jQuery UI 为什么使用部件库 jQuery UI API 文档

jQuery UI 如何使用部件库（Widget Factory）

我们将创建一个进度条。正如下面实例所示，这可以通过调用 `jQuery.widget()` 来完成，它带有两个参数：一个是要创建的插件名称，一个是包含支持插件的函数的对象文字。当插件被调用时，它将创建一个新的插件实例，所有的函数都将在该实例的语境中被执行。这与两种重要方式的标准 jQuery 插件不同。首先，语境是一个对象，不是 DOM 元素。其次，语境总是一个单一的对象，不是一个集合。

```
$.widget( "custom.progressbar", {
    _create: function() {
        var progress = this.options.value + "%";
        this.element
            .addClass( "progressbar" )
            .text( progress );
    }
});
```

插件的名称必须包含命名空间，在这个实例中，我们使用了 `custom` 命名空间。您只能创建一层深的命名空间，因此，`custom.progressbar` 是一个有效的插件名称，而 `very.custom.progressbar` 不是一个有效的插件名称。

我们看到部件库（Widget Factory）为我们提供了两个属性。`this.element` 是一个包含一个元素的 jQuery 对象。如果我们的插件在包含多个元素的 jQuery 对象上调用，则会为每个元素创建一个单独的插件实例，且每个实例都会有自己的 `this.element`。第二个属性，`this.options`，是一个包含所有插件选项的键名/键值对的哈希（hash）。这些选项可以被传给插件，如下所示：

```
$( "<div></div>" )
    .appendTo( "body" )
    .progressbar({ value: 20 });
```

当我们调用 `jQuery.widget()`，它通过给 `jQuery.fn`（用于创建标准插件的系统）添加函数来扩展 jQuery。所添加的函数名称是基于您传给 `jQuery.widget()` 的名称，不带命名空间 - "progressbar"。传给插件的选项是在插件实例中获取设置的值。如下面的实例所示，我们可以为任意一个选项指定默认值。当设计您的 API 时，您应该清楚你的插件的最常见的使用情况，以便您可以设置适当的默认值，且确保使所有的选项真正可选。

```
$.widget( "custom.progressbar", {

    // Default options.
    options: {
        value: 0
    },
    _create: function() {
```

```
        var progress = this.options.value + "%";
        this.element
            .addClass( "progressbar" )
            .text( progress );
    }
});
```

调用插件方法

现在我们可以初始化我们的进度条，我们将通过在插件实例上调用方法来执行动作。为了定义一个插件方法，我们只在我们传给 `jQuery.widget()` 的对象中引用函数。我们也可以通过给函数名加下划线前缀来定义 "private" 方法。

```
$.widget( "custom.progressbar", {

    options: {
        value: 0
    },

    _create: function() {
        var progress = this.options.value + "%";
        this.element
            .addClass( "progressbar" )
            .text( progress );
    },

    // Create a public method.
    value: function( value ) {

        // No value passed, act as a getter.
        if ( value === undefined ) {
            return this.options.value;
        }

        // Value passed, act as a setter.
        this.options.value = this._constrain( value );
        var progress = this.options.value + "%";
        this.element.text( progress );
    },

    // Create a private method.
    _constrain: function( value ) {
        if ( value > 100 ) {
            value = 100;
        }
        if ( value < 0 ) {
            value = 0;
        }
        return value;
```

```
    }
});
```

为了在插件实例上调用方法，您可以向 jQuery 插件传递方法的名称。如果您调用的方法接受参数，您只需简单地在方法名后面传递这些参数即可。

注意：通过向同一个用于初始化插件的 jQuery 函数传递方法名来执行方法。这样做是为了在保持链方法调用时防止 jQuery 命名空间污染。在本章节的后续，我们将看到看起来更自然的其他用法。

```
var bar = $( "<div></div>" )
    .appendTo( "body" )
    .progressbar({ value: 20 });

// Get the current value.
alert( bar.progressbar( "value" ) );

// Update the value.
bar.progressbar( "value", 50 );

// Get the current value again.
alert( bar.progressbar( "value" ) );
```

使用选项

`option()` 方法是自动提供给插件的。`option()` 方法允许您在初始化后获取并设置选项。该方法像 jQuery 的 `.css()` 和 `.attr()` 方法：您可以只传递一个名称作为取值器来使用，也可以传递一个名称和值作为设置器使用，或者传递一个键名/键值对的哈希来设置多个值。当作为取值器使用时，插件将返回与传入名称相对应的选项的当前值。当作为设置器使用时，插件的 `_setOption` 方法将被每个被设置的选项调用。我们可以在我们的插件中指定一个 `_setOption` 方法来反应选项更改。对于更改选项要独立执行的动作，我们可以重载 `_setOptions`。

```
$.widget( "custom.progressbar", {
    options: {
        value: 0
    },
    _create: function() {
        this.options.value = this._constrain(this.options.value);
        this.element.addClass( "progressbar" );
        this.refresh();
    },
    _setOption: function( key, value ) {
        if ( key === "value" ) {
            value = this._constrain( value );
        }
        this._super( key, value );
    },
    _setOptions: function( options ) {
```

```
        this._super( options );
        this.refresh();
    },
    refresh: function() {
        var progress = this.options.value + "%";
        this.element.text( progress );
    },
    _constrain: function( value ) {
        if ( value > 100 ) {
            value = 100;
        }
        if ( value < 0 ) {
            value = 0;
        }
        return value;
    }
});
```

添加回调

最简单的扩展插件的方法是添加回调，这样用户就可以在插件状态发生变化时做出反应。我们可以看下面的实例如何在进度达到 100% 时添加回调到进度条。`_trigger()` 方法有三个参数：回调名称，一个启动回调的 jQuery 事件对象，以及一个与事件相关的数据哈希。回调名称是唯一一个必需的参数，但是对于想要在插件上实现自定义功能的用户，其他的参数是非常有用的。例如，如果我们创建一个可拖拽插件，我们可以在触发拖拽回调时传递 mousemove 事件，这将允许用户对基于由事件对象提供的 x/y 坐标上的拖拽做出反应。请注意，传递到 `_trigger()` 的原始的事件必须是一个 jQuery 事件，而不是一个原生的浏览器事件。

```
$.widget( "custom.progressbar", {
    options: {
        value: 0
    },
    _create: function() {
        this.options.value = this._constrain(this.options.value);
        this.element.addClass( "progressbar" );
        this.refresh();
    },
    _setOption: function( key, value ) {
        if ( key === "value" ) {
            value = this._constrain( value );
        }
        this._super( key, value );
    },
    _setOptions: function( options ) {
        this._super( options );
        this.refresh();
    },
    refresh: function() {
```

```
        var progress = this.options.value + "%";
        this.element.text( progress );
        if ( this.options.value == 100 ) {
            this._trigger( "complete", null, { value: 100 } );
        }
    },
    _constrain: function( value ) {
        if ( value > 100 ) {
            value = 100;
        }
        if ( value < 0 ) {
            value = 0;
        }
        return value;
    }
});
```

回调函数本质上只是附加选项，所以您可以像其他选项一样获取并设置它们。无论何时执行回调，都会有一个相对应的事件被触发。事件类型是通过连接插件的名称和回调函数名称确定的。回调和事件都接受两个相同的参数：一个事件对象和一个与事件相关的数据哈希，具体如下面实例所示。

您的插件可能需要包含防止用户使用的功能，为了做到这点，最好的方法就是创建一个可撤销的回调。用户可以撤销回调或者相关的事件，与他们撤销任何一个原生事件一样，都是通过调用 `event.preventDefault()` 或返回 `false` 来实现的。如果用户撤销回调，`_trigger()` 方法将返回 `false`，这样您就能在插件内实现合适的功能。

```
var bar = $( "<div></div>" )
    .appendTo( "body" )
    .progressbar({
        complete: function( event, data ) {
            alert( "Callbacks are great!" );
        }
    })
    .bind( "progressbarcomplete", function( event, data ) {
        alert( "Events bubble and support many handlers for extreme flexibility." );
        alert( "The progress bar value is " + data.value );
    });

bar.progressbar( "option", "value", 100 );
```

本质

现在我们已经看到如何使用部件库（Widget Factory）创建一个插件，接下来让我们看看它实际上是如何工作的。当您调用 `jQuery.widget()` 时，它将为插件创建一个构造函数，并设置您为插件实例传入的作为原型的对象。所有自动添加到插件的功能都来自一个基本的小部件原型，该原型定义为 `jQuery.Widget.prototype`。当创建插件实例时，会使用 `jQuery.data` 把它存储在原始的 DOM 元素上，插件名作为键名。

由于插件实例直接链接到 DOM 元素上，您可以直接访问插件实例，而不需要遍历插件方法。这将允许您直接在插件实例上调用方法，而不需要传递字符串形式的方法名，同时您也可以直接访问插件的属性。

```
var bar = $( "<div></div>" )
    .appendTo( "body" )
    .progressbar()
    .data( "progressbar" );

// Call a method directly on the plugin instance.
bar.option( "value", 50 );

// Access properties on the plugin instance.
alert( bar.options.value );
```

您也可以在不遍历插件方法的情况下创建一个实例，通过选项和元素直接调用构造函数即可：

```
var bar = $.custom.progressbar( {}, $( "<div></div>" ).appendTo( "body") );

// Same result as before.
alert( bar.options.value );
```

扩展插件的原型

插件有构造函数和原型的最大好处是易于扩展插件。通过添加或修改插件原型上的方法，我们可以修改插件所有实例的行为。例如，如果我们想要向进度条添加一个方法来重置进度为 0%，我们可以向原型添加这个方法，它将在所有插件实例上可调用。

```
$.custom.progressbar.prototype.reset = function() {
    this._setOption( "value", 0 );
};
```

如需了解扩展小部件的更多细节，以及如何在一个已有的小部件上创建一个全新的小部件的更多细节，请查看 通过部件库（Widget Factory）扩展小部件（Widget）。

清理

在某些情况下，允许用户应用插件，然后再取消应用。您可以通过 `_destroy()` 方法做到这一点。在 `_destroy()` 方法内，您应该撤销在初始化和后期使用期间插件所做的一切动作。`_destroy()` 是通过 `.destroy()` 方法被调用的，`.destroy()` 方法是在插件实例绑定的元素从 DOM 上移除时被自动调用的，所以这可被用于垃圾回收。基本的 `.destroy()` 方法也处理一些常用的清理操作，比如从小部件的 DOM 元素上移除实例引用，从元素上解除绑定小部件命名空间中的所有事件，解除绑定所有使用 `_bind()` 添加的事件。

```
$.widget( "custom.progressbar", {
    options: {
```

```
        value: 0
    },
    _create: function() {
        this.options.value = this._constrain(this.options.value);
        this.element.addClass( "progressbar" );
        this.refresh();
    },
    _setOption: function( key, value ) {
        if ( key === "value" ) {
            value = this._constrain( value );
        }
        this._super( key, value );
    },
    _setOptions: function( options ) {
        this._super( options );
        this.refresh();
    },
    refresh: function() {
        var progress = this.options.value + "%";
        this.element.text( progress );
        if ( this.options.value == 100 ) {
            this._trigger( "complete", null, { value: 100 } );
        }
    },
    _constrain: function( value ) {
        if ( value > 100 ) {
            value = 100;
        }
        if ( value < 0 ) {
            value = 0;
        }
        return value;
    },
    _destroy: function() {
        this.element
            .removeClass( "progressbar" )
            .text( "" );
    }
});
```

关闭注释

部件库（Widget Factory）只是创建有状态插件的一种方式。这里还有一些其他不同的模型可以使用，且每个都有各自的优势和劣势。部件库（Widget Factory）解决了很多常见的问题，且大大提高了效率，同时也大大提高了代码的重用性，使它适合于 jQuery UI 及其他有状态的插件。

请注意，在本章节中我们使用了 `custom` 命名空间。`ui` 命名空间被官方的 jQuery UI 插件保留。当创建您自己的插件时，您应该创建自己的命名空间。这样才能更清楚插件来自哪里，属于哪个范围。

jQuery UI 为什么使用部件库 jQuery UI API 文档

点我分享笔记