CKEditor 4 开发人员指南

本文由 Jack Soong 根据英文版 CKEditor 4 Developer's Guide 进行翻译，希望对使用 CkEditor 4 的开发人员有所帮助。2013.04.09

一、安装

安装 CKEditor 是很容易的。选择合适的过程（新安装或升级），按下面描述的步骤安装。

1、新安装

第一次安装 CKEditor，按如下方式进行：

1) 从 CKEditor 网站：http://ckeditor.com/download 下载最新版本的 CKEditor 。
2) 将下载的文档解压到你的 website 根目录下的 ckeditor 目录中。

你可以将文档放在你的 website 的任意路径中。ckeditor 目录是缺省目录。

2、升级

要升级现有的 CKEditor 安装，按如下方式进行：

1) 将你的旧编辑文件夹更名为备份文件夹，如 ckeditor_old。
2) 从 CKEditor 网站：**http://ckeditor.com/download** 下载最新版本的 CKEditor 。
3) 将下载的文档解压到原编辑器的目录，如 ckeditor。
4) 将你已做过更改的配置文件从备份目录复制到其新目录的相应位置。这些配置文件可能包括（但不限于）下列文件：
   - **config.js**
   - **contents.css**
   - **styles.js**

3、验证安装

CKEditor 带有样例页面，可以用于验证安装是否正常。为了查看编辑器是否工作，看一下样例目录。

要检查安装，在你的站点中调用下面的页面：

http://<your site>/<CKEditor installation path>/samples/index.html

例如：http://www.example.com/ckeditor/samples/index.html

二、集成

1、加载 CKEditor

CKEditor 是 JavaScript 应用程序。要加载它，需要在页面中包含一个简单的文件引用。如果已在站点根目录下的 ckeditor 目录中安装了 CKEditor ，你需要将下面的代码片段插入到页面的<head>部分：

```
<head>
    ...
    <script src="/ckeditor/ckeditor.js"></script>
```

</head>

当上述文件加载后，就可以使用 CKEditor JavaScript API。

当将 CKEditor 加入到 web 页面时，使用原始的文件名（ckeditor.js）。如果想使用不同的文件名，或者甚至想将 CKEditor 合并到另外一个 JavaScript 文件中，请先参考开发人员指南中的 Specifying the Editor Path 部分。

生成编辑器

现在已经可以在页面中使用 CKEditor JavaScript API 了，可以使用它生成编辑器。为此，有两种可用的选项：

- 框架式编辑（Framed Editing）：最常用的使用方式，通常的表现形式是在页面中指定位置放置工具栏和编辑区。
- 内联式编辑（Inline Editing）：用于看起来像最终的页面中。通过 HTML5 的 contenteditable（内容可编辑的）属性，可以将编辑直接应用于 HTML 元素上。

点击你偏爱的选项获得更多信息。

2、框架式编辑（Framed Editing）

框架式编辑是使用 CKEditor 最常用的方式。它通常的表现形式是在页面中指定位置放置工具栏和编辑区。

在加载 CKEditor 的脚本后，就可以准备生成编辑器了。

生成框架式编辑式

在框架式编辑时，CKEditor 的工作就像页面中的文本域元素。编辑器提供了用户界面，可以容易在编写、格式化，并可与富文本一起处理，但是可以用<textarea>元素完成同样的事情（尽管不是那么容易），需要用户在其中输入 HTML 代码。

实际上，CKEditor 使用文本域将其数据传给服务器。对于终端用来来说，文本域是不可见的。为了生成编辑器的实例，必须首先将<textarea>元素加入到 HTML 页面的源代码中。

```
<textarea name="editor1">&lt;p&gt;Initial value.&lt;/p&gt;</textarea>
```

注意如果你想将数据加载到编辑器中，如从数据中，你需要将数据置入<textarea>元素中，就像上面例子中 HTML 编码的<p>元素一样。在此例中，文本域被命名为 editor1。在后面当收到提交的数据后，可以将此名字用于服务器端的代码中。

插入文本域后，就可以使用 CKEditor JavaScript API 将 HTML 元素替换为编辑器的实例。为此，需要调用简单的 CKEDITOR.replace 方法：

```
<script>
    CKEDITOR.replace( 'editor1' );
</script>
```

此脚本块必须被包含在<textarea>标记后的页面源代码的任意位置。你也可以在<head>部分调用 replace 函数，但此时需要监听 window.onload 事件：

```
<script>
    window.onload = function() {
        CKEDITOR.replace( 'editor1' );
    };
</script>
```

保存编辑器的数据

如上所述，编辑器的工作方式类似于<textarea>域。这意味着当提交包含编辑器实例的

表单时，数据将很简单地提交，使用<textarea>元素名作为键来检查它。

例如，接着上面的例子，我们可以生成下面的 PHP 代码：

```
<?php
    $editor_data = $_POST[ 'editor1' ];
?>
```

客户端数据处理

一些应用程序（如基于 Ajax 的）需要在客户端处理所有的数据，用其特定的方法将数据发给服务器。如果是这样，使用 CKEditor JavaScript API 就足以很容易地检索编辑器实例的数据。为此，你可以使用 getData 方法：

```
<script>
    var editor_data = CKEDITOR.instances.editor1.getData();
</script>
```

完整的样例

要插入一个 CKEditor 实例，可以使用下面的样例来生成一个页面，它包含一个用 CKEditor 替换过的表单。

```
<html>
<head>
    <title>CKEditor Sample</title>
    <script src="/ckeditor/ckeditor.js"></script>
</head>
<body>
    <form method="post">
        <p>
            My Editor:<br>
            <textarea name="editor1">&lt;p&gt;Initial value.&lt;/p&gt;</textarea>
            <script>
                CKEDITOR.replace( 'editor1' );
            </script>
        </p>
        <p>
            <input type="submit">
        </p>
    </form>
</body>
</html>
```

3、内联式编辑（Inline Editing）

内联式编辑是一种新技术，使得可以编辑看起来像最终样子的页面。它是一种完全的“所见即所得”（WYSIWYG ）体验，因为不仅是编辑的内容看起来像最终样子，而且内容所在的页面和上下文也是这样。

启用内联式编辑

通过 HTML5 的 contenteditable（内容可编辑的）属性，内联式编辑可以直接在 HTML 元素上启用。

例如，假设你想使用一个 元素可编辑。可以这样做：

```
<div id="editable" contenteditable="true">
    <h1>Inline Editing in Action!</h1>
    <p>The div element that holds this text is now editable.
</div>
```

也可以通过代码来启用编辑，通过调用 CKEDITOR.inline：

```
<div id="editable" contenteditable="true">
    <h1>Inline Editing in Action!</h1>
    <p>The div element that holds this text is now editable.
</div>
<script>
    // Turn off automatic editor creation first.
    CKEDITOR.disableAutoInline = true;
    CKEDITOR.inline( 'editable' );
</script>
```

当在上面的 内容中点击时，CKEditor 的工具栏就是出现。

检索编辑器的数据

跟框架式编辑不同，当使用内联式编辑时，用 CKEditor 编辑的数据没有放在<textarea>中，而是直接存在于页面的 DOM 中。因此，应用程序就要完成检查数据和存储所需的操作。

要检查编辑器的数据，简单地调用编辑器实例的 CKEDITOR.editor.getData 方法。对于上面的例子，如下：

```
<script>
    var data = CKEDITOR.instances.editable.getData();
// Your code to save "data", usually though Ajax.
</script>
```

注意为了检索编辑器实例，最初的 id 已传给了 CKEDITOR.instances 对象。

三、配置

1、设置 CKEditor 的配置

CKEditor 具有丰富的一组配置选项，可以定制其外观、功能和行为。主配置文件名字为 config.js。此文件可以在 CKEditor 安装文件夹的根目录中找到。

可用的配置选项

在 API 文档中可以找到所有可用的配置选项。参考 CKEDITOR.config 对象的定义。

在页面中（In-Page）定义配置

设置 CKEditor 的最好方法是当生成了编辑器实例时，在页面中设置。此方法可以使你避免修改 CKEditor 安装文件夹中最初发布的文件，使升级任务更容易。

在页面中可将设置传给任意编辑器实例生成函数，即 CKEDITOR.replace 和 CKEDITOR.appendTo。例如：

```
CKEDITOR.replace( 'editor1', {
    toolbar : 'Basic',
    uiColor : '#9AB8F3'
```

});

注意通过一个字面对象定义（以“{”符号开始，以“}”符号结束）来传递配置选项。因此，每个选项的正确语法是（"配置的名字"）:（"配置的值"）。一定不要在冒号字符（:）处使用“等于”字符（=）。

使用 config.js 文件

也可以通过使用 config.js 文件来配置 CKEditor 的设置。此文件缺省情况几乎为空。要改变 CKEditor 的配置，将你想要修改的设置增加到 config.js 文件中。例如：

```
CKEDITOR.editorConfig = function( config ) {
    config.language = 'fr';
    config.uiColor = '#AADC6E';
};
```

为了应用配置设置，必须总是要定义 CKEDITOR.editorConfig 函数。config.js 文件将在页面范围中被执行，故你还可以引用在页面（In-Page）或者甚至是其它 JavaScript 文件中定义的变量。

使用定制的配置文件

使用定制的配置文件是另一种推荐的设置 CKEditor 配置的方法。不使用缺省的 config.js 文件，你可以将此文件复制到站点的任何地方，并简单地引导编辑器加载它。此方法的优点是采用此方法可以避免改变原始的文件，从而使以后升级 CKEditor 更容易，只是简单地覆盖所有的文件。

假设你将 config.js 文件复制到你的站点根目录下一个叫 custom 的文件夹中。你还将此文件更名为 ckeditor_config.js。此时，要使用定义在文件中的定制设置，只需在生成编辑器实例时，设置 customConfig 配置选项即可。例如：

```
CKEDITOR.replace( 'editor1', {
    customConfig : '/custom/ckeditor_config.js'
});
```

定制的配置文件必须看起来像缺省的 config.js 文件。

配置的加载顺序

不要求你仅使用一种上述配置选项。可以混用上述方法，并且可以正确地加载配置。下面的列表表示在生成编辑器实例时所使用的配置加载顺序：

- 生成一个编辑器的实例。此时，其所有配置项设置为缺少值。
- 如果在页面中（in-page）中有配置设置，该文件被加载，否则加载缺省的 config.js 文件。在定制配置文件中的所有设置覆盖当前实例的配置。
- 如果在第二步加载的设置中还定义了新的 customConfig 值，就加载另一个配置文件，且其设置覆盖实例的当前设置。对所有文件递归，直到没有再定义 customConfig。
- 最终，在页面中(in-page)中定义的设置覆盖实例的当前设置（除了 customConfig，它已在第一步中使用过了）。

避免加载外部的设置文件

也可能完全避免加载外部的配置文件，以减少加载的文件数。为此，你需要将 CKEDITOR.config.customConfig 设置为空字符串。例如：

```
CKEDITOR.replace( 'editor1', {
    customConfig : ' '
});
```

如果你在 config.js 文件中没有配置，也没有定制的配置文件，则一定推荐此设置。

2、工具栏的定制

虽然 CKEditor 是一个功能齐全的“所见即所得”编辑器，但是并不是其所有选项在所有情况下都需要。因此，工具栏定制是最常用需求之一。

有两种方法配置工具栏来匹配需求：

- 工具栏组配置
- “逐项”配置

工具栏组配置

CKEditor 4 为组织工具栏引入了新概念，它基于“分组”，而不是传统的“逐项定位”方式。

配置分组是由 toolbarGroups 设置定义的。下面是 CKEditor 标准发布所使用的配置：

```
config.toolbarGroups = [
	{ name: 'clipboard',	groups: [ 'clipboard', 'undo' ] },
	{ name: 'editing',	groups: [ 'find', 'selection', 'spellchecker' ] },
	{ name: 'links' },
	{ name: 'insert' },
	{ name: 'forms' },
	{ name: 'tools' },
	{ name: 'document',	groups: [ 'mode', 'document', 'doctools' ] },
	{ name: 'others' },
	'/',
	{ name: 'basicstyles', groups: [ 'basicstyles', 'cleanup' ] },
	{ name: 'paragraph',	groups: [ 'list', 'indent', 'blocks', 'align' ] },
	{ name: 'styles' },
	{ name: 'colors' },
	{ name: 'about' }
];
```

它是一个对象的列表（数组），每一项有一个“name”（如"clipboard"或"links"）和一个可选的“子组”列表。

改变组的顺序

可以通过改变上述配置，可以容易地定制组的顺序和位置。

通过加入'/'，正如在上面看到的，可以在工具栏中强制断行（row-break）。

注意：在上面的配置中，有没有使用的组。这是因为“设计”的原因（参见“组配置的好处”）。

组配置的好处

与“逐项”配置相比，配置工具栏分组的最重要好处是：自动化。

现在可以让插件开发人员决定其插件应该将按钮加到工具栏的哪个组中。例如，“图像”插件，可将其按钮包括到“插入”组中，而撤销和重做按钮加到“撤销”子组中。

虽然不是强制性的，但推荐要配置所有的组和子组（包括没有使用的），因为在将来的任何时候，当安装了新的插件时，其按钮将会自动出现在工具栏中，而不用再配置。

组配置的缺点

分组配置的最明显问题是不可能精确控制每项在工具栏中的放置位置。它是由插件自身

来决定的。

“逐项”配置

除了分组配置，还可以通过确定精确的位置来控制工具栏中的每一个单独的元素。这是通过配置“工具栏定义”来完成的。

工具栏定义是一个 JavaScript 数组，它含有在编辑器中可以使用的在所有工具栏行中显示的元素。下面是一个例子：

```
config.toolbar = [
    { name: 'document', items: [ 'Source', '-', 'NewPage', 'Preview', '-', 'Templates' ] },
    { name: 'clipboard', items: [ 'Cut', 'Copy', 'Paste', 'PasteText', 'PasteFromWord', '-', 'Undo', 'Redo' ] },
    '/',
    { name: 'basicstyles', items: [ 'Bold', 'Italic' ] }
];
```

这里每个工具栏组都给定了一个名字，并定义了其中项目的精确清单。

也可以用简单的语法实现上面的（参见后面的“访问问题”）：

```
config.toolbar = [
    [ 'Source', '-', 'NewPage', 'Preview', '-', 'Templates' ],
    [ 'Cut', 'Copy', 'Paste', 'PasteText', 'PasteFromWord', '-', 'Undo', 'Redo' ],
    '/',
    [ 'Bold', 'Italic' ]
];
```

通过将'-'(短线)加到项目列表中，将项目分隔符包括其中，如上面看到的。

将'/'加到组之间，可以在工具栏中强制断行。它们可以用于在其放置的位置强制断行，使得下一个工具栏组处于新的一行。

“逐项”配置的好处

这种配置最明显的好处是每个单独的项目在工具栏中的位置都可以控制。

逐项”配置的缺点

最大的问题是当安装新的插件时没有了自动化。这意味着，如果有任何新的插件加入到你的编辑器中，你将不得不手动改变你的配置，将插件按钮包含于希望的位置。

可访问性问题

用于每个工具栏组的名字将被帮助技术所使用，如屏幕阅读器（screen readers）。该名字将被 CKEditor 用于在编辑器语言文件（工具栏组条目）中查找每个工具栏组的“可读”名字。

屏幕阅读器将通过使用其可读名字（如果可用）或其已定义的名字属性显示每个工具栏组。

3、样式

样式组合（Stryles Combo）插件将一个组合加入到 CKEditor 的工具栏中，包含一个样式列表。此列表使很容易地将定制的样式和语义值应用于在编辑器中生成的内容上。

可以很容易地定制组合下拉列表中可用的条目，以满足你的需要。

定义样式

样式定义是一个 JavaScript 数组，通过调用 CKEDITOR.stylesSet.add 函数注册。必须给样

式定义赋以唯一的名字，这样后面就可以配置每个编辑器实例来加载它。这种方法可以使一个样式定义被页面中多个 CKEditor 实例共享。

下面的代码展示了如何注册一个例子样式定义。

```
CKEDITOR.stylesSet.add( 'my_styles', [
    // Block-level styles
    { name: 'Blue Title', element: 'h2', styles: { 'color': 'Blue' } },
    { name: 'Red Title' , element: 'h3', styles: { 'color': 'Red' } },
// Inline styles
    { name: 'CSS Style', element: 'span', attributes: { 'class': 'my_style' } },
    { name: 'Marker: Yellow', element: 'span', styles: { 'background-color': 'Yellow' } }
]);
```

如上面所示的样式定义注册可以内置于页面源代码中，或者存在于外部文件中，只要需要时，“按需”加载。（见下面）。

当准备好定义后，你必须通过使用 stylesSet 设置使编辑器应用新注册的样式。这可以在 config.js 文件中设置，例如：

```
config.stylesSet = 'my_styles';
```

使用外部样式定义文件

可以将样式定义的注册调用包含在一个外部 JavaScript 文件中。缺省情况，CKEditor 从其安装文件夹中的 styles.js 文件中加载样式定义。

你的样式定义文件可以保存在站点的任意地方（或互联网的某个地方）。但，你必须知道访问它所需的 URL。例如，你可以将此文件保存在站点的根目录中，然后按/styles.js 调用它，或者放在其它任意地方，并使用其完整的 URL 引用它，如 http://www.example.com/styles.js。

此时，改变 stylesSet 设置，使编辑器指向此文件：

```
config.stylesSet = 'my_styles:/styles.js';
```

或者

```
config.stylesSet = 'my_styles:http://www.example.com/styles.js';
```

样式定义设置的语法总是：样式定义名称:文件的 URL。

注意，必须使用在此文件中注册样式定义时使用过的唯一的名称。

样式规则

在样式定义中的输入项称为“样式规则”。每条规则定义了单个样式的显示名字和元素、属性以及所使用的 CSS 样式。下面是它的一般表现形式：

```
{
    name: 'Name displayed in the Styles drop-down list',
    element: 'HTML element name (for example "span")',
    styles: {
        'css-style1': 'desired value',
        'css-style2': 'desired value',
        ...
    }
    attributes: {
        'attribute-name1': 'desired value',
        'attribute-name2': 'desired value',
        ...
```

```
    }
}
```

名称和元素值是必须的，而其它的值是可选的。

样式的类型

有三种样式类型，每一种都与在样式规则中使用的元素相关:

- 块级样式（**Block-level styles**）-作为整体应用于文本块（段落），不限于文本选择。应用于下面的元素：address, div, h1, h2, h3, h4, h5, h6, p, 和 pre。
- 对象样式（**Object styles**）- 用于特殊的可选择的对象（不是文本的），不论何时只要你的浏览器支持这样的选择。应用于下面的元素：a, embed, hr, img, li, object, ol, table, td, tr 和 ul。
- 内联样式（**Inline styles** ）- 应用于文本选择，其样式规则所使用的元素没有在其它样式类型中定义。

样式单解析器插件（Stylesheet Parser Plugin）

存在另一种简化的方法，用于定制在 CKEditor 中生成文档的样式，用外部 CSS 样式单文件中添加的样式定义来填充下拉清单。样式单解析器插件允许使用现有的 CSS 样式，而不必按上面给出的格式专门为 CKEditor 定义样式。

安装完样式单解析器后，需要使用 contentsCss 配置设置，提供含有样式定义的 CSS 文件的位置：

```
config.contentsCss = 'sample_CSS_file.css';
```

最后，如果你想跳过加载 CKEditor 缺省使用的样式，可以将 stylesSet 设置为空值：

```
config.stylesSet = [];
```

这种解决方法使得可以配置编辑器使用现有的 CSS 样式单规则，而不需要为 CKEditor 生成单独的样式定义。另一方面，前面使用的方法提供了对用户可以使用样式的更多控制，因此，两种方法可以按需要互换使用。

选择 CSS 选择器

可以细调此插件，以使只考虑与 stylesheetParser_validSelectors 配置值相匹配的 CSS 选择器。默认的正则表达式接受 element.class 形式的所有 CSS 规则，但可以修改它，以引用有限的元素集，如下面的例子：

```
// Only add rules for <p> and <span> elements.（仅为<p>和<span>元素增加规则
config.stylesheetParser_validSelectors = /\^(p|span)\.\w+/;
```

限制 CSS 选择器

通过设置 stylesheetParser_skipSelectors 的配置值，也可以进行定制。那么，该插件将会角力忽略与正则表达式相匹配的 CSS 规则，不会在下拉列表中显示它们，也不会使用它们输出文档内容。默认值不包括为<body>的所有元素和为非具体元素定义的类，但可以修改它，以忽略更广泛的元素集，如下例：

```
// Ignore rules for <body> and <caption> elements, classes starting with "high", and any class defined for no specific element.（忽略<body>和<caption>元素、以“high”开头的类，以及为非具体元素定义的类的规则）
config.stylesheetParser_skipSelectors = /(^body\.|^caption\.|\.high|^\.)/i;
```

4、高级内容过滤器

介绍

注意：高级内容过滤器是在 CKEditor 4.1 中引入的。

什么是高级内容过滤器（ACF）？

ACF 是自 CKEditor 4.1 可用的高度可配置的 CKEditor 核心特色功能。它限制和适配输入的数据（在源代码模式中或由 editor.setData 方法添加的 HTML 代码，粘贴的 HTML 代码，等等），因此它可以以更可行的方式配置编辑器的配置。它也可以使生成配置不允许的 HTML 代码的功能无效。

高级内容过滤器有两种工作模式：

- **自动模式**-过虑器由编辑器功能（如插件，按钮和命令）进行配置，这些功能是由配置选项，如 CKEDITOR.config.plugins、CKEDITOR.config.extraPlugins 和 CKEDITOR.config.toolbar，来启用的。
- **定制模式**-由 CKEDITOR.config.allowedContent 选项来配置过滤器，并且只有与此设置匹配的功能才被激活。

在两种模式中，可以通过使用 CKEDITOR.config.extraAllowedContent 设置来扩展过滤器的配置。

如果想使高级内容过滤器不起作用，将 CKEDITOR.config.allowedContent 设置为"true"。所有可用的编辑器功能将被激活，并且将不会过滤输入的数据。

自动模式

当没有提供 CKEDITOR.config.allowedContent 设置时，高级内容过滤器工作于自动模式。在编辑器初始化时，编辑功能将其规则添加到过滤器。因此，只允许使用当前加载的功能可以编辑的内容，其余所有内容都被过滤掉。

下面的例子或许可以较容易地理解 ACF 的自动模式。

1. 从完全或标准 CKEditor 包中打开 datafiltering.html 例子（基本包提供的功能集太有限了）。
2. 勾选 *editor 1*。它使用默认的配置，所以所有包中可用的按钮、按键或样式都是有效的，并且编辑器的内容也与最初加载的内容一样（除了标准包中的很小细节--因为标准包不包括 Justify（调整版面）插件，页脚没有向右对齐）。
3. 现在勾选 *editor 4*。你可以看到很多插件和按钮被 CKEDITOR.config.removePlugins 和 CKEDITOR.config.removeButtons 移除了；CKEDITOR.config.format_tags 列表也被缩减了。配置的改变自动反映在了编辑器的内容中了--Image 工具栏按钮没有了，故在内容中也就不存在图像了；Table 插件没有了，故添加在原始内容中的表格也被移除了。你可以在源代码模式中看一下编辑器是如何清除粘贴的内容或 HTML 代码。

如果想将编辑器配置为工作于自动模式，但需要附加的 HTML 标记、属性、样式或类生效，可以使用 CKEDITOR.config.extraAllowedContent 配置选项。

定制模式

当定义了 CKEDITOR.config.allowedContent 设置后，高级内容过滤器就工作于定制模式。此配置选项告诉过滤器允许哪些 HTML 元素、属性、样式和类。基于默认的规则（称为允许内容规则--Allowed Content Rules），编辑器过滤输入的数据，决定哪些功能生效。

允许内容规则（Allowed Content Rules）可以被设置为两种不同的格式：紧凑的字符串格式（string format）和更强大的对象格式（object format）。关于允许内容规则，请阅读允许内容规则。

下面的例子可以使较容易地理解 ACF 的定制模式。

1. 从完全或标准 CKEditor 包中打开 datafiltering.html 例子（基本包提供的功能集太有限了）。
2. 勾选 *editor 1*。它使用默认的配置，所以所有包中可用的按钮、按键或样式都是有效的，并且编辑器的内容也与最初加载的内容一样（除了标准包中的很小细节--因为标准包不包括 Justify（调整版面）插件，页脚没有向右对齐）。
3. 勾选 *editor 2*。CKEDITOR.config.allowedContent 选项定义了允许内容规则使用字符串格式。注意到，因为这些规则不允许's'元素，删去（Strike Through）按钮从工具栏中被删除了，并且文本中的删去格式化也被删除了。对于带有水平线（Horizontal Line）、下标（Subscript）和上标（Superscript）功能的例子同样如此。

再看一下 Styles 和 Format 下拉列表，仅含有允许内容规则中定义的项目。而且，在某些对话框中可用的选项也受到了限制。例如，图象对话窗口仅含有 URL、替代文字（Alternative Text）、宽度和高度值，因为仅有这些属性在 CKEDITOR.config.allowedContent 中定义了。

4. 另外，*editor 3* 被配置为使用对象格式定义的不同规则集。

内容转换（Content Transformations）

高级内容过滤器不仅可以删除不允许的 HTML 元素、它们的类、样式和属性，而且可以试着通过将元素从一种形式转换为另一种更喜欢的形式，从而统一输入数据。

考虑加粗（Bold）功能。在 HTML 代码中，它可以用、 <b>，或者甚至 元素来表示。假设 CKEDITOR.config.allowedContent 设置仅含有对<b>的规则。这是否意味着当解析<string>或元素时，会删除它们？

不会。编辑器会将它们都转换为<b>元素。结果是编辑器将只含有<b>元素，并且会保留粘贴内容的可视化形式。如果保留默认的 CKEDITOR.config.allowedContent 值（在自动模式中），也同样如此--所有加粗的功能形式将被统一成偏爱的形式。

假设将'img[!src,alt,width,height]'设置（<img>标记带有必需的 src 和三个可选的属性）添加到允许内容规则中。图像大小应该用属性来设置，因此，作为例子，用样式设置大小的粘贴图像将被转换成具有属性的图像（注意，不是所有情况都可能--只有基于像素的大小可以被转换）。

内容转换功能完全是自动的，不需要配置它。需要做的唯一的事情是设置 CKEDITOR.config.allowedContent 选项，或使用默认的值（自动模式）

目前，我们只是为几个编辑器功能定义了内容转换，但是在将来发布的版本中数量会增加。

允许内容规则（Allowed Content Rules）

注意：高级内容过滤器是在 CKEditor 4.1 中引入的。

简介

允许内容规则定义了允许哪些 HTML 元素、属性、样式和类。当配置 CKEditor 时，你将对设置 CKEDITOR.config.allowedContent 选项最感兴趣。插件开发人员也需要设置 CKEDITOR.config.allowedContent 属性，告诉编辑器在自动模式中一项功能所允许哪些内容。

允许内容规则通常包括四个主要部分：

- 允许的元素，
- 这些元素可以有的属性，
- 这些元素可以有的样式，
- 这些元素可以有的类。

注意：替代书写“属性(attributes)、样式(styles)和类(classes)”，使用“属性”（properties）作为简便的描述方法。

对于一个元素可以存在多项规则，一个元素可以被包含在多个元素列表中。例如，每一

个规则可以允许另一元素属性集。

规则是一个一个地应用的。初始时，被过滤的元素是无效的，其所有属性都被拒绝。应用于该元素的第一条规则校验它（它不会被删除），并且该规则可能接受某些元素的属性。另一条规则可能使编辑器接受更多的元素属性。因此：

- 如果一个元素不存在规则，则删除它。
- 可能接受一个元素，但拒绝其所有的属性，然后就被删除。
- 一旦被验证，一个元素和它的属性不能被另一规则使之失效。

字符串格式（String Format）

字符串格式是允许内容规则的紧凑表示法，但它没有提供对象格式中可用的所有功能。但是，在大多数情况下，应该够用了。

规则格式：elements [attributes]{styles}(classes)

正则表达式模板：

< elements >< styles, attributes, and classes >< separator >

/^([a-z0-9*\s]+)((?:\s*\{[!\w\-,\s*]+\}\s*|\s*\[[!\w\-,\s*]+\]\s*|\s*\([!\w\-,\s*]+\)\s*){0,3})(?:;\s*|$)/i,

其中:

- elements – 一列由空格分隔的元素名或一个星号（*）字符，
- attributes –一列由逗号分隔的属性名或一个星号（*）字符，
- styles –一列由逗号分隔的样式名或一个星号（*）字符，
- classes –一列由逗号分隔的类或一个星号（*）字符，

特殊字符：

- 用于元素列表中的星号意思是：“该规则对所有元素接受后面的属性，但不是这些元素本身；必须有另一条规则显式地接受这些元素”。
- 属性列表中的星号意思是：“接受所有属性”。
- 用于属性列表中条目名前的感叹号（!）（如：[!href]）意思是：“此属性是必需的。如果一个元素不具有此属性，此规则不应应用于该元素（故此元素将不会由它得到验证）”。

例子：

```
// 一条规则接受<p>和<h1>元素，但没有任何属性。
p h1
//一条规则接受具有可选的"left"和 "right"类的<p>和<h1>元素
//注意：两个元素都可能含有这些类，而不仅仅是<h1>
p h1(left,right)
//一条规则接受具有其所有属性的<p>和<h1>元素
p h1[*]
//一条规则接受<a>，只有当它含有"href"属性时。
a[!href]
//一条规则接受<img>，它具有必需的"src"属性和一个可选的"alt"属性，加上可选的"width"和"height"样式
img[alt,!src]{width,height}
//与上面相同，因为属性的顺序和其清单是不相关的，并且忽略空格
img { height, width } [ !src, alt ]
```

允许内容规则集可以含有由分号（;）分隔的多条规则。如：

```
// 这些规则允许：
// 具有可选的"text-align"样式的<p 和><h1>元素
//具有必需的"href"属性的<a>
//  <strong> 和 <em> 元素,
//具有可选的"tip"类的<p>（故<p>元素可以同时具有"text-align"样式和"tip"类）
p h1{text-align}; a[!href]; strong em; p(tip)
//这些规则允许：
// 具有可选的"id"属性的<p>和<h1>元素
//具有必需的"href"属性和可选的"id"属性的<a>
p h1; a[!href]; *[id]
```

调试

为了验证允许内容规则是否被正确解析了，可以检查 CKEDITOR.editor.filter 对象的 CKEDITOR.filter.allowedContent 属性。

```
var editor = CKEDITOR.replace( 'textarea_id', {
    allowedContent: 'a[!href]; ul; li{text-align}(someclass)'
} );
editor.on( 'instanceReady', function() {
    console.log( editor.filter.allowedContent );
} );
// 这将对下列数组做日志：
// { elements: 'p br', ... } (默认编辑器规则)
// { elements: 'a', attributes: '!href' }
// { elements: 'ul' }
// { elements: 'li', styles: 'text-align', classes: 'someclass' }
```

对象格式（Object Format）

为方便使用，允许内容规则可以定义成标准的对象字面量，故下面的：

```
allowedContent: 'p h1{text-align}; a[!href]; strong em; p(tip)'
```

与下面的相对应：

```
allowedContent: {
    'p h1': {
        styles: 'text-align'
    },
    a: {
        attributes: '!href'
    },
    'strong em': true,
    p: {
        classes: 'tip'
    }
}
```

采用此方法，允许内容规则可以由 JavaScript 动态地生成，并因任意目的存储成 JSON 数据格式。注意：对象字面量的键必须是唯一的，因此：

```
allowedContent: {
    p: {
```

```
            styles: 'text-align'
        },
        p: {
            classes: 'tip'
        }
    }
```

等价于：

```
allowedContent: 'p(tip)'
```

但永远不会是：

```
allowedContent: 'p{text-align}(tip)'
```

5、输出格式化（Output Formatting）

CKEditor 提供了强大而又灵活的输出格式化系统。它使开发人员可以完全控制由编辑器生成的 HTML 代码的样子。此系统可以控制所有 HTML 标记，并且可以给出每种标记不同的结果。

HTML 编写器（HTML Writer）

HTML 编写器插件可以使 CKEditor 生成高级格式化的输出。

CKEDITOR.htmlDataProcessor 类使用"writer"来编写输出数据。因此，可以用 editor.dataProcessor.writer 属性检索出具体编辑器实例的当前编写器。

通过设置 writer 属性可以配置几种输出格式化选项。下面的例子概括了最常用的几种，给出了其默认值：

```
var writer = editor.dataProcessor.writer;
// 生一缩进步使用的字符串
writer.indentationChars = '\t';
//自结束标记的方式，就像<br />
writer.selfClosingEnd = ' />';
// 断行所使用的字符串
writer.lineBreakChars = '\n';
// 标记的编写规则
writer.setRules( 'p', {
    // 表示在此标记中断行时缩进
    indent: true,
//在<p>的开始标记前插入断行符
    breakBeforeOpen: true,
//在<p>的开始标记后插入断行符
    breakAfterOpen: true,
// 在<p>的结束标记前插入断行符
    breakBeforeClose: false,
// 在<p>的结束标记后插入断行符
    breakAfterClose: true
});
```

设置编写器规则

因为编辑器是每个编辑器实例的属性，并且由于对加载的编写器插件的依赖，对它进行修改的最好方法是监听 CKEDITOR.instanceReady 事件；这样含义可以安全地假设

CKEDITOR.editor.dataProcessor 属性将会被加载，并准备好可以修改。下面的代码展示了当生成编辑器实例时使用此方法的例子：

```
CKEDITOR.replace( 'editor1', {
    on: {
        instanceReady: function( ev ) {
            // Output paragraphs as <p>Text</p>.
            this.dataProcessor.writer.setRules( 'p', {
                indent: false,
                breakBeforeOpen: true,
                breakAfterOpen: false,
                breakBeforeClose: false,
                breakAfterClose: true
            });
        }
    }
});
```

另一种方法是使用 CKEDITOR 对象，这样所有的编辑器实例会被改变：

```
CKEDITOR.on( 'instanceReady', function( ev ) {
    // Ends self closing tags the HTML4 way, like <br>.
    ev.editor.dataProcessor.writer.selfClosingEnd = '>';
});
```

6、拼写检查（Spell Checking）

CKEditor 可以被配置成使用本地由浏览器提供的拼写检查功能，也可以使用外部的拼写检查 web 服务。

本地拼写检查器

本地拼写检查功能默认情况在编辑器中是被禁用的，可以使用 disableNativeSpellChecker 来使之生效：

```
config.disableNativeSpellChecker = false;
```

在重新加载编辑器后，应该可以看到内容下面的拼写下划线。

注意：如果上下文菜单插件生效，需要在右击时按信 CTRL 键来查看建议。

注意：一般情况下，拼写检查并不是对所有浏览器都是可用的。

SpellCheckAsYouType 插件

SpellCheckAsYouType (SCAYT)插件提供了内联（inline）拼写检查，更像本地的拼写检查器，与 CKEditor 上下文菜单很好地集成在一起。

它是由 WebSpellChecker.net 提供的。它使用其 web 服务，将文本传递到其服务器执行拼写检查。这是跨浏览器的解决方案。

WebSpellChecker 插件

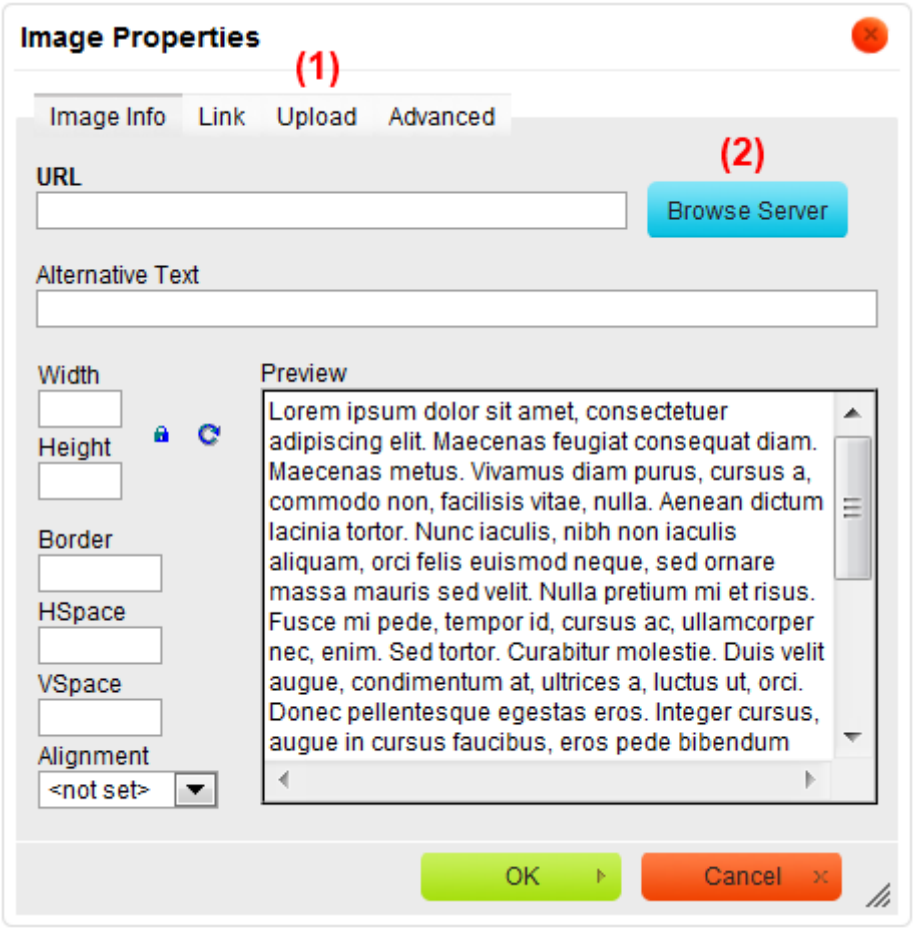
WebSpellChecker 插件是由 WebSpellChecker.net 提供的另一个拼写检查方案，它是通过一个对话窗口来运行查检，而不是内联标记拼写不正确的单词。

7、文件浏览器和上传

与文件浏览器集成

CKEditor 可以容易地与外部文件浏览器/上传器进行集成。

一旦正确地配置了，所有的文件浏览器功能将会自动变成可用。这包括 Link、Image 和 Flash Properties 对话窗口中的 Upload 标签（1），以及 Browse Server 按钮(2)中。



基本配置

- filebrowserBrowseUrl 设置含有外部浏览器的位置，当按了 **Browse Server** 按钮后会启动该浏览器。
- filebrowserUploadUrl 设置含有处理文件上传脚本的位置。如果设置了，在一些对话窗口中会出现 **Upload** 标签--如果该功能在这些窗口中可用的话，即 **Link**, **Image** and **Flash Properties**。

例 1

下面的例子显示了基本配置代码，可以用于插入到配置了文件浏览的 CKEditor 实例中。

```
CKEDITOR.replace( 'editor1', {
    filebrowserBrowseUrl: '/browser/browse.php',
    filebrowserUploadUrl: '/uploader/upload.php'
});
```

例 2

也可以通过在文件浏览器设置：filebrowserBrowseUrl 和 filebrowserUploadUrl 中，使用对话窗口名来为所选的对话窗口设置单独的 URL。

对于为图像对话窗口设置特殊的上传 URL 的例子，使用 filebrowserImageUploadUrl 属性。

```
CKEDITOR.replace( 'editor1', {
    filebrowserBrowseUrl: '/browser/browse.php',
    filebrowserImageBrowseUrl: '/browser/browse.php?type=Images',
    filebrowserUploadUrl: '/uploader/upload.php',
    filebrowserImageUploadUrl: '/uploader/upload.php?type=Images'
});
```

在上面的例子中，filebrowserBrowseUrl 和 filebrowserUploadUrl 设置将在默认情况下使用。而在 Image Properties 对话窗口中，CKEditor 将使用 filebrowserImageBrowseUrl 和 filebrowserImageUploadUrl 的配置设置。

文件浏览器窗口的大小

在 CKEditor 中，文件浏览器的默认宽度设置为屏幕宽度的 80%，而默认高度设置为屏幕高度的 70%。

如果因为任意原因，默认值不适合你，你可以按需要进行调整，使用 filebrowserWindowWidth 改变窗口的宽度，使用 filebrowserWindowHeight 改变窗口的高度。

要将文件浏览器窗口的大小定义成像素，将其设置为一个数字（如"800"）。如果你喜欢将窗口的高度和宽度设置为屏幕的百分数，不要忘记在数字后面加上百分号（如"60%"）。

例 3

下面例子给出的基本配置代码可以用于插入到配置了文件浏览器路径和窗口大小的 CKEditor 中。

```
CKEDITOR.replace( 'editor1', {
    filebrowserBrowseUrl: '/browser/browse.php',
    filebrowserUploadUrl: '/uploader/upload.php',
    filebrowserWindowWidth: '640',
    filebrowserWindowHeight: '480'
});
```

要为指定的对话窗口设置文件浏览器的窗口大小，使用 filebrowserWindowWidth 和 filebrowserWindowHeight 设置。

例如，只想改变"Image"对话框中的文件浏览器的窗口大小，改变 filebrowserImageWindowWidth 和 filebrowserImageWindowHeight 的设置。

例 4

下面例子给出的基本配置代码可以用于插入到配置了文件浏览器路径的 CKEditor 中。它还改变了文件浏览器窗口的默认尺寸，但仅当从 Image Properties 对话窗口中打开时。

```
CKEDITOR.replace( 'editor1', {
    filebrowserBrowseUrl: '/browser/browse.php',
    filebrowserUploadUrl: '/uploader/upload.php',
    filebrowserImageWindowWidth: '640',
    filebrowserImageWindowHeight: '480'
});
```

使用 CKFinder

CKEditor 可以很容易地与 CKFinder 集成，它是一个高级的 Ajax 文件浏览器。看这里的现场演示。

可以用两种方式实现集成：通过设置 CKEditor 配置选项（下面的例子）或通过使用

CKFinder API 中可用的 CKFinder.SetupCKEditor()方法。

例 5

下面例子给出的配置代码可以用于插入到与 CKFinder 集成的 CKEditor 中。对于图像和 Flash 对象的浏览和上传路径与 CKFinder 的默认路径分开配置。

```
CKEDITOR.replace( 'editor1', {
    filebrowserBrowseUrl: '/ckfinder/ckfinder.html',
    filebrowserImageBrowseUrl: '/ckfinder/ckfinder.html?Type=Images',
    filebrowserFlashBrowseUrl: '/ckfinder/ckfinder.html?Type=Flash',
    filebrowserUploadUrl:
'/ckfinder/core/connector/php/connector.php?command=QuickUpload&type=Files',
    filebrowserImageUploadUrl:
'/ckfinder/core/connector/php/connector.php?command=QuickUpload&type=Images',
    filebrowserFlashUploadUrl:
'/ckfinder/core/connector/php/connector.php?command=QuickUpload&type=Flash'
});
```

上面的例子对 PHP 环境有效。注意：/ckfinder/ 是 CKFinder 安装目标的基路径。

如果你想在 ASP, ASP.NET, 或 ColdFusion 中使用 CKFinder，记住将上面的 php 修改为正确的扩展名：

- asp – CKFinder for ASP
- aspx – CKFinder for ASP.NET
- cfm – CKFinder for ColdFusion
- php – CKFinder for PHP

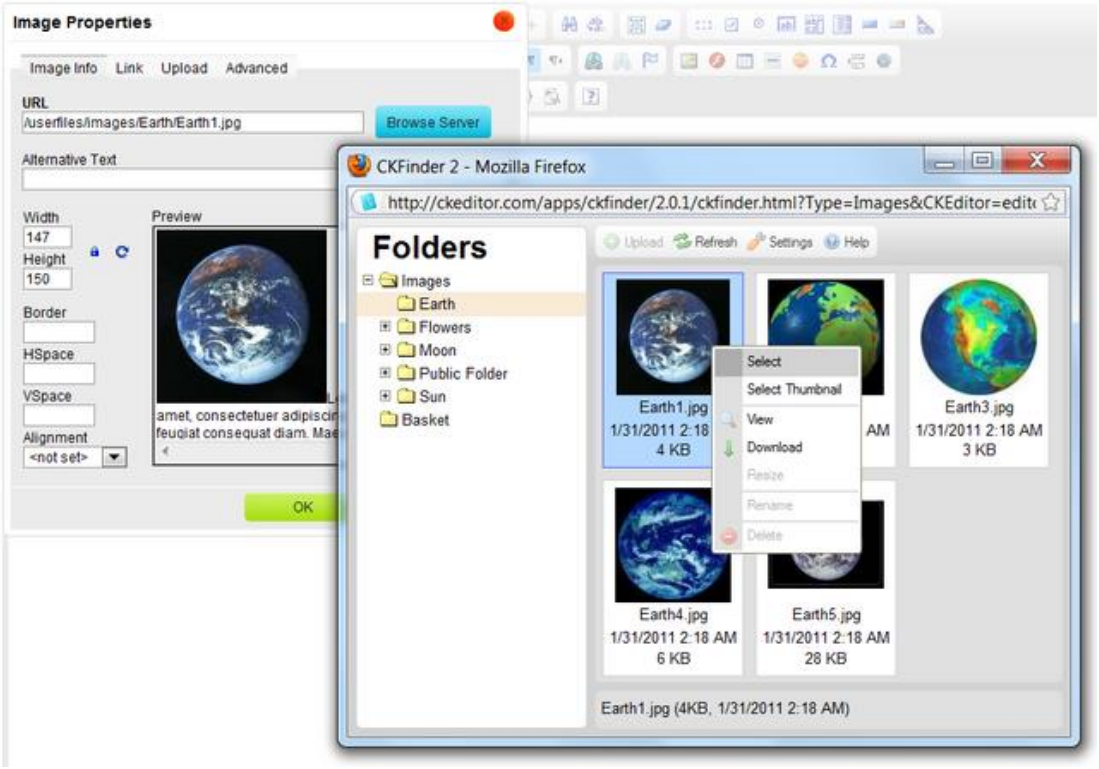
例 6

下面例子说明了 CKFinder.SetupCKEditor()的使用，可以插入到与 CKFinder 集成的 CKEditor 实例中。

```
var editor = CKEDITOR.replace( 'editor1' );
CKFinder.SetupCKEditor( editor, '/ckfinder/' );
```

SetupCKEditor()方法的第二个参数是 CKFinder 的安装路径。

请查看与 CKFinder 一起发布的 _samples/js/ckeditor.html 例子，查看此集成方法的完整工作示例。



PHP API

至于 CKFinder 1.4.2 和 CKEditor 3.1，可以使用 PHP API 将 CKFinder 与 CKEditor 集成。更多细节，参见 CKFinder for PHP 文档。

其它资源

关于将 CKEditor 与文件浏览器集成的更高级信息，参阅下面的文章：

- Creating a Custom File Browser（生成定制的文件浏览器）
- Adding the File Browser to Dialog Windows（将文件浏览器添加到对话窗口中）

文件浏览器 API

CKEditor 可以很容易地与你自己的文件浏览器进行集成。

要连接与 CKEditor 兼容的文件浏览器(如 CKFinder)，遵循 File Browser (Uploader)文档。

CKEditor 与文件浏览器之间的交互

CKEditor 自动向文件浏览器发送一些附加的参数：

- CKEditor - CKEditor 实例名，
- langCode - CKEditor 的语言 (英语为 en)，
- CKEditorFuncNum - 匿名函数引用编号，用于将文件的 URL 传给 CKEditor（一个随机数）。

例如：

CKEditor=editor1&CKEditorFuncNum=1&langCode=en

例 1

假设 CKEditor 由下面的 JavaScript 调用生成：

```
CKEDITOR.replace( 'editor2', {
    filebrowserBrowseUrl: '/browser/browse.php?type=Images',
    filebrowserUploadUrl: '/uploader/upload.php?type=Files'
});
```

为了浏览文件，CKEditor 将调用：

/browser/browse.php?type=Images&CKEditor=editor2&CKEditorFuncNum=2&langCode=de

该调用包括下面的元素：

- /browser/browse.php?type=Images – filebrowserBrowseUrl 参数的值
- &CKEditor=editor2&CKEditorFuncNum=2&langCode=de – 由 CKEditor:添加的信息。
    - CKEditor=editor2 – CKEditor 实例名 (editor2),
    - CKEditorFuncNum=2 – 匿名函数的引用编号，将用于 callFunction 中，
    - langCode=de – 语言编码(此例中: German). 此参数可以用于发送本地化的错误信息。

传递所选文件的 URL

要从外部文件浏览器中传回文件的 URL，调用 CKEDITOR.tools.callFunction，并将 CKEditorFuncNum 作为第一个参数进行传递：

```
window.opener.CKEDITOR.tools.callFunction( funcNum, fileUrl [, data] );
```

如果 data（第三个参数）是字符串，将由 CKEditor 进行显示。如果在文件上传时发生了问题，此参数通常作为错误信息显示。

例 2

下面的例子说明了如何使用 JavaScript 代码从文件浏览器中传递 URL：

```
//从查询字符串中获得参数的 Helper 函数
function getUrlParam( paramName ) {
    var reParam = new RegExp( '(?:[\?&]|&)' + paramName + '=([^&]+)', 'i' ) ;
    var match = window.location.search.match(reParam) ;
return ( match && match.length > 1 ) ? match[ 1 ] : null ;
}
var funcNum = getUrlParam( 'CKEditorFuncNum' );
var fileUrl = '/path/to/file.txt';
window.opener.CKEDITOR.tools.callFunction( funcNum, fileUrl );
```

例 3

下面的代码说明了如何从 PHP 连接器中传回上传文件的 URL：

```
<?php
    // Required: anonymous function reference number as explained above.
    $funcNum = $_GET['CKEditorFuncNum'] ;
    // Optional: instance name (might be used to load a specific configuration file or anything else).
    $CKEditor = $_GET['CKEditor'] ;
    // Optional: might be used to provide localized messages.
    $langCode = $_GET['langCode'] ;
// Check the $_FILES array and save the file. Assign the correct path to a variable ($url).
    $url = '/path/to/uploaded/file.ext';
    // Usually you will only assign something here if the file could not be uploaded.
    $message = ;
echo "<script type='text/javascript'>window.parent.CKEDITOR.tools.callFunction($funcNum, '$url', '$message');</script>";
?>
```

例 4

如果 data 是一个函数，将在调用此文件浏览器的按钮范围内执行它。这意味着服务器

连接器可以直接访问 CKEditor 和按钮所属的对话窗口。

假设除了传递根据对话窗口定义自动赋给相应域的 fileUrl 外，如果文件浏览器在 Image Properties 对话窗口中打开，你还想设置 alt 属性。为此，将匿名函数作为第三个参数传递：

```
window.opener.CKEDITOR.tools.callFunction( funcNum, fileUrl, function() {
    // 获得对话窗口的引用
    var element,
        dialog = this.getDialog();
    //检查是否为 Image 对话窗口
    if ( dialog.getName() == 'image' ) {
        // 获得具有"alt"属性的文本域的引用
        element = dialog.getContentElement( 'info', 'txtAlt' );
        // 赋新值
        if ( element )
            element.setValue( 'alt text' );
    }
    ...
    // 返回 false，停止进一步执行- - 在这种情况下，CKEditor 将忽略第二个参数 (fileUrl)
    // 和赋给调用文件浏览器按钮的 onSelect 函数 (如果定义了).
    [return false;]
});
```

将文件浏览器添加到对话框中

CKEditor 可以很容易地与你自己的文件浏览器集成。

要连接与 CKEditor 兼容的文件浏览器（如 CKFinder），遵循 File Browser (Uploader)文档。

对话框

请参考 Dialog definition API，获得如何生成一个对话框的一般帮助。

Filebrowser 插件

filebrowser 插件内置于 CKEditor 中。它的唯一目的是在 CKEditor 中提供一个 API，可以集成任意外部文件浏览器，并为各种 CKEditor 组件（通常为对话框）增加文件浏览功能。

添加"Browse Server"按钮

要将文件浏览器插件添加到一个对话框中的元素，设置"filebrowser"属性。在 image 插件的例子中，有：

```
{
    type: 'button',
    hidden: true,
    id: 'browse',
    filebrowser: 'Link:txtUrl',
    label: editor.lang.common.browseServer,
    style: 'float:right',
},
```

此按钮默认是隐藏的 (hidden:true)。文件浏览器插件查找具有 filebrowser 属性的所有元素，并且如果有可用的相应的配置设置(filebrowserBrowseUrl/filebrowserUploadUrl)，就不再隐藏它们。

插件所执行的操作取决于元素的类型，对于 fileButton，操作为 QuickUpload，对于其它

元素，默认的操作是 Browse。在上面的例子中，当点击按钮时，将会启动文件浏览器（在弹出窗口中）。

'Link:txtUrl' 值告诉插件当调用 CKEDITOR.tools.callFunction(funcNum)时，更新 Link 标签中 id 为 txtUrl 的元素。

添加 "Quick Upload"支持

再次看看如何在我们的对话框中处理文件上传，我们将使用来自 CKEditor 的工作示例。在 image 插件中，有一个 Upload 标签定义：

```
{
    id: 'Upload',
    hidden: true,
    filebrowser: 'uploadButton',
    label: editor.lang.image.upload,
    elements: [
        {
            type: 'file',
            id: 'upload',
            label: editor.lang.image.btnUpload,
            style: 'height:40px',
            size: 38
        },
        {
            type: 'fileButton',
            id: 'uploadButton',
            filebrowser: 'info:txtUrl',
            label: editor.lang.image.btnUpload,
            'for': [ 'Upload', 'upload' ]
        }
    ]
},
```

此例较前一例子稍复杂一些，因为 1）这里有整个标签的定义 2）我们需要两个元素：file 和 fileButton 来上传文件。

在上面的例子中，标签的 id 为'Upload'。它默认是隐藏的（hidden:true）。正如已提到的，文件浏览器插件查找具有 filebrowser 属性的所有元素，并且如果相应的配置设置可用就不隐藏它们。在此例中，如果'uploadButton'文件浏览器设置（因为 filebrowser:'uploadButton'）将成为可用(filebrowserUploadUrl)，则将自动不再隐藏此标签。

file 元素很简单，不需要做解释，它只是一个 input 元素，将保存将要上传的文件名。

fileButton 元素较有趣。'info:txtUrl' 值使文件浏览器插件当调用 CKEDITOR.tools.callFunction(funcNum)时（参见 Custom File Browser），更新 info 标签中 id 为 txtUrl 的元素。'for': ['Upload', 'upload'] 一行用于将 fileButton 与 file 元素相关联。它是 CKEditor 上传文件的一条指令，在'Upload 标签'（第一个值）中使用 id 为'upload'的'file'元素。

高级配置（浏览）

可以定义自己的函数，当选择/上传文件时可以调用它。

```
{
    type: 'button',
```

```
        hidden: true,
        id: 'id0',
        label: editor.lang.common.browseServer,
        filebrowser: {
            action: 'Browse',
            // target: 'tab1:id1',
            onSelect: function( fileUrl, data ) {
                alert( 'The selected file URL is "' + fileUrl + '"' );
    for ( var _info in data )
                    alert( 'data[ "' + _info + '" ]' + ' = ' + data[ _info ] );
    var dialog = this.getDialog();
                dialog.getContentElement( 'tab1', 'id1' ).setValue( data[ 'fileUrl' ] );
    //不要调用内置的 onSelect 命令
                return false;
            }
        }
    }
```

在此例中，我们设置了'Browse'操作，当点击按钮时调用文件浏览器。'target'不是必需的，因为我们将在定制的 onSelect 函数中更新目标元素。

如在此文档中说明的，当用户选择了一个文件时，我们已调用了 CKEDITOR.tools.callFunction(funcNum, fileUrl, data)；。fileUrl 和 data 参数被传递给我们定制的 onSelect 函数，我们可以使用它来更新目标元素。

高级配置（快速上传）

用我们配置按钮打开文件浏览器相同的方法，我们配置 fileButton。

```
    {
        type: 'file',
        label: editor.lang.common.upload,
        labelLayout: 'vertical',
        id: 'id2'
    },
    {
        type: 'fileButton',
        label: editor.lang.common.uploadSubmit,
        id: 'id3',
        filebrowser: {
            action: 'QuickUpload',
            params: { type: 'Files', currentFolder: '/folder/' },
            target: 'tab1:id1',
            onSelect: function( fileUrl, errorMessage ) {
                alert( 'The url of uploaded file is: ' + fileUrl + '\nerrorMessage: ' +
errorMessage );
                // Do not call the built-in onSelect command
                // return false;
            }
```

```
        },
        'for': [ 'tab1', 'id2' ]
    }
```

在 filebrowser.params 属性中，我们可以添加其它参数，在查询字符串中传递给外部文件浏览器。filebrowser.target 是当通过服务器连接器（uploader）返回文件时被更新的目标元素--如果我们定义了自己的 onSelect 函数(filebrowser.onSelect)并在此函数中更新了目标元素，就像前一例子中的作法，我们就不不需要此目标元素。

四、定制

1、插件

CKEditor 是完全基于插件的。实际上，编辑器的核心是一个空盒子，它由插件提供的功能来进行填充。甚至编辑器的界面，像工具栏、按钮和编辑区也是插件。

CKEditor 的默认安装，可能你正在使用，具有一些展现于其中的插件。你可以将插件添加到你的编辑器中，给你的用户带来更好更有用的功能。

在哪里寻找插件？

CKEditor Add-ons Repository 是寻找和共享插件的在线服务。它可以很容易地理解插件功能，与 CKEditor 群体进行交往。如果你是一个插件开发者，它也是一个展示你的技能，获得大量用户基础的最佳地方。

用 CKBuilder 生成定制的编辑器

CKBuilder 是 CKEditor Add-ons Repository 的姊妹服务，可以通过选择更适合你需要的插件来生成定制的编辑器。

通过插件件的导航，你可以使用 "Add to my editor button" 将你喜欢的插件发到你定制的编辑器中。一旦完成，你可以简单地下载插件，尽享完美实现你需要的编辑体验。

手动安装插件

如果你不喜欢使用 CKBuilder，如果你有自己的或第三方插件，或如果你只想在经历 CKBuilder 过程前测试插件，你还可以将插件添加到你本地的安装中，通过下面的几个步骤：

1. 解压 zip 文件：插件通常是 zip 文件。所以要想开始，要确保将 zip 文件解压到一个文件夹中。
2. 复制文件到 CKEditor 中：最简单的安装文件的方法是通过简单地将文件复制到 CKEditor 安装目录的 plugins 文件夹中。必须放到与插件"技术"名字相匹配的子文件夹中。例如， Magic Line plugin 将被安装到这样的文件夹中：<CKEditor folder>/plugins/magicline。
3. 使插件生效：现在是时候告诉 CKEditor 你为它加了新的插件。为此，你简单地使用 extraPlugins 配置选项：config.extraPlugins = 'magicline';

就是这些。现在你的插件就可以在 CKEditor 中生效了。

2、皮肤

CKEditor 的用户界面外观完全可以通过皮肤来进行定制。像工具栏、对话框、按钮以及甚至其图标这些元素都可以按你喜欢的样式进行改变。

CKEditor 的默认安装具有 Moono skin。

到哪里寻找皮肤？

CKEditor Add-ons Repository 是寻找和共享皮肤的在线服务。如果你是一个皮肤开发者，它也是一个展示你的技能，获得大量用户基础的最佳地方。

下载带有你喜欢皮肤的 CKEditor

CKBuilder 是 CKEditor Add-ons Repository 的姊妹服务，可以生成具有所需皮肤的定制编辑器。

手动安装皮肤

如果你不喜欢使用 CKBuilder，如果你有自己的或第三方皮肤，或如果你只想在经历 CKBuilder 过程前测试皮肤，你还可以将皮肤添加到你本地的安装中，通过下面的几个步骤：

1. 解压 zip 文件：皮肤通常是 zip 文件。所以要想开始，要确保将 zip 文件解压到一个文件夹中。
2. 复制文件到 CKEditor 中：最简单的安装文件的方法是通过简单地将文件复制到 CKEditor 安装目录的 skins 文件夹中。必须放到与皮肤"技术"名字相匹配的子文件夹中。例如，**Kama skin** 将被安装到这样的文件夹中：<CKEditor folder>/skins/kama。
3. **Enabling the plugin:** Now you must just setup CKEditor, by using the **skin** configuration option:
4. 使插件生效：现在你只需要配置 CKEditor，通过使用 **skin** 配置选项：
5. config.skin = 'kama';

就是这些。现在新皮肤就可以在 CKEditor 中生效了。