**版本**

针对iScroll的优化。为了达到更高的性能，iScroll分为了多个版本。你可以选择最适合你的版本。  
目前我们有以下版本：

* iscroll.js，这个版本是常规应用的脚本。它包含大多数常用的功能，有很高的性能和很小的体积。
* iscroll-lite.js，精简版本。它不支持快速跳跃，滚动条，鼠标滚轮，快捷键绑定。但如果你所需要的是滚动(特别是在移动平台) iScroll 精简版 是又小又快的解决方案。
* iscroll-probe.js，探查当前滚动位置是一个要求很高的任务,这就是为什么我决定建立一个专门的版本。如果你需要知道滚动位置在任何给定的时间,这是iScroll给你的。（我正在做更多的测试,这可能最终在常规iscroll.js脚本，请留意）。
* iscroll-zoom.js，在标准滚动功能上增加缩放功能。
* iscroll-infinite.js，可以做无限缓存的滚动。处理很长的列表的元素为移动设备并非易事。 iScroll infinite版本使用缓存机制,允许你滚动一个潜在的无限数量的元素。

**入门**

**IScroll是一个类，每个需要使用滚动功能的区域均要进行初始化**。每个页面上的iScroll实例数目在设备的CPU和内存能承受的范围内是没有限制的。

尽可能保持DOM结构的简洁。iScroll使用硬件合成层但是有一个限制硬件可以处理的元素。  
最佳的HTML结构如下：

<div id="wrapper">

<ul>

<li>...</li>

<li>...</li>

...

</ul>

</div>

iScroll作用于滚动区域的外层。在上面的例子中，UL元素能进行滚动。只有容器元素的第一个子元素能进行滚动，其他子元素完全被忽略。

最基本的脚本初始化的方式如下：

<script type="text/javascript">

var myScroll = new IScroll(‘#wrapper‘);

</script>

**第一个参数可以是滚动容器元素的DOM选择器字符串**，也可以是滚动容器元素的引用对象。下面是一个有效的语法：

var wrapper = document.getElementById(‘wrapper‘);

var myScroll = new IScroll(wrapper);

所以基本上你要么直接传递元素，要么传递一个querySelector字符串。因此可以使用css名称代替ID去选择一个滚动器容器,如下:

var myScroll = new IScroll(‘.wrapper‘);

注意，iScroll使用的是querySelector 而不是 querySelectorAll，所以iScroll只会作用到选择器选中元素的第一个。如果你需要对多个对象使用iScroll，你需要构建自己的循环机制。

**初始化**

当DOM准备完成后iScroll需要被初始化。最保险的方式是在window的onload事件中启动它。  
在DOMContentLoaded事件中或者inline initialization中做也可以，需要记住的是脚本需要知道滚动区域的高度和宽度。如果你有一些图片在滚动区域导致不能立马获取区域的高度和宽度，iScroll的滚动尺寸有可能会错误。

为滚动起容器增加position:relative或者absolute样式。这将解决大多数滚动器容器大小计算不正确的问题。  
综上所述，最小的iScroll配置如下：

<head>

...

<script type="text/javascript" src="iscroll.js"></script>

<script type="text/javascript">

var myScroll;

function loaded() {

myScroll = new IScroll(‘#wrapper‘);

}

</script>

</head>

...

<body onload="loaded()">

<div id="wrapper">

<ul>

<li>...</li>

<li>...</li>

...

</ul>

</div>

</body>

如果你有一个复杂的DOM结构，最好在onload事件之后适当的延迟，再去初始化iScroll。最好给浏览器100或者200毫秒的间隙再去初始化iScroll。

**配置**

在iScroll初始化阶段可以通过构造函数的第二个参数配置它。

var myScroll = new IScroll(‘#wrapper‘, {

mouseWheel: true,

scrollbars: true

});

上面的例子示例了在iScroll初始化时开启鼠标滚轮支持和滚动条支持。

在初始化后你可以通过options对象访问标准化值。例如： js console.dir(myScroll.options);  
上面的语句将返回myScroll实例的配置信息。所谓的标准化意味着如果你设置useTransform:true，但是浏览器并不支持CSS  
transforms，那么useTransform属性值将为false。

**理解核心**

iScroll使用基于设备和浏览器性能的各种技术来进行滚动。**通常不需要你来配置引擎，iScroll会为你选择最佳的方式。**

尽管如此，理解iScroll工作机制和了解如何去配置他们也是很重要的。

**options.useTransform**

默认情况下引擎会使用CSStransform属性。如果现在还是2007年，那么可以设置这个属性为false，这就是说：引擎将使用top/left属性来进行滚动。  
这个属性在滚动器感知到Flash，iframe或者视频插件内容时会有用，但是需要注意：性能会有极大的损耗。  
默认值：true

**options.useTransition**

iScroll使用CSS transition来实现动画效果（动量和弹力）。如果设置为false，那么将使用requestAnimationFrame代替。  
在现在浏览器中这两者之间的差异并不明显。在老的设备上transitions执行得更好。  
默认值：true

**options.HWCompositing**

这个选项尝试使用translateZ(0)来把滚动器附加到硬件层，以此来改变CSS属性。在移动设备上这将提高性能，但在有些情况下,你可能想要禁用它(特别是如果你有太多的元素和硬件性能跟不上)。  
默认值：true

如果不确定iScroll的最优配置。从性能角度出发，上面的所有选项应该设置为true。（或者更好的方式，让他们自动设置属性为true）。你可以尝试这配置他们，但是要小心内存泄漏。

**基本功能**

**options.bounce**

当滚动器到达容器边界时他将执行一个小反弹动画。在老的或者性能低的设备上禁用反弹对实现平滑的滚动有帮助。  
默认值：true

**options.click**

为了重写原生滚动条，iScroll禁止了一些默认的浏览器行为，比如鼠标的点击。如果你想你的应用程序响应click事件，那么该设置次属性为true。请注意，建议使用自定义的tap 事件来代替它（见下面）。  
默认属性：false

**options.disableMouse**

**options.disablePointer**

**options.disableTouch**

默认情况下，iScroll监听所有的指针事件，并且对这些事件中第一个被触发的做出反应。这看上去是浪费资源，但是在大量的浏览器/设备上兼容，特定的事件侦测证明是无效的，所以listen-to-all是一个安全的做法。

如果你有一种设备侦测的内部机制，或者你知道你的脚本将在什么地方运行，你可以把你不需要的事件禁用（鼠标，指针或者触摸事件）。  
下面的例子是禁用鼠标和指针事件：

var myScroll = new IScroll(‘#wrapper‘, {

disableMouse: true,

disablePointer: true

});

默认值：false

**options.eventPassthrough**

有些时候你想保留原生纵向的滚动条但想为横向滚动条增加iScroll功能（比如走马灯）。设置这个属性为true并且iScroll区域只将影响横向滚动，纵向滚动将滚动整个页面。

在移动设备上访问[event passthrough demo](http://lab.cubiq.org/iscroll5/demos/event-passthrough/)。注意，这个值也可以设置为horizontal，其作用和上面介绍的相反（横向滚动条保持原生，纵向滚动条使用iScroll）。

**options.freeScroll**

此属性针对于两个两个纬度的滚动条（当你需要横向和纵向滚动条）。通常情况下你开始滚动一个方向上的滚动条，另外一个方向上会被锁定不动。有些时候，你需要无约束的移动（横向和纵向可以同时响应），在这样的情况下此属性需要设置为true。请参考 [2D scroll demo](http://lab.cubiq.org/iscroll5/demos/2d-scroll/)。  
默认值：false

**options.keyBindings**

此属性为true时激活键盘（和远程控制）绑定。请参考下面的Key bindings内容。  
默认值：false

**options.invertWheelDirection**

当鼠标滚轮支持激活后，在有些情况下需要反转滚动的方向。（比如，鼠标滚轮向下滚动条向上，反之亦然）。  
默认值：false

**options.momentum**

在用户快速触摸屏幕时，你可以开/关势能动画。关闭此功能将大幅度提升性能。  
默认值：true

**options.mouseWheel**

侦听鼠标滚轮事件。  
默认值：false

**options.preventDefault**

当事件触发时师傅执行preventDefault()。此属性应该设置为true，除非你真的知道你需要怎么做。  
请参考下面的Advanced features中的preventDefaultException，可以获取更多控制preventDefault行为的信息。  
默认值：true

**options.scrollbars**

是否显示为默认的滚动条。更多信息请查看Scrollbar  
默认值：false

**options.scrollX**

**options.scrollY**

默认情况下只有纵向滚动条可以使用。如果你需要使用横向滚动条，需要将scrollX 属性值设置为 true。请参考示例[horizontal demo](http://lab.cubiq.org/iscroll5/demos/horizontal/)。

也可以参考**freeScroll**选项。

默认值：scrollX: false，scrollY: true

注意属性 scrollX/Y: true 与overflow: auto有相同的效果。设置一个方向上的值为 false 可以节省一些检测的时间和CPU的计算周期。

**options.startX**

**options.startY**

默认情况下iScroll从0, 0 (top left)位置开始，通过此属性可以让滚动条从不同的位置开始滚动。  
默认值：0

**options.tap**

设置此属性为true，当滚动区域被点击或者触摸但并没有滚动时，可以让iScroll抛出一个自定义的tap事件。  
这是处理与可以点击元素之间的用户交互的建议方式。侦听tap事件和侦听标准事件的方式一致。示例如下：

element.addEventListener(‘tap‘, doSomething, false); \\ Native

$(‘#element‘).on(‘tap‘, doSomething); \\ jQuery

你可以通过传递一个字符串来自定义事件名称。比如：

tap: ‘myCustomTapEvent‘

在这个示例里你应该侦听名为myCustomTapEvent的事件。  
默认值：false

**滚动条**

滚动条不只是像名字所表达的意义一样，在内部它们是作为indicators的引用。

一个指示器侦听滚动条的位置并且现实它在全局中的位置，但是它可以做更多的事情。  
先从最基本的开始。

**options.scrollbars**

正如我们在基本功能介绍中提到的，激活滚动条只需要做一件事情，这件事情就是：

var myScroll = new IScroll(‘#wrapper‘, {

scrollbars: true

});

当然这个默认的行为是可以定制的。

**options.fadeScrollbars**

不想使用滚动条淡入淡出方式时，需要设置此属性为false以便节省资源。  
默认值：false

**options.interactiveScrollbars**

此属性可以让滚动条能拖动，用户可以与之交互。  
默认值：false

**options.resizeScrollbars**

滚动条尺寸改变基于容器和滚动区域的宽/高之间的比例。此属性设置为false让滚动条固定大小。这可能有助于自定义滚动条样式（参考下面的**滚动条样式**）。  
默认值：true

**options.shrinkScrollbars**

当在滚动区域外面滚动时滚动条是否可以收缩到较小的尺寸。  
有效的值为：clip 和 scale。  
clip是移动指示器到它容器的外面，效果就是滚动条收缩起来，简单的移动到屏幕以外的区域。属性设置为此值后将大大的提升整个iScroll的性能。  
scale是关闭属性useTransition，之后所有的动画效果将使用requestAnimationFrame实现。指示器实际上有各种尺寸，并且最终的效果最好。  
默认值：false

注意改变大小不是在GPU上执行的，所以‘scale‘ 是在CPU上执行。

请参考 [滚动条示例](http://lab.cubiq.org/iscroll5/demos/scrollbars/)。

**滚动条样式**

如果你不喜欢默认的滚动条样式，而且你认为你可以做的更好，你可以自定义滚动条样式。第一步就是设置选项scrollbars的值为custom：

var myScroll = new IScroll(‘#wrapper‘, {

scrollbars: ‘custom‘

});

使用下面的CSS类可以简单的改变滚动条样式。

* .iScrollHorizontalScrollbar，这个样式应用到横向滚动条的容器。这个元素实际上承载了滚动条指示器。
* .iScrollVerticalScrollbar，和上面的样式类似，只不过适用于纵向滚动条容器。
* .iScrollIndicator，真正的滚动条指示器。
* .iScrollBothScrollbars，这个样式将在双向滚动条显示的情况下被加载到容器元素上。通常情况下其中一个（横向或者纵向）是可见的

[自定义滚动条样式示例](http://lab.cubiq.org/iscroll5/demos/styled-scrollbars/)。

如果你设置resizeScrollbars: false，滚动条将是固定大小，否则它将基于滚动区域的尺寸变化。

**指示**

上面所有关于滚动条的选项实际上是包装了一个底层的选项indicators。它看起来或多或少像这样：

var myScroll = new IScroll(‘#wrapper‘, {

indicators: {

el: [element|element selector]

fade: false,

ignoreBoundaries: false,

interactive: false,

listenX: true,

listenY: true,

resize: true,

shrink: false,

speedRatioX: 0,

speedRatioY: 0,

}

});

**options.indicators.el**

这是一个强制性的参数，它保留了指向滚动条容器元素的引用。容器里的第一个子元素就是指示器。  
有效的语法如下：

indicators: {

el: document.getElementById(‘indicator‘)

}

更简单的方式：

indicators: {

el: ‘#indicator‘

}

注意，滚动条可以在你的文档的任意地方，它不需要在滚动条包装器内。

**options.indicators.ignoreBoundaries**

这个属性是告诉指示器忽略它容器所带来的边界。当我们能改变滚动条速度的比率，在让滚动条滚动时这个属性很有用。比如你想让指示器是滚动条速度的两倍，指示器将很快到达它的结尾。这个属性被用在**视差滚动**。  
默认值：false

**options.indicators.listenX**

**options.indicators.listenY**

指示器的那一个轴（横向和纵向）被侦听。可以设置一个或者都设置  
默认值：true

**options.indicators.speedRatioX**

**options.indicators.speedRatioY**

指示器移动的速度和主要滚动条大小的关系。默认情况下是设置为自动。你很少需要去改变这个值。  
默认值：0

**options.indicators.fade**

**options.indicators.interactive**

**options.indicators.resize**

**options.indicators.shrink**

这几个选项和我们已经介绍过的**滚动条**中的一样，在这里不重复介绍。  
请参考[迷你地图](http://lab.cubiq.org/iscroll5/demos/minimap/)示例，你将体验indicators选项的神奇力量。

你应该已经注意到选项indicators是复数，是的，实际上，传递一个对象数组你可以得到一个虚拟的无限大小的指示器。我不知道你是否需要，但是，这里我是想你介绍参数设置，所以要提及此。

**视差滚动**

视差滚动是指示器功能的一个 *附属功能*  
指示器是一个遵循主滚动条移动和动画的层。如果你了解它你就会理解这个功能背后的力量。增加这个功能你就可以创建任意数量的指示器和视差滚动。  
请参考 [视差滚动示例](http://lab.cubiq.org/iscroll5/demos/minimap/).

**滚动的编程接口**

当然还可以通过编程来进行滚动。

scrollTo(x, y, time, easing)

对应存在的一个叫做myScroll的iScroll实例，可以通过下面的方式滚动到任意的位置：

myScroll.scrollTo(0, -100);

通过上面的方式将向下滚动100个像素。记住：0永远是左上角。需要滚动你必须传递负数。  
time 和 easing是可选项。他们控制滚动周期（毫秒级别）和动画的擦除效果。  
擦除功能是一个有效的IScroll.utils.ease对象。例如应用一个一秒的经典擦除动画你应该这么做：

myScroll.scrollTo(0, -100, 1000, IScroll.utils.ease.elastic);

擦除动画的类型选项有：quadratic, circular, back, bounce, elastic。

**scrollBy(x, y, time, easing)**

和上面一个方法类似，但是可以传递X和Y的值从当前位置进行滚动。

myScroll.scrollBy(0, -10);

上面这个语句将在当前位置向下滚动10个像素。如果你当前所在位置为-100，那么滚动结束后位置为-110.

**scrollToElement(el, time, offsetX, offsetY, easing)**

这是一个很有用的方法，你会喜欢它的。  
在这个方法中只有一个强制的参数就是el。传递一个元素或者一个选择器，iScroll将尝试滚动到这个元素的左上角位置。  
time是可选项，用于设置动画周期。  
offsetX 和 offsetY定义像素级的偏移量，所以你可以滚动到元素并且加上特别的偏移量。但并不仅限于此。如果把这两个参数设置为true，元素将会位于屏幕的中间。请参考例子 滚动到元素 example。  
easing参数和scrollTo方法里的一样。

**对齐**

iScroll能对齐到固定的位置和元素。

**options.snap**

最简单的对齐配置如下：

var myScroll = new IScroll(‘#wrapper‘, {

snap: true

});

这将按照页面容器的大小自动分割滚动条。  
snap属性也可以传递字符类型类型的值。这个值是滚动条将要对齐到的元素的选择器。比如下面：

var myScroll = new IScroll(‘#wrapper‘, {

snap: ‘li‘

});

这个示例中滚动条将会对齐到每一个LI标记的元素。  
下面将帮助你快速浏览iScroll提供的关于对齐的一系列有趣的方法。

**goToPage(x, y, time, easing)**

x 和 y呈现你想滚动到横向轴或者纵向轴的页面数。如果你需要在单个唯独上使用滚动条，只需要为你不需要的轴向传递0值。  
time属性是动画周期，easing属性是滚动到指定点使用的擦除功能类型。请参考**高级功能**中的option.bounceEasing。这两个属性都是可选项。

myScroll.goToPage(10, 0, 1000);

上面这个例子将在一秒内沿着横向滚动到第10页。

**next()**

**prev()**

滚动到当前位置的下一页或者前一页。

**缩放**

为了使用缩放功能，你最好使用iscroll-zoom.js脚本。

**options.zoom**

此属性设置为true启用缩放功能。  
默认值：false

**options.zoomMax**

最大缩放级数。  
默认值：4

**options.zoomMin**

最小缩放级数。  
默认值：1

**options.zoomStart**

初始的缩放级数。  
默认值：1

**options.wheelAction**

鼠标滚轮的动作可以设置为zoom，这样在滚动滚轮时缩放操作会代替原来的滚动操作。  
默认值：undefined（即：鼠标滚轮滚动）

和前面的示例一样，一个好的缩放功能的配置如下：

myScroll = new IScroll(‘#wrapper‘, {

zoom: true,

mouseWheel: true,

wheelAction: ‘zoom‘

});

缩放功能使用的CSS的转换功能。iScroll只能在支持此CSS功能的浏览器上执行。

一些浏览器（特别是基于webkit的）采取的快照缩放区域就放在硬件合成层(比如当你申请转换)。该快照作为纹理的缩放区域,它几乎不能被更新。这意味着您的纹理将基于  
scale 1 进行缩放,将导致文本和图像模糊,清晰度低。

一个简单的解决方案是使用实际分辨率双倍（或者三倍）装载内容，然后  
放到一个按照scale(0.5)比例缩小的div中。这种方法大多数情况下能适用。

请参考 [缩放示例](http://lab.cubiq.org/iscroll5/demos/zoom/)。

**zoom(scale, x, y, time)**

一个有意思的的方法，能让你进行缩放编程。  
scale是缩放因子。  
x 和 y是缩放关注点，即缩放的中心。如果没有指定，这个中心就是屏幕中心。  
time是毫秒级别的动画周期（可选项）。

**无限滚动**

iScroll集成智能缓存系统，允许处理的使用(重用)一群元素几乎无限数量的数据。  
无限滚动开发的早期阶段，尽管它可以被认为是稳定的，但它还没有准备好被广泛使用。  
请参考 [无限滚动示例](http://lab.cubiq.org/iscroll5/demos/infinite/) 并请提交你的建议和报告bug。

**高级选项**

下面这些选项主要针对核心开发人员。

**options.bindToWrapper**

move事件通常绑定到文档而不是滚动器容器（wrapper）。当你在滚动器容器（wrapper）外移动光标/手指，滚动条将不断滚动。这通常是你想要的,但是你也可以绑定事件转移到滚动器容器（wrapper）本身。这样做一旦指针离开了容器，滚动就会停止。  
默认值：false

**options.bounceEasing**

擦除功能在弹跳动画过程中执行。有效的值为：‘quadratic‘, ‘circular‘, ‘back‘, ‘bounce‘, ‘elastic‘. 参见bounce easing demo，往下拽滚动条然后释放。  
bounceEasing比上面的示例更强大。你可以自定义一个消除的方式，比如：

bounceEasing: {

style: ‘cubic-bezier(0,0,1,1)‘,

fn: function (k) { return k; }

}

上面这个示例将执行一个线性的擦出。style选项将在在每次动画执行时使用CSS转场执行。fn和requestAnimationFrame一起使用。如果一个擦出功能太复杂，不能由一个三次贝塞尔曲线展现，那么为style属性传递 ‘‘ （空字符串）。  
默认值：circular

注意：bounce 和 elastic这两种方式不能被CSS转场执行。

**options.bounceTime**

弹跳动画的持续时间，使用毫秒级。  
默认值：600

**options.deceleration**

这个值可以改变改变动画的势头持续时间/速度。更高的数字使动画更短。你可以从0.01开始去体验，这个值和基本的值比较，基本上没有动能。  
默认值：0.0006

**options.mouseWheelSpeed**

设置鼠标滚轮滚动的速度值。  
默认值：20

**options.preventDefaultException**

调用preventDefault()方法时所有的异常将被触发，尽管preventDefault设置了值。  
这是一个强大的选项，如果你想为所有包含formfield样式名称的元素上应用preventDefault()方法，你可以设置为下面的值：

preventDefaultException: { className: /(^|\s)formfield(\s|$)/ }

默认值：{ tagName: /^(INPUT|TEXTAREA|BUTTON|SELECT)$/ }.

**options.resizePolling**

当你改变窗口的大小iScroll重新计算元素的位置和尺寸。这可能是一个相当艰巨的任务。轮询设置为60毫秒。  
通过降低这个值你获得更好的视觉效果，但会占用更多的CPU资源。默认值是一个很好的折中。  
默认值：60

**刷新**

iScroll需要知道包装器和滚动器确切的尺寸，在iScroll初始化的时候进行计算，如果元素大小发生了变化，需要告诉iScroll DOM发生了变化。  
下面将提供调用refresh方法的正确时机。

每次触摸DOM，浏览器渲染器重绘页面。一旦发生了重画我们可以安全地读新的DOM属性。重新绘制阶段不是瞬时发生的只是范围结束时触发。这就是为什么我们需要给渲染器刷新iScroll之前一点时间。

为了确保javascript得到更新后的属性，应该像下面的例子这样使用刷新方法：

ajax(‘page.php‘, onCompletion);

function onCompletion () {

// Update here your DOM

setTimeout(function () {

myScroll.refresh();

}, 0);

};

这里调用refresh()使用了零秒等待，如果你需要立即刷新iScroll边界就是如此使用。当然还有其他方法可以等待页面重绘，但零超时方式相当稳定。

如果你有一个相当复杂的HTML结构，你应该给浏览器更多的执行事件，可以设置100到200毫秒的超时时间。

这通常适用于所有任务必须在DOM上进行。通常给渲染器一些执行的时间。

**自定义事件**

**Custom events**

iScroll还提供额一些你可以挂靠的有用的自定义事件。  
使用on(type, fn)方法注册事件。

myScroll = new IScroll(‘#wrapper‘);

myScroll.on(‘scrollEnd‘, doSomething);

上面的代码会在每次滚动停止是执行doSomething方法。  
可以挂靠的事件如下：

* beforeScrollStart，在用户触摸屏幕但还没有开始滚动时触发。
* scrollCancel，滚动初始化完成，但没有执行。
* scrollStart，开始滚动
* scroll，内容滚动时触发，只有在scroll-probe.js版本中有效，请参考onScroll event。
* scrollEnd，停止滚动时触发。
* flick，用户打开左/右。
* zoomStart，开始缩放。
* zoomEnd，缩放结束。

**onScroll事件**

scroll事件在iScroll probe edition版本中有效（仅包含在iscroll-probe.js脚本文件中）。可以通过改变probeType选项值来改变scroll事件的触发时机。

**options.probeType**

这个属性是调节在scroll事件触发中探针的活跃度或者频率。有效值有：1, 2, 3。数值越高表示更活跃的探测。探针活跃度越高对CPU的影响就越大。

* probeType: 1 对性能没有影响。scroll事件只有在滚动条不繁忙的时候触发。
* probeType: 2 除了在势能和反弹范围内，将在scroll事件周期内一直执行。这类似于原生的onScroll事件。
* probeType: 3 像素级的触发scroll事件。注意，此时滚动只关注requestAnimationFrame (即：useTransition:false).

请参考 [probe demo](http://lab.cubiq.org/iscroll5/demos/probe/).

**按键绑定**

你可以激活keyBindings选项来支持键盘控制。默认情况下iScroll监听方向键，上下翻页建，home/end键，但这些按键绑定完全可以自定义。  
你可以通过传递一个包含按键代码列表的对象来进行按键绑定。  
默认的按键值如下:

keyBindings: {

pageUp: 33,

pageDown: 34,

end: 35,

home: 36,

left: 37,

up: 38,

right: 39,

down: 40

}

当然你也可以传递字符串进行按键绑定（例如：pageUp: ‘a‘）。只要你设置了对于的按键值，那么iScroll就会响应你的设置。

**滚动条信息**

iScroll存储了很多有用的信息，您可以使用它们来增强您的应用。  
你可能会发现有用的：

* myScroll.x/y，当前位置
* myScroll.directionX/Y，最后的方向 (-1 down/right, 0 still, 1 up/left)
* myScroll.currentPage，当前对齐捕获点

下面是关于处理时间的代码示例：

myScroll = new IScroll(‘#wrapper‘);

myScroll.on(‘scrollEnd‘, function () {

if ( this.x < -1000 ) {

// do something

}

});

如果 x 位置是低于-1000 像素滚轮停止时，上述执行某些代码。请注意我用这个产品而不是 myScroll，您可以使用两个当然，但 iScroll 传递本身作为这种情况下，当触发自定义事件的功能。

**销毁**

在不需要使用iScoll的时候调用iScroll实例的公共方法destroy()可以释放一些内存。

myScroll.destroy();

myScroll = null;