

Eclipse 常用快捷键

Eclipse 的编辑功能非常强大，掌握了 Eclipse 快捷键功能，能够大大提高开发效率。Eclipse 中有如下一些和编辑相关的快捷键。

1. 【ALT+/】

此快捷键为用户编辑的好帮手，能为用户提供内容的辅助，不要为记不全方法和属性名称犯愁，当记不全类、方法和属性的名字时，多体验一下【ALT+/】快捷键带来的好处吧。

2. 【Ctrl+O】

显示类中方法和属性的大纲，能快速定位类的方法和属性，在查找 Bug 时非常有用。

3. 【Ctrl+/】

快速添加注释，能为光标所在行或所选定行快速添加注释或取消注释，在调试的时候可能总会需要注释一些东西或取消注释，现在好了，不需要每行进行重复的注释。

4. 【Ctrl+D】

删除当前行，这也是笔者的最爱之一，不用为删除一行而按那么多次的删除键。

5. 【Ctrl+M】

窗口最大化和还原，用户在窗口中进行操作时，总会觉得当前窗口小(尤其在编写代码时)，现在好了，试试【Ctrl+M】快捷键。

查看和定位快捷键

在程序中，迅速定位代码的位置，快速找到 Bug 的所在，是非常不容易的事，Eclipse 提供了强大的查找功能，可以利用如下的快捷键帮助完成查找定位的工作。

1. 【Ctrl+K】、【Ctrl++Shift+K】

快速向下和向上查找选定的内容，从此不再需要用鼠标单击查找对话框了。

2. 【Ctrl+Shift+T】

查找工作空间（Workspace）构建路径中的可找到 Java 类文件，不要为找不到类而痛苦，而且可以使用“*”、“?”等通配符。

3. 【Ctrl+Shift+R】

和【Ctrl+Shift+T】对应，查找工作空间（Workspace）中的所有文件（包括 Java 文件），也可以使用通配符。

4. 【Ctrl+Shift+G】

查找类、方法和属性的引用。这是一个非常实用的快捷键，例如要修改引用某个方法的代码，可以通过【Ctrl+Shift+G】快捷键迅速定位所有引用此方法的位置。

5. 【Ctrl+Shift+O】

快速生成 import，当从网上拷贝一段程序后，不知道如何 import 进所调用的类，试试【Ctrl+Shift+O】快捷键，一定会有惊喜。

6. 【Ctrl+Shift+F】

格式化代码，书写格式规范的代码是每一个程序员的必修之课，当看见某段代码极不顺眼时，选定后按【Ctrl+Shift+F】快捷键可以格式化这段代码，如果不选定代码则默认格式化当前文件（Java 文件）。

7. 【ALT+Shift+W】

查找当前文件所在项目中的路径，可以快速定位浏览器视图的位置，如果想查找某个文件所在的包时，此快捷键非常有用（特别在比较大的项目中）。

8. 【Ctrl+L】

定位到当前编辑器的某一行，对非 Java 文件也有效。

9. 【Alt+←】、【Alt+→】

后退历史记录和前进历史记录，在跟踪代码时非常有用，用户可能查找了几个有关联的地方，但可能记不清楚了，可以通过这两个快捷键定位查找的顺序。

10. 【F3】

快速定位光标位置的某个类、方法和属性。

11. 【F4】

显示类的继承关系，并打开类继承视图。

调试快捷键

Eclipse 中有如下一些和运行调试相关的快捷键。

1. 【Ctrl+Shift+B】：在当前行设置断点或取消设置的断点。
2. 【F11】：调试最后一次执行的程序。
3. 【Ctrl+F11】：运行最后一次执行的程序。
4. 【F5】：跟踪到方法中，当程序执行到某方法时，可以按【F5】键跟踪到方法中。
5. 【F6】：单步执行程序。
6. 【F7】：执行完方法，返回到调用此方法的后一条语句。
7. 【F8】：继续执行，到下一个断点或程序结束。

常用编辑器快捷键

通常文本编辑器都提供了一些和编辑相关的快捷键，在 Eclipse 中也可以通过这些快捷键进行文本编辑。

1. 【Ctrl+C】：复制。
2. 【Ctrl+X】：剪切。
3. 【Ctrl+V】：粘贴。
4. 【Ctrl+S】：保存文件。
5. 【Ctrl+Z】：撤销。
6. 【Ctrl+Y】：重复。

7. 【Ctrl+F】: 查找。

其他快捷键

Eclipse 中还有很多快捷键,无法一一列举,用户可以通过帮助文档找到它们的使用方式,另外有几个常用的快捷键如下。

1. 【Ctrl+F6】: 切换到下一个编辑器。
2. 【Ctrl+Shift+F6】: 切换到上一个编辑器。
3. 【Ctrl+F7】: 切换到下一个视图。
4. 【Ctrl+Shift+F7】: 切换到上一个视图。
5. 【Ctrl+F8】: 切换到下一个透视图。
6. 【Ctrl+Shift+F8】: 切换到上一个透视图。

Eclipse 中快捷键比较多,可以通过帮助文档找到所有快捷键的使用,但要掌握所有快捷键的使用是不可能的,也没有必要,如果花点时间熟悉本节列举的快捷键,必将会事半功倍

1. edit->content Assist -> add Alt+/ 代码关联
2. Window -> Next Editor -> add Ctrl+Tab 切换窗口
3. Run/Debug Toggle Line Breakpoint -> add Ctrl+' 在调试的时候 增删断点
4. Source-> Surround with try/catch Block -> Ctrl+Shift+v 增加 try catch 框框
5. Source -> Generate Getters and Setters -> Ctrl+Shift+. 增加 get set 方法

-----有用的快捷键-----

Alt+/ 代码助手完成一些代码的插入(但一般和输入法有冲突,可以修改输入法的热键,也可以暂用 Alt+/来代替)

Ctrl+I:光标停在某变量上,按 Ctrl+I键,可以提供快速重构方案。选中若干行,按 Ctrl+I键可将此段代码放入 for、while、if、do 或 try 等代码块中。

双击左括号(小括号、中括号、大括号),将选择括号内的所有内容。

Alt+Enter 显示当前选择资源(工程,or 文件 or 文件)的属性

-----Ctrl 系列-----

Ctrl+K:将光标停留在变量上,按 Ctrl+K 键可以查找到下一个同样的变量

Ctrl+Shift+K:和 Ctrl+K 查找的方向相反

Ctrl+E 快速显示当前 Editor 的下拉列表(如果当前页面没有显示的用黑体表示)

Ctrl+Shift+E 显示管理当前打开的所有的 View 的管理器(可以选择关闭,激活等操作)

Ctrl+Q 定位到最后编辑的地方

Ctrl+L 定位在某行(对于程序超过100的人就有福音了)

Ctrl+M 最大化当前的 Edit 或 View (再按则反之)

Ctrl+/ 注释当前行,再按则取消注释

Ctrl+T 快速显示当前类的继承结构

Ctrl+Shift-T: 打开类型(Open type)。如果你不是有意磨洋工,还是忘记通过源码树(source tree)打开的方式吧。

Ctrl+O:在代码中打开类似大纲视图的小窗口

Ctrl+鼠标停留:可以显示类和方法的源码

Ctrl+H:打开搜索窗口

Ctrl+/ (小键盘) 折叠当前类中的所有代码

Ctrl+× (小键盘) 展开当前类中的所有代码

-----Ctrl+Shift 系列-----

Ctrl+Shift+F 格式化当前代码

Ctrl+Shift+X 把当前选中的文本全部变味小写

Ctrl+Shift+Y 把当前选中的文本全部变为小写

Ctrl+Shift+O:快速地导入 import

Ctrl+Shift+R:打开资源 open Resource

-----F 快捷键 系列-----

F3:打开声明该引用的文件

F4:打开类型层次结构

F5:单步跳入

F6:单步跳过

F7:单步跳出

F8:继续, 如果后面没有断点, 程序将运行完

-----行编辑用-----

Ctrl+D: 删除当前行

Ctrl+Alt+↓ 复制当前行到下一行(复制增加)

Ctrl+Alt+↑ 复制当前行到上一行(复制增加)

Alt+↓ 当前行和下面一行交互位置(特别实用,可以省去先剪切,再粘贴了)

Alt+↑ 当前行和上面一行交互位置(同上)

Shift+Enter 在当前行的下一行插入空行(这时鼠标可以在当前行的任一位置,不一定是最后)

Ctrl+Shift+Enter 在当前行插入空行(原理同上条)

-----不常用的-----

Alt+← 前一个编辑的页面

Alt+→ 下一个编辑的页面(当然是针对上面那条来说了)

Ctrl+Shift+S:保存全部

Ctrl+W 关闭当前 Editer

Ctrl+Shift+F4 关闭所有打开的 Editer

Ctrl+Shift+G: 在 workspace 中搜索引用

Ctrl+Shift+P 定位到对于的匹配符(譬如{ }) (从前面定位后面时,光标要在匹配符里面,后面到前面,则反之)

-----不明白-----

Ctrl+J 正向增量查找(按下 Ctrl+J 后,你所输入的每个字母编辑器都提供快速匹配定位到某个单词,如果没有,则在 stutes line 中显示没有找到了,查一个单词时,特别实用,这个功能 Idea 两年前就有了)

Ctrl+Shift+J 反向增量查找(和上条相同,只不过是后往前查)

attribute : `$("p").addClass(css 中定义的样式类型);` 给某个元素添加样式
`$("img").attr({ src: "test.jpg", alt: "test Image" });` 给某个元素添加属性/值, 参数是 map
`$("img").attr("src", "test.jpg");` 给某个元素添加属性/值 `$("img").attr("title", function() { return this.src });` 给某个元素添加属性/值 `$("元素名称").html();` 获得该元素内的内容 (元素, 文本等) `$("元素名称").html("new stuff");` 给某元素设置内容 `$("元素名称").removeAttr("属性名称")` 给某元素删除指定的属性以及该属性的值 `$("元素名称").removeClass("class")` 给某元素删除指定的样式 `$("元素名称").text();` 获得该元素的文本 `$("元素名称").text(value);` 设置该元素的文本值为 value `$("元素名称").toggleClass(class)` 当元素存在参数中的样式的时候取消, 如果不存在就设置此样式 `$("input 元素名称").val();` 获取 input 元素的值 `$("input 元素名称").val(value);` 设置 input 元素的值为 value
Manipulation : `$("元素名称").after(content);` 在匹配元素后面添加内容 `$("元素名称").append(content);` 将 content 作为元素的内容插入到该元素的后面 `$("元素名称").appendTo(content);` 在 content 后接元素 `$("元素名称").before(content);` 与 after 方法相反 `$("元素名称").clone(布尔表达式)` 当布尔表达式为真时, 克隆元素 (无参时, 当作 true 处理) `$("元素名称").empty();` 将该元素的内容设置为空 `$("元素名称").insertAfter(content);` 将该元素插入到 content 之后 `$("元素名称").insertBefore(content);` 将该元素插入到 content 之前 `$("元素").prepend(content);` 将 content 作为该元素的一部分, 放到该元素的最前面 `$("元素").prependTo(content);` 将该元素作为 content 的一部分, 放 content 的最前面 `$("元素").remove();` 删除所有的指定元素 `$("元素").remove("exp");` 删除所有含有 exp 的元素 `$("元素").wrap("html");` 用 html 来包围该元素 `$("元素").wrap(element);` 用 element 来包围该元素
Traversing : `add(expr)` `add(html)` `add(elements)` `children(expr)` `contains(str)` `end();` `filter(expression)` `filter(filter)` `find(expr)` `is(expr)` `next(expr)` `not(el)` `not(expr)` `not(elems)` `parent(expr)` `parents(expr)` `prev(expr)` `siblings(expr)`
Core: `$(html).appendTo("body")` 相当于在 body 中写了一段 html 代码 `$(elems)` 获得 DOM 上的某个元素 `$(function() { });` 执行一个函数 `$("div > p").css("border", "1px solid gray");` 查找所有 div 的子节点 p, 添加样式 `$("input:radio", document.forms[0])` 在当前页面的第一个表单中查找所有的单选按钮
`$.extend(prop)` prop 是一个 jquery 对象, 举例: `jQuery.extend({ min: function(a, b) { return a < b ? a : b; }, max: function(a, b) { return a > b ? a : b; } });` `jQuery(expression, [context])` — `$(expression, [context]);` 在默认情况下, `$()` 查询的是当前 HTML 文档中的 DOM 元素。
`each(callback)` 以每一个匹配的元素作为上下文来执行一个函数 举例: 1 `$("span").click(function() { $("li").each(function() { $(this).toggleClass("example"); }); });` 举例: 2 `$("button").click(function () { $("div").each(function (index, domEle) { // domEle == this $(domEle).css("backgroundColor", "yellow"); if ($(this).is("#stop")) { $("span").text("Stopped at div index #" + index); return false; } }); });` `jQuery Event: ready(fn); $(document).ready();` 注意在 body 中没有 onload 事件, 否则该函数不能执行。在每个页面中可以 有很多个函数被加载执行, 按照 fn 的顺序来执行。 `bind(type, [data], fn)` 为每一个匹配元素的特定事件 (像 click) 绑定一个或多个事件处理器函数。可能的事件属性有: blur, focus, load, resize, scroll, unload, click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, mouseenter, mouseleave, change, select, submit, keydown, keypress, keyup, error one(type, [data], fn) 为每

一个匹配元素的特定事件（像 click）绑定一个或多个事件处理器函数。在每个对象上，这个事件处理函数只会被执行一次。其他规则与 bind() 函数相同。 trigger(type, [data]) 在每一个匹配的元素上触发某类事件。 triggerHandler(type, [data]) 这一特定方法会触发一个元素上特定的事件(指定一个事件类型)，同时取消浏览器对此事件的默认行动 unbind([type], [data]) 反绑定，从每一个匹配的元素中删除绑定的事件。 \$('p').unbind() 移除所有段落上的所有绑定的事件 \$('p').unbind("click") 移除所有段落上的 click 事件 hover(over, out) over, out 都是方法，当鼠标移动到一个匹配的元素上面时，会触发指定的第一个函数。当鼠标移出这个元素时，会触发指定的第二个函数。 \$('p').hover(function(){\$(this).addClass("over"); }, function(){\$(this).addClass("out"); }); toggle(fn, fn) 如果点击了一个匹配的元素，则触发指定的第一个函数，当再次点击同一元素时，则触发指定的第二个函数。 \$('p').toggle(function(){\$(this).addClass("selected"); }, function(){\$(this).removeClass("selected"); });

元素事件列表说明 注：不带参数的函数，其参数为可选的 fn。jQuery 不支持 form 元素的 reset 事件。事件 描述 支持元素或对象 blur() 元素失去焦点 a, input, textarea, button, select, label, map, area change() 用户改变域的内容 input, textarea, select click() 鼠标点击某个对象 几乎所有元素 dblclick() 鼠标双击某个对象 几乎所有元素 error() 当加载文档或图像时发生某个错误 window, img focus() 元素获得焦点 a, input, textarea, button, select, label, map, area keydown() 某个键盘的键被按下 几乎所有元素 keypress() 某个键盘的键被按下或按住 几乎所有元素 keyup() 某个键盘的键被松开 几乎所有元素 load(fn) 某个页面或图像被完成加载 window, img mousedown(fn) 某个鼠标按键被按下 几乎所有元素 mousemove(fn) 鼠标被移动 几乎所有元素 mouseout(fn) 鼠标从某元素移开 几乎所有元素 mouseover(fn) 鼠标被移到某元素之上 几乎所有元素 mouseup(fn) 某个鼠标按键被松开 几乎所有元素 resize(fn) 窗口或框架被调整尺寸 window, iframe, frame scroll(fn) 滚动文档的可视部分时 window select() 文本被选定 document, input, textarea submit() 提交按钮被点击 form unload(fn) 用户退出页面 window

jQuery Ajax 方法说明：load(url, [data], [callback]) 装入一个远程 HTML 内容到一个 DOM 结点。 \$('#feeds').load("feeds.html"); 将 feeds.html 文件载入到 id 为 feeds 的 div 中 \$(" #feeds").load("feeds.php", { limit: 25 }, function(){ alert("The last 25 entries in the feed have been loaded"); });

jQuery.get(url, [data], [callback]) 使用 GET 请求一个页面。 \$.get("test.cgi", { name: "John", time: "2pm" }, function(data){ alert("Data Loaded: " + data); });

jQuerygetJSON(url, [data], [callback]) 使用 GET 请求 JSON 数据。 \$.getJSON("test.js", { name: "John", time: "2pm" }, function(json){ alert("JSON Data: " + json.users[3].name); });

jQuery.getScript(url, [callback]) 使用 GET 请求 javascript 文件并执行。 \$.getScript("test.js", function(){ alert("Script loaded and executed."); });

jQuery.post(url, [data], [callback], [type]) 使用 POST 请求一个页面。

ajaxComplete(callback) 当一个 AJAX 请求结束后，执行一个函数。这是一个 Ajax 事件 \$('#msg').ajaxComplete(function(request, settings){\$(this).append("Request Complete."); });

ajaxError(callback) 当一个 AJAX 请求失败后，执行一个函数。这是一个 Ajax 事件 \$(" #msg").ajaxError(function(request, settings){\$(this).append("Error requesting page " + settings.url + ""); });

ajaxSend(callback) 在一个 AJAX 请求发送时，执行一个函数。这是一个 Ajax 事件 \$(" #msg").ajaxSend(function(evt, request,

settings){\$(this).append("Starting request at " + settings.url + ""); }); ajaxStart(callback) 在一个 AJAX 请求开始但还没有激活时, 执行一个函数。这是一个 Ajax 事件 当 AJAX 请求开始 (并 还没有 激活 时) 显示 loading 信息 \$("#loading").ajaxStart(function(){\$(this).show(); }); ajaxStop(callback) 当所有的 AJAX 都停止时, 执行一个函数。这是一个 Ajax 事件 当所有 AJAX 请求都停止时, 隐藏 loading 信息。 \$("#loading").ajaxStop(function(){\$(this).hide(); }); ajaxSuccess(callback) 当一个 AJAX 请求成功完成后, 执行一个函数。这是一个 Ajax 事件 当 AJAX 请求成功完成时, 显示信息。 \$("#msg").ajaxSuccess(function(evt, request, settings){\$(this).append("Successful Request!"); }); jQuery.ajaxSetup(options) 为所有的 AJAX 请求进行全局设置。查看\$.ajax 函数取得所有选项信息。 设置默认的全局 AJAX 请求选项。 \$.ajaxSetup({url: "/xmlhttp/", global: false, type: "POST" }); \$.ajax({ data: myData }); serialize() 以名称和值的方式连接一组 input 元素。实现了正确表单元素序列 function showValues() {var str = \$("form").serialize(); \$("#results").text(str); } \$("checkbox, :radio").click(showValues); \$("select").change(showValues); showValues(); serializeArray() 连接所有的表单和表单元素 (类似于.serialize()方法), 但是返回一个 JSON 数据格式。 从 form 中取得一组值, 显示出来 function showValues() {var fields = \$("input").serializeArray();alert(fields); \$("#results").empty(); jQuery.each(fields, function(i, field){ \$("#results").append(field.value + " "); }); } \$("checkbox, :radio").click(showValues); \$("select").change(showValues); showValues();

jQuery Effects 方法说明 show() 显示隐藏的匹配元素。 show(speed, [callback]) 以优雅的动画显示所有匹配的元素, 并在显示完成后可选地触发一个回调函数。 hide() 隐藏所有的匹配元素。 hide(speed, [callback]) 以优雅的动画隐藏所有匹配的元素, 并在显示完成后可选地触发一个回调函数 toggle() 切换元素的可见状态。如果元素是可见的, 切换为隐藏的; 如果元素是隐藏的, 切换为可见的。 slideDown(speed, [callback]) 通过高度变化 (向下增大) 来动态地显示所有匹配的元素, 在显示完成后可选地触发一个回调函数。这个动画效果只调整元素的高度, 可以使匹配的元素以 “滑动” 的方式显示出来。 slideUp(speed, [callback]) 通过高度变化 (向上减小) 来动态地隐藏所有匹配的元素, 在隐藏完成后可选地触发一个回调函数。这个动画效果只调整元素的高度, 可以使匹配的元素以 “滑动” 的方式隐藏起来。 slideToggle(speed, [callback]) 通过高度变化来切换所有匹配元素的可见性, 并在切换完成后可选地触发一个回调函数。这个动画效果只调整元素的高度, 可以使匹配的元素以 “滑动” 的方式隐藏或显示。 fadeIn(speed, [callback]) 通过不透明度的变化来实现所有匹配元素的淡入效果, 并在动画完成后可选地触发一个回调函数。这个动画只调整元素的不透明度, 也就是说所有匹配的元素的高度和宽度不会发生变化。 fadeOut(speed, [callback]) 通过不透明度的变化来实现所有匹配元素的淡出效果, 并在动画完成后可选地触发一个回调函数。这个动画只调整元素的不透明度, 也就是说所有匹配的元素的高度和宽度不会发生变化。 fadeTo(speed, opacity, [callback]) 把所有匹配元素的不透明度以渐进方式调整到指定的不透明度, 并在动画完成后可选地触发一个回调函数。这个动画只调整元素的不透明度, 也就是说所有匹配的元素的高度和宽度不会发生变化。 stop() 停止所有匹配元素当前正在运行的动画。如果有动画处于队列当中, 他们就会立即开始。 queue() 取得第一个匹配元素的动画序列的引用(返回一个内容为函数的数组) queue(callback) 在每一个匹配元素的事件序列的末尾添加一个可执行函数, 作为此元素的事件函数 queue(queue) 以一个新的动画序列代替所有匹配元素的原动画序列 dequeue() 执行并移除动画序列前端的动画 animate(params, [duration], [easing], [callback]) 用于创建自定义动画的函数。 animate(params, options) 创建自定义动画的另一个方法。作用同上。 JQuery Traversing 方法说明 eq(index) 从匹配的元素集合中取得一个指定位置的元素, index 从0

开始 `filter(expr)` 返回与指定表达式匹配的元素集合, 可以使用`,”`号分割多个 `expr`, 用于实现多个条件筛选 `filter(fn)` 利用一个特殊的函数来作为筛选条件移除集合中不匹配的元素。`is(expr)` 用一个表达式来检查当前选择的元素集合, 如果其中至少有一个元素符合这个给定的 表达式就返回 `true`。 `map(callback)` 将 jQuery 对象中的一组元素利用 `callback` 方法转换其值, 然后添加到一个 jQuery 数组中。 `not(expr)` 从匹配的元素集合中删除与指定的表达式匹配的元素。 `slice(start, [end])` 从匹配元素集合中取得一个子集, 和内建的数组的 `slice` 方法相同。 `add(expr)` 把与表达式匹配的元素添加到 jQuery 对象中。 `children([expr])` 取得一个包含匹配的元素集合中每一个元素的所有子元素的元素集合。可选的过滤器 将使这个方法只匹配符合的元素(只包括元素节点, 不包括文本节点)。 `contents()` 取得一个包含匹配的元素集合中每一个元素的所有子孙节点的集合(只包括元素节点, 不包括文本节点), 如果元素为 `iframe`, 则取得其中的文档元素 `find(expr)` 搜索所有与指定表达式匹配的元素。 `next([expr])` 取得一个包含匹配的元素集合中每一个元素紧邻的后面同辈元素的元素集合。 `nextAll([expr])` 取得一个包含匹配的元素集合中每一个元素所有的后面同辈元素的元素集合 `parent([expr])` 取得一个包含着所有匹配元素的唯一父元素的元素集合。 `parents([expr])` 取得一个包含着所有匹配元素的唯一祖先元素的元素集合(不包含根元素)。 `prev([expr])` 取得一个包含匹配的元素集合中每一个元素紧邻的前一个同辈元素的元素集合。 `prevAll([expr])` 取得一个包含匹配的元素集合中每一个元素的之前所有同辈元素的元素集合。 `siblings([expr])` 取得一个包含匹配的元素集合中每一个元素的所有同辈元素的元素集合。 `andSelf()` 将前一个匹配的元素集合添加到当前的集合中 取得所有 `div` 元素和其中的 `p` 元素, 添加 `border` 类属性。取得所有 `div` 元素中的 `p` 元素, 添加 `background` 类属性 `$(“div”).find(“p”).andSelf().addClass(“border”); $(“div”).find(“p”).addClass(“background”);` `end()` 结束当前的操作, 回到当前操作的前一个操作 找到所有 `p` 元素其中的 `span` 元素集合, 然后返回 `p` 元素集合, 添加 `css` 属性 `$(“p”).find(“span”).end().css(“border”, “2px red solid”);`

jQuery Selectors 方法说明

基本选择器 `$(“#myDiv”)` 匹配唯一的具有此 `id` 值的元素 `$(“div”)` 匹配指定名称的所有元素 `$(“.myClass”)` 匹配具有此 `class` 样式值的所有元素 `$(“*”)` 匹配所有元素 `$(“div,span,p.myClass”)` 联合所有匹配的选择器 层叠选择器 `$(“form input”)` 后代选择器, 选择 `ancestor` 的所有子孙节点 `$(“#main > *”)` 子选择器, 选择 `parent` 的所有子节点 `$(“label + input”)` 临选择器, 选择 `prev` 的下一个临节点 `$(“#prev ~ div”)` 同胞选择器, 选择 `prev` 的所有同胞节点

基本过滤选择器 `$(“tr:first”)` 匹配第一个选择的元素 `$(“tr:last”)` 匹配最后一个选择的元素 `$(“input:not(:checked) + span”)` 从原元素集合中过滤掉匹配 `selector` 的所有元素 (这里有一个临选择器) `$(“tr:even”)` 匹配集合中偶数位置的所有元素(从0开始) `$(“tr:odd”)` 匹配集合中奇数位置的所有元素(从0开始) `$(“td:eq(2)”)` 匹配集合中指定位置的元素(从0开始) `$(“td:gt(4)”)` 匹配集合中指定位置之后的所有元素(从0开始) `$(“td:lt(4)”)` 匹配集合中指定位置之前的所有元素(从0开始) `$(“:header”)` 匹配所有标题 `$(“div:animated”)` 匹配所有正在运行动画的所有元素

内容过滤选择器 `$(“div:contains(‘John’)”)` 匹配含有指定文本的所有元素 `$(“td:empty”)` 匹配所有空元素(只含有文本的元素不算空元素) `$(“div:has(p)”)` 从原元素集合中再次匹配所有至少含有一个 `selector` 的所有元素 `$(“td:parent”)` 匹配所有不为空的元素(含有文本的元素也算) `$(“div:hidden”)` 匹配所有隐藏的元素, 也包括表单的隐藏域 `$(“div:visible”)` 匹配所有可见的元素

属性过滤选择器 `$(“div[id]”)` 匹配所有具有指定属性的元素 `$(“input[name=‘newsletter’]”)` 匹配所有具有指定属性值的元素 `$(“input[name!=‘newsletter’]”)` 匹配所有不具有指定属性值的元素 `$(“input[name^=‘news’]”)` 匹配所有指定属性值以 `value` 开头的元素 `$(“input[name$=‘letter’]”)` 匹配所有指定属性值以 `value` 结尾的元素 `$(“input[name*=‘man’]”)` 匹配所有指定属性值含有 `value` 字符的元素 `$(“input[id][name$=‘man’]”)` 匹配同时符合多个选择器的所有元素

子

元素过滤选择器 `$("ul li:nth-child(2)")`, `$("ul li:nth-child(odd)")`, 匹配父元素的第 n 个子元素 `$("ul li:nth-child(3n + 1)")` `$("div span:first-child")` 匹配父元素的第 1 个子元素 `$("div span:last-child")` 匹配父元素的最后 1 个子元素 `$("div button:only-child")` 匹配父元素的唯一 1 个子元素 表单元素选择器 `$(":input")` 匹配所有的表单输入元素, 包括所有类型的 `input`, `textarea`, `select` 和 `button` `$(":text")` 匹配所有类型为 `text` 的 `input` 元素 `$(":password")` 匹配所有类型为 `password` 的 `input` 元素 `$(":radio")` 匹配所有类型为 `radio` 的 `input` 元素 `$(":checkbox")` 匹配所有类型为 `checkbox` 的 `input` 元素 `$(":submit")` 匹配所有类型为 `submit` 的 `input` 元素 `$(":image")` 匹配所有类型为 `image` 的 `input` 元素 `$(":reset")` 匹配所有类型为 `reset` 的 `input` 元素 `$(":button")` 匹配所有类型为 `button` 的 `input` 元素 `$(":file")` 匹配所有类型为 `file` 的 `input` 元素 `$(":hidden")` 匹配所有类型为 `hidden` 的 `input` 元素或表单的隐藏域 表单元素过滤选择器 `$(":enabled")` 匹配所有可操作的表单元素 `$(":disabled")` 匹配所有不可操作的表单元素 `$(":checked")` 匹配所有已点选的元素 `$("select option:selected")` 匹配所有已选择的元素

JQuery CSS 方法说明 `css(name)` 访问第一个匹配元素的样式属性。 `css(properties)` 把一个 "名 / 值对" 对象设置为所有匹配元素的样式属性。 `$("p").hover(function () {$(this).css({ backgroundColor:"yellow", fontWeight:"bolder" }); }, function () {var cssObj = { backgroundColor: "#ddd", fontWeight: "", color: "rgb(0,40,244)" }$(this).css(cssObj); });` `css(name, value)` 在所有匹配的元素中, 设置一个样式属性的值。 `offset()` 取得匹配的的第一个元素相对于当前可视窗口的位置。返回的对象有 2 个属性, `top` 和 `left`, 属性值为整数。这个函数只能用于可见元素。 `var p = $("p:last"); var offset = p.offset(); p.html("left: " + offset.left + ", top: " + offset.top);` `width()` 取得当前第一匹配的元素宽度值, `width(val)` 为每个匹配的元素设置指定的宽度值。 `height()` 取得当前第一匹配的元素高度值, `height(val)` 为每个匹配的元素设置指定的高度值。 JQuery Utilities 方法说明 `jQuery.browser.msie` 表示 ie `jQuery.browser.version` 读取用户浏览器的版本信息 `jQuery.boxModel` 检测用户浏览器针对当前页的显示是否基于 w3c CSS 的盒模型 `jQuery.isFunction(obj)` 检测传递的参数是否为 function `function stub() { } var objs = [function () {}, { x:15, y:20 }, null, stub, "function"]; jQuery.each(objs, function (i) {var isFunc = jQuery.isFunction(objs[i]); $("span:eq(" + i + ")").text(isFunc);});` `jQuery.trim(str)` 清除字符串两端的空格, 使用正则表达式来清除给定字符串两端的空格 `jQuery.each(object, callback)` 一个通用的迭代器, 可以用来无缝迭代对象和数组 `jQuery.extend(target, object1, [objectN])` 扩展一个对象, 修改原来的对象并返回, 这是一个强大的实现继承的工具, 这种继承是采用传值的方法来实现的, 而不是 JavaScript 中的原型链方式。合并 `settings` 和 `options` 对象, 返回修改后的 `settings` 对象 `var settings = { validate: false, limit: 5, name: "foo" }; var options = { validate: true, name: "bar" }; jQuery.extend(settings, options);` 合并 `defaults` 和 `options` 对象, `defaults` 对象并没有被修改。 `options` 对象中的值代替了 `defaults` 对象的值传递给了 `empty`。 `var empty = {} var defaults = { validate: false, limit: 5, name: "foo" }; var options = { validate: true, name: "bar" }; var settings = $.extend(empty, defaults, options);` `jQuery.grep(array, callback, [invert])` 通过一个筛选函数来去除数组中的项 `$().grep([0,1,2], function(n,i){return n > 0; });` `jQuery.makeArray(obj)` 将一个类似数组的对象转化为一个真正的数组 将选取的 `div` 元素集合转化为一个数组 `var arr = jQuery.makeArray(document.getElementsByTagName("div"));`

`arr.reverse();` // use an Array method on list of dom elements `$(arr).appendTo(document.body);`
`jQuery.map(array, callback)` 使用某个方法修改一个数组中的项，然后返回一个新的数组
`jQuery.inArray(value, array)` 返回 `value` 在数组中的位置，如果没有找到，则返回 -1
`jQuery.unique(array)` 删除数组中的所有重复元素，返回整理后的数组