例清单
- 步骤清单
- 修订历史
- 关于本书
- GNU 通用公共许可证
 - 1. 序言
 - 2. 有关复制，发布和修改的条款和条件
 - 3. 如何将这些条款用到你的新程序

第 1 章 开始

1.3. 安装用户脚本

Greasemonkey “用户脚本”是用 Javascript 编写的独立文件，用来定制一个或多个网页。

|



您可以在[Greasemonkey脚本库](#)，找到许多用户脚本。尽管没人要求您必须把脚本放到那儿去，但是实际上，您可以把您的脚本共享到任何地方，这样别人就可以安装它了。甚至您根本不需要一台网络服务器，因为你可以从本地文件中安装用户脚本。

|



用户脚本的文件名必须以 `.user.js` 结尾。

我写的第一个用户脚本叫做“Butler”。它增强了Google的搜索结果的功能。

过程 1.2. 安装 Butler 用户脚本

1. 访问[Butler 的主页](#)，可看到有关 Butler 的功能简述。(并不是所有的用户脚本都有主页，Greasemonkey 只要有用户脚本就够了。)
2. 点击“Download version...”链接 (0.3截至发稿时)。
3. 弹出一个标题为“Greasemonkey 安装过程”的对话框，其中显示了将要安装的用户脚本名称，简介以及作用与排除的页面列表。所有这些信息都包含在脚本之中；以后您会在用元数据描述您的用户脚本学到定义的方法。
4. 点击确定安装用户脚本。

没有意外的话，Greasemonkey 会在状态栏滑出一个提示，“‘Butler’ 安装成功”。

现在，可以在 [Google](#) 中任意搜索一些东西。在搜索结果页面的顶部会有一行文字：“Try your search on: Yahoo, Ask Jeeves, AlltheWeb, ...”。在其下面会有一个标签：“Enhanced by Butler”。这些都是由 Butler 用户脚本加进去的。

参考资料

- [Greasemonkey 脚本库](#)有上百个 Greasemonkey 脚本。

1.4. 管理用户脚本

如果愿意可以安装很多个 Greasemonkey 脚本。Greasemonkey 带有配置对话框来管理用户脚本：暂时禁用，改变配置或卸载脚本。

过程 1.3. 暂时禁用 Butler

1. 在菜单中，选择工具 (T) → **Greasemonkey** → 管理用户脚本 (U)...，Greasemonkey 会弹出一个对话框：“管理用户脚本”。
2. 在对话框左方一栏中列出了已安装的所有用户脚本 (如果您从本文开始一步步来的话

，这里只有一个脚本：Butler。)

3. 选中列表中的 **Butler**，然后取消启用复选框。左方列表中的“**Butler**”就会由黑色转为灰色。(当它还是选中状态的时候，看起来比较费劲，但是当安装了很多脚本的时候，这个特性就非常有用了。)
4. 点击 **确定**，退出“管理用户脚本”对话框。

现在 **Butler** 已经安装，但是未被启用。您在Google上随便搜索下就会发现确实如此。在页面顶端的“Enhanced by Butler”应该没有了。您可以在“管理用户脚本”对话框中重复刚才的步骤，重新选中 **Butler**，重新启用启用复选框。



虽然我用“暂时”来形容禁用用户脚本，但是如果您不重新启用它，它就始终会被禁用。之所以是暂时，只因为您可以方便的启用它，而不需要再到我的网站上来找原始脚本，而且还要重新安装。

也可以用“管理用户脚本”对话框来彻底得卸载脚本。

过程 1.4. 卸载 Butler

1. 在菜单中，选择工具 (T) → **Greasemonkey** → 管理用户脚本 (U)...。Greasemonkey 会弹出“管理用户脚本”对话框。
2. 在左方列表栏中，选中 **Butler** 再点击卸载。不需要确认；用户脚本马上就被卸载掉了。
3. 第三步... 没有第三步！(感谢 Jeff Goldblum。)

先等一下，还没完呢！您也可以修改您之前安装过的用户脚本的配置。记得

第一次安装 **Butler** 时看到的对话框吗

，上面有两个列表：包含和排除的网站？嗯，您可以在“管理用户脚本”对话框中编辑这两个列表，不管是第一次安装时或者其它什么时候。

好！继续。假如您觉得 **Butler** 不错，但是它在[Froogle](#)上毫无用处。Google的商品对比网站。那么可以编辑用户脚本的配置来排除这个网站，而让它在别的 Google 网站上仍然生效。

过程 1.5. 重新配置 Butler 排除 Froogle

1. 在菜单中，选择工具 (T) → **Greasemonkey** → 管理用户脚本 (U)...。Greasemonkey 会弹出“管理用户脚本”对话框。

2. 在左方面板中，选择“Butler”。接着在右方面板中就会显示两个列表，一个是执行的页面 (“http://*.google.*/)，另一个是豁免的页面(空白)。
3. 接下来在“不包含下列网页”的列表中，点击添加...。
4. Greasemonkey 会弹出另一个对话框：“添加网页地址”，并提示您输入新的 **Illegal HTML tag removed : URL**。在其中输入 `http://froogle.google.com/*` 然后点击确定。
5. 回到“管理用户脚本”对话框中，豁免页面列表中现在就有了您新添的 **Illegal HTML tag removed : URL** 和通配符， `http://froogle.google.com/*`，表示 Butler 不会在 `froogle.google.com` 站点的任何页面上执行。星号被当做通配符使用，可以在 **Illegal HTML tag removed : URL: 域名，路径或者任意 Illegal HTML tag removed : URL schema(http://)**中使用。
6. 点击确定，退出“管理用户脚本”对话框。在 Froogle 中搜索一下，验证下 Butler 会不会执行。但是它仍然会在普通搜索、图片搜索以及 Google 站点的其他页面中执行。

第 2 章 您的第一个用户脚本

2.1. Hello World

我们步入 Greasemonkey 美妙世界的万里长征将从第一步开始，所有读过技术手册的读者都会很熟悉这一步：让您的电脑打出“Hello world”。

例 2.1. `helloworld.user.js`


```
// Hello World! example user script
// version 0.1 BETA!
// 2005-04-22
// Copyright (c) 2005, Mark Pilgrim
// Released under the GPL license
// http://www.gnu.org/copyleft/gpl.html
//
// -----
//
// This is a Greasemonkey user script.
//
// To install, you need Greasemonkey:
http://greasemonkey.mozdev.org/
// Then restart Firefox and revisit this script.
// Under Tools, there will be a new menu item to "Install User
Script".
// Accept the default configuration and install.
//
// To uninstall, go to Tools/Manage User Scripts,
// select "Hello World", and click Uninstall.
//
// -----
//
// ==UserScript==
// @name          Hello World
// @namespace      http://diveintogreasemonkey.org/download/
// @description    example script to alert "Hello world!" on
every page
// @include       *
// @exclude       http://diveintogreasemonkey.org/*
// @exclude       http://www.diveintogreasemonkey.org/*
// ==/UserScript==

alert('Hello world!');
```

正如您所见到的，这个 `Hello World` 脚本的大部分都是注释。有些注释，比如如何安装，没什么特殊含义；那只是对初学者的一些指导。但是，有一节注释确实有特殊含义，下一节会有详细的解释。

要看到脚本的效果，您首先要

安装

，然后访问一个不在 `diveintogreasemonkey.org` 域名下的网站(例如，[Google](#))。这个页面将会像平时一样显示出来，还会弹出一个对话框：“Hello world!”

下载

- `helloworld.user.js`

2.3. 编写用户脚本代码

我们的第一个用户脚本是在执行时简单地显示一条提示信息：“Hello world!”。

例 2.3. 显示“Hello world!”提示信息

```
alert('Hello world!');
```

尽管这段代码仿佛够用了，而且也达到了目的。**Greasemonkey** 实际上在幕后做了很多的事情来确保用户脚本不会与页面所包含的原有脚本发生严重的冲突。特别是它会自动的把您的用户脚本封装在一个匿名的函数包里。一般情况下，您可以忽视，但是终究有一天会让您遇到麻烦。所以最好现在就了解一下。

最经常遇到的麻烦之一是在用户脚本里定义的变量和函数不能被别的脚本访问。事实上，只要用户脚本运行完了，所有的变量和函数就都不能使用了。如果您期望使用

`window.setTimeout` 函数，或者在链接挂上字符串式的 `onclick` 属性然后期望 Javascript 稍后调用您的函数，那么您会遇到问题。

例如，下面这个用户脚本中定义了一个函数 `helloworld`，然后尝试设置一个计数器来在一秒后调用这个函数。

例 2.4. 延迟调用函数的错误方法

```
function helloworld() {  
  alert('Hello world!');  
}  
  
window.setTimeout("helloworld()", 60);
```

这段代码没有起任何作用；不会弹出提示窗口。如果您打开

错误控制台

，会看到一个异常：`Error: helloworld is not defined.` 这是因为当延迟结束，开始调用 `helloworld()` 时，`helloworld` 函数已经不存在了。

如果您需要引用用户脚本中的变量或者函数，应该显式的把它们定义为 `window` 对象的属性，它是始终存在的。

例 2.5. 延迟调用函数的更好方法

```
window.helloworld = function() {  
  alert('Hello world!');  
}  
  
window.setTimeout("helloworld()", 60);
```

目的达到了！页面完成加载一秒后，一个提示框骄傲的弹了出来，写着：“Hello world!”

然而，在 `window` 上设置属性依然不太理想；这有点像用全局变量来做局部变量该做的事。（事实上，就是那么回事，`window` 是全局的，可以被页面中的所有脚本访问。更实际的讲，它可能会与页面自身的脚本，甚至是其它的用户脚本相互干扰。

最佳的解决方案是定义匿名函数，把它作为第一个参数传递给 `window.setTimeout`。

例 2.6. 延迟调用函数的最好方法

```
window.setTimeout(function() { alert('Hello world!') }, 60);
```

我在这里所做的是建立一个没有名字的函数（一个“匿名函数”），然后直接把它传递给 `window.setTimeout`。这样可以完成与上个例子相同的事，而不会留下痕迹。例如不会被其它的脚本检测到。

我发现我在写用户脚本时经常使用匿名函数。它们很适合创建“一次性”函数，然后当作参数传递给类似 `window.setTimeout`，`document.addEventListener` 或者赋值给事件句柄像 `click` 或 `submit`。

参考资料

- [Javascript 中的匿名函数](#)
- [Block Scope in Javascript](#) 和 [associated discussion thread](#)

2.4. 修改用户脚本

对于脚本的作者来讲，“管理用户脚本”对话框有个很实用的功能：编辑按钮可以“动态的 (live)”修改已安装的用户脚本。

过程 2.1. 编辑 Hello World 的源代码然后观察运行结果

1. 在菜单中，选择工具 (T) → **Greasemonkey** → 管理用户脚本 (U).... Greasemonkey 会弹出“管理用户脚本”对话框。
2. 在左方面板中，选择 Hello World 再点击编辑。Hello World 的已安装版本将会在你喜好的文本编辑器中打开。(否则，请检查 .js 文件是否已经关联到您喜好的文本编辑器上。)
3. 修改 `alert` 语句，替换显示内容为“Live editing!”。
4. 在编辑器中保存所做的更改，然后回到浏览器，随意刷新某个页面来测试所做的更改，你将能立刻看到效果。你不需要重新安装或者“刷新”用户脚本，您是在“动态”修改。

|



在“管理用户脚本”对话框中，点击编辑，您正在“动态”修改脚本的副本，它在 **Firefox** 的个人配置文件夹中。我形成了一个习惯，每“动态”修改完毕，又回到编辑器，选择文件 (F) → 另存为 (a)...，把用户脚本保存到另一个文件夹中。尽管这不是必须的 (**Greasemonkey** 只会用您配置文件夹中的那个脚本)，但是我喜欢在完成全部修改后把脚本在其它文件夹里保存一个“原本”。

第 3 章 调试用户脚本

3.1. 用错误控制台追踪错误

如果用户脚本好似没有正常执行，第一个要检查的地方是错误控制台，那里列出了所有与脚本有关的错误，包括用户脚本在内。

过程 3.1. 打开错误控制台

1. 在 Firefox 菜单中，选择 工具 (T) → 错误控制台 (C)。
2. 控制台中列出了从打开 Firefox 以来在所有页面上发生的所有错误，应该会有很多的(您会惊奇地发现，很多大的网站的脚本是那么的糟糕)。在开始调试您的用户脚本前，请点击清空 (C)来清空列表。

现在刷新页面来测试出现错误的用户脚本。如果它的确出错了，一条异常会显示在错误控制台中。

|



如果您的用户脚本出错了，错误控制台会显示一条异常(exception)和一个行号。由于 Greasemonkey 将用户脚本插入到页面中，所以行号没有实际的用处，应该忽略这个行号。这并不是您的用户脚本中发生异常的行号。

3.2. 用 `GM_log` 记日志

Greasemonkey 提供了一个记录用的函数，`GM_log`，这个函数可以将消息写入错误控制台。这些消息在发布前应当剔出掉，不过在调试时却非常有用。此外，看调试信息比在调试用的警告窗口中一次次点确定好得多。

`GM_log` 有一个参数：日志的字符串。在将信息输出到错误控制台后，用户脚本会一如既往地执行。

例 3.1. 记录到错误控制台然后继续(`gmlog.user.js`)

```
if
(/^http:\\\\diveintogreasemonkey\\.org\\/\\.test(window.location.hr
ef)) {
    GM_log('running on Dive Into Greasemonkey site w/o www
prefix');
} else {
    GM_log('running elsewhere');
}
GM_log('this line is always printed');
```

安装这个用户脚本后打开 <http://diveintogreasemonkey.org/>，这两行就会出现在错误控制台中：

```
Greasemonkey:
http://diveintomark.org/projects/greasemonkey//Test Log:
running on Dive Into Greasemonkey site w/o www prefix
Greasemonkey:
http://diveintomark.org/projects/greasemonkey//Test Log:
this line is always printed
```

如您所见，Greasemonkey 从

用户脚本元数据段

中取得命名空间和脚本名称，再把作为传给 `GM_log` 的参数日志消息算进来，做为显示在错误控制台中显示的信息。

如果您访问的不是<http://diveintogreasemonkey.org/>，那么下面这两条信息会显示在错误控制台中。

```
Greasemonkey:
http://diveintomark.org/projects/greasemonkey//Test Log:
running elsewhere
Greasemonkey:
http://diveintomark.org/projects/greasemonkey//Test Log:
this line is always printed
```

我已经厌倦去挖掘日志信息的最大长度。它超过了255个字符。还有，输出的信息在错误控制台中可以正确断行，可以向下滚动来查看日志消息其余部分。为日志着迷吧！

|



在错误控制台中可以用右键(**Mac**用户用 **control-click**)点击任意行选中，然后选择复制(**C**)，将信息复制到剪贴板。

参见

- GM_log

3.3. 用 DOM 查看器查看元素

DOM 查看器能够查看任何页面的已解析的文档对象模型(DOM)。可以获得每个 **Illegal HTML tag removed** : HTML 元素、属性和文本节点的详细信息；也可以看到每个页面样式表中的所有 CSS 规则；也可以看到每个对象的所有脚本属性。它的功能非常强大。

DOM 查看器包含在 Firefox 的安装程序中，但是由您的平台而定，它可能默认没装。如果您在工具 (**T**)菜单中看不到**DOM 查看器(N)**，那么您需要重新安装 Firefox。重新安装 Firefox 不会影响您现有的书签，设置，扩展以及用户脚本。

过程 3.2. 安装 DOM 查看器

1. 运行 Firefox 的安装程序。
2. 接受用户协议后，选择定制 (**C**)安装。
3. 选择安装路径后，安装向导会询问安装的组件。选择开发工具。
4. 安装结束后，运行 Firefox。您会看到工具 (**T**) → **DOM 查看器 (N)**。

下面我们将要访问深入浅出 Greasemonkey的主页，让您了解 DOM 查看器的强大功能。

过程 3.3. 查看和编辑深入浅出 Greasemonkey的主页

1. 访问<http://www.firefox.net.cn/dig/>。
2. 在菜单中，选择工具 (**T**) → **DOM 查看器 (N)**打开 DOM 查看器。
3. 在 DOM 查看器的左侧视图中，会看到 DOM 节点的树状图。如果看不到，打开左上角的下拉菜单，选择**DOM Nodes**。
4. DOM 的树状列表从 document 节点开始，标记为 `#document`。展开此节点，可以看到 **HTML** 节点。

5. 展开 `HTML` 元素后可以看到三个节点：`HEAD`、`#text` 和 `BODY`。注意 `BODY` 的 `id` 为 `diveintogreasemonkey-org`。如果没看到，调整一下列宽度。
6. 展开 `BODY` 可以看到五个节点：`#text`，`DIV id="intro"`，`#text`，`DIV id="main"` 和 `#text`。
7. 展开 `DIV id="intro"` 可以看到两个节点：`#text` 和 `DIV class="sectionInner"`。
8. 展开 `DIV class="sectionInner"` 可以看到两个节点：`#text` 和 `DIV class="sectionInner2"`。
9. 展开 `DIV class="sectionInner2"` 可以看到五个节点：`#text`，`DIV class="s"`，`#text`，`DIV class="s"` 和 `#text`。
10. 展开第一个 `DIV class="s"` 可以看到五个节点：`#text`，`H1`，`#text`，`P` 和 `#text`。
11. 选择 `H1` 节点。在原始页面上(DOM 查看器后面)，`H1` 元素会闪现红色的边框。在右侧面板中可以看到节点的名称("H1")、命名空间 **Illegal HTML tag removed : URI** (空白，因为 **Illegal HTML tag removed : HTML** 没有命名空间 -- 只有在页面被当作 `application/xhtml+xml` 时有效，或者显示一些其他名字空间 **Illegal HTML tag removed : XML** 的页面)、节点类型(1 代表元素)和节点值(空白，因为标题没有值 -- 标题上的文字有自己的节点)。
12. 在右侧面板的顶部，有个下拉菜单，在其中可以看到数个选项：**DOM Node**, **Box Model**, **XBL Bindings**, **CSS Style Rules**, **Computed Style** 和 **Javascript Object**。它们提供了当前选中的节点的不同信息。有些是可以修改的，变更会立即反映到原始页面上。选择 **Javascript Object** 可以看到选中的 `H1` 元素的所有可脚本控制的属性和方法。
13. 选择 **CSS Style Rules**。右侧面板会分为两部分，顶部的列出了所有可以作用于这个元素的样式(包括浏览器已有的默认样式)，下面的显示了样式所定义的属性。
14. 在右上方的视图中选中第二条规则，这条样式规则是在 <http://www.firefox.net.cn/dig/css/dig.css> 样式表中定义的。
15. 在右下面板中双击 `font-variant` 属性，然后输入新值：`normal`。在原始页面中(DOM 查看器后面)，"深入浅出 Greasemonkey" 标志文字会立即从 `small-caps` 变为正常的大小写字母。
16. 在右下面板中任意位置点击右键(Mac 用户 `control-click`)，然后选择 新建属性。会弹出一个对话框："新样式规则"。输入属性名称：`background-color`，然后点击确定，接下来属性值：`red`，然后点击确定应用新属性。新属性会显示在右下面板中，并且原始页面会立即变成红色背景。

如果觉得依次展开 DOM 节点树中每层很不方便，那么我强烈推荐您使用 **Inspect Element** 扩展，它能迅速找到 DOM 查看器中的指定元素。

|



安装 **Inspect Element** 扩展前，您必须先安装 **DOM** 查看器，否则 **Firefox** 就不能正常启动。如果已经遇到了这种情况，打开命令行窗口，进入 **Firefox** 的安装目录，输入 `firefox -safe-mode` 。**Firefox** 会以安全模式启动，不加载任何扩展，然后选择工具 (T) → 附加软件 (A) 接着卸载 **Inspect Element** 。

过程 3.4. 用 **Inspect Element** 直接查看元素

1. 访问 [Inspect Element](#) 下载页面，然后点击 **Install Now** 。
2. 重新启动 **Firefox** 。
3. 再访问<http://www.firefox.net.cn/dig/> 。
4. 右击(Mac 用户 control-click) 标志语：`深入浅出 Greasemonkey` 。
5. 在上下文菜单中，选择 **Inspect element** 。**DOM** 查看器会打开，并且选中了 `H1` 元素，然后您可以马上开始查看和/或编辑它的属性。

|



DOM 查看器不会“跟着”您一起浏览网页。如果打开 **DOM** 查看器然后在原始窗口中浏览别的网页，**DOM** 查看器会变得非常困惑。好的做法是，去您想去的地方，查看您想查看的页面，在做别的事情前先关闭 **DOM** 查看器。

参考资料

- [DOM 查看器介绍](#)
- [Inspect Element 扩展](#)
- [Inspector Widget 扩展](#) 具有 **Inspect Element** 相同功能的扩展，它增加了一个工具条，而不是一个菜单项。

3.4. 用 **Javascript Shell** 计算表达式

Javascript Shell 是一个 bookmarklet，可以在当前页面的内容中计算任意 **Javascript** 表达式。

过程 3.5. 安装 **Javascript Shell**

1. 访问 [Jesse's Web Development Bookmarklets](#)。
2. 将 **Shell** bookmarklet 拖放到您的书签工具栏。
3. 访问一个网页(例如, 深入浅出 [Greasemonkey](#) 主页), 然后点击书签工具栏上的 **Shell** bookmarklet。Javascript Shell 窗口会在后台打开。

Javascript Shell 提供了与

DOM 查看器

一样强大的功能, 但是环境更加宽松。就把它当作 DOM 的命令行版本吧。游戏开始。

例 3.2. Javascript Shell 介绍

```
**`document.title`**
深入浅出 Greasemonkey
**`document.title = 'Hello World'`**
Hello World
**`var paragraphs = document.getElementsByTagName('p')`**
**`paragraphs`**
[object HTMLCollection]
**`paragraphs.length`**
5
**`paragraphs[0]`**
[object HTMLParagraphElement]
**`paragraphs[0].innerHTML`**
教老网玩新把戏
**`paragraphs[0].innerHTML = 'Live editing, baby! '`**
Live editing, baby!
```

当输入完上面的内容后, 按回车键, 变更就会反映到原始网页上。

我想提一下 Javascript Shell 的 `props` 函数。

例 3.3. 获取元素属性

```
**`var link = document.getElementsByTagName('a')[0]`**
**`props(link)`**
Methods of prototype: blur, focus
Fields of prototype: id, title, lang, dir, className, accessKey,
charset, coords, href, hreflang, name, rel, rev, shape,
```

```

tabIndex,
target, type, protocol, host, hostname, pathname, search, port,
hash, text, offsetTop, offsetLeft, offsetWidth, offsetHeight,
offsetParent, innerHTML, scrollTop, scrollLeft, scrollHeight,
scrollWidth, clientHeight, clientWidth, style
Methods of prototype of prototype of prototype: insertBefore,
replaceChild, removeChild, appendChild, hasChildNodes,
cloneNode,
normalize, isSupported, hasAttributes, getAttribute,
setAttribute,
removeAttribute, getAttributeNode, setAttributeNode,
removeAttributeNode, getElementsByTagName, getAttributeNS,
setAttributeNS, removeAttributeNS, getAttributeNodeNS,
setAttributeNodeNS, getElementsByTagNameNS, hasAttribute,
hasAttributeNS, addEventListener, removeEventListener,
dispatchEvent,
compareDocumentPosition, isSameNode, lookupPrefix,
isDefaultNamespace,
lookupNamespaceURI, isEqualNode, getFeature, setUserData,
getUserData
Fields of prototype of prototype of prototype: tagName,
nodeName,
nodeValue, nodeType, parentNode, childNodes, firstChild,
lastChild,
previousSibling, nextSibling, attributes, ownerDocument,
namespaceURI,
prefix, localName, ELEMENT_NODE, ATTRIBUTE_NODE, TEXT_NODE,
CDATA_SECTION_NODE, ENTITY_REFERENCE_NODE, ENTITY_NODE,
PROCESSING_INSTRUCTION_NODE, COMMENT_NODE, DOCUMENT_NODE,
DOCUMENT_TYPE_NODE, DOCUMENT_FRAGMENT_NODE, NOTATION_NODE,
baseURI, textContent, DOCUMENT_POSITION_DISCONNECTED,
DOCUMENT_POSITION_PRECEDING, DOCUMENT_POSITION_FOLLOWING,
DOCUMENT_POSITION_CONTAINS, DOCUMENT_POSITION_CONTAINED_BY,
DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC
Methods of prototype of prototype of prototype of prototype: toString

```

“哇哦！”这些是什么？它列出了元素 `<a>` 在 Javascript 中所有有效的属性和方法，并且按照 DOM 的对象层级分类。先列出的是链接元素的特有方法和属性(比如 `blur` 和 `focus` 方法，`href` 和 `hreflang` 属性)，然后列出了所有节点公有的方法和属性(比如

`insertBefore`)，等等。

此外，这和 DOM 查看器上的内容一样.....只不过是多些打字多些试验，少点些鼠标。

|



| 同

DOM 查看器

一样，Javascript Shell 也不会“跟着”您浏览网页。如果打开 Javascript Shell 然后在原始窗口中浏览别的网页，Javascript Shell 会变得非常困惑。好的做法是，去您想去的地方，查看您想查看的页面，在做别的事情前先关闭 Javascript Shell。|| --- | --- |

3.5. 其他调试工具

下面是我觉得有用的其他调试工具，由于时间有限，我没有全部列出。

参考资料

- [Web Developer 扩展](#) 有很多分析页面的函数。
- [Aardvark](#) 互动的显示标签名称、`id` 和 `class` 的属性。
- [Venkman Javascript Debugger](#) 完整的 Javascript 运行时调试器。
- [Web Development Bookmarklets](#) 有一些有用的函数，可以把它们拖放到工具栏中。
- [JSUnit](#) 是 Javascript 单元测试框架。
- [js-lint](#) 可检查出 Javascript 代码的常见错误。

第 4 章 公共模式

4.1. 在域名以及它所有子域名上执行用户脚本

对于许多网站，无论是否有 `www.` 前缀，访问网站都是等效的。如果要为这样的站点写用户脚本，需要能匹配这两种地址。

例 4.1. 匹配域名和它所有子域名的元数据标签

```
// ==UserScript==
// @include http://example.com/*
// @include http://*.example.com/*
// ==/UserScript==
```

实例

- [Butler](#)
- [Salon Auto-Pass](#)

4.2. 测试 **Greasemonkey** 函数是否有效

Greasemonkey 的新版本给用户脚本提供了新函数。如果准备发布用户脚本，那么应该测试下所用的 Greasemonkey 函数是否存在。

例 4.2. **GM_xmlHttpRequest** 函数无效时警告用户

```
if (!GM_xmlHttpRequest) {
    alert('请升级到最新版本的 Greasemonkey.');
```

return;

```
}
// 使用 GM_xmlHttpRequest 的其他代码
```

4.3. 测试页面中是否有

Illegal HTML tag removed : HTML

元素 {#4-3-illegal-html-tag-removed-html}

可以用 `getElementsByTagName` 函数来测试页面中是否有 **Illegal HTML tag removed : HTML** 元素。

例 4.3. 检查页面中是否有 **<textarea>** 元属

```
var textareas = document.getElementsByTagName('textarea');
if (textareas.length) {
// 页面中有至少一个 textarea
} else {
// 页面中没有 textarea
}
```

实例

- [BetterDir](#)
- [Zoom Textarea](#)

4.4. 操作所有

Illegal HTML tag removed : HTML

元素 {#4-4-illegal-html-tag-removed-html}

有时需要操作页面中的所有 **Illegal HTML tag removed : HTML** 元素。Firefox支持

`getElementsByTagName('*')`，它返回一个可遍历操作的元素集合。

例 4.4. 遍历所有元素

```
var allElements, thisElement;
allElements = document.getElementsByTagName('*');
for (var i = 0; i < allElements.length; i++) {
    thisElement = allElements[i];
    // 使用 thisElement
}
```

|



| 只有真需要操作所有元素时才能采用这种方法。如果只要操作特定的一些元素，可使用 XPath 查询可以正确快速地得到这些元素。更多资料，请查看

操作所有有特定属性的元素

◦ ||---|---|

实例

- [Anti-Disabler](#)

4.5. 操作特定

Illegal HTML tag removed : HTML

元素的所有实例 {#4-5-illegal-html-tag-removed-html}

有时您需要操作页面中特定 **Illegal HTML tag removed : HTML** 元素的所有实例。例如改变所有 `<textarea>` 的字体大小。最简单的办法就是调用

`getElementsByTagName('tagname')`，它返回一个可遍历操作的元素集合。

例 4.5. 获取页面上所有的 **textarea**

```
var allTextareas, thisTextarea;
allTextareas = document.getElementsByTagName('textarea');
for (var i = 0; i < allTextareas.length; i++) {
    thisTextarea = allTextareas[i];
    // 使用 thisTextarea
}
```

|



| 不应该用这个方法来自操作页面中所有的链接，因为 `<a>` 元素也可以用作页面中的锚。要知道如何查找页面中所有的链接，请查看

操作所有有特定属性的元素

◦ ||---|---|

实例

- [Zoom Textarea](#)

4.6. 操作所有有特定属性的元素

在 Greasemonkey 的兵器库中最强悍的一个就是 `evaluate` 方法。利用 XPath 查询语言，它可以用来获取页面中的元素，属性以及文本。

举个例子来说，如果您想获得页面中的全部链接。您也许会想到用

```
document.getElementsByTagName('a')
```

，但是您还要检查每个元素是否具有 `href` 属性，因为 `<a>` 元素还可以用作有名称的锚。

然而可以用 Firefox 内建的 XPath 功能来查找具有 `href` 属性的 `<a>` 元素。

例 4.6. 获取页面中的所有链接

```
var allLinks, thisLink;
allLinks = document.evaluate(
    '//a[@href]',
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i = 0; i < allLinks.snapshotLength; i++) {
    thisLink = allLinks.snapshotItem(i);
    // 使用 thisLink
}
```

`document.evaluate` 方法是关键。它有一个代表 XPath 查询语句的字符串参数以及一些其它参数，接下来解释一下。这条 XPath 查询语句找到了具有 `href` 属性的 `<a>` 元素，用随机的次序排列后返回。(也就是说，集合中的第一个元素并不一定是页面中的第一个元素。)然后您可以用 `allLinks.snapshotItem(i)` 方法访问找到的元素。

XPath 表达式所能做到的甚至会使您惊讶。请看下面这个例子，它获取了全部具有 `title` 属性的元素。

例 4.7. 获取所有具有 `title` 属性的元素


```

var allElements, thisElement;
allElements = document.evaluate(
    '//*[ @title ]',
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i = 0; i < allElements.snapshotLength; i++) {
    thisElement = allElements.snapshotItem(i);
    switch (thisElement.nodeName.toUpperCase()) {
    case 'A':
        // 这是链接，在这里完成您的操作
        break;
    case 'IMG':
        // 这是图片，在这里完成您的操作
        break;
    default:
        // 其他类型的 HTML 元素，在这里完成您的操作
    }
}

```

|



如果已有一个元素的引用(例如 `thisElement`)，可以用 `thisElement.nodeName` 来判断它的 **Illegal HTML tag removed : HTML** 标签。如果页面被当作 `text/html` 类型，标签名称就总是大写，不论它在原始页面是如何定义的。如果页面被当作 `application/xhtml+xml` 类型，那么标签名称就总是小写的。我总是用 `thisElement.nodeName.toUpperCase()` 这样我就可以不用管这些了。

这是另一个 XPath 查询，它获取了具有特定 `class` 属性的 `<div>` 元素。

例 4.8. 获取所有 `class` 为 `sponsoredlink` 的 `<div>`;

```
var allDivs, thisDiv;
allDivs = document.evaluate(
    "//div[@class='sponsoredlink']",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i = 0; i < allDivs.snapshotLength; i++) {
    thisDiv = allDivs.snapshotItem(i);
    // 使用 thisDiv
}
```

附注：在 XPath 查询语句外使用双引号，这样在语句内就可以使用单引号了。

`document.evaluate` 方法中有很多参数。第二个参数(在前两个例子中都是 `document`)可以是任意元素。XPath 查询只返回这个元素的子元素结点。如果已有一个元素的引用(比如，从 `document.getElementById` 或者 `document.getElementsByTagName` 数组的一项中得到的引用)，您就可以限制查询只返回这个元素的子元素。

第三个参数是对名称空间解析函数的引用，只有在 `application/xhtml+xml` 类型页面执行的用户脚本中才会用到。即使对它不了解也没关系，因为那种类型的页面不是很多，您可能一次也遇不到。如果您很想知道它的用法，[Mozilla XPath 文档](#)解释了它的用法。

第四个参数是结果的返回方式。在前面的两个例子中都使用了

`XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE`，它返回的元素是随机次序的。我99%都是使用这种方式，但是，可能出于某种原因，想以在页面上出现的顺序返回结果，您可以使用

`XPathResult.ORDERED_NODE_SNAPSHOT_TYPE`。 [Mozilla XPath 文档](#)还提供了一些其他返回方式的例子。

第五个参数用来合并两次 XPath 查询的结果。传入以前调用 `document.evaluate` 结果，它将返回两次查询的合并结果。在前面的两个例子中，这个参数都用了 `null`，这意味着我们只想获得本次 XPath 查询的结果。

现在明白了吗？XPath 既可简单，也可难，这就看您怎么用了。我强烈推荐您阅读[这个优秀的 XPath 教程](#)，可以学到更多的 XPath 语法。关于 `document.evaluate` 的其他参数，我很少用除非您已经在这里看到他们了。事实上，您可以定义一个函数来封装它们。

例 4.9. xpath 函数

```
function xpath(query) {  
    return document.evaluate(query, document, null,  
        XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE, null);  
}
```

现在您可以简单的调用 `xpath('//a[@href]')` 来获得页面上的全部链接，或者用 `xpath('//*[@title]')` 获得具有 `title` 属性的元素。您仍然需要用 `snapshotItem` 方法访问结果中的每个元素；它不是一个规则的 Javascript 数组。

实例

- [Access Bar](#)
- [BetterDir](#)
- [Blogdex Display Title](#)
- [Butler](#)
- [Frownies](#)
- [Offsite Blank](#)
- [Rotten Reviews](#)
- [Stop The Presses](#)

参考资料

- [Mozilla XPath 文档](#)
- [XPath 实例教程](#)
- [XPathResult 参考手册](#)

4.7. 在元素前插入内容

如果已找到某个元素(不论用何种方法)，您想在它前面插入额外的内容。您可以使用 `insertBefore` 函数。

例 **4.10.** 在主内容前插入 `<hr>`;

假设有个元素的 ID 为 `"main"`。

```
var main, newElement;
main = document.getElementById('main');
if (main) {
    newElement = document.createElement('hr');
    main.parentNode.insertBefore(newElement, main);
}
```

实例

- [Butler](#)
- [Zoom Textarea](#)

4.8. 在元素后插入内容

如果已找到某个元素，您想在它后面插入额外的内容。您也可以使用 `insertBefore` 函数，要与 `nextSibling` 属性联合使用。

例 4.11. 在导航条后面插入 `<hr>`

假设有个元素的 ID 为 `"navbar"`。

```
var navbar, newElement;
navbar = document.getElementById('navbar');
if (navbar) {
    newElement = document.createElement('hr');
    navbar.parentNode.insertBefore(newElement,
    navbar.nextSibling);
}
```

|



即使 `someExistingElement` 是它的父元素的最后一个孩子(在它之后没有下一个元素)，仍然可以在 `someExistingElement.nextSibling` 之前插入新内容。在这种情况下，`someExistingElement.nextSibling` 将返回一个空值，`insertBefore` 函数将把新内容追加到最后。(也许这对你来说没有太大意义，但我想让您知道的是，尽管这种方法似乎是不太对，但它却总是有效的。)

实例

- [Blogdex Display Title](#)
- [Butler](#)
- [Frownies](#)

4.14. 获取元素样式

当多个 CSS 规则同时应用到某个元素上时，获取这个元素的实际样式有时候是有帮助的。您可能会简单的认为，通过元素的 `style` 属性就可以得到，这样的话您就错了。它只会返回 `style` 属性的内容。要获得元素的最终样式（包括在外部定义的样式），您需要使用 `getComputedStyle` 函数。

为了演示，我们创建一个简单的测试页面。它首先对页面中所有的 `< p >` 元素定义了一个样式，但随后对其中的一个 `< p >` 元素用它的 `style` 属性重新定义了样式。

```
<html>
<head>
<title>样式测试页</title>
<style type="text/css">
p { background-color: white; color: red; }
</style>
</head>
<body>
<p id="p1">这行是红色的。</p>
<p id="p2" style="color: blue">这行是蓝色的。</p>
</body>
</html>
```

例 4.17. 获取由元素的 `style` 属性定义的样式

```
var p1elem, p2elem;
p1elem = document.getElementById('p1');
p2elem = document.getElementById('p2');
alert(p1elem.style.color); // 提示为空字符串
alert(p2elem.style.color); // 提示 "blue"
```

用这种方法获得的信息并不是很有用,所以您不应该再用 `element.style` 来获取单独的样式。(您可以用它来设置元素的样式,详见

设置元素样式

。)

那么您该用什么方法来取代它呢? `getComputedStyle()` 。

例 4.18. 获取元素的真实样式

```
var p1elem, p2elem, p1style, p2style;
p1elem = document.getElementById('p1');
p2elem = document.getElementById('p2');
p1style = getComputedStyle(p1elem, '');
p2style = getComputedStyle(p2elem, '');
alert(p1style.color); // 提示 "rgb(255, 0, 0)"
alert(p2style.color); // 提示 "rgb(0, 0, 255)"
```

实例

- [Butler](#)
- [Zoom Textarea](#)

4.15. 设置元素样式

如果只需要设置单个元素的少数样式,可以“手动地”在这个元素的 `style` 属性中设置多个子属性。

例 4.19. 设置单个元素的样式

假设有个元素 ID 为 `"logo"` 。

```
var logo = document.getElementById('logo');
logo.style.marginTop = '2em';
logo.style.backgroundColor = 'white';
logo.style.color = 'red';
```

|



并不是每个样式的属性名称都是显然的。通常，它们有相似的模式，例如 `margin-top` 对应 `someElement.style.marginTop`。但是也有例外：`float` 属性对应到 `someElement.style.cssFloat`，因为 `float` 是 **Javascript** 的保留字。

实例

- [Access Bar](#)
- [BetterDir](#)
- [Blogdex Display Title](#)
- [Zoom Textarea](#)

参考资料

- [CSS 属性](#)

4.16. 处理已渲染的页面

由于 Firefox 保存已渲染页面的内在方式，应该在页面加载完成以后，才能对页面上的 DOM 做出大规模的改动，那么这个操作应该发生在页面完成载入之后，这是由已执行页面在 Firefox 内的存储方式决定的。您可以用 `addEventListener` 函数来延迟函数的执行。

例 4.20. 替换全部页面为自定义的内容

```
var newBody =
'<html>' +
'<head>' +
'<title>My New Page</title>' +
'</head>' +
'<body>' +
'<p>This page is a complete replacement of the original.</p>' +
'</body>' +
'</html>';
window.addEventListener(
  'load',
  function() { document.body.innerHTML = newBody; },
  true);
```

仔细看这段代码，您可能自然地想知道它是怎么执行的。我声明了一个匿名函数来做为 `window.addEventListener` 的第二个参数传入，它访问了在这个函数外定义的变量 `newBody`。这种方法称为“闭包(closure)”，在 Javascript 中是完全有效的。一般说来，在“外层”函数中定义的“内层”函数可以访问外层函数的所有局部变量 -- 甚至在外层函数执行结束以后。这是一个很有用的特性，可以通过它来创建事件句柄和用来在运行时建立的数据构架的其他函数。



删除和替换 `document.body.innerHTML` 并不会更改页面。原页面在 `<head>` 中定义的所有内容仍然是有效的，包括页面的标题，**CSS** 样式，以及脚本。可以分开修改或者移除。

实例

- [Access Bar](#)
- [BetterDir](#)

4.17. 匹配大小写无关的属性值

在 **Illegal HTML tag removed** : HTML 中，许多属性的值是大小写无关的，有的还允许首尾空格。如果想获取到所有这样的值，需要在 XPath 查询中使用一点技巧。

例 4.21. 获取 **method** 为 "POST" 或 "post" 的表单(form)

```
var postforms = document.evaluate(
    "//form[translate(@method, 'POST ', 'post')='post']",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
```

本次 XPath 查询可以获取所有以 POST 方式提交的表单。首先，我们需要用 `translate` 函数将 `method` 属性的值中的字母大写变成小写。(XPath 2.0 有 `lowercase` 函数，但我还从来没用成功过。)其次，我们需要将首尾的空格去掉。我们可以将其整合到调用 `translate` 函数中，只要在第一个参数中加个额外的空格。因为在第二个参数中没有对应的字母，这样一来所有的空格就被去掉了。最后，我们就能将获得的属性值同 `'post'` 做比较。

实例

- [ForceGet](#)

4.18. 获取当前域名

用户脚本操作多个域名(或者所有域名)时，常常需要检测当前页面的域名。可以用

`window.location.href` 获取当前页面的完整 **Illegal HTML tag removed** : URL，或者用 `window.location.host` 只获取域名。

例 4.22. 获取当前域名

```
var href = window.location.host;
```

实例

- [Offsite Blank](#)

4.21. 俘获用户点击

虽然

改写链接很简单

，您也可以走的更远，使用 `addEventListener` 函数，俘获页面上任何地方的每次点击。(这也能俘获点击链接。)您可以考虑下怎么做：是否让可点击元素的点击操作“失效”，或者做些完全不同的操作。

例 4.25. 当用户点击页面上任意地方时做点事

```
document.addEventListener('click', function(event) {  
  // event.target 是被点击的元素  
  
  // 把你的代码放在这里  
  
  // 如果您想阻止默认点击动作  
  // (例如链接转向)，使用下面这两条命令：  
  event.stopPropagation();  
  event.preventDefault();  
}, true);
```

注释：用

匿名函数

作为传入 `document.addEventListener` 函数的一个参数。

4.22. 覆盖内建的 Javascript 方法

您可以用 `prototype` 属性 覆盖原有的对象方法。

例 4.26. 当表单提交时做点事

```
function newsubmit(event) {
  var target = event ? event.target : this;

  // 在这里定义想做的操作
  alert('Submitting form to ' + target.action);

  // 调用真正的提交函数
  this._submit();
}

// 捕获所有表单的 onsubmit 事件
window.addEventListener('submit', newsubmit, true);

// 如果脚本调用 someForm.submit(), onsubmit 事件不会发生,
// 所以我们需要重新定义 HTMLFormElement 类的 submit 方法。
HTMLFormElement.prototype._submit =
HTMLFormElement.prototype.submit;
HTMLFormElement.prototype.submit = newsubmit;
```

在这里做了两件事。首先，我加入了一个用来捕获 `submit` 事件的监听函数。当用户点击表单的提交按钮时，触发 `submit` 事件。然而，当别的脚本调用表单的 `submit()` 方法时，并不会触发 `submit` 事件。所以，我做的第二件事是覆盖 `HTMLFormElement` 类的 `submit` 方法。

但是等等，还有几点需要说明。事件监听函数和重新定义的方法都指向同一个函数，`newsubmit`。如果 `newsubmit` 被一个 `submit` 事件调用，`event` 参数将会被包含在事件对象中，这个事件对象包含事件的信息（例如，`event.target` 是被提交的表单）。然而，

如果脚本手动调用了 `submit` 方法，`event` 事件会被忽略掉，但是全局变量 `this` 将会指向这个表单。因此，在我的 `newsubmit` 函数中，我先判断 `event` 是否为空；如果为空，就用 `this` 来得到这个表单。

|



正常情况下，当用户提交一个表单时，例如，点击表单中的 提交 按钮或者按 回车键），都会触发 `submit` 事件。但是，当脚本调用 `aForm.submit()` 提交表单时，却不会触发 `submit` 事件。因此，您必须做两件事来捕获表单的提交事件：给 `submit` 事件增加事件监听，并且修改 `HTMLFormElement` 类的原型来重定向 `submit()` 方法到您的自定义函数上。

参考资料

- [Javascript 事件兼容性表](#)
- [Mozilla 中的 Javascript-DOM 原型](#)
- [显示事件对象常量](#)

4.23. 解析

Illegal HTML tag removed : XML

{#4-23-illegal-html-tag-removed-xml}

Firefox 自动将当前页面解析为 DOM，但是您也可以手动创建 DOM 而不使用任何 **Illegal HTML tag removed : XML** 字符串，可以是您自己建立的，也可以是您从远端主机得到的 **Illegal HTML tag removed : XML**。

例 **4.27**. 将任意字符串解析为

Illegal HTML tag removed : XML

{#4-27-illegal-html-tag-removed-xml}

```

var xmlString = '<passwd>' +
  '  <user id="101">' +
  '    <login>mark</login>' +
  '    <group id="100"/>' +
  '    <displayname>Mark Pilgrim</displayname>' +
  '    <homedir>/home/mark/</homedir>' +
  '    <shell>/bin/bash</shell>' +
  '  </user>' +
  '</passwd>'
var parser = new DOMParser();
var xmlDoc = parser.parseFromString(xmlString,
  "application/xml");

```

这里关键的地方是 `DOMParser` 对象，有个 `parseFromString` 方法。(它也有其他的方法，但在这里我们用不到。) `parseFromString` 方法有两个参数：解析用的 **Illegal HTML tag removed** : XML 字符串，和内容类型。两者都是必需的。

|



`DOMParser` 的 `parseFromString` 方法将内容的类型作为它的第二个参数。这个方法可以接受 `application/xml` , `application/xhtml+xml` 和 `text/xml` 。由于深究起来，理由会很荒唐，您应该始终使用 `application/xml` 。

这个模式非常强大，如果把它与 `GM_xmlHttpRequest` 函数结合起来解析远程源的 **Illegal HTML tag removed** : XML 。

例 4.28. 解析远程源的

Illegal HTML tag removed : XML

{#4-28-illegal-html-tag-removed-xml}

```
GM_xmlHttpRequest({
  method: 'GET',
  url: 'http://greaseblog.blogspot.com/atom.xml',
  headers: {
    'User-agent': 'Mozilla/4.0 (compatible) Greasemonkey/0.3',
    'Accept': 'application/atom+xml,application/xml,text/xml',
  },
  onload: function(responseDetails) {
    var parser = new DOMParser();
    var dom =
parser.parseFromString(responseDetails.responseText,
  "application/xml");
    var entries = dom.getElementsByTagName('entry');
    var title;
    for (var i = 0; i < entries.length; i++) {
      title = entries[i].getElementsByTagName('title')
[0].textContent;
      alert(title);
    }
  }
});
```

这段代码会载入<http://greaseblog.blogspot.com/atom.xml>的 Atom feed，把它解析为 DOM，然后查询 DOM 得到条目列表。再对每个条目做 DOM 查询，获得条目的标题(title)，然后在对话框中显示出来。

参见

- 操作特定

Illegal HTML tag removed : HTML

元素的所有实例

- GM_xmlHttpRequest

第 5 章 实例教学

5.3. 案例：Ain't It Readable

覆盖页面样式

```
thisdomain = window.location.host;
```

获取页面中所有链接也一样容易，尽管我应该注意到我忽视了自己

的建议

简单地使用 `document.getElementsByTagName('a')` 而不用 XPath 查询。其实也没什么不同.....

```
links = document.getElementsByTagName('a');
```

接下来，我遍历了所有的链接(实际上是所有的 `<a>` 元素，它们中的一些可能是链接)然后检查链接的域名和当前页面的域名是否匹配。由于一些链接可能指向非 **Illegal HTML tag removed** : HTTP 的 **Illegal HTML tag removed** : URL(例如，本地文件或者 **Illegal HTML tag removed** : FTP 服务器)，我需要检查 `a.host` 是否存在，然后再检查它与当前域名是否相等。

```
for (var i = 0; i < links.length; i++) {  
  a = links[i];  
  if (a.host && a.host != thisdomain) {  
    ...  
  }  
}
```

如果我找到一个包含域名的链接，但是它的域名与当前域名不相同，我只需设置它的 `target` 属性为 `"_blank"` 就可以强制链接在新窗口打开了。

```
a.target = "_blank";
```

下载

- `offsiteblank.user.js`

参见

- 获取当前域名
- 操作特定

Illegal HTML tag removed : HTML

元素的所有实例

- 操作所有有特定属性的元素

5.5. 案例：Dumb Quotes

转换智能引号为原始引号

DumbQuotes 是应许多网络博客的需要而生的：大多数出版软件可以自动转换直接的 ASCII 引号为“智能引号(smart quotes)”，但是当作者在文章中复制粘贴文本时，同样是这些软件处理“智能引号”却很愚笨。常见的例子是博客想引用其他网站的一段文字。在浏览器中选中几句，粘贴到自己网站的表单中提交，这段文字却看起来完全不同，因为他们的出版软件没有处理好字符编码。

当然，我不能修复世界上每个出版系统，但是我可以为自己写个用户脚本来解决，自动转换智能引号和其他有问题的高位字符为等价的7位 ASCII 字符。

例 5.5. `dumbquotes.user.js`

```
// ==UserScript==
// @name          DumbQuotes
// @namespace      http://diveintogreasemonkey.org/download/
// @description    straighten curly quotes and apostrophes,
//                  simplify fancy dashes, etc.
// @include        *
// ==/UserScript==

var replacements, regex, key, textnodes, node, s;

replacements = {
  "\xa0": " ",
  "\xa9": "(c)",
  "\xae": "(r)",
  "\xb7": "*",
  "\u2018": "'",
  "\u2019": "'",
  "\u201c": '"',
  "\u201d": '"',
```

```
"\u2026": "...",
"\u2002": " ",
"\u2003": " ",
"\u2009": " ",
"\u2013": "-",
"\u2014": "--",
"\u2122": "(tm)"};
regex = {};
for (key in replacements) {
    regex[key] = new RegExp(key, 'g');
}

textnodes = document.evaluate(
    "//text()",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i = 0; i < textnodes.snapshotLength; i++) {
    node = textnodes.snapshotItem(i);
    s = node.data;
    for (key in replacements) {
        s = s.replace(regex[key], replacements[key]);
    }
    node.data = s;
}
```

这段代码分为四个步骤：

1. 先定义字符替换规则列表，映射某些8位字符到对应的7位字符。
2. 获取当前页面中的所有文字结点。
3. 遍历文字结点清单。
4. 在每个文字结点中，替换每个8位字符为等价的7位字符。

第一步其实是两步。Javascript 的字符串替换基于正则表达式。所以要替换8位字符为等价的7位字符，需要创建一个正则表达式集合。


```

replacements = {
  "\xa0": " ",
  "\xa9": "(c)",
  "\xae": "(r)",
  "\xb7": "*",
  "\u2018": "'",
  "\u2019": "'",
  "\u201c": '"',
  "\u201d": '"',
  "\u2026": "...",
  "\u2002": " ",
  "\u2003": " ",
  "\u2009": " ",
  "\u2013": "-",
  "\u2014": "--",
  "\u2122": "(tm)"};
regex = {};
for (key in replacements) {
  regex[key] = new RegExp(key, 'g');
}

```

我使用花括号语法迅速建立了一个关联数组。等价于(但是键入更少)单独把每个关键字和值对应起来：

```

replacements["\xa0"] = " ";
replacements["\xa9"] = "(c)";
replacements["\xae"] = "(r)";
// 等等

```

每个8位的字符都是用十六进制值表示的，是用了逃逸语法(escaping syntax) `"\xa0"` 或 `"\u2018"`。我们完成字符到字符串的关联数组后，我会遍历整个数组，然后建立正则表达式对象列表。每个正则表达式对象会在全局范围搜索8位字符。(第二个参数 `'g'` 意义为全局搜索；否则每个正则表达式只会搜索和替换第一次出现的特定8位字符，这样我可能会漏掉很多。)

下一步是获取当前文档中的全部文本节点。您可以非常想说，“嗨，我只要用 `document.body.innerHTML` 就得到全部页面的字符串，然后搜索替换就成了。”

```
var tmp = document.body.innerHTML;
// 在 tmp 上完成一批搜索和替换
document.body.innerHTML = tmp;
```

但这是个坏习惯，因为 `innerHTML` 会返回页面中的全部源代码：所有的标签、所有的脚本、所有的属性等等。在这种情况下，有可能不会造成问题(**Illegal HTML tag removed** : HTML 标签并不包含8位字符)，但它在其他情况下不堪设想，很难调试。你要问你自己到底要搜索和替换什么。如果答案是“原始的页面源代码”，那么就去使用 `innerHTML`。然而在这种情况下，答案是“全部的页面文字”，所以正确的方法是使用 XPath 查询获取所有文本节点。

```
textnodes = document.evaluate(
    "//text()",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
```

这里使用了 XPath 函数，`text()`，用来匹配任意文本节点。您可能对查询元素节点很熟悉了：所有 `<a>` 元素集，或者所有具有 `alt` 属性的 `` 元素集。但是 DOM 也包含有节点中存在的文本内容。(也有其他类型的节点，例如注释和处理指令。)现在它也是我们感兴趣的文本节点。

第3步是遍历所有文本节点。这完全跟遍历像 `//a[@href]` XPath 查询返回的节点集一样；唯一的不同就是遍历中的是文本节点，而不是元素节点。

```
for (var i = 0; i < textnodes.snapshotLength; i++) {
    node = textnodes.snapshotItem(i);
    s = node.data;
    // 做替换操作
    node.data = s;
```

`node` 是循环中的当前文本节点，而 `s` 是 `node` 中存在文本的字符串变量。我打算使用 `s` 完成替换操作，然后把处理结果复制回原始节点中。

所以现在我有单个节点的文本，我需要完成替换操作。因为我已经建好了正则表达式列表和替换字符串列表，所以这相对简单。

```

    for (key in replacements) {
    s = s.replace(regex[key], replacements[key]);
    }

```

下载

- [dumbquotes.user.js](#)

参见

- 操作所有有特定属性的元素

5.6. 案例：Frownies

转换图片表情为文字

Frownies 是对一个玩笑的回应。[Greasemonkey 邮件列表](#)有人宣称他们开发了一个用户脚本，可以把 ASCII “表情符号” 比如 `: -)` 转换为与它们相同含义的图片。其他人答复道，想知道要完成逆向转换要花多久时间：转换图片表情为文本。

为了创纪录，我花了大约20分钟。大部分时间是花在研究已发布的自动转换图片表情的软件上了，然后编写了详尽的转换表。

这个脚本依赖于能自动生成表情图片的大多数软件有如下特征：这些软件会把相同含义的文字表情符号放在 `` 元素的 `alt` 属性里。所以，这个脚本实际上所做的是：如果 ALT 文本匹配预定义的常量列表中的某一条，就用图片的 ALT 文本来替换图片。

例 5.6. `frownies.user.js`

```

// ==UserScript==
// @name          Frownies
// @namespace      http://diveintogreasemonkey.org/download/
// @description    convert graphical smilies to their text
// @include        *
// ==/UserScript==

var smilies, images, img, replacement;
smilies = [":)", ":-)" ":-(", ":((", ";-)", ";)", ":-D", ":D",

```

```
":-/",  
    ":", "/", ":X", ":-X", ":\>", ":P", ":-P", ":O", ":-O", "X-(",  
    "X(", ":->", ":", ">", "B-", "B)", ">:", ":(((", ":-",  
    ((("  
        "(:))", ":-))", ":-|", "|:", "0:-)", "0:)", ":-B", ":B",  
";",  
    "I)", "I-)", "|-)", "|)", ":-&", ":&", ":-$", "$", "[-((",  
":0)",  
    ":@", "3:-O", ":(|)", "@};-", "**==", "(~)", "*-:)", "8-X",  
  
    "8X", "=:)", "<:) ", ";;) ", "=" ":*", "=", "-*", "=", "s", "=",  
":-s", "=", " :)", "=", " :-)", "=", "8- |", "=", "8|", "=", "8- }", "=",  
"8}", "=", " (: |", "=", " :-?", "=", " :?", "=", "#-o", "=", "#o", "=", ">",  
"~:>", "%%- ", "~O)", ":-L", ":L", "[-O<", "[o<", "=", "@-)", "=",  
"@)", "=", "$-)", "=", "$)", "=", "=>-)", ":-\\"", "^\0", "B-(",  
    "B(", " :)>- ", " [-X", "[X", "\\:D/", ">:D<", "(%)", "=", "  
(((" "#:-s", "=", "#:s", "=", "l-) ", "= l)", "=", "<:-p", "=", "  
<:p", "=", " :-ss", "=", ":ss", "=", " :-w", "=", ":w", "=", " :-<", "=", " :  
<=", "=", ">:P", ">:-P", ">:/", ";)))", ":-@", "^:)^",  
    ":-J", "(*)", ":GRIN:", ":-)", ":SMILE:", ":SAD:", ":EEK:",  
    ":SHOCK:", ":", "???:", "8)", "8-)", ":COOL:", ":LOL:", ":MAD:",  
    ":RAZZ:", ":OOPS:", ":CRY:", ":EVIL:", ":TWISTED:",  
":ROLL:",  
    ":WINK:", ":", "!", ":", "?:", ":", "IDEA:", ":", "ARROW:", ":", "NEUTRAL:",  
    ":MRGREEN:"];  
images = document.evaluate(  
    '//img[@alt]',  
    document,  
    null,  
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,  
    null);  
for (var i = 0; i < images.snapshotLength; i++) {  
    img = images.snapshotItem(i);  
    alt = img.alt.toUpperCase();  
    for (var j in smilies) {  
        if (alt == smilies[j]) {  
            replacement = document.createTextNode(alt);  
            img.parentNode.replaceChild(replacement, img);  
        }  
    }  
}
```

```
}
```

这段代码分为四个步骤：

1. 定义表情符号的列表(文本格式)。
2. 找出页面中所有含有 `alt` 属性的图片。
3. 对每一幅图像，检测它的 ALT 文本是否与列表中的某个 ASCII 表情符号匹配。
4. 如果匹配，就替换 `` 元素为包含 ASCII 表情的文本节点。

第一步简单的定义了一个列表，用 Javascript 的 `[]` 语法。

```
smilies = [":)", ":-)" ":-(", ":((", ";-)", ";)", ":-D", ":D",
":-/",
":/", ":X", ":-X", ":\>", ":P", ":-P", ":O", ":-O", "X-(",
"X(", ":->", ":", ">", "B-)", "B)", ">:)", ":((", ":((", ":-
(((",
":))", ":-))", ":-|", ":", "|", "O:-)", "O:)", ":-B", ":B",
"=;",
"I)", "I-)", "|-)", "|)", ":-&", ":", "&", ":-$", ":", "$", "[-((",
":O)",
":@)", "3:-O", ":(|)", "@};-", "**==", "(~~)", "*-:)", "8-
X",
"8X", "=:)", "<):)", ";;)", "=", ":", "*", "=", ":", "-*", "=", ":", "s", "=",
":-s", "=", ":", ":", ":", ":", ":", "8-|", "=", "8|", "=", "8-}", "=",
"8}", "=", ":", "|", "=", ":", "-?", "=", ":", "?", "=", "#-o", "=", "#o", "=", ">",
"~:~>", "%%-", "~o)", ":-L", ":L", "[-o<", "[o<", "=", "@-)", "=",
"@)", "=", "$-)", "=", "$)", "=", "=" ">-)", ":-\\", ":", "^o", "B-(",
"B(", ":", ">-", "[-X", "[X", "\\:D/", ">:D<", "(%)", "=", "="
(((", ":", "#:-s", "=", "#:s", "=", "l-)", "=", "l)", "=", "<:-p", "=", ":",
<:p", "=", ":", "-ss", "=", ":", "ss", "=", ":", "-w", "=", ":", "w", "=", ":", "-<", "=", ":",
<=", "=", ">:P", ">:-P", ">:/", ";))", ":-@", "^(^)",
":-J", "(*)", ":GRIN:", ":-)", ":SMILE:", ":SAD:", ":EEK:",
":SHOCK:", ":", "???:", "8)", "8-)", ":COOL:", ":LOL:", ":MAD:",
":RAZZ:", ":OOPS:", ":CRY:", ":EVIL:", ":TWISTED:",
":ROLL:",
":WINK:", ":", "!", ":", "?:", ":IDEA:", ":ARROW:", ":NEUTRAL:",
":MRGREEN:"];
```

接下来，我在页面中使用 XPath 查询语句搜索所有的含有 `alt` 属性的 `` 元素。有关 XPath 查询语句的更多信息，请阅读

操作所有有特定属性的元素

。

```
images = document.evaluate(
  '//img[@alt]',
  document,
  null,
  XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
  null);
```

第三步，遍历所有的 `` 元素,检验 `alt` 属性是否与我们定义的表情符号匹配。由于一些表情符号含有字母，我用`toUpperCase()`方法在比较前把 `alt` 属性转换为大写。

```
for (var i = 0; i < images.snapshotLength; i++) {
  img = images.snapshotItem(i);
  alt = img.alt.toUpperCase()
  for (var j in smilies) {
    if (alt == smilies[j]) {
      // ...
    }
  }
}
```

最后，我创建了一个包含文字表情符号的新文本节点，然后用它替换掉已有的 `` 元素。更多的信息，请阅读

替换元素为新内容

。

```
replacement = document.createTextNode(alt);
img.parentNode.replaceChild(replacement, img);
```

下载

- [frownies.user.js](#)

参见

- 操作所有有特定属性的元素
- 替换元素为新内容

第 6 章 高级话题

6.2. 在菜单栏上添加菜单项

Greasemonkey 定义了一个函数，`GM_registerMenuCommand`，让用户脚本可添加菜单项到 Firefox 菜单栏上。当用户脚本激活时，注册的菜单项会出现在 **User Script Commands** 子菜单中。

```
function GM_registerMenuCommand(menuText, callbackFunction);
```

`menuText` 参数是一个字符串，新增的菜单项的名称。`callbackFunction` 参数是函数对象。选中此菜单项，`callbackFunction` 函数就会被调用。

这个函数由 Greasemonkey 0.2.6 引入。您应该

参考资料

- [POST Interceptor](#) 新增了菜单项来切换脚本是否有效。

参见

- `GM_registerMenuCommand`

6.3. 整合其他网站的数据

Greasemonkey 定义了一个函数，`GM_xmlHttpRequest`，允许用户脚本使用从任意 **Illegal HTML tag removed** : URL `GET` (获取)数据，或者 `POST` (发送)数据到任意 **Illegal HTML tag removed** : URL。

更多细节和示例，请阅读

`GM_xmlHttpRequest`

。

这个函数是在 Greasemonkey 0.2.6 中引入的。您应该

测试它是否存在

，否则完全降低(degrade gracefully)。

参考资料

- [LibraryLookup](#) 整合 [Amazon.com](#) 和您的本地图书馆。
- [Annotate Google](#) 整合 [Google](#) 和 [del.icio.us](#)。
- [Bloglines Tweaks](#) 在 [Bloglines](#) 的文章汇总上添加了一个 **Expand** 按钮，可以获取并显示全部文章。
- [Flick Batch Enhancer](#) 使用 `GM_xmlHttpRequest`，利用 Flickr 自己的

Illegal HTML tag removed : REST

API，在 [Flickr](#) 上新增功能。

- [Hide Google Redirects](#) 改编 [Google 个人搜索历史](#)，使用普通的 `` 链接，但是仍然可以跟踪点击。在适当的跟踪

Illegal HTML tag removed : URL

上调用 `GM_xmlHttpRequest`。

6.4. 把您的用户脚本编译为扩展

想让您的用户脚本“长出”Greasemonkey 的架构吗？您的用户脚本需要访问 Javascript 特许函数，本地文件，或者只有全资格的扩展才具有的其它 Firefox 特性？可以把您的用户脚本转换为全资格的 `XPI`，仅仅点几下就行了。感谢 Adrian Holovaty 惊人的 [Greasemonkey 编译器](#)！

过程 6.1. 把 Butler 编译成浏览器扩展

1. 访问 [Butler 用户脚本源代码](#)。在菜单中，选择编辑 (E) → 全选 (A) (**Ctrl-A**)复制用户脚本源代码到剪贴板。
2. 访问[Greasemonkey 编译器](#)。
3. 转到“Javascript” field. 在菜单中，选择编辑 (E) → 粘贴 (P) (**Ctrl-V**)粘贴 Butler 用户脚本源代码。
4. 在“Creator(作者)”栏中，填入 `Mark Pilgrim`。
5. 在“Version(版本)”栏中，填入 Butler 用户脚本的当前版本(0.3截至发稿时， 否则请检查脚本的

元数据段

找到确切的版本)。

6. 打开一个新窗口或者标签，访问 [GUID Generator\(GUID 产生器\)](#)，产生一个随机的 GUID。把这个 GUID 复制到剪贴板，包含花括号。
7. 切换回“Greasemonkey 编译器”页面。在“GUID”栏中，粘贴刚才从 GUID 产生器上获得的 GUID。
8. 在“Homepage(主页)”栏中，填入 `http://diveintomark.org/projects/butler/`。
9. 点击 **Create the Firefox extension(创建 Firefox 扩展)**。Firefox 会弹出一个对话框：“打开 butler.xpi”。选择保存到磁盘，并且选择下载目录

好啦！您先在有 Butler 的浏览器扩展了。在安装 Butler 扩展前，应该先

禁用 Butler 用户脚本

然后 [Google](#) 一下禁用是否生效。然后安装 Butler 扩展，选择文件 **(F)** → 打开... **(O)** 然后选中用 Greasemonkey 编辑器创建的 `butler.xpi` 文件。务必在安装完成后重新启动浏览器。

`.xpi` 文件其实就是有确定的目录结构的 ZIP 压缩包。可以用任一 ZIP 程序(例如 Windows 下用 [7-zip](#) 或者在 Mac OS X 下用 [Stuffit Expander](#))解压缩文档，然后看下浏览器扩展是由哪些文件组成。

```
butler.xpi
|
+-- install.rdf
|
+-- chrome/
|
+-- butler/
|
+-- content/
|
+-- browser.xul
|
+-- contents.rdf
|
+-- javascript.js
```

这里有四个有关的文件。两个 RDF 文件是大多数 Firefox 的样板文件。其他两个包含有实现代码。

- `install.rdf`
- 与扩展自身有关的元数据，包括名称、版本、描述和兼容的 Firefox 版本。
- `browser.xul`
- 引导代码。检查对比当前的

Illegal HTML tag removed : URL

与脚本的 `@include` 和 `@exclude` 参数，然后注入执行脚本。

- `contents.rdf`
- 额外的元数据，Firefox 样板文件。
- `javascript.js`
- 原用户脚本的源代码。

基本上，Greasemonkey 编译器创建了真正的 Greasemonkey 缩减版，没有用户脚本，在适当的 **Illegal HTML tag removed : URL** 上自动加载一个用户脚本。但是如果所有的源文件，您可以自己创建自定义对话框、修改配置面板，或者注册 **User Script Commands** 以外的菜单项.....发狂了！

`value` 参数可以是字符串，布尔值或者整数。这两个参数都是必需的。

Greasemonkey 的配置值与浏览器 cookies 很相似，但也有重要的区别。两者都存放在本地电脑中，但是 cookies 是域名专用的而且只能被创建它们的域名访问，Greasemonkey 配置值是脚本专用的而且只能被创建这些参数的脚本访问(与执行用户脚本的地址也无关)。而且与 cookies 不同的是，配置值绝不会发送到远程服务器上去。

名称

GM_registerMenuCommand — 在 用户脚本命令 (C) 子菜单中添加菜单项

大纲

<code>function **GM_registerMenuCommand**</code> (Illegal HTML tag removed : commandName
	Illegal HTML tag removed : commandFunc
	Illegal HTML tag removed : accelKey
	Illegal HTML tag removed : accelModifiers
	Illegal HTML tag removed : accessKey

描述

`GM_registerMenuCommand` 在 工具 (T) → **Greasemonkey** → 用户脚本命令 (C) 子菜单中添加菜单项。 `commandName` 参数是一个字符串，新增的菜单项的名称。 `commandFunc` 参数是函数对象。选中此菜单项， `commandFunc` 函数就会被调用。 `accelKey` 单字符(例如：'g')或者键盘码，用来触发命令。 `accessKey` 字符串，列出了必须与 `accelKey` 同时按下的修饰键。如果多于一个功能键，用空格分开。例如，'shift' 或者 'ctrl alt'。可用的功能键有：shift, alt, meta, control 和 accel。 `accessKey` 单字符(例如：'g')当菜单打开时用来跳转到命令。它最好是 `commandName` 中的一个字母。

```
function **callbackFunction** ( Illegal HTML tag removed : e );
```

参数 `e` 是菜单选择事件。我不知道这有什么用。

历史

`GM_registerMenuCommand` 由 Greasemonkey 0.2.6 引入。

名称

`GM_xmlHttpRequest` — 进行任意的 **Illegal HTML tag removed : HTTP** 请求

大纲

```
**GM_xmlHttpRequest** ( Illegal HTML tag removed : details );
```

描述

`GM_xmlHttpRequest` 建立任意的 **Illegal HTML tag removed : HTTP** 请求。 [详细](#) 参数是一个有七个域的对象。

- `method`
- 字符串，请求使用的方法

Illegal HTML tag removed : HTTP

。必需。 通常使用 `GET`，但也可使任一个

Illegal HTML tag removed : HTTP

动词，包含 `POST`，`PUT` 和 `DELETE`。

- `url`
- 字符串，请求使用的

Illegal HTML tag removed : URL

。必需。

- `headers`
- 这个请求包含的

Illegal HTML tag removed : HTTP

头的关联数组。可选，默认为空字符串。例如：

```
headers: {'User-Agent': 'Mozilla/4.0 (compatible)
Greasemonkey',
'Accept': 'application/atom+xml,application/xml,text/xml'}
```

- `data`
- 字符串，

Illegal HTML tag removed : HTTP

请求的主体。可选，默认是空字符串。如果您在模拟提交表单(`method == 'POST'`)，在 `headers` 域中必需有 `Content-type`，值为 `'application/x-www-form-urlencoded'`，而且在 `data` 域包含有

Illegal HTML tag removed : URL

编码后的表单数据。

- `onload`
- 函数对象，请求成功完成调用的回调函数。
- `onerror`
- 函数对象，执行请求发生错误时调用的回调函数。
- `onreadystatechange`
- 函数对象，请求进行时反复调用的回调函数。

onload 回调

`onload` 的回调函数，有一个变量，`responseDetails`。

```
function **onloadCallback** (
```

Illegal HTML tag removed : responseDetails

`responseDetails` 是有五个域的对象。

- `status`
- 整数，

Illegal HTML tag removed : HTTP

应答的状态代码。 `200` 意为请求正常完成。

- `statusText`
- 字符串，

Illegal HTML tag removed : HTTP

状态文字。状态文字是依赖于服务器的。

- `responseHeaders`
- 字符串，应答包含的

Illegal HTML tag removed : HTTP

头部。

- `responseText`
- 字符串，应答的主体。
- `readyState`
- 未使用

onerror 回调

`onerror` 的回调函数，有一个参数， `responseDetails` 。

```
function **onerrorCallback** (
```

Illegal HTML tag removed : responseDetails

`responseDetails` 是有五个域的对象。

- `status`
- 整数，

Illegal HTML tag removed : HTTP

的错误代码。 `404` 意为页面无法找到。

- `statusText`
- 字符串，

Illegal HTML tag removed : HTTP

状态文字。状态文字是依赖于服务器的。

- responseHeaders
- 字符串，应答包含的

Illegal HTML tag removed : HTTP

头部。

- responseText
- 字符串，应答的主体。

Illegal HTML tag removed : HTTP

错误页面的主体是依赖于服务器的。

- readyState
- 未使用

onreadystatechange 回调

`onreadystatechange` 的回调函数，当请求正在进行时反复调用。有一个参数，`responseDetails`。

<pre>function **onreadystatechangeCallback** (</pre>	Illegal HTML tag removed : responseDetails
--	---

`responseDetails` 是有五个域的对象。`responseDetails.readyState` 指示了请求当前所在的阶段。

- status
- 整数，应答的

Illegal HTML tag removed : HTTP

状态代码。当 `responseDetails.readyState` `<` `4` 时，这个值是 `0`。

- statusText
- 字符串，

Illegal HTML tag removed : HTTP

状态文字。当 `responseDetails.readyState` `<` `4` 时，这个值是空字符串。

- responseHeaders
- 字符串，应答包含的

Illegal HTML tag removed : HTTP

头。当 `responseDetails.readyState` `<` `4` 时，这个值是空字符串。

- `responseText`
- 字符串， 应答的主体。当 `responseDetails.readyState` `<` `4` 时，这个值是空字符串。
- `readyState`
- 整数，the stage of the

Illegal HTML tag removed : HTTP

request。

- `1`
- 正在载入。请求已准备好。
- `2`
- 已加载。请求准备发送到服务器，但是还什么都没发。
- `3`
- 交互状态。请求已经发送，并且客户端等待服务器完成发送数据。
- `4`
- 完成。请求已完成，并且其他域中的所有应答数据已可用。

例子

下面的代码从<http://greaseblog.blogspot.com/>获取 Atom feed，然后用警告窗口显示结果。

```
GM_xmlHttpRequest({
  method: 'GET',
  url: 'http://greaseblog.blogspot.com/atom.xml',
  headers: {
    'User-agent': 'Mozilla/4.0 (compatible) Greasemonkey',
    'Accept': 'application/atom+xml,application/xml,text/xml',
  },
  onload: function(responseDetails) {
    alert('Request for Atom feed returned ' + responseDetails.status
    +
    ' ' + responseDetails.statusText + '\n\n' +
    'Feed data:\n' + responseDetails.responseText);
  }
});
```

Bugs

`onreadystatechange` 回调函数当 `readyState` `< 4` 时，工作不正常。

注释

与 `XMLHttpRequest` 对象不同，`GM_xmlHttpRequest` 并不限制与当前域名；它能够从任何 **Illegal HTML tag removed : URL** `GET` 或者 `POST` 数据。

历史

`GM_xmlHttpRequest` 由 Greasemonkey 0.2.6 引入。

参考资料

- `XMLHttpRequest` 在 Mozilla 中的支持
- `XMLHttpRequest` 在 Internet Explorer 中的支持
- `XMLHttpRequest` 在 Safari 中的支持
- [

Illegal HTML tag removed : HTTP

状态代码](<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>)

- [RFC 2616](#)

名称

`GM_openInTab` — 在新标签中打开指定的 **Illegal HTML tag removed : URL**

大纲

```
function **GM_openInTab** ( Illegal HTML tag removed : url ) ;
```

描述

`GM_openInTab` 在新标签中打开指定的 **Illegal HTML tag removed : URL**。

历史

`GM_openInTab` 由 Greasemonkey 0.5 beta 引入。

名称

`GM_addStyle` — 给页面添加 **Illegal HTML tag removed : CSS** 样式

大纲

```
function **GM_addStyle** ( Illegal HTML tag removed : CSS ) ;
```

描述

`GM_addStyle` 给页面添加样式。创建一个新的 `<style>` 元素，在其中加入给定的 **Illegal HTML tag removed : CSS**，然后插入到 `<head>` 中。

历史

`GM_addStyle` 由 Greasemonkey 0.6 引入。

参见

- 添加 CSS 样式

“参考资料”链接清单

|



当您在本书中看到“参考资料”链接列表时，您可以点“参考资料”的标题就可以跳转到这个所有参考资料链接清单。

1. [Greasemonkey 脚本库](#)有上百个 Greasemonkey 脚本。

1.3. 安装用户脚本

2. [tag:

Illegal HTML tag removed : URI

s](<http://taguri.org/>)

2.2. 用元数据描述您的用户脚本

3. [Javascript 中的匿名函数](#)

2.3. 编写用户脚本代码

4. [Block Scope in Javascript](#) 和 [associated discussion thread](#)

2.3. 编写用户脚本代码

5. [DOM 查看器介绍](#)

3.3. 用 DOM 查看器查看元素

6. [Inspect Element 扩展](#)

3.3. 用 DOM 查看器查看元素

7. [Inspector Widget 扩展](#) 具有 [Inspect Element](#) 相同功能的扩展，它增加了一个工具条，而不是一个菜单项。

3.3. 用 DOM 查看器查看元素

8. [Web Developer 扩展](#) 有很多分析页面的函数。

3.5. 其他调试工具

9. [Aardvark](#) 互动的显示标签名称、`id` 和 `class` 的属性。

3.5. 其他调试工具

10. [Venkman Javascript Debugger](#) 完整的 Javascript 运行时调试器。

3.5. 其他调试工具

11. [Web Development Bookmarklets](#) 有一些有用的函数，可以把它们拖放到工具栏中。

3.5. 其他调试工具

12. [JSUnit](#) 是 Javascript 单元测试框架。

3.5. 其他调试工具

13. [js-lint](#) 可检查出 Javascript 代码的常见错误。

3.5. 其他调试工具

14. [Mozilla XPath 文档](#)

4.6. 操作所有有特定属性的元素

15. [XPath 实例教程](#)

4.6. 操作所有有特定属性的元素

16. [XPathResult 参考手册](#)

4.6. 操作所有有特定属性的元素

17. [CSS 属性](#)

4.15. 设置元素样式

18. [Javascript 事件兼容性表](#)

4.22. 覆盖内建的 Javascript 方法

19. [Mozilla 中的 Javascript-DOM 原型](#)

4.22. 覆盖内建的 Javascript 方法

20. [显示事件对象常量](#)

4.22. 覆盖内建的 Javascript 方法

21. [Event 文档](#)

5.7. 案例：Zoom Textarea

22. [MyPIPsTag](#) 首次运行时提示您输入用户名。

6.1. 存取持久数据

23. [POST Interceptor](#) 添加菜单项(使用

```
GM_registerMenuCommand
```

)来切换脚本是否生效。

6.1. 存取持久数据

24. [

Illegal HTML tag removed : MSDN

Language Filter]

(<http://blog.monstuff.com/archives/http://mit00.02753.com/paper/greasemonkey/images/MSDNLanguageFilter.user.js>) 插入设置选项到页面中。

6.1. 存取持久数据

25. [POST Interceptor](#) 新增了菜单项来切换脚本是否有效。

6.2. 在菜单栏上添加菜单项

26. [LibraryLookup](#) 整合 [Amazon.com](#) 和您的本地图书馆。

6.3. 整合其他网站的数据

27. [Annotate Google](#) 整合 [Google](#) 和 [del.icio.us](#)。

6.3. 整合其他网站的数据

28. [Bloglines Tweaks](#) 在 [Bloglines](#) 的文章汇总上添加了一个 **Expand** 按钮，可以获取并显示全部文章。

6.3. 整合其他网站的数据

29. [Flick Batch Enhancer](#) 使用 `GM_xmlHttpRequest`，利用 Flickr 自己的

Illegal HTML tag removed : REST

API，在 [Flickr](#) 上新增功能。

6.3. 整合其他网站的数据

30. [Hide Google Redirects](#) 改编 [Google 个人搜索历史](#)，使用普通的 `` 链接，但是仍然可以跟踪点击。在适当的跟踪

Illegal HTML tag removed : URL

上调用 `GM_xmlHttpRequest`。

6.3. 整合其他网站的数据

31. [扩展开发者的扩展](#)，调试和测试 Firefox 扩展的无价之宝。

6.4. 把您的用户脚本编译为扩展

32. `XMLHttpRequest` 在 [Mozilla](#) 中的支持

`GM_xmlHttpRequest`

33. `XMLHttpRequest` 在 [Internet Explorer](#) 中的支持

`GM_xmlHttpRequest`

34. `XMLHttpRequest` 在 [Safari](#) 中的支持

`GM_xmlHttpRequest`

35. [

Illegal HTML tag removed : HTTP

状态代码](<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>)

GM_xmlHttpRequest

36. RFC 2616

GM_xmlHttpRequest

技巧清单

|



当您在本书中看到技巧或者警告时，您可以点击前面的提示小图标，然后跳转到此页。

1. 您可以在[Greasemonkey脚本库](#)，找到许多用户脚本。尽管没人要求您必须把脚本放到那儿去，但是实际上，您可以把您的脚本共享到任何地方，这样别人就可以安装它了。甚至您根本不需要一台网络服务器，因为你可以从本地文件中安装用户脚本。

1.3. 安装用户脚本

2. 元数据可以以任意次序排列。笔者推荐使用 `@name`，`@namespace`，`@description`，`@include`，最后是 `@exclude`，但是其它的顺序也没关系。

2.2. 用元数据描述您的用户脚本

3. 在“管理用户脚本”对话框中，点击编辑，您正在“动态”修改脚本的副本，它在 Firefox 的个人配置文件夹中。我形成了一个习惯，每“动态”修改完毕，又回到编辑器，选择文件 **(F)** → 另存为 **(a)...**，把用户脚本保存到另一个文件夹中。尽管这不是必须的 (Greasemonkey 只会用您配置文件夹中的那个脚本)，但是我喜欢在完成全部修改后把脚本在其它文件夹里保存一个“原本”。

2.4. 修改用户脚本

4. 在错误控制台中可以用右键 (Mac 用户用 **control-click**) 点击任意行选中，然后选择复制 **(C)**，将信息复制到剪贴板。

3.2. 用 `GM_log` 记日志

5. 如果已有一个元素的引用 (例如 `thisElement`)，可以用 `thisElement.nodeName` 来判断它的 **Illegal HTML tag removed : HTML** 标签。如果页面被当作 `text/html` 类型，标签名称就总是大写，不论它在原始页面是如何定义的。如果页面被当作

`application/xhtml+xml` 类型，那么标签名称就总是小写的。我总是用 `thisElement.nodeName.toUpperCase()` 这样我就可以不用管这些了。

4.6. 操作所有有特定属性的元素

- 即使 `someExistingElement` 是它的父元素的最后一个孩子(在它之后没有下一个元素)，仍然可以在 `someExistingElement.nextSibling` 之前插入新内容。在这种情况下，`someExistingElement.nextSibling` 将返回一个空值，`insertBefore` 函数将把新内容追加到最后。(以许这对你来说没有太大意义，但我想让您知道的是，尽管这种方法似乎是不太对，但它却总是有效的。)

4.8. 在元素后插入内容

- 如果要移除广告，比起写用户脚本来，更容易的办法是安装 [AdBlock](#) 然后导入 [最新的过滤列表](#)。

4.9. 删除元素

- 使用 `data:`

Illegal HTML tag removed : URI

[kitchen](#) 来建立您自己的 `data: Illegal HTML tag removed : URLs`。

4.12. 在没有服务器的情况下添加图片

- 可以使用 `addGlobalStyle` 函数样式化插入到页面中的元素或者部分原页面中已有的元素。然而，如果要样式化已有的元素，应该在您定义的每条规则上使用 `! important` 关键字，保证您的样式覆盖原页面中已定义的规则。

4.13. 添加 CSS 样式

- 删除和替换 `document.body.innerHTML` 并不会更改页面。原页面在 `<head>` 中定义的所有内容仍然是有效的，包括页面的标题，CSS 样式，以及脚本。可以分开修改或者移除。

4.16. 处理已渲染的页面

- 正常情况下，当用户提交一个表单时，例如，点击表单中的提交按钮或者按回车键)，都会触发 `submit` 事件。但是，当脚本调用 `aForm.submit()` 提交表单时，却不会触发 `submit` 事件。因此，您必须做两件事来捕获表单的提交事件：给 `submit` 事件增加事件监听，并且修改 `HTMLFormElement` 类的原型来重定向 `submit()` 方法到您的自定义函数上。

4.22. 覆盖内建的 Javascript 方法

- 访问 `about:config`，然后再过滤器中填 `greasemonkey.scriptvals`，就可以查看已存储的配置值。


GM_getValue

13. 当您在本书中看到“参考资料”链接列表时，您可以点“参考资料”的标题就可以跳转到这个所有参考资料链接清单。

“参考资料”链接清单

14. 当您在本书中看到技巧或者警告时，您可以点击前面的提示小图标，然后跳转到此页。

技巧清单

15. 当您在本书中看到视频的链接时，点击提示  小图标，可以跳转到全部视频清单。

步骤清单

实例清单

1. 例 2.1. `helloworld.user.js`
2. 例 2.2. Hello World 元数据
3. 例 2.3. 显示“Hello world!”提示信息
4. 例 2.4. 延迟调用函数的错误方法
5. 例 2.5. 延迟调用函数的更好方法
6. 例 2.6. 延迟调用函数的最好方法
7. 例 3.1. 记录到错误控制台然后继续(`gmlog.user.js`)
8. 例 3.2. Javascript Shell 介绍
9. 例 3.3. 获取元素属性
10. 例 4.1. 匹配域名和它所有子域名的元数据标签
11. 例 4.2. `GM_xmlHttpRequest` 函数无效时警告用户
12. 例 4.3. 检查页面中是否有 `<textarea>` 元属
13. 例 4.4. 遍历所有元素
14. 例 4.5. 获取页面上所有的 `textarea`
15. 例 4.6. 获取页面中的所有链接
16. 例 4.7. 获取所有具有 `title` 属性的元素
17. 例 4.8. 获取所有 `class` 为 `sponsoredlink` 的 `<div>`

- 18. 例 4.9. `xpath` 函数
- 19. 例 4.10. 在主内容前插入 `<hr>`
- 20. 例 4.11. 在导航条后面插入 `<hr>`
- 21. 例 4.12. 删除广告侧边栏
- 22. 例 4.13. 替换图片为替代(`alt`)文本
- 23. 例 4.14. 在页面顶部加上标语
- 24. 例 4.15. 在页面顶部添加图形 logo
- 25. 例 4.16. 放大段落文字
- 26. 例 4.17. 获取由元素的 `style` 属性定义的样式
- 27. 例 4.18. 获取元素的真实样式
- 28. 例 4.19. 设置单个元素的样式
- 29. 例 4.20. 替换全部页面为自定义的内容
- 30. 例 4.21. 获取 `method` 为 "POST" 或 "post" 的表单(form)
- 31. 例 4.22. 获取当前域名
- 32. 例 4.23. 在链接末尾添加查询参数
- 33. 例 4.24. 重定向一个站点到它的安全站点
- 34. 例 4.25. 当用户点击页面上任意地方时做点事
- 35. 例 4.26. 当表单提交时做点事
- 36. 例 4.27. 将任意字符串解析为

Illegal HTML tag removed : XML

- 37. 例 4.28. 解析远程源的

Illegal HTML tag removed : XML

- 38. 例 5.1. 重定向 GMail 到等同的 `https://` 地址
- 39. 例 5.2. 让 Bloglines 自动显示所有未读条目
- 40. 例 5.3. `aintitreadable.user.js`
- 41. 例 5.4. `offsiteblank.user.js`
- 42. 例 5.5. `dumbquotes.user.js`

- 43. 例 5.6. `frownies.user.js`
- 44. 例 5.7. `zoomtextarea.user.js`
- 45. 例 5.8. `accessbar.user.js`

步骤清单

|



当您在本书中看到视频的链接时，点击提示  小图标，可以跳转到全部视频清单。

- 1. 过程 1.1. 安装 Greasemonkey 扩展
- 2. 过程 1.2. 安装 Butler 用户脚本
- 3. 过程 1.3. 暂时禁用 Butler
- 4. 过程 1.4. 卸载 Butler
- 5. 过程 1.5. 重新配置 Butler 排除 Froogle
- 6. 过程 2.1. 编辑 Hello World 的源代码然后观察运行结果
- 7. 过程 3.1. 打开错误控制台
- 8. 过程 3.2. 安装 DOM 查看器
- 9. 过程 3.3. 查看和编辑深入浅出 Greasemonkey的主页
- 10. 过程 3.4. 用 Inspect Element 直接查看元素
- 11. 过程 3.5. 安装 Javascript Shell
- 12. 过程 6.1. 把 Butler 编译成浏览器扩展

修订历史

修订历史	
修订 2005-05-09	2005-05-09

- 新增
 案例：Zoom Textarea

。

- 新增

存取持久数据

。

- 新增

在菜单栏上添加菜单项

。

- 新增

整合其他网站的数据

。

- 新增

把您的用户脚本编译为扩展

。

|| 修订 2005-05-06 | 2005-05-06 ||

- 新增

案例：Frownies

。

- 修改

Greasemonkey API 参考

中的 `void` 为 `function` 。（“您可以继续使用这一词。我不认为这就是您的想法。”）

|| 修订 2005-05-05 | 2005-05-05 ||

- 新增

Greasemonkey 是什么？

. 感谢 Dennis 。

- 新增

案例：Dumb Quotes

。

- 新增可下载的 Palm OS™ 数据库，可在移动设备上阅读。

|| 修订 2005-05-04 | 2005-05-04 || **Illegal HTML tag removed : XML** || 修订 2005-05-03 | 2005-05-03 || **Illegal HTML tag removed : HTML** || 修订 2005-05-02 | 2005-05-02 ||

- 新增

GM_xmlHttpRequest

。

- 新增

案例：Ain't It Readable

。

- 重构 [Offsite Blank](#)。
- 更新代码示例为 Greasemonkey 0.3 格式。

|| 修订 2005-05-01 | 2005-05-01 ||

- 新增

Greasemonkey API 参考

。

- 纳入 Simon Willison 的大量反馈。感谢 Simon。
- 纳入 Jeremy Dunck 的大量反馈。感谢 Jeremy。

|| 修订 2005-04-30 | 2005-04-30 ||

- 新增

开始

。

- 新增

您的第一个用户脚本

。

- 新增

调试用户脚本

。

- 新增

视频

(现在只能在线观看)。

|| 修订 2005-04-25 | 2005-04-25 ||

- 初次出版

|

关于本书

我使用 [Emacs](#)，在 [DocBook](#)

Illegal HTML tag removed : XML

中撰写了此书。[SAXON](#) 把它转换为了 **Illegal HTML tag removed : HTML** (使用了 Norman Walsh 的 [Norman Walsh 的

Illegal HTML tag removed : XSL

样式表](<http://docbook.sourceforge.net/projects/xsl/index.html>)的自定义版本)。[\[HTMLDoc\]](#) (<http://www.easysw.com/htmldoc/>) 把它转为了 **Illegal HTML tag removed : PDF**。[w3m](#) 把它转为了纯文本。[Plucker Distiller](#) 把它转换为了 Palm OS™ 数据库所以您能在移动设备上阅读它。由 [Ant](#) 脚本自动化了上述过程。

我用 [CamStudio](#) 和 [TMPGEnc](#) 制作了附带的视频。

Dean Edwards 写了 [js-highlight](#)。它提供了 Javascript 范例的自动语法高亮的功能。

如果您对学习 DocBook 的技术写作感兴趣，您可以下载[这本书的 XML 源代码](#)，其中包含此书的 DocBook 版本和所有 **Illegal HTML tag removed : XSL** 样式表以及我用来生成此书的不同版本的安装脚本。而且，您也该阅读下这本权威的书 [DocBook: The Definitive Guide](#)。您也可以订阅[DocBook 邮件列表](#)。

简体中文版主要翻译人员：[Fiag\(fiag.hit at gmail dot com\)](#)；[Drsu](#)；[Baggio](#)。

由于 HTMLDoc 不支持中文，所以暂时无中文PDF文档。

GNU 通用公共许可证

1991年6月 第二版

版权 © 1989, 1991 Free Software Foundation, Inc.

Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

允许每个人复制和发布这一许可证原始文档的副本，但绝对不允许对它进行任何修改。

Version 2, June 1991

1. 序言

大多数软件许可证决意剥夺你的共享和修改软件的自由。对比之下，GNU通用公共许可证力图保证你的共享和修改自由软件的自由——保证自由软件对所有用户是自由的。GPL适用于大多数自由软件基金会的软件，以及由使用这些软件而承担义务的作者所开发的软件。(自由软件基金会的其他一些软件受GNU库通用许可证的保护)。你也可以将它用到你的程序中。

当我们谈到自由软件(free software)时，我们指的是自由而不是价格。我们的GNU通用公共许可证决意保证你有发布自由软件的自由(如果你愿意，你可以对此项服务收取一定的费用)；保证你能收到源程序或者在你需要时能得到它；保证你能修改软件或将它的一部分用于新的自由软件；而且还保证你知道你能做这些事情。

为了保护你的权利，我们需要作出规定：禁止任何人不承认你的权利，或者要求你放弃这些权利。如果你修改了自由软件或者发布了软件的副本，这些规定就转化为你的责任。

例如，如果你发布这样一个程序的副本，不管是收费的还是免费的，你必须将你具有的一切权利给予你的接受者；你必须保证他们能收到或得到源程序；并且将这些条款给他们看，使他们知道他们有这样的权利。

我们采取两项措施来保护你的权利：

1. 给软件以版权保护。
2. 给你提供许可证。它给你复制，发布和修改这些软件的法律许可。

同样，为了保护每个作者和我们自己，我们需要清楚地让每个人明白，自由软件没有担保(no warranty)。如果由于其他某个人修改了软件，并继续加以传播。我们需要它的接受者明白：他们所得到的并不是原来的自由软件。由其他人引人的任何问题，不应损害原作者的声誉。

最后，任何自由软件不断受到软件专利的威胁。我们希望避免这种风险：自由软件的再发布者以个人名义获得专利许可证，从而将软件变为私有。为防止这一点，我们必须明确：任何专利必须以允许每个人自由使用为前提，否则就不准许有专利。

以下是有关复制，发布和修改的条款和条件。

2. 有关复制，发布和修改的条款和条件

第 0 款

此许可证适用于任何包含版权所有者声明的程序和其他作品，版权所有者在声明中明确说明程序和作品可以在GPL条款的约束下发布。下面提到的“程序”指的是任何这样的程序或作品。而“基于程序的作品”指的是程序或者任何受版权法约束的衍生作品。也就是说包含程序或程序的一部分的作品。可以是原封不动的，或经过修改的和/或翻译成其他语言的(程序)。(在下文中，翻译包含在“修改”的条款中。) 每个许可证接受人(licensee)用“你”来称呼。

许可证条款不适用于复制，发布和修改以外的活动。这些活动超出这些条款的范围。运行程序的活动不受条款的限止。仅当程序的输出构成基于程序作品的内容时，这一条款才适用(如果只运行程序就无关)。是否普遍适用取决于程序具体用来做什么。

第 1 款

只要你在每一副本上明显和恰当地出版版权声明和不承担担保的声明，保持此许可证的声明和没有担保的声明完整无损，并和程序一起给每个其他的程序接受者一份许可证的副本，你就可以用任何媒体复制和发布你收到的原始的程序的源代码。

你可以为转让副本的实际行动收取一定费用。你也有权选择提供担保以换取一定费用。

第 2 款

你可以修改程序的一个或几个副本或程序的任何部分，以此形成基于程序的作品。只要你同时满足下面的所有条件，你就可以按前面

第 1 款

的要求复制和发布这一经过修改的程序或作品：

1. 你必须在修改的文件中附有明确的说明：你修改了这一文件及具体的修改日期。
2. 你必须使你发布或出版的作品(它包含程序的全部或一部分，或包含由程序的全部或部分衍生的作品)允许第三方作为整体按许可证条款免费使用。
3. 如果修改的程序在运行时以交互方式读取命令，你必须使它在开始进入常规的交互使用方式时打印或显示声明：包括适当的版权声明和没有担保的声明(或者你提供担保的声明)；用户可以按此许可证条款重新发布程序的说明；并告诉用户如何看到这一许可证的副本。

|



| 如果原始程序以交互方式工作，它并不打印这样的声明，你的基于程序的作品也就不需要打印声明 || --- | --- |

这些要求适用于修改了的作品整体。如果能够确定作品的一部分并非程序的衍生产品，可以合理地认为这部分是独立的，是不同的作品。当你将它作为独立作品发布时，它不受此许可证和它的条款的约束。但是当你将这部分作为基于程序的作品的一部分发布时，作为整体

它将受到许可证条款约束。准予其他许可证持有人的使用范围扩大到整个产品。也就是每个部分，不管它是谁写的。

因此，本条款的意图不在于索取权利；或剥夺全部由你写成的作品的权利。而是履行权利来控制基于程序的集体作品或衍生作品的发布。

此外，将与程序无关的作品和该程序或基于程序的作品一起放在存贮体或发布媒体的同一卷上，并不导致将其他作品置于此许可证的约束范围之内。

第 3 款

你可以以目标码或可执行形式复制或发布程序(或符合

第 2 款

的基于程序的作品)，只要你遵守前面的第

1

,

2

款，并同时满足下列3条中的1条：

1. 在通常用作软件交换的媒体上，和目标码一起附有机可读的完整的源码。这些源码的发布应符合上面第1，2款的要求。或者
2. 在通常用作软件交换的媒体上，和目标码一起，附有给第三方提供相应的机器可读的源码的书面报价。有效期不少于3年，费用不超过实际完成源程序发布的实际成本。源码的发布应符合上面的第

1

,

2

款的要求。或者

3. 和目标码一起，附有你收到的发布源码的报价信息。(这一条款只适用于非商业性发布，而且你只收到程序的目标码或可执行代码和按第2项要求提供的报价。)

作品的源码指的是对作品进行修改最优先择取的形式。对可执行的作品讲，完整的源码包括：所有模块的所有源程序，加上有关的接口的定义，加上控制可执行作品的安装和编译的 script。作为特殊例外，发布的源码不必包含任何常规发布的供可执行代码在上面运行的操作系统的主要组成部分(如编译程序，内核等)。除非这些组成部分和可执行作品结合在一起。

如果采用提供对指定地点的访问和复制的方式发布可执行码或目标码，那么，提供对同一地点的访问和复制源码可以算作源码的发布，即使第三方不强求与目标码一起复制源码。

第 4 款

除非你明确按许可证提出的要求去做，否则你不能复制，修改，转发许可证和发布程序。任何试图用其他方式复制，修改，转发许可证和发布程序是无效的。而且将自动结束许可证赋予你的权利。然而，对那些从你那里按许可证条款得到副本和权利的人们，只要他们继续全面履行条款，许可证赋予他们的权利仍然有效。

第 5 款

你没有在许可证上签字，因而你没有必要一定接受这一许可证。然而，没有任何其他东西赋予你修改和发布程序及其衍生作品的权利。如果你不接受许可证，这些行为是法律禁止的。因此，如果你修改或发布程序(或任何基于程序的作品)，你就表明你接受这一许可证以及它的所有有关复制，发布和修改程序或基于程序的作品条款和条件。

第 6 款

每当你重新发布程序(或任何基于程序的作品)时，接受者自动从原始许可证颁发者那里接受这些条款和条件支配的复制，发布或修改程序的许可证。你不可以对接受者履行这里赋予他们的权利强加其他限制。你也没有强求第三方履行许可证条款的义务。

第 7 款

如果由于法院判决或违反专利的指控或任何其他原因(不限于专利问题)的结果，强加于你的条件(不管是法院判决，协议或其他)和许可证的条件有冲突。他们也不能用许可证条款为你开脱。在你不能同时满足本许可证规定的义务及其他相关的义务时，作为结果，你可以根本不发布程序。例如，如果某一专利许可证不允许所有那些直接或间接从你那里接受副本的人们在不付专利费的情况下重新发布程序，唯一能同时满足两方面要求的办法是停止发布程序。

如果本条款的任何部分在特定的环境下无效或无法实施，就使用条款的其余部分。并将条款作为整体用于其他环境。

本条款的目的不在于引诱你侵犯专利或其他财产权的要求，或争论这种要求的有效性。本条款的主要目的在于保护自由软件发布系统的完整性。它是通过通用公共许可证的应用来实现的。许多人坚持应用这一系统，已经为通过这一系统发布大量自由软件作出慷慨的供献。作者／捐献者有权决定他／她是否通过任何其他系统发布软件。许可证持有人不能强制这种选择。

本节的目的在于明确说明许可证其余部分可能产生的结果。

第 8 款

如果由于专利或者由于有版权的接口问题使程序在某些国家的发布和使用受到限止，将此程序置于许可证约束下的原始版权拥有者可以增加限止发布地区的条款，将这些国家明确排除在外。并在这些国家以外的地区发布程序。在这种情况下，许可证包含的限止条款和许可证正文一样有效。

第 9 款

自由软件基金会可能随时出版通用公共许可证的修改版或新版。新版和当前的版本在原则上保持一致，但在提到新问题时或有关事项时，在细节上可能出现差别。

每一版本都有不同的版本号。如果程序指定适用于它的许可证版本号以及“任何更新的版本”。你有权选择遵循指定的版本或自由软件基金会以后出版的新版本，如果程序未指定许可证版本，你可选择自由软件基金会已经出版的任何版本。

第 10 款

如果你愿意将程序的一部分结合到其他自由程序中，而它们的发布条件不同。写信给作者，要求准予使用。如果是自由软件基金会加以版权保护的软件，写信给自由软件基金会。我们有时会作为例外的情况处理。我们的决定受两个主要目标的指导。这两个主要目标是：我们的自由软件的衍生作品继续保持自由状态。以及从整体上促进软件的共享和重复利用。

第 11 款

由于程序准予免费使用，在适用法准许的范围内，对程序没有担保。除非另有书面说明，版权所有者和／或其他提供程序的人们“一样”不提供任何类型的担保。不论是明确的，还是隐含的。包括但不限于隐含的适销和适合特定用途的保证。全部的风险，如程序的质量和性能问题都由你来承担。如果程序出现缺陷，你承担所有必要的服务，修复和改正的费用。

第 12 款

除非适用法或书面协议的要求，在任何情况下，任何版权所有者或任何按许可证条款修改和发布程序的人们都不对你的损失负有任何责任。包括由于使用或不能使用程序引起的任何一般的，特殊的，偶然发生的或重大的损失(包括但不限于数据的损失，或者数据变得不精确，或者你或第三方的持续的损失，或者程序不能和其他程序协调运行等)。即使版权所有者和其他人提到这种损失的可能性也不例外。

条款和条件结束

3. 如何将这些条款用到你的新程序

如果你开发了新程序，而且你需要它得到公众最大限度的利用。要做到这一点的最好办法是将它变为自由软件。使得每个人都能在遵守条款的基础上对它进行修改和重新发布。

为了做到这一点，给程序附上下列声明。最安全的方式是将它放在每个源程序的开头，以便最有效地传递拒绝担保的信息。每个文件至少应有“版权所有”行以及在什么地方能看到声明全文的说明。

<用一行空间给出程序的名称和它用来做什么的简单说明。> 版权所有 (C) <年份> <作者姓名>

这一程序是自由软件，你可以遵照自由软件基金会出版的GNU通用公共许可证条款来修改和重新发布这一程序。或者用许可证的第二版，或者(根据你的选择)用任何更新的版本。

发布这一程序的目的是希望它有用，但没有任何担保。甚至没有适合特定目的的隐含的担保。更详细的情况请参阅GNU通用公共许可证。

你应该已经和程序一起收到一份GNU通用公共许可证的副本。如果还没有，写信给：The Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

当然，请加上通过电子或者纸质邮件联系你的信息。

如果程序以交互方式进行工作，当它开始进入交互方式工作时，使它输出类似下面的简短声明：

Gnomovision第69版，版权所有(C) 年份，作者姓名 Gnomovision绝对没有担保。要知道详细情况，请输入‘show w’。这是自由软件，欢迎你遵守一定的条件重新发布它，要知道详细情况，请输入‘Show c’。

假设的命令‘show w’和‘show c’应显示通用公共许可证的相应条款。当然，你使用的命令名称可以不同于‘show w’和‘show c’。根据你的程序的具体情况，也可以用菜单或鼠标选项来显示这些条款。

如果需要，你应该取得你的上司(如果你是程序员)或你的学校签署放弃程序版权的声明。下面只是一个例子，你应该改变相应的名称：

Ynyodyne公司以此方式放弃 James Harker 所写的 Gnomovision 程序的全部版权利益。

，1989年4月1日 Ty Coon, 副总裁

这一许可证不允许你将程序并入专用程序。如果你的程序是一个子程序库。你可能会认为用库的方式和专用应用程序连接更有用。如果这是你想做的事，使用GNU库通用公共许可证代替本许可证。

appdir @ssv

1.1. Greasemonkey 是什么？

Greasemonkey 是一个 Firefox 扩展，它具有通过编写脚本来改变被访问网页的功能。使用它，能使您访问的网站更便于阅读或者更便于使用。使用它，您能修复网页渲染的缺陷，而无需烦扰网站管理员。使用它，您能让网页更好地使用残疾人援助技术，清楚响亮地说出网页内容，或者将网页内容变为盲文。使用它，您能自动地获得其它网站的数据，从而使两个网站更好地相互链接起来。

然而 Greasemonkey 本身并没有作这些事。实际上，在您安装它之后，您注意到根本没有任何变动...直到您开始安装一种叫做“用户脚本”的东西。用户脚本（user script）就是一大块 Javascript 代码，还有些附加信息，用来告诉 Greasemonkey 脚本应该在何时何地运行。每个用户脚本能够针对具体页面，具体网站，或者一批网站。用户脚本能做到您在 Javascript 中可做到的任何事情。实际上，它能做得更多，因为 Greasemonkey 提供了专供用户脚本使用的函数。

这里是[Greasemonkey 脚本库](#)，含了上百个用户脚本，这些都是用户为了满足自己的需要而写的。一旦您写了自己的用户脚本，只要您认为别人也许发现它有用，您可以把它添加到脚本库中。您也可以自己使用，因为从编写过程中获得知识，获得满足感，才是更重要的。

这是[Greasemonkey 的邮件列表](#)，您可以在那里提问，发表用户脚本，和讨论新特性的想法。Greasemonkey 开发人员常去这个邮件列表；他们也许会回答您的问题！

为什么写这本书？

深入浅出 Greasemonkey 是从 Greasemonkey 邮件列表中的用户讨论和作者本人编写用户脚本的经验中发展而来。仅仅一个星期，作者就发现，新用户经常会提出重复的问题，而这些问题是被回答过的。此外，在写了几个用户脚本以后，作者发现，一些常用的模式，以及可以解决某一类问题的成块的可重用代码会反复出现。因此，作者开始整理这些有用的模式，解释编程思路，同时作者也从中写作中得到不少锻炼。

如果没有 Greasemonkey 的开发者 Aaron Boodman 和 Jeremy Dunck 的大力帮助，没有那些对我的初稿提出宝贵的反馈建议的用户，就不会有现在的这本书。谢谢大家。

1.2. 安装 Greasemonkey

要写用户脚本，您首先要安装 Greasemonkey 扩展，0.3 或以上版本。

安装 Greasemonkey 扩展

Chrome

在 chrome 上安装 [Tampermonkey](#)。

火狐

在 Firefox 上安装 [Greasemonkey](#)。

一般来说，安装好 Greasemonkey，(除了四个菜单项外)并不会给浏览器添加任何功能。但是它能让您添加别的东西，名叫“用户脚本”，用户脚本可以用来定制指定的网页。

2.2. 用元数据描述您的用户脚本

每个用户脚本都含有一段元数据，用来向 Greasemonkey 描述这个脚本自身的信息：发行者，执行规则等等。

```
// ==UserScript==
// @name           Hello World
// @namespace      http://diveintogreasemonkey.org/download/
// @description    example script to alert "Hello world!" on every page
// @include        *
// @exclude        http://diveintogreasemonkey.org/*
// @exclude        http://www.diveintogreasemonkey.org/*
// ==/UserScript==
```

这里有六条独立的元数据信息，作为一个整体包含在注释中。元数据以 `// ==UserScript==` 开始，以 `// ==/UserScript==` 结束，其内容要包含在 `//` 注释里面。这段注释可以放在用户脚本的任何部位，但经常会放在靠近顶部的地方。

现在让我们按顺序逐条解释。

字段名以 `@` 开始

字段名	必要	多次定义	说明	格式
name	N	N	名字	
namespace	N	N	命名空间	
description	N	N	描述	
include		Y	包含	通配符
exclude		Y	排除	通配符

namespace

用它来区分名称相同但是作者不同的用户脚本。如果您有一个域名，您可以使用它作命名空间。如果不存在，将会默认使用下载用户脚本的网站域名。

description

这是关于用户脚本功能的描述。在您第一次安装脚本时，它将会在安装对话框中显示，之后会在“管理用户脚本”对话框中显示。描述不应多于两句。

exclude

`exclude` 规则优先于 `include` 。

6.1 存取持久数据

Greasemonkey 定义了两个函数，`GM_setValue` 和 `GM_getValue`，用户脚本就可以存取“私有”数据，并且只有这个用户脚本才能访问（其他脚本无法访问这些数据，并不仅仅是用户脚本）。可以使用这两个函数存储脚本专有的配置，维护在页面之间持续的缓存，或者记持续的活动日志。

```
function GM_getValue(key, defaultValue=undefined) { }  
function GM_setValue(key, value) { }
```

value 可以是字符串、数字或布尔值。

API 编程接口

调用声明

API 使用要求使用 `@grant` 元数据声明后再使用，如：

```
@grant GM_getValue
```

API 表

函数名	定义	说明
GM_log	GM_log(message, level=0)	记录日志到错误控制台
GM_getValue	GM_getValue(key, defaultValue=undefined)	读取脚本专用的配置值
GM_setValue	GM_setValue(key, value)	设置脚本专用的配置值
GM_xmlHttpRequest	GM_xmlHttpRequest(requestInfo)	进行 HTTP 请求（允许跨域）

GM_log

记录日志到错误控制台。

在错误控制台中输入信息。主要用于调试。`level` 是可选的，默认值是 0。

其有效的值有：

- 0 - 信息
- 1 - 警告
- 2 - 错误

GM_getValue

读取脚本专用的配置值

GM_setValue

设置脚本专用的配置值

`value` 可以是：

- string
- integer
- boolean

GM_xmlHttpRequest