扩展机制

ZIN 提供了丰富的扩展机制，可以通过扩展机制来修改页面。

渲染流程

在了解扩展机制之前，需要了解 ZIN 的渲染流程。当用户请求后，ZIN 会根据请求的路径找到对应的页面，然后根据页面上定义的内容进行渲染。主要渲染流程如下：

- 构建节点树
- 渲染节点树
- 输出渲染结果

ZIN 扩展机制允许在上述流程中对节点中的数据、内容、事件进行修改，也可以对最终输出的 HTML 进行修改。

节点构建前扩展

节点构建前扩展允许在构建节点树之前通过回调函数 `onBeforeBuildNode()` 对节点树上的内容进行修改，该方法定义如下：

```php
function onBeforeBuildNode(callable|Closure $beforeBuildNodeCallback): void;
```

其中 `$beforeBuildNodeCallback` 为回调函数，函数签名如下：

```php
function beforeBuildNodeCallback(node $node): void;
```

其中 `$node` 为节点对象。通过在回调函数中修改 `$node` 可以对节点的内容进行修改，下面为一个例子：

```php
/* 下面为页面原定义： */
h1('Hello world!');
p('这是 zin 的示例页面！');
div
(
    setID('myContent'),
    btn('了解更多'),
    html('<p>仍然支持添加复杂的 <strong>HTML</strong></p>')
);

/* 下面为扩展代码： */
onBeforeBuildNode(function(node $node)
{
    if($node->is('h1'))
    {
        $node->props->class->add('text-xl');
    }

    if($node->is('p'))
    {
        /* 标记为从构建中移除。 */
        $node->remove = true;
    }

    if($node->is('#myContent'))
    {
        $node->add('这是一个扩展的内容');
    }
});
```

节点构建扩展

节点构建扩展允许在构建每个节点时通过回调函数 `onBuildNode()` 对节点上的内容进行修改，该方法定义如下：

```php
function onBuildNode(callable|Closure $buildNodeCallback): void;
```

其中 `$buildNodeCallback` 为回调函数，函数签名如下：

```php
function buildNodeCallback(stdClass &$data, node $node): void;
```

其中 `$data` 为节点的构建数据，`$node` 为节点对象。节点数据 `$data` 包含如下属性：

- `before`：需要插入到节点之前的内容；
- `children`：节点内部内容；
- `build`：节点默认构建数据，通常包含节点内部内容；
- `after`：需要插入到节点之后的内容；

需要注意的是上面所有内容属性都是数组。通过在回调函数中修改 `$data` 可以对节点的内容进行修改，下面为一个例子：

```php
/* 下面为页面原定义： */
h1('Hello world!');
p('这是 zin 的示例页面！');
div
(
    setID('myContent'),
    btn('了解更多'),
    html('<p>仍然支持添加复杂的 <strong>HTML</strong></p>')
);

/* 下面为扩展代码： */
onBuildNode(function(stdClass &$data, node $node)
{
    if($node->is('h1'))
    {
        $data->build = array('Hello zin');
    }

    if($node->is('p'))
    {
        /* 标记为从构建中移除。 */
        $data->remove = true;
    }

    if($node->is('#myContent'))
    {
        $data->children[] = '这是一个扩展的内容';
    }
});
```

节点渲染扩展

节点渲染扩展允许在渲染每个节点时通过回调函数 `onRenderNode()` 对节点上的内容进行修改，该方法定义如下：

```php
function onRenderNode(callable|Closure $renderNodeCallback): void;
```

其中 `$renderNodeCallback` 为回调函数，函数签名如下：

```php
function renderNodeCallback(stdClass &$data, node $node): void;
```

其中 `$data` 为节点的渲染数据，`$node` 为节点对象。节点数据 `$data` 包含如下属性：

- `html`：渲染生成的 HTML。

通过修改 `$data->html` 属性可以对节点的渲染结果进行修改，下面为一个例子：

```php
/* 下面为页面原定义： */
h1('Hello world!');
p('这是 zin 的示例页面！');
div
(
    setID('myContent'),
    btn('了解更多'),
    html('<p>仍然支持添加复杂的 <strong>HTML</strong></p>')
);

/* 下面为扩展代码： */
onRenderNode(function(stdClass &$data, node $node)
{
    if($node->is('h1'))
    {
        $data->html = '<h1 class="text-xl">Hello zin</h1>';
    }

    if($node->is('#myContent'))
    {
        $data->html .= '<p>这是一个扩展的内容</p>';
    }
});
```

节点查询

节点查询允许通过函数 `query()` 来查询页面上的节点，并对节点进行修改，该方法定义如下：

其中 `$selector` 为节点选择器字符串或回调函数。选择器字符串支持如下形式：

- `#id`：通过 ID 查询节点；
- `.class`：通过类名查询节点；
- `name`：通过节点名称，包括（HTML5 标签名或部件名）称查询节点；

选择器还支持通过组合使用，规则如下：

- `name#id.class1.class2`：直接组合多个形式作为合并查询条件；
- `#id .class name`：通过空格来组合匹配节点的层级关系，并找到最终的节点；
- `#id1, #id2 name.class`：通过 `,` 来组合可以任意满足的多个条件。

也可以通过回调函数来自定逻辑判断是否匹配给定的节点，回调函数签名如下：

```php
function selectorCallback(node $node): bool;
```

`query` 方法会返回 `query` 实例，该实例包含大部分类 jQuery 方法方便对节点进行修改：

- `remove()`：移除匹配的节点；
- `prop(string|array $name, mixed $value)`：设置节点属性；
- `data(string|array|object $keyOrData, mixed $value = null)`：设置节点 `data-*` 属性；
- `addClass(string ...$classes)`：添加类名；
- `removeClass(string ...$classes)`：移除类名；
- `toggleClass(string $class, ?bool $toggle)`：切换类名；
- `html(string $html)`：设置节点的 HTML 内容；
- `text(string $text)`：设置节点的文本内容；
- `before(mixed ...$content)`：在匹配的节点之前插入内容；
- `after(mixed ...$content)`：在匹配的节点之后插入内容；
- `append(mixed ...$content)`：向匹配的节点添加内容；
- `prepend(mixed ...$content)`：在匹配的节点内部前插入内容；
- `empty()`：清空匹配的节点内部内容；
- `on(string $event, jsCallback $callback)`：绑定事件；
- `off(string $event)`：解绑事件，注意此行为会移除节点上之前绑定的所有指定类型的事件；
- `replaceWith(mixed ...$content)`：替换匹配的节点。
- `closest(string $selector)`：向上查找节点，如果找到第一个节点则返回。
- `find(string $selector)`：在内部查找节点。
- `first(string $selector)`：在内部查找节点，如果找到第一个节点则返回。
- `last(string $selector)`：在内部查找节点，如果找到第一个节点则返回。
- `each(callable|Collator $callback)`：遍历节点。

`query` 实例上的方法都会返回实例本身，这样可以进行链式调用，下面为一个例子：

```php
/* 下面为页面原定义： */
h1('Hello world!');
p('这是 zin 的示例页面！');
div
(
    setID('myContent'),
    btn(setID('myBtn'), '了解更多'),
    html('<p>仍然支持添加复杂的 <strong>HTML</strong></p>')
);

/* 下面为扩展代码： */
query('h1')->addClass('text-xl');
query('p')->remove();
query('#myContent')->append('这是一个扩展的内容');
query('#myBtn')
    ->prop('text', '按钮')
    ->on('click', jsCallback()->call('alert', 'Hello zin!'));
```

提示

建议优先使用 `query()` 的方式来对页面进行扩展，只有 `query()` 无法满足需求时再使用节点构建扩展和节点渲染扩展。

输出扩展

输出扩展允许在输出整个页面渲染结果时通过回调函数 `onRender()` 对渲染结果进行修改，该方法定义如下：

```php
function onRender(callable|Closure $renderCallback): void;
```

其中 `$renderCallback` 为回调函数，函数签名如下：

```php
function renderCallback(stdClass &$data, node $rootNode): void;
```

其中 `$data` 为渲染数据，`$rootNode` 为根节点对象。渲染数据 `$data` 包含如下属性：

- `html`：渲染生成的 HTML。

通过修改 `$data->html` 属性可以对渲染结果进行修改，下面为一个例子：

```php
/* 下面为页面原定义： */
h1('Hello world!');
p('这是 zin 的示例页面！');
div
(
    setID('myContent'),
    btn('了解更多'),
    html('<p>仍然支持添加复杂的 <strong>HTML</strong></p>')
);

/* 下面为扩展代码： */
onRender(function(stdClass &$data, node $rootNode)
{
    $data->html = '<div class="container">' . $data->html . '</div>';
});
```

应用示例

为标签页增加新的标签页

```php
query('#tabs')->append
(
    /* 新增一个标签页： */
    tabPane
    (
        set::title('新标签页'),
        set::key('newTab'),
        div('新标签页内容')
    ),

    /* 自定义 CSS 来对导航菜单重新排序： */
    css
    (
      <<<'CSS'
        #tabs .nav-tabs .nav-item[data-key="key1"] {order: 1}
        #tabs .nav-tabs .nav-item[data-key="key3"] {order: 2}
        #tabs .nav-tabs .nav-item[data-key="key2"] {order: 3}
      CSS
    )
);
```

修改表单部件的 fields 属性

```php
query('formGridPanel')->each(function($node)
{
    /* 获取 fields 属性： */
    $fields = $node->prop('fields');

    /* 修改 name 字段： */
    $fields->field('name')
        ->label('名称')
        ->required();

    /* 添加新字段： */
    $fields->field('newField')
        ->label('新字段')
        ->required();

    /* 对字段列表中的字段进行排序： */
    $fields->sort('$BEGIN,newField,name');

    /* 重新设置 fields 属性： */
    $node->setProp('fields', $fields);
});
```