皮肤制作指南

知识产权

开源代码许可关联告知

在各种开源代码许可中，XE遵循 LGPL(GNU Lesser General Public License) v2版本。因LGPL v2与v3两个版本之间有着细微的差异，用户须知悉其具体版本。LGPL许可大体上等同于GPL许可之外，有着更限制性的适用范围。
与GPL相同，修改或衍生遵循LGPL许可的代码，则所有的代码，涉及修改衍生的代码都必须采用相同的许可。但是与修改或衍生遵循该许可的代码必须要无条件开源的GPL许可不同，采用了LGPL许可的代码在特定条件下可以允许其代码不开源。因此修改或衍生LGPL许可的代码可以用于开发私有软件。
更多信息请参见下列站点。
LGPL 许可: http://www.gnu.org/copyleft/lesser.html
GPL 许可: http://www.gnu.org/licenses/gpl.html

文档信息

文档概要

本文档阐述XE皮肤的制作方法。该文档以 XE core 1.4.4.2版本为基准。

目标读者

本文档的主要目标读者为希望制作各式各样的XE皮肤，用来构筑自主站点的用户。XE皮肤主要采用HTML, CSS, Javascript, jQuery 来制作。因此，如果事先对相关知识有所了解，可以更快地熟悉制作皮肤的方法。

关于XE的安装与使用，请参考“XE用户指南”

问题与建议

如果您查阅本文档发现有什么错误或对本文档有任何疑问，请按下文所列的电子邮件地址联系我们。

电子邮件: developers@xpressengine.com

文档版本及历史

版本	日期	备注
1.0	2010.12.31	发布 1.0
1.1	2011.12.06	基于 XE core 1.5 修订
1.1-zh	2011.11.30	中文版

标注规则

参考

> **参考**
>
> 阐述读者可以参考的内容。

注意

> **注意**
>
> 阐述读者务必要知悉，或可能导致系统错误，又或没有履行导致的财产上的损失等等事项。

窗口名称 / 站点名称 / 菜单名称 / 填写项名称 / 可选项以及符号标记

本文档中对窗口名称、站点名称、菜单名称、填写项名称、可选项做如下标记。

- 窗口名称: **窗口名**称(代码中使用的符号不用遵守本规则)
- 站点名称: 'Naver桌面客户端下载'站点
- 菜单名称: **菜单 > 子菜单**
- 填写项: 填*home page*.

代码范例

本文档中将代码范例用灰底黑字来标识。

```
COPYDATASTRUCT st;
st.dwData = PURPLE_OUTBOUND_ENDING;
st.cbData = sizeof(pp);
st.lpData = &pp;
::SendMes(GetTargetHwnd(), WM_COPYDATA, (WPARAM)this->m_hWnd, (LPARAM)&st);
```

目录

表、图示目录

表目录

图示目录

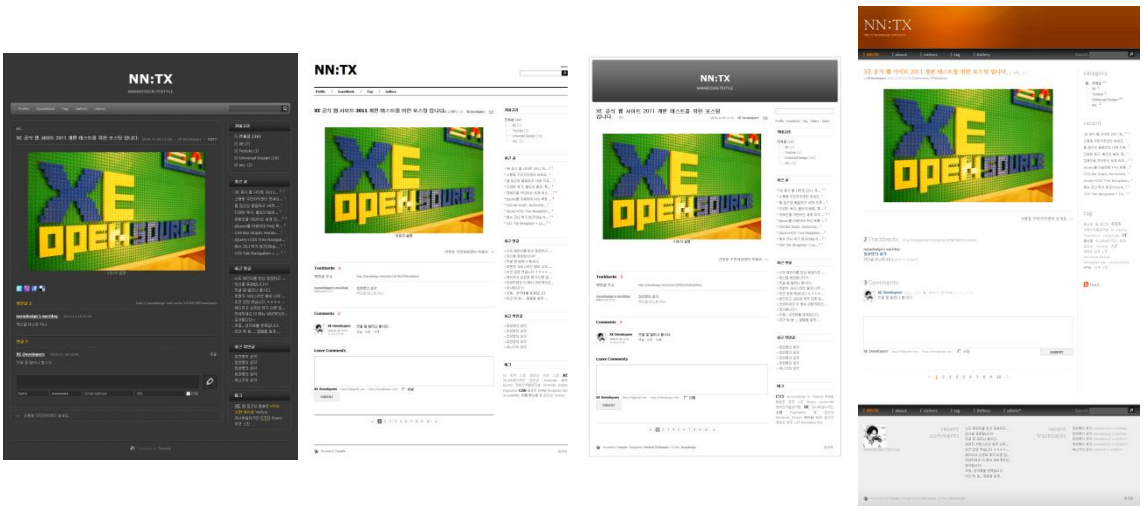
1. XE 皮肤制作概要

这章主要介绍XE皮肤的定义与制作皮肤所必需的要素。

1.1 什么是 XE 皮肤

皮肤是用来把XE生成保存的数据显示在用户界面的样式。模块(module)，布局(Layout)，控件(Widget)，控件样式(Widget Style)等等都拥有一个以上的自己的皮肤。XE用户可以从XE的官方站点(http://www.xpressengine.com)下载各种皮肤，或可以参考本文档自行制作一套属于自己的皮肤。

下图为一个站点(Textyle)套用不同皮肤的示例。



图示 1-1 套用不同皮肤的页面

模块和控件的皮肤位于该模块或控件的’ skins’ 路径中。布局与控件样式的皮肤位于该布局或控件样式的安装文件夹中。

1.2 XE 皮肤制作所必需的要素

制作XE皮肤，事先要了解HTML, CSS, Javascript, XE模板语言。

HTML (Hyper Text Markup Language)

HTML是一种描述网页页面结构的语言。把所要展现内容的标题、段落、目录等等要素使用HTML标签编写后通过浏览器解析并最终会渲染成网页。当按照HTML标准语法并赋予其语义编写后的HTML文档，将便于搜索引擎抓取，也可在各种环境便于访问。

CSS (Cascading Style Sheet)

如果说HTML是负责网页中结构的语言，那么CSS则是负责网页样式的语言。网页浏览器预先定制有默认的表现形式，也可以自己利用CSS使这些修改成更好的，更符合自己要求的样式。选择一个元素并赋予该元素CSS属性后，可以对其在页面中的布置、颜色、线条、样式、背景、图片等等因素都可以随心所欲地展现出来。

Javascript 与 jQuery

HTML, CSS主要负责静态的显示，Javascript则主要负责XE文档的动态的表现。使用Javascript可以不用移动页面也可以把运行结果显示在当前页。Javascript也可以预先下载内容并按需求显示或隐藏这些内容，又可以检查用户填写的内容后标识出现的错误或遗漏。 为了便于利用Javascript，XE使用jQuery库。

因为Javascript不是在所有的环境中得到支持，可以仅使用HTML来实现的功能严重依赖于Javascript, 会导致互用性(Interoperability)下降。因此使用Javascript时，始终要考虑那些不能使用Javascript的环境。

XE 模板语言

XE 模板语言在服务器端会解释为PHP语言，并从数据库查询数据显示在用户界面。使用XE语言能在被允许的范围内扩展或缩小模块、控件的功能。

2. XE 皮肤制作的基础

XE主要介绍XE皮肤制作中所必需的HTML. CSS, Javascript/jQuery，和XE模板语言。

2.1 了解 HTML

HTML具有一种给数据赋予含义使其章乱无序的数据变成有用信息的魔力。要布置和修饰HTML文档，就要使用CSS. 区分好数据的’含义’与’表现’后使用编写的HTML便于他人理解编辑。将文档的‘表现’用CSS分离出来，此后就可以设计改版时减少对HTML的修改工作。

2.1.1 元素、属性、值

HTML由元素、属性和与其匹配的值来组成。

元素（Element)

```
<h1 title="Xpress Engine">XE</h1>
```

由<h1>开始到</h1>结束的部分为“元素”。这一元素被称为<h1> 元素，而<h1>与</h1>被各自称为“开始标签”与“结束标签”。中间部分的’XE’则称为它的内容。

'h'为'heading'的缩略词，有很多标签都属于缩略词。 数字则表示标题的级别，'1'表示最高等级。大部分浏览器都会给标题加大加粗。不过Web开发者不能仅仅为了使字体加大加粗就是用这个元素。没有赋予数据任何语义仅仅是加大加粗属于‘表现’的范畴，应该交由CSS来处理。

属性(Attribute)

```
<h1 title="Xpress Engine">XE</h1>
```

这段代码中’title’为属性。 ‘属性’表示该元素相比别的元素有什么区别。’title’ 属性起着模糊说明该元素的意义的参考标题作用。不同的元素可以设定的属性都不同。

值(Value)

```
<h1 title="Xpress Engine">XE</h1>
```

属性必须去匹配一个值。'Xpress Engine'为 'title' 这一属性的值。值具体定义该元素有着什么样的属性，属性也会直接影响画面上的显示。有时输入了错误的值则有可能不能正确地显示画面。值按属性有时是已定义好的，有时则开发者自己需要自定义属性的值。这里的'Xpress Engine'为自定义值。

2.1.2 HTML 的开始与结束

为了明示数据的范围，所有的HTML元素都有开始与结束。

```
<h1 title="Xpress Engine">XE</h1>
```

- 这一元素的开始部分是 <h1>. 首级标题就是从这里开始。
- 这一元素的结束部分为 </h1>. 首级标题到这里便结束。

当不是文本内容的情况下，元素本身可能会是数据本身。此时，元素即为开始部分也是结束部分。下列情况，图片不包含其他内容，因此元素标签在开始的同时又被闭合。.

```
<img />
```

2.1.3 父级元素与子元素

HTML中元素可以包含另一个元素。

如下例，让我们做一个当点击‘ XE’ 时，移动到初始画面的内容。

```
<h1 title="Xpress Engine">
   <a href="index.html">XE</a>
</h1>
```

<h1>元素包含着 <a> 元素。如上例所示，一个数据可以被多层元素嵌套。此时，如<h1>位于外层的元素叫‘父级元素’ ，而像<a>元素位于里面被父级元素所包含的元素叫‘子级元素’ 。

<a>元素帮助数据链接到另外的可参照资源。href作为<a>的属性指定参照对象。即如上例代码所定义的是：‘ XE为’ 首级标题并参照index.html文件。

需要注意的是，多重嵌套元素时必须保持其开始结束的顺序。 如下代码是违背了嵌套规则的错误代码。

```
<h1 title="Xpress Engine">
   <a href="index.html">XE</h1>
</a>
```

2.1.4 内联元素与块元素

块元素被视为一个‘ 独立的块’ ，可以占据一整行。 ，<h>，<p> 等都属于块元素。内联元素作为‘行’里的一部分，并不能换行只可以并排连续使用。，<img>，<a> 等属于内联元素。

块元素可以包含内联元素而内联元素不可以。要分清哪些是块元素或内联元素，可以参照HTML标准明细。

下例代码为块元素包含内联元素的正确范例。<h1>元素为块元素，<a>元素为内联元素。

```
<h1><a>XE</a></h1>      <!—正确-->
```

下例代码则是内联元素包含块元素的错误示例.

```
<a><h1>XE</h1></a>      <!—错误-->
```

参考

W3C HTML 4.01 规格: http://trio.co.kr/webrefer/html/cover.html
W3C HTML 4.01 元素索引: http://trio.co.kr/webrefer/html/index/elements.html
W3C HTML 4.01 属性索引: http://trio.co.kr/webrefer/html/index/attributes.html
W3C XHTML 1.0 规格: http://trio.co.kr/webrefer/xhtml/overview.html
W3C 标准编码验证: http://validator.w3.org/
W3 Schools 在线资料库: http://www.w3schools.com/

2.2 了解 CSS

如果说HTML是给Web文档赋予语义将数据变成‘数据’，那么CSS则负责关于文档的排编与表现，使内容变得易于理解并变得好看。一般情况下，XE皮肤中CSS文件与HTML文件是分别存放的，但是HTML文件始终可以参照CSS文件，所以用户看到的是两者都得到反映的最终页面。学会CSS语法后，可以使文档变得很好看，更可以将HTML使用得更合理更富有语义。

2.2.1 选择器、属性、值

CSS由‘选择器’、‘属性’、‘值’组成。‘选择器’选择HTML的指定元素，‘属性’与‘值’来定义这些HTML元素在页面怎样去显示。

选择器(selector)

```
h1 { font-size:24px; }
```

'h1'为‘选择器’。指<h1> 元素。

属性(property)

```
h1 { font-size:24px; }
```

'font-size'为属性。用来控制<h1> 元素的字体大小。

值(value)

```
h1 { font-size:24px; }
```

'24px'为值。具体指定 <h1> 元素的 font-size 属性(字体大小)值。

2.2.2 CSS 选择器的种类

CSS 选择器起着选择所需的HTML元素的作用。使用选择器可以给特定的元素应用应有的CSS样式。CSS支持多种类型的选择器。

类型选择器선택자(type selector)

```
h1 { ... }
```

指直接借用 HTML 名来使用的方式。只要从HTML文档中发现与之符合的相同元素名便可以选择这些元素。可以使用标准的HTML元素名作为选择器。

ID 选择器 (id selector)

```
#selector { ... }
```

id选择器在选择名前加数字符号(#). 从HTML文档中找到与该id名相符的元素。id名可以由用户自己定义。其命名规则如下：

- 可以任意使用所有自然语言里的词语。推荐使用大小写字母和数字。
- 不能用特殊字符来作为选择器名，但可以使用下划线（_)与连字符（-).
- 首字符不能为数字。

HTML 文档不允许一个页面有两个以上的id. id名在一个HTML页面中必须是唯一的。

类选择器（class selector)

```
.selector { ... }
```

类选择器在选择名前加点(.)字符。从HTML文档中找到与该类名相符的元素。类名可以由用户自己自己定义。类命名规则如下：

- 可以任意使用所有自然语言里的词语。推荐使用大小写字母和数字。
- 不能用特殊字符来作为选择器名，但可以使用下划线（_)与连字符（-).
- 首字符不能为数字。

HTML 文档允许一个页面使用两个以上的元素使用相同的类名。

子元素选择器（child selector)

```
h1>a { ... }
```

子元素选择器在两个选择器之间使用大于号(>)来发生作用。所罗列的选择器符合HTML元素的层级顺序便可选择成功。其中有层叠超过两层，便不会发生作用。子元素选择器只选择紧跟着父级元素的子元素。

> **注意**
>
> IE6 浏览器不支持子元素选择器。

后代选择器（descendant selector)

```
h1 a { ... }
```

后代选择器在两个选择器之间使用空格来发生作用。 所罗列的选择器符合HTML元素的层级顺序便可选择成功。如下例代码会选择被 <h1>所包含的 <a> .

```
<h1><a>XE</a></h1>
```

在<h1> 元素外层的 <a> 元素不受此影响。

通用选择器（universal selector)

```
* { ... }
```

通用选择器使用星号(*).该选择器所有的HTML元素。可以用来把浏览器预先定制的默认表现形式初始化。

使用通用选择器导致页面显示速度下降。因此在没有必要的情况下不推荐使用。

伪选择器(pseudo selector)

```
a:focus { ... }
```

伪选择器紧跟在另外一个选择器后使用冒号(:)来选择。伪选择器具体分为伪类选择器与伪元素选择器。与其他选择器选择静态的HTML元素不同，伪选择器用来捕捉HTML元素的动态变化或没有直接写在代码里的虚拟的元素。

伪类选择器的种类如下所示。

表 2-1 伪类选择器

伪类选择器	说明	实例
:link	选择没有访问过的链接。 结合 'a' 类型选择器来使用	a:link
:visited	选择以访问过的链接 结合 'a' 类型选择器来使用	a:visited
:hover	选择当鼠标移动至其上方时的状态。	a:hover
:active	选择鼠标点击的瞬间或按回车时的状态。	a:active
:focus	选择被聚焦的元素的状态	a:focus
:first-child	选择首个子元素	li:first-child
:lang(language)	选择语言属性相一致的元素	p:lang(en)

注意

IE6浏览器不支持 ':first-child, :lang()' 伪类浏览器。 IE6 浏览器下 :hover, :active, :focus 伪类选择器只能对 'a'元素起作用。

'a' 在给'a'元素指定多种伪类选择器时，推荐保持 :link -> :visited -> :hover -> :active -> :focus 的顺序。因为后定义的选择器的优先级要高于早先定义的选择器会覆盖前面的定义。比如先定义 :hover 后再定义a:visited,会导致鼠标移到已访问过的链接上方后:hover的定义也不会起作用。要使鼠标移到已访问连接上方时还发生作用，就必须把 :hover放在:visited 后。

伪元素选择器的种类如下所示。

表 2-2 伪元素选择器

伪元素选择器	说明	实例
:first-line	选择首行 只适用于块元素。	p:first-line
:first-letter	选择首字 只适用于块元素。	p:first-letter
:before	选择元素开始部分内部的伪元素。	div:before{ content:"..." }

伪元素选择器	说明	实例
		(选择 div 元素的开始部分内部生成:"..." 伪字符串。)
:after	选择元素结束部分的伪元素。	div:after{ content:"..." } (选择 div 元素的结束部分内部生成:"..." 伪字符串。)

注意

IE6 浏览器不支持伪元素选择器。

选择器的分组(Selector Grouping)

```
.apple,
.tomato { color:red }
```

用逗号将需要分组的选择器分开后，其属性与值只需定义一遍。需要多个选择器共享相同的属性与值时，可以将这些选择器分组。

上面的代码与下列代码是起相同作用的。

```
.apple { color:red }
.tomato { color:red }
```

相邻兄弟选择器（Adjacent Sibling Selector)

```
h1+h2 { color:red }
```

相邻兄弟选择器在选择器与选择器之间使用加号(+)来发生作用。紧接在另一个元素后的元素， 所罗列的选择器符合HTML元素的位置顺序便可选择成功。

如上例代码。<h2>元素按顺序紧跟在<h1> 元素后，因此<h2>元素将被显示为红色字体。

```
<h1>XE</h1>
<h2>Textyle</h2>
```

注意

IE6 浏览器不支持相邻兄弟选择器。

属性选择器（Attribute Selector)

```
input[checked] { ... } /* input 使用到 checked 属性，则会被选择。 */
input[type=text] { ... } /* input 元素中 type 属性为 text时会被选择。 */
img[alt~=dog] { ... }  /* img 元素中 alt 属性值包含 'dog'字符时被选择。 */
p[lang|=en] { ... } /* p 元素中 lang 属性如 en-us 由 en-开始，则会被选择。*/
```

利用HTML 元素中定义的属性来选择该元素。属性选择器的分类如下：

表 2-3 属性选择器

属性选择器	说明	实例
[attribute]	HTML 元素中使用到 attribute 属性，则不管其值是否被定义，都会被选择。	input[checked]
[attribute=value]	HTML 元素中使用了 attribute 属性，而且只定义了单个值为 value，则选择该元素。	input[type=text]
[attribute~=value]	HTML 元素中使用了 attribute 属性，并且定义有两个以上的值被空格分开时，其中有一个值为 value，则会被选择。	img[alt~=dog]
[attribute|= value]	HTML 元素中使用了 attribute 属性，并且其值以 value 开头，则会被选择。	p[lang|=en]

注意

IE6 浏览器不支持属性选择器。

参考

CSS 규격 한글 번역문: http://trio.co.kr/webrefer/css2/cover.html

CSS 参考手册 http://w3school.com.cn/css/css_reference.asp

CSS Reference: http://www.w3schools.com/css/css_reference.asp

2.3 使用 Javascript 与 jQuery

XE为了方便使用Javascript 采用了jQuery库。皮肤制作中即使对Javascript不太熟悉，使用jQuery可以很方便的操作Javascript.

这里将对制作XE皮肤时在HTML文档载入Javascript的方法进行说明。

2.3.1 调用 jQuery 库

XE core中已包含jQuery库并可以在任何页面使用它，因此无需另外载入jQuery库。使用XE生成的所有页面中<head>元素内部都会包含载入 jQuery 的代码。

```
<script type="text/javascript" src="./common/js/jquery.js"></script>
```

Javascript代码虽然可以写在HTML文档中的<head>元素或<body>元素内部。但是在XE内部jQuery库的载入代码会写在<head>元素内部。这是因为，浏览器解析Javascript代码是按照载入代码的顺序来解析的，而且Javascript的执行顺序与页面的显示结果也有密切的关系。为防止依赖于jQuery的js代码比jQuery库优先载入，XE把jQuery库的载入代码放到了HTML文档里<head>元素的内部。

> **参考**
>
> XE 安装路径不同，jquery.js文件的载入路径可能与上例代码不同。

2.3.2 使用 XE 模板语言来调入 Javascript.

jQuery库需要使用Javascript来写代码。Javascript的调用要写在HTML文档的<head>或<body>内部而不可以写在别的位置导致HTML语法错误。一些严格遵守标准的的浏览器可能不会执行在HTML语法不允许的位置调用的Javascript代码，调用即可以将Javascript代码直接写在HTML文档，也可以分离为单独的*.js文件

Javascript调用位置按需求将分为两个部分。

<head> 元素中的 JS 文件调用

如果需要浏览器在完成页面渲染之前对一些内容隐藏或显示，那么这些JS代码最好写在<head>元素内部。如果此时JS代码是写在<body>元素内部，那么Javascript会比HTML解析的速度慢导致页面显示异常。

包含在<head>元素内部的Javascript会早于HTML文档被解析，编写时须注意要等到HTML文档全部载入完毕后才可以执行。

<body> 元素中的 JS 文件调用

如果Javascript在HTML文档载入时没有需求去决定关于一些内容的显示隐藏，那么最好写在HTML文档的<body>元素内部的最后。这是因为浏览器不会同时解析HTML代码与Javascript代码，而是按照编写的顺序来解析的。因此将Javascript代码放置在HTML文档的最后，则可以让浏览器优先执行HTML部分的显示工作。

使用XE 语法来调用Javascript部分可以参照"通过使用 index 属性可以改变CSS文件调用顺序。 <load /> 与index属性可以在XE core 1.4.4 以上版本使用。

index属性值可以为正•负整数。使用负整数可以优先调用，而使用正整数可以将载入优先度滞后。指定 index="-1" 时，在其他CSS文件的上一行载入。

当CSS 中出现属性值冲突时，只有最后定义的值才会生效。因此推荐将优先度高的文件放到最后去调用。

"部分。

2.3.3 使用标准语言调入 Javascript 脚本

需要调用单独的JS文件，可以使用XE模板语言选择<head>元素内部或<body>元素内部两个位置。但是需要精确的调用到所需要的位置可以使用HTML标准语法。如下例代码。

```
<body>
    ...
    <script type="text/javascript" src="xxx.js"></script>
    ...
</body>
```

Javascript可以直接放在HTML文档里而不用另行保存单独的JS文件。如下例代码。

```
<body>
    ...
    <script type="text/javascript">
    // <![CDATA[
        在这里编辑Javascript代码。
    // ]]>
    </script>
    ...
</body>
```

参考

在Javascript代码的开始与结束部分定义 <![CDATA[...]]> 是为了防止Javascript代码被解释为HTML代码。没有 <![CDATA[...]]> 时，' <' ,' >' 和一些Javascript运算符会被解释为HTML元素或HTML实体(Entity)的开始。 <![CDATA[...]]> 因为不是Javascript语法中的一部分，所以要加上 // 注释。

2.3.4 解析 HTML 后执行 jQuery

由XE生成的文档都可以调用jQuery库，因此<script>内部可以使用jQuery语法。可如果在HTML文档被完整解析前执行Javascript代码，会使一些参照HTML结构的Javascript语句出错。为防止这些错误，可以让jQuery的解析与执行非同步。

下方法定义jQuery函数，可以等到HTML文档完全载入玩后去执行jQuery语句。

```
<script type="text/javascript">
// <![CDATA[
```

```
    jQuery(function($){
        在这里编辑jQuery 语句。
    });
// ]]>
</script>
```

注意

jQuery 语法中 'jQuery'可以用美元符号($) 替换使用。但是在XE中为了避免与其它使用美元符号($) 的Javascript库发生冲突，在jQuery函数开始部分并不允许使用 美元符号($)来替换。因此在XE编写第一个jQuery函数时须写成 jQuery(function($){ ... }) .

2.3.5 体验 jQuery 的工作方式

jQuery 的工作方式可以简单归纳为“选择并执行”。这句话可以解释为“选择HTML元素后在必要时执行动作”。

先参考简单的例题

```
<head>
    <style type="text/css">
    #help { display:none; }
    </style>
    <script type="text/javascript" src="jquery.js"></script>
</head>
<body>
    <a href="#help" class="helpBtn">도움말</a>
    <p id="help"> 这里被CSS样式定义为隐藏。 <p>
    <script type="text/javascript">
    // <![CDATA[
    jQuery(function($){
        $('a.helpBtn').click(function(){ // 选择HTML 元素
            $('p#help').css('display','block'); // 执行动作
        });
    });
    // ]]>
    </script>
</body>
```

这段代码执行结果为：当点击'a.helpBtn'的元素后，将把样式状态为display:none的'p#help'的状态变成display:block . jQuery函数先选择了HTML元素并等待用户发起的事件去变更该元素的CSS样式。

因为显示或隐藏的动作是使用比较频繁的功能，所以方便处理这些事情jQuery内藏show(), hide()函数。 上例代码可以写的更简单。

```
<head>
    <style type="text/css">
    #help { display:none; }
    </style>
```

```
    <script type="text/javascript" src="jquery.js"></script>
</head>
<body>
    <a href="#help" class="helpBtn">  帮助  </a>
    <p id="help">  这里被CSS样式定义为隐藏。<p>
    <script type="text/javascript">
    // <![CDATA[
        jQuery(function($){
            $('a.helpBtn').click(function(){ // 选择 HTML 元素
                $('p#help').show(); // 执行 show() 函数
            });
        });
    // ]]>
    </script>
</body>
```

下例代码改进了动作，使之拥有在点击 'a.helpBtn' 链接时实现显示或隐藏的开关(Toggle)效果。

```
<head>
    <style type="text/css">
    #help { display:none; }
    </style>
    <script type="text/javascript" src="jquery.js"></script>
</head>
<body>
    <a href="#help" class="helpBtn">  帮助  </a>
    <p id="help">  这里被CSS样式定义为隐藏。<p>
    <script type="text/javascript">
    // <![CDATA[
        jQuery(function($){
            $('a.helpBtn').click(function(){ // 选择 HTML 元素
                $('p#help').toggle(); // 执行 toggle() 函数
            });
        });
    // ]]>
    </script>
</body>
```

到这里大概地了解了jQuery 的使用与工作方式。更多更详细的方法与功能(很多种选择元素方法、更多动作)可以参考jQuery 帮助文档或有关书籍。

参考

关于使用jQuery来选择HTML元素并执行动作的详细内容可以参照 jQuery 官方网站(http://jquery.com/).

2.4 XE 模板语言

XE 模板语言在服务器端被转换为PHP语法，并从数据库查询所需要的数据显示在用户界面。使用XE 模板语言可以在允许的方位内扩展或限制模块或控件的功能。

使用XE 模板语言制作 XE 皮肤的方法共有四种。

- 在HTML 注释 <!--...-->内部编写的方法， 例如： <!--@if(...)-->...<!--@end-->
- 在虚拟的 <block> 元素内部编写的方法，例如: <block>...</block>
- 直接编写在 HTML 元素内部的方法，例如: ...
- 不依赖注释或元素直接编写的方法，例如: 使用{$content} 内容变量来输出数据。

> **参考**
> <block> 元素并不是 HTML 标准元素，而是 XE core 1.4.4 版新增的虚拟元素。它假借HTML元素的形式来执行语句而并不输出到页面。cond 属性也是 XE core 1.4.4 版新增的虚拟属性起着条件语句功能。.

2.4.1 变量

变量可以输出程序事先定义的内容或用户输入的内容。用户在页面上看到的大部分内容几乎全部都是依赖变量来输出的。

变量的使用方法如下。

基本形式

```
{$string}
```

变量的基本形式是在花括号（{ }）里填写变量名并在变量名前加（$）. 变量名必须用字符串'문자열(string)'来命名并且只能使用已定义过的变量。

变量嵌套另一个变量

```
{$string->string}
{$string->string->string}
```

变量可以包含另一个变量，要使用它需要用连字符与大于号（->）来连接变量与变量。

变量嵌套函数

```
{$string->string()}
{$string->string(string | integer)}
```

连接变量与函数也需使用连字符与大于号(->). 函数名后要加括号（( )). 括号里可以包含字符串（string)或整数(integer). 函数名只能使用已定义的函数。

2.4.2 XE core 变量

'XE core 变量'是指 XE core内部使用的变量。XE core变量可以在大部分模块使用。还有一种变量只能在特定模块中使用。这些只称为 '变量'. XE core 变量如下所示：

表 2-4 XE core 变量

变量	说明
$is_logged	确认用户的登录状态 <!--@if($is_logged)-->Welcome!<!--@end--> Welcome!
$current_url	当前页 URL
$request_uri	XE core 安装 URL
$logged_info	给已登录会员显示自己的会员信息
$logged_info->member_srl	会员固有号
$logged_info->user_id	已登录用户 id
$logged_info->email_address	已登录用户电子邮件地址
$logged_info->email_id	已登录用户电子邮件 id
$logged_info->email_host	已登录用户电子邮件主机名
$logged_info->user_name	已登录用户姓名
$logged_info->nick_name	已登录用户昵称
$logged_info->homepage	已登录用户个人主页
$logged_info->blog	已登录用户博客
$logged_info->birthday	已登录用户生日 (YYYYMMDD)
$logged_info->profile_image	已登录用户 Profile Image
$logged_info->image_name	已登录用户图片昵称
$logged_info->image_mark	已登录用户组图片路径
$logged_info->signature	已登录用户个人签名
$logged_info->group_list	已登录用户参与组(Group)列表
$logged_info->is_admin	确认已登录用户是否为管理员
$logged_info->is_site_admin	确认已登录用户是否为虚拟站点管理员
$module_info	显示当前模块信息
$module_info->module	模块名称

变量	说明
$module_info->use_mobile	是否使用移动版皮肤
$module_info->mid	模块 id
$module_info->skin	模块皮肤名
$module_info->mskin	模块移动版皮肤名
$module_info->browser_title	模块文档标题
$module_info->string	模块扩展变量

2.4.3 条件句

在特定条件下将所需的内容按需求显示或不显示时，需要用到条件句。条件句由 ‘if’, elseif, else, end’ 与 ‘条件式’ 构成。如果以 ‘if’ 句开始，则需要 ‘end’ 句来结束条件句。因为条件句里的内容最终会利用PHP来解析，因此可以使用PHP里一些常用的运算符((&&, ||, ==, !) 。

以下是为显示 ” Welcome XE” 而使用的各种条件句。

表 2-5 条件句使用范例

条件句	说明	备注
<!--@if(条件句)--> Welcome XE! <!--@end-->	条件为真时，显示内部内容。	
<block cond=" 条件句 "> Welcome XE! </block>	条件为真时，显示内部内容。	XE core 1.4.4 以上
 Welcome XE! 	条件为真时，显示包含 元素的内容。	XE core 1.4.4 以上
<p attr="value"|cond="조건식"> Welcome XE! 	显示 元素，条件为真时，显示 attr=” value” 属性值。	XE core 1.4.4 以上

参考

<block> 元素并不是 HTML 标准元素，而是 XE core 1.4.4 版新增的虚拟元素。它假借HTML元素的形式来执行语句而并不输出到页面。cond 属性也是 XE core 1.4.4 版新增的虚拟属性起着条件语句功能

一并使用if, elseif, else语句，除了可以按’ TRUE’ 或’ FALSE’ 来判断，还可以赋予各种条件来显示不同结果。

```
<!--@if(条件A)-->
    <p> 满足条件A,则显示此内容。 </p>
<!--@elseif(条件B)-->
    <p> 不满足条件A,满足B,则显示此内容。 </p>
<!--@else-->
    <p> 同时不满足条件A与条件B则显示此内容。 </p>
<!--@end-->
```

以上代码中的条件句通过使用 XE core 1.4.4版本新增的’ cond’ 条件句来简化，如下例代码：

```
<p cond="条件A"> 满足条件A,则显示此内容。 </p>
<p cond="条件B"> 不满足条件A,满足B,则显示此内容。 </p>
<p cond="!条件A && ! 条件B"> 同时不满足条件A与조건B则显示此内容。 </p>
```

2.4.4 循环句

需要按照特定条件来反复循环显示必要的内容，则需要使用循环句。条件句由'foreach, end'与 ‘条件式’ 构成。一旦使用foreach语句，则必须使用 end句来结束循环句。

表 2-6 循环句使用范例

循环句	说明	备注
<!--@foreach(变量名 as $val)--> <tr>…</tr> <!--@end-->	不包含$key 值, 循环<tr>…</tr>	
<block loop=" 变量名 =>$val"> <tr>…</tr> </block>	不包含$key 值, 循环 <tr>…</tr>	XE core 1.4.4 以上
<tr loop=" 变量名 =>$val">…</tr>	不包含$key 值, 循环 <tr>…</tr>	XE core 1.4.4 以上
<!--@foreach(变量名 as $key => $val)--> <tr>…</tr> <!--@end-->	包含$key 值, 循环 <tr>…</tr>	
<block loop=" 变量名 =>$key, $val"> <tr>…</tr> </block>	包含$key 值, 循环 <tr>…</tr>	XE core 1.4.4 以上
<tr loop=" 变量名 =>$key,$val">…</tr>	包含$key 值, 循环 <tr>…</tr>	XE core 1.4.4 以上
<!--@foreach($i=0;$i<100;$i++)--> <tr>…</tr> <!--@end-->	初始值为 0, 循环 <tr>…</tr>100 次	
<block loop="$i=0;$i<100;$i++"> <tr>…</tr> </block>	初始值为 0, 循环 <tr>…</tr>100 次	XE core 1.4.4 以上
<tr loop="$i=0;$i<100;$i++">…</tr>	初始值为 0, 循环 <tr>…</tr>100 次	XE core 1.4.4 以上

> **参考**
>
> 'loop' 属性并不是 HTML 标准属性，而是 XE core 1.4.4 版新增的模板语法之一。 'loop'是编写 foreach句时使用的虚拟属性。

2.4.5 使用简单的 PHP 语句

XE 皮肤文件中并不能直接使用 PHP语法。但是可以用花括号({})里包含@符号来使用简单的 PHP 句。

```
{@$is_logged=Context::get('is_logged')}
```

直接使用PHP语句时，须一句一行。如下例代码可以在PHP中毫无问题，在XE中可能会引起致命错误。是因为这一行写了多个PHP句。

```
{@$test=364; $test=$test*$test}
```

上述代码要分开来写，如下例所示：

```
{@
    $test=364;
    $test=$test*$test
}
```

2.4.6 include 句

制作皮肤时部分内容块出现在多个页面可以另存为一个单独文件来管理。这样，一次修改一个文件就可以方便应用到多个页面。include就是引用别的文件到当前页面的命令。

表 2-7 include句使用范例

Include 句	说明	备注
<!--#include(“header.html")-->	引用 header.html (include)	
<include target="header.html" />	引用 header.html (include)	XE core 1.4.4 以上

> **参考**
>
> <include /> 元素并不是 HTML 标准元素，而是 XE core 1.4.4 版新增的虚拟元素。它假借HTML元素的形式来执行语句而并不输出到页面。target 属性也是 XE core 1.4.4 版新增的模板语法。 target指定需要引用的文件路径。.

2.4.7 调用 CSS 文件

这里将阐述关于调用CSS文件到HTML页面的方法。因为从XE core 1.4.4 版开始去除了将多个CSS文件整合为一个文件的optimizer功能，因此制作模块时尽量要写在一个文件而不要写在多个CSS文件。

CSS 调用

```
<!--%import( “xe.css")-->
或
```

```
<load target="xe.css" />
```

HTML 文档的 <head> 元素中调用 xe.css 文件。<load /> 元素可以在 XE core 1.4.4 以上版本 使用。最终代码实现结果如下：

```
<head>
    <link rel="stylesheet" type="text/css" href="xe.css" />
</head>
```

CSS 文件只可以在HTML文档的<head>元素内部被调用。当<body> 元素内部试图载入CSS文件会出现HTML语法错误。

指定 media

```
<load target= “xe.css" media="print" />
```

可以选择指定CSS 文件的媒体对象。<load />元素与 media属性可以在XE core 1.4.4 以上版本使用，最终代码实现结果如下：

```
<head>
    <link rel="stylesheet" type="text/css" href="xe.css" media="print" />
</head>
```

media 属性值可以用逗号(,)来指定多个值。默认值为’ all’ . 可以使用的值如下所示：

表 2-8 media 属性值

属性值	说明
all	没有指定 media 时的默认值. 对应所有设备。
aural	语音合成器（画面朗读装置，屏幕阅读机）
braille	盲人点字法反馈设备
embossed	盲人打印机
handheld	移动设备
print	打印机
projection	投影设备
screen	屏幕（显示器）
tty	电传打字机（Tele Type Writer）
tv	电视机

参考

指定CSS media 属性时，事先必须确认目标设备是否遵守CSS标准。如果该设备不支持，则无法适用指定media属性。.

使用 IE 条件注释

```
<load target= “xe.css" targetie="IE 6" />
```

可以使用targetie 属性来选择性的让CSS文件在指定版本的IE浏览器解释。<load /> 元素与 targetie 属性可以在XE core 1.4.4 以上版本使用。最终代码如下：

```
<!--[if IE 6]>
    <link rel="stylesheet" type="text/css" href="xe.css" />
<![endif]-->
```

这段代码会被IE6浏览器解析并执行，但除此之外的所有浏览器都会当作注释并不会去解析。

变更 CSS 文件调用顺序

```
<load target= “xe.css" index="-1" />
```

通过使用 index 属性可以改变CSS文件调用顺序。<load /> 与index属性可以在XE core 1.4.4 以上版本使用。

index属性值可以为正•负整数。使用负整数可以优先调用，而使用正整数可以将载入优先度滞后。指定 index="-1" 时，在其他CSS文件的上一行载入。

当CSS 中出现属性值冲突时，只有最后定义的值才会生效。因此推荐将优先度高的文件放到最后去调用。

2.4.8 调用 JS 文件

这里将阐述关于调用JS文件到HTML页面的方法。 因为从XE core 1.4.4 版开始去除了将多个JS文件整合为一个文件的optimizer功能，因此制作模块时尽量要写在一个文件而不要写在多个CSS文件。

CSS 文件只可以在HTML文档的<head>, <body>元素内部被调用。当要在<head>, <body>元素外部试图载入JS文件时，会出现HTML语法错误而导致有些浏览器无法解析JS文件。

<head> 元素内部调用 JS 文件

下例代码提供在<head>元素内部调用xe.js文件的方法。

```
<!--%import( “xe.js")-->
或
<load target="xe.js" />
```

<load /> 句式可以在 XE core 1.4.4 以上版本使用。最终代码实现结果如下：

```
<head>
    <script type="text/javascript" src="xe.js"></script>
</head>
```

<body> 元素内部调用 JS 文件

下例代码提供在<body>元素内部调用xe.js文件的方法。

```
<load target="xe.js" type="body" />
```

<load /> 元素可以在HTML文档载入指定的文件。以下是可以使用的属性与值。

- media="all | aural | braille | embossed | handheld | print | projection | screen | tty | tv" – 可以指定CSS文件适用的目标媒体，可以使用逗号 (,)来指定多个值。默认值为 all, 因此省略不填则 media="all".
- targetie="IE 6 | IE 7 | IE8 | ..." – 可以使用 IE 条件注释在各个IE浏览器之中指定对应的CSS/JS文件。默认值为空，不使用。
- index="integer" – 可以变更CSS/JS 调用位置。可以使用正•负整数。取正整数值可以比当前位置滞后调用，而取负整数值可以比当前位置优先调用。默认值为空，不改变顺序。当前顺序为默认值，则比其他文件延迟调用。
- type="head | body" – 可以选择在<head>或<body>内部调用JS文件。默认值为head, 省略时在<head>元素内部调用。

<load /> 语法在 XE core 1.4.4 以上版本可以使用。 <body>

元素内部调用JS文件时，其代码将位于<body>元素结束部分，如下所示：

```
<body>
    ...
    <script type="text/javascript" src="xe.js"></script>
</body>
```

使用 IE 条件注释

```
<load target= "xe.js" targetie="IE 6" />
```

可以使用targetie 属性来选择性的让JS文件在指定版本的IE浏览器解释。<load /> 元素与 targetie 属性可以在XE core 1.4.4 以上版本使用。最终代码如下：

```
<!--[if IE 6]>
    <script type="text/javascript" src="xe.js"></script>
<![endif]-->
```

这段代码会被IE6浏览器解析并执行，但除此之外的所有浏览器都会当作注释并不会去解析。

变更 JS 文件调用顺序

```
<load target= "xe.js" index="-1" />
```

通过使用 index 属性可以改变JS文件载入顺序。<load /> 与index属性可以在XE core 1.4.4 以上版本使用。

index属性值可以为正•负整数。使用负整数可以优先载入，而使用正整数可以将载入优先度滞后。指定 index="-1" 时，在其他CSS文件的上一行载入。

2.4.9 应用 XML JS Filter

XML JS Filter是用XML来定义填写项后，在传送Form数据时自动进行有效性检查的功能。由XML转换过来的Javascript将自动地执行有效性检查，因此用户无须再编写复杂难懂的Javascript代码。XML JS Filter又可以将通过有效性检查的Form数据派送到指定模块的指定命令。

应用XML JS Filter的方法与调用CSS, JS文件的方法相同。

```
<!--%import( “xe.xml")-->
```

如上例代码，调用XML JS Filter后，XML 文档将被编译为 JS 代码。XE core 1.4.4版本以前这些代码会放在<head>元素内部调用。

其结果如下：

```
<head>
    <script type="text/javascript" src="xe.js"></script>
</head>
```

XE core 1.4.4 版本开始会在<body>结束部分调用该代码。

```
<!--%import( “xe.xml")-->
或
<load target= “xe.xml" />
```

结果如下：

```
<body>
    ...
    <script type="text/javascript" src="xe.js"></script>
</body>
```

与XE core 1.4.4以前版本不同点就在于其调用的位置。.

2. 4. 10 嵌入控件(Widget)

控件作为一种与用户交互的界面把XE core或模块(Module)里的数据处理为有意义的信息输出。站点初始页面用来显示最近帖子的控件与会员登录的控件是包含在 XE core里的。控件是为皮肤开发者 不用考虑后台复杂难懂的逻辑也可以实现简单并且有用的功能而存在的。

皮肤开发者用一行模板代码也可以实现所需要的功能。<img />元素添加一个widget属性可为嵌入控件的模板代码。虽然 <img />元素是 HTML 标准元素，但包含 widget属性，则会被解释为 XE 模板语言在服务器端会转换为标准 HTML语言。

下例代码为输出会员登录样式的代码。

```
<img widget="login_info" skin="xe_official" />
```

嵌入控件的代码时，在 XE 管理者页面选择 界面管理 > 控件管理，并使用 **生成代码** 功能后自动生成。详细内容请参照 "XE 用户指南"

2. 4. 11 XE core 1.4.4 新模板語言

这里只整理了 XE core 1.4.4版本新添加的模板语法。如果是从前版本升级到 XE core 1.4.4版本的请参照下列表。

表 2-9 XE core 1.4.4开始新添加的模板语法

新添加的模板语法	说明
<block cond="条件句"> Welcome XE! </block>	条件为真时，显示内部内容。
   Welcome XE! 	条件为真时，显示包含 元素的内容。
<p attr="value"|cond=" 条件句 "> Welcome XE! 	显示 元素，条件为真时，显示 attr=”value” 属性值。
<block loop="变量名=>$val"> <tr>...</tr> </block>	不包含$key 值，循环<tr>...</tr>
<tr loop=" 变量名 =>$val">...</tr>	不包含$key 值，循环<tr>...</tr>
<block loop=" 变量名 =>$key, $val"> <tr>...</tr> </block>	包含$key 值，循环 <tr>...</tr>
<tr loop=" 变量名 =>$key,$val">...</tr>	包含$key 值，循环 <tr>...</tr>
<block loop="$i=0;$i<100;$i++"> <tr>...</tr> </block>	初始值为 0，循环 <tr>...</tr>100 次
<tr loop="$i=0;$i<100;$i++">...</tr>	初始值为 0，循环 <tr>...</tr>100 次
<include target="header.html" />	引用（include）header.html
<load target="xxx.xxx" />	载入 CSS/JS/XML JS Filter 文件到 <head>内部
<load target="xxx.xxx" type="body" />	载入 JS/XML JS Filter 文件到 <body>内部
<unload target="xxx.xxx" />	不载入目标路径一致的 CSS/JS/XML JS Filter 文件

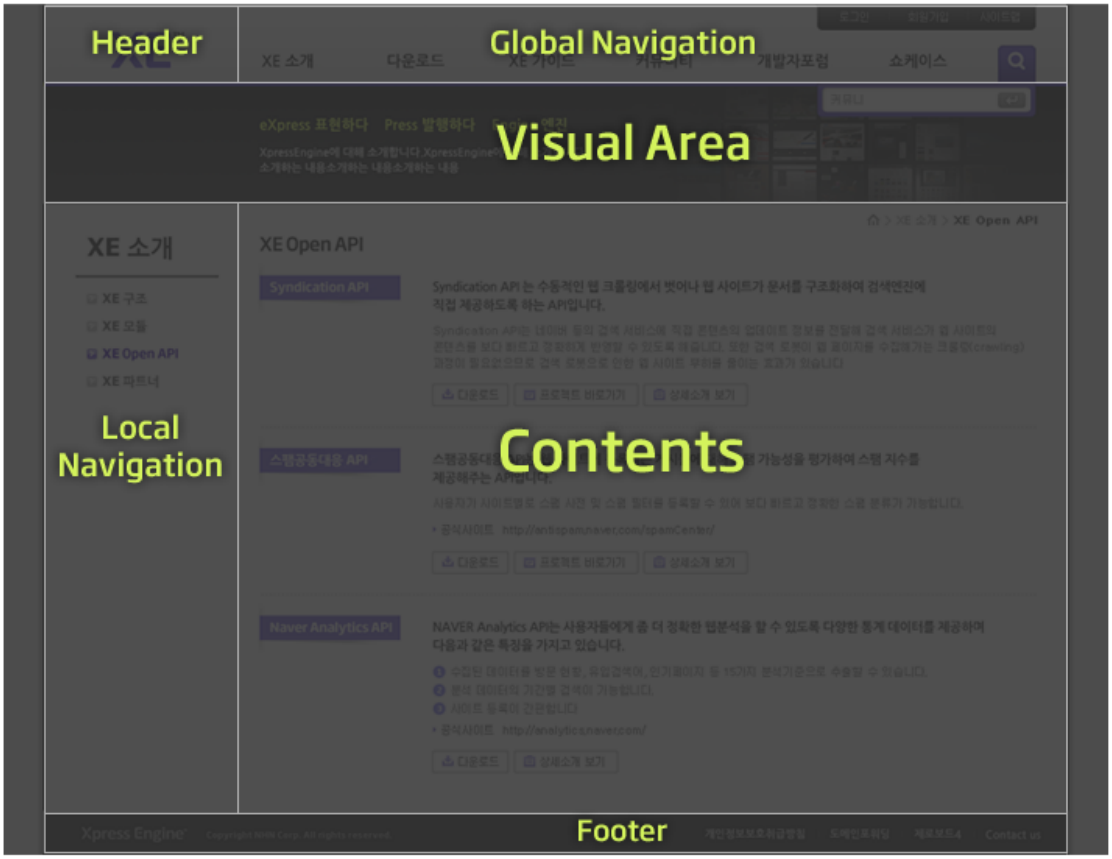
3. 制作布局皮肤(Layout Skin)

这一章主要介绍如何使用布局皮肤样本来制作布局皮肤并使用。

3.1 什么是布局皮肤

在XE里，布局与内容是独立的。布局是一种承载XE内容的一种构造或结构体。一般的网站由头部、全局导航、局部导航、内容、尾部构成。布局的功能就是布置这些结构体在页面中的位置。如果仅仅是把版面(board), Wiki等内容显示出来，则不需要使用布局皮肤。但是除这些内容之外还需要站点有连贯性的导航，并且想布置头部、全局导航、局部导航、尾部等等部分，则必须使用布局皮肤。

下图是使用了布局皮肤的一般页面结构。



图示 3-1 应用布局皮肤的效果图

XE core里包含了不止一个布局皮肤。这些皮肤位于 XE core安装位置的 /layouts/ 里。皮肤制作者即可以使用这些皮肤，也可以自己新制作布局皮肤。

3.2 下载布局皮肤样本

要新制作一个布局皮肤，首先需要按照布局皮肤的结构来生成路径与文件。本文档为方便皮肤制作者提供了布局皮肤样本。皮肤样本是按照制作皮肤所必须遵守的路径与文件结构来制作的，因此可以很方便地改造为自己需要的皮肤样式。

布局皮肤的下载路径为：

- http://doc.xpressengine.com/manual/user_layout.zip

3.3 布局皮肤的位置与路径

3.3.1 确认布局皮肤的位置

如果XE是安装在 /xe/ 这一用户自定义的路径，则布局皮肤的具体位置如下：

```
/xe/layouts/
```

先解压缩已下载的布局皮肤(user_layout)，并复制到 '/xe/layouts/'.

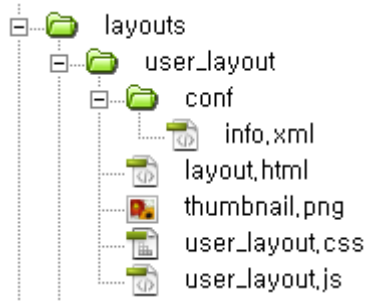
```
/xe/layouts/user_layout/
```

> **参考**
>
> 如果要在本地PC修改布局皮肤，则应把修改的内容反映到站点的布局皮肤路径里。.

3.3.2 布局皮肤路径结构

制作布局皮肤首先必须遵循一定的路径规则。不按照这一结构则不能识别为布局皮肤。

布局皮肤(user_layout)样本的结构如下：



图示 3-2 布局皮肤的路径与结构

info.xml

info.xml包含布局皮肤的基本信息并提供给管理员后台需要显示的说明和选项。info.xml必须存在于以下路径，'conf' 路径名与 'info.xml'文件名不可变更。

```
/xe/layouts/user_layout/conf/info.xml
```

layout.html

layout.html包含布局皮肤的 HTML, CSS, JS的调用信息。'layout.html' 文件名不可修改。

```
/xe/layouts/user_layout/layout.html
```

thumbnail.png

thumbnail.png是布局的预览图。将其宽度设定为180px，高度设定为120px以上的图片放到布局皮肤文件夹，则可以在管理员后台显示为缩略图。 'thumbnail.png'文件名不可修改。

```
/xe/layouts/user_layout/thumbnail.png
```

CSS, JS, IMG

CSS, JS, IMG 文件按实际需求来生成使用，并不是布局皮肤里所必需的。

可以放在除'conf'路径之外的任意位置。可以生成 ‘css, js, img’ 等文件夹进行管理。

3.4 编辑布局皮肤信息

布局皮肤信息需要在 info.xml文档编辑。 info.xml文档的基本结构如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<layout version="0.2">
    <title xml:lang="ko">테스트 레이아웃</title>
    <title xml:lang="en">Test Layout</title>
    <title xml:lang="zh-CN">测试用布局</title>
    <title xml:lang="jp">テストのレイアウト</title>
    <description xml:lang="ko">디자인이 없는 테스트 레이아웃입니다. 새 스킨을 만들 때 이 레이아웃 사본을 사용하면 좋습니다.</description>
    <description xml:lang="zh-CN">无设计要素的测试用布局。新制作布局皮肤时推荐使用该皮肤的副本。</description>
    <description xml:lang="en">Is not designed for testing the layout. When you create a new skin, we recommend you copy the layout.</description>
    <description xml:lang="jp">デザインがないテストレイアウトです。新しいスキンを作成するときに、レイアウトのコピーを使用してください。</description>
    <version>1.0</version>
    <date>2010-12-24</date>
    <author email_address="user@user.com" link="http://user-define.com/">
        <name xml:lang="ko">제작자 이름</name>
        <name xml:lang=" zh-CN">制作者名 </name>
        <name xml:lang="en">Maker Name</name>
        <name xml:lang="jp">製作者名</name>
    </author>
    <menus>
        <menu name="main_menu" maxdepth="3" default="true">
            <title xml:lang="ko">전역 메뉴</title>
            <title xml:lang="ko">全局菜单</title>
            <title xml:lang="en">Global Menu</title>
            <title xml:lang="jp">グローバルメニュー</title>
        </menu>
    </menus>
</layout>
```

以上代码的内容说明如下：

代码	说明
<?xml version="1.0" encoding="UTF-8"?>	XML 文档格式声明
<layout version="0.2">	声明该文档为布局信息文档。version 里应表示 XE core 所支持的版本。支持以 XE core 1.4.4.2 为标准的布局版本为 0.2 版。
<title xml:lang="ko">...</title>	布局名称

代码	说明
<description xml:lang="ko">...</description>	布局说明
<version>...</version>	布局皮肤版本
<date>YYYY-MM-DD</date>	布局皮肤制作日期。 以年-月-日(YYYY-MM-DD) 形式来标识。
<author email_address="..." link="..."> <name xml:lang="ko">...</name> </author>	布局皮肤制作者信息。如电子邮件、个人主页、制作者名等。
<menus> <menu name="main_menu" maxdepth="2" default="true"> <title xml:lang="ko">...</title> </menu> </menus>	可以连接到 XE 管理员后台的控制面板生成的菜单。 <menus>...</menus> 元素内部可以使用两个以上的 <menu>...</menu>元素。

除此之外 info.xml还可以把用户自定义的各种类型变量包含在 <extra_vars> 元素内部。以下就是皮肤制作者把接收到的如 select, image, text, textarea 等类型的数据扩展为可使用变量的实例。

```
<?xml version="1.0" encoding="UTF-8"?>
<layout version="0.2">

    ...
    <extra_vars>
        <var name="colorset" type="select">
            <title xml:lang="zh-CN">色彩方案</title>
            <description xml:lang=" zh-CN ">请选择色彩方案</description>
            <options value="black">
                <title xml:lang=" zh-CN ">Black (默认)</title>
            </options>
            <options value="white">
                <title xml:lang=" zh-CN ">White</title>
            </options>
        </var>
        <var name="logo_image" type="image">
            <title xml:lang=" zh-CN ">LOGO图片</title>
            <description xml:lang=" zh-CN ">请上传页面头部的LOGO图片。</description>
        </var>
        <var name="logo_url" type="text">
            <title xml:lang=" zh-CN ">LOGO图片链接 </title>
            <description xml:lang=" zh-CN ">请填写点击LOGO图片时的前往目标URL.</description>
        </var>
        <var name="title_description" type="textarea">
            <title xml:lang=" zh-CN ">正文头部内容</title>
            <description xml:lang=" zh-CN ">请填写正文头部显示区域的内容。(可以使用HTML)</description>
```

```
        </var>
    </extra_vars>

    ...

</layout>
```

以上代码的内容说明如下：

代码	说明
<extra_vars>...</extra_vars>	扩展变量容器
<var name="colorset" type="select">...</var>	select 类型的扩展变量。皮肤制作者可以用 {$layout_info->colorset} 形式来使用变量。
<var name="logo_image" type="image">...</var>	image 类型的扩展变量。皮肤制作者可以用 {$layout_info->image} 形式来使用变量。
<var name="logo_url" type="text">...</var>	text 类型的扩展变量。皮肤制作者可以用 {$layout_info->logo_url} 形式来使用变量。
<var name="title_description" type="textarea">...</var>	textarea 类型的扩展变量。皮肤制作者可以用 {$layout_info->title_description} 形式来使用变量。

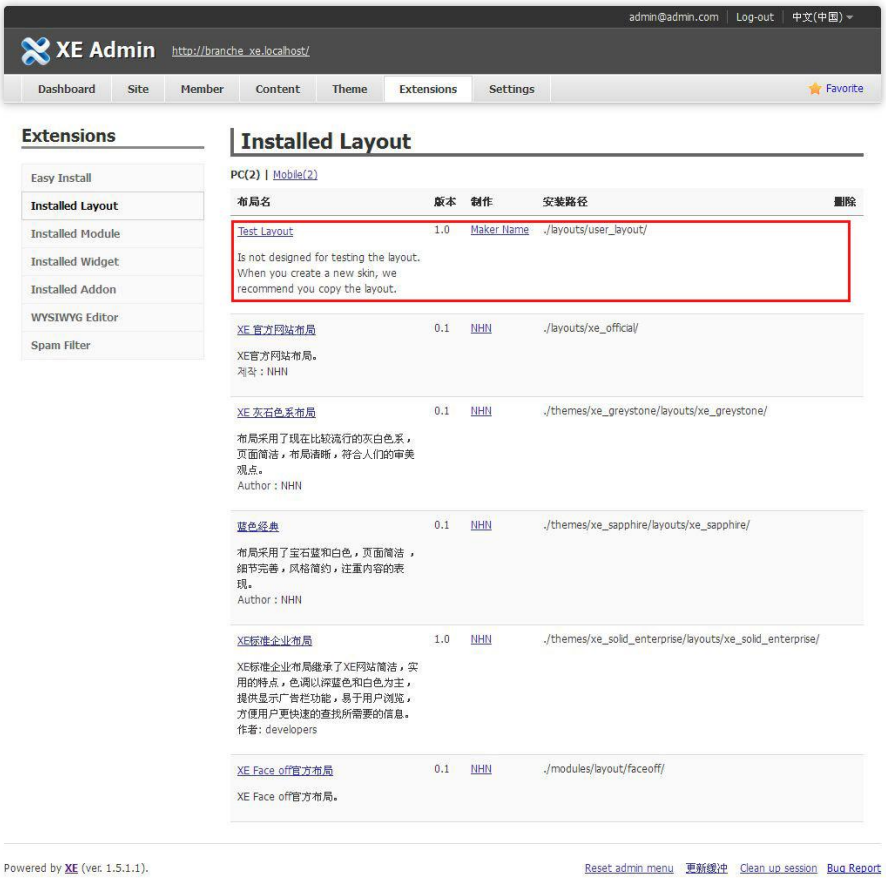
info.xml里新添加的扩展变量在生成布局后，显示在布局设定页面。网站管理员给扩展变量赋值后可以在皮肤中显示出来。

3.5 生成布局

确认用户自定义布局

确认是否正确的编写了info.xml的方法如下：

1. 在XE 管理者页面打开 **扩展功能> 已安装的布局。**
2. 确认**测试用布局使用**是否正常显示。



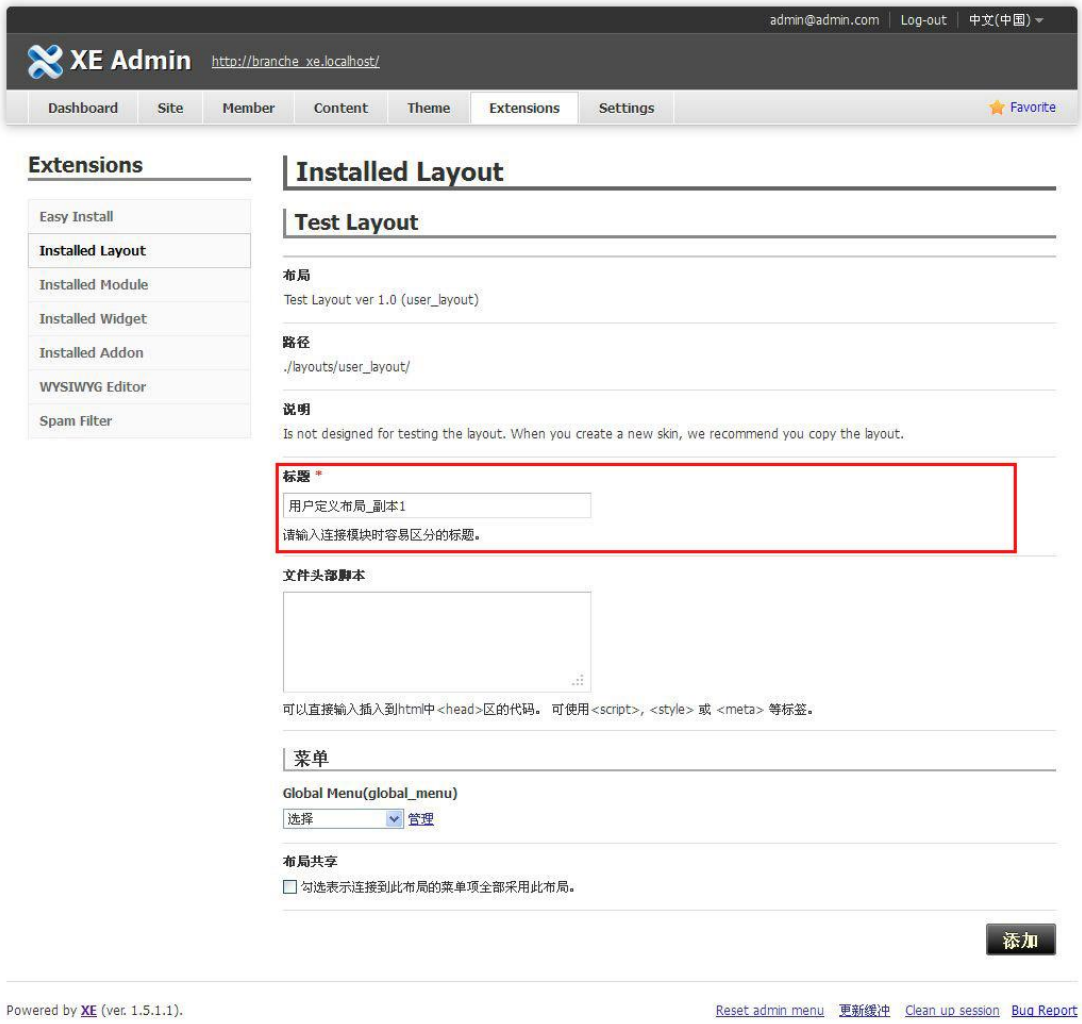
因为info.xml只包含一个 <menu> 元素，因此**菜单(Menu)数量**显示为1. **菜单(Menu)数量**是对应<menu> 的数量显示的。

生成布局副本

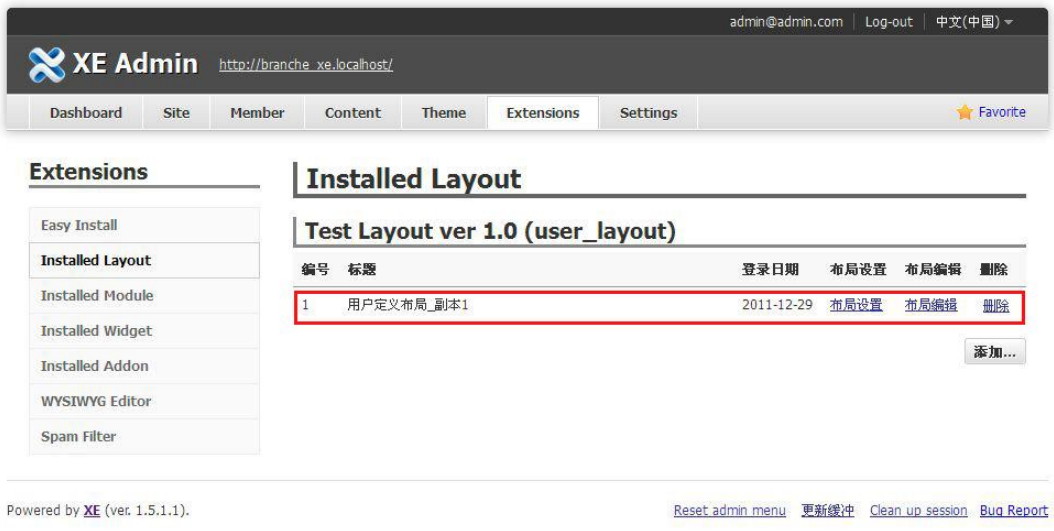
在**已安装的布局** 中的'测试用布局'选择生成布局的 '副本'后才可以使用它。用户可以重复生成多个'user_layout'的副本。

生成布局副本的方法如下：

1. 打开" 测试用布局" 页面后点击" **添加**..." 。
2. 在" **标题**" 栏填写副本名后，点击" **添加**" 按 钮。填写**标题名**应方便与其他副本区别开来。在本次实例中填入了" 用户定义布局_副本1" 。



3. “**测试用布局**”页面可以看到”**用户定义布局_副本1**”已成功生成。



现在可以在需要用到该布局的模块设置页面中选择使用”**用户定义布局_副本1**”了。

参考

在没有生成 layout.html 的状态下，试图打开使用该布局的模块页面会出现 "Err : "./layouts/user_layout/layout.html" template file does not exist." 错误信息。

3.6 制作布局皮肤

布局皮肤的内容在 layout.html 文件编辑.

XE是自动处理 'DOCTYPE, html, head, body' 元素的。因此皮肤制作者不必在皮肤文件使用这些元素。如果在皮肤文件中包含了上述的几个元素，则会被认为是HTML错误或者发生布局显示错位的现象。

适用了布局皮肤的页面节本结构如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html lang=".." xml:lang=".." xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <!—这一部分内容是在layout.html文档或在管理员后台编辑的。 -->
</head>
<body>
    <!—在这里显示 layout.html 文档的代码与内容。 -->
</body>
</html>
```

'layout.html' 页面中编辑的HTML代码与内容都会显示为<body>元素内部的HTML代码。
因此，皮肤制作者必须使用<body>内部有效的HTML语法来编辑layout.html文件。<head>元素内部包含的内容可以使用XE模板语法在<body>元素内部编辑。

布局皮肤的文档结构

测试用布局皮肤中的文件由 header, gnb(global navigation bar), body, lnb(local navigation bar), content, footer 来构成。因此layout.html的文档结构如下：

```
<div class="user_layout">
    <div class="header">
        <h1>Site Logo</h1>
        <hr />
        <div class="gnb">.gnb</div>
    </div>
    <hr />
    <div class="body">
        <div class="lnb">.lnb</div>
        <hr />
        <div class="content">.content</div>
    </div>
    <hr />
    <div class="footer">.footer</div>
</div>
```

由HTML编写的 元素之所以有Class 名的原因是因为需要在CSS文件里选择该HTML元素并使之具有符合要求的布局结构。

使用{$content} 变量来输出正文内容

layout.html现在还不能在用户界面得到确认，因为当前没有模块选择使用该布局。布局皮肤的页面是无法独立存在的，只有被其他模块选择使用时才可以在该模块页面看到布局。因此布局皮肤是“依赖”于模块的。

要想在用户界面看到测试用布局皮肤，须生成一个特定模块(页面、版面、维基等)并选择使用一个布局皮肤。皮肤制作者须事先编写‘内容变量’来显示连接到模块的布局皮肤。

测试用布局皮肤中是在布局的正文区域使用{$content}变量来编写‘内容变量’的。

```
<div class="user_layout">
    <div class="header">
        <h1>Site Logo</h1>
        <hr />
        <div class="gnb">.gnb</div>
    </div>
    <hr />
    <div class="body">
        <div class="lnb">.lnb</div>
        <hr />
        <div class="content">
            .content{$content}
        </div>
    </div>
    <hr />
    <div class="footer">.footer</div>
</div>
```

{$content}是布局皮肤中使用的最重要的变量，它使得当前布局皮肤连接到任何模块都可以在正文区域显示该模块的内容部分。呈现为静态的页面还是动态的版面、维基页面取决于连接模块的种类。

XE中模块与布局的相对关系并不是‘布局皮肤’承载各种‘模块’的关系，而是各种‘模块’连接不同‘布局皮肤’的关系。XE生成的模块页面拥有自己的固有链接URL，但是布局皮肤并不拥有自己的URL. 只有在连接到‘模块’的情况下才能显示当前页面的链接。

输出全局菜单

可以在 .gnb 元素内部输出已连接到布局的全局菜单。在XE管理员后台连接布局与菜单后，布局始终会显示已连接的菜单。为了显示这些菜单，布局皮肤必须事先编写好相应代码。

测试用布局皮肤中的代码如下，这段代码会把一个菜单显示到页面里，菜单深度为二级菜单。

```
<div class="gnb">
    .gnb
    <ul>
        <li loop="$global_menu->list=>$key1,$val1" class="active"|cond="$val1['selected']"><a
href="{$val1['href']}" target="_blank"|cond="$val1['open_window']=='Y'">{$val1['link']}</a>
```

```
            <ul cond="$val1['list']">
                <li loop="$val1['list']=>$key2,$val2" class="active"|cond="$val2['selected']"><a
href="{$val2['href']}" target="_blank"|cond="$val2['open_window']=='Y'">{$val2['link']}</a></li>
            </ul>
        </li>
    </ul>
</div>
```

上述代码使用到的模板语法和变量如下：

模板语法/变量	说明	种类
loop="$global_menu->list=>$key1,$val1"	当有连接的菜单时显示	循环句
cond="$val1['list']"	当连接的菜单有子菜单时显示。	条件句
loop="$val1['list']=>$key2,$val2"	当连接的菜单有子菜单时显示	循环句
class="active"|cond="$val1['selected'] class="active"\|cond="$val2['selected']"	连接的菜单或子菜单包含当前页，给 元素添加 class="active" 属性。	条件句
target="_blank"|cond="$val1['open_window']=='Y'" target="_blank"\|cond="$val2['open_window']=='Y'"	连接的菜单有新窗口属性，则显示 target="_blank"	条件句
{$val1['href']} {$val2['href']}	菜单链接 URL	变量
{$val1['link']} {$val2['link']}	菜单链接文本	变量

输出局部菜单

可以在 .gnb 元素内部输出已连接到布局的局部菜单。显示局部菜单的语法 与显示全局菜单的方法基本相似。 全局菜单与局部菜单的不同点就在于，局部菜单并不显示以及菜单。全局菜单已在 .gnb 区域输出了一、二级菜单，因此局部菜单只显示以当前页为基准的二、三级菜单。

本地菜单只会在有连接的页面时显示，因此生成菜单时必须输入实际可以链接到的URL.

下面的一段代码是测试用布局皮肤中的显示局部菜单输出代码。

```
<div class="lnb">
    .lnb
    <h2 loop="$global_menu->list=>$key1,$val1" cond="$val1['selected']"><a href="{$val1['href']}"
target="_blank"|cond="$val1['open_window']=='Y'">{$val1['link']}</a></h2>
    <ul loop="$global_menu->list=>$key1,$val1" cond="$val1['selected'] && $val1['list']">
        <li loop="$val1['list']=>$key2,$val2" class="active"|cond="$val2['selected']"><a href="{$val2['href']}"
target="_blank"|cond="$val2['open_window']=='Y'">{$val2['link']}</a>
            <ul cond="$val2['list']">
                <li loop="$val2['list']=>$key3,$val3" class="active"|cond="$val3['selected']"><a
href="{$val3['href']}" target="_blank"|cond="$val3['open_window']=='Y'">{$val3['link']}</a>
```

```
                </li>
            </ul>
        </li>
    </ul>
</div>
```

上述代码使用到的模板语法和变量如下：

模板语言/变量	说明	种类
loop="$global_menu->list=>$key1,$val1"	当有连接的菜单时显示	循环句
cond="$val1['selected']"	连接的菜单与当前页一致时显示。	条件句
cond="$val1['selected'] && $val1['list']"	连接的菜单与当前页一致，并且包含子菜单时显示。	
loop="$val1['list']=>$key2,$val2" loop="$val2['list']=>$key3,$val3"	连接的菜单包含子菜单时显示。	循环句
cond="$val2['list']"	连接的菜单包含子菜单时显示。	条件句
class="active"|cond="$val2['selected']" class="active"\|cond="$val3['selected']"	连接的菜单或子菜单包含当前页，给 元素添加 class="active" 属性。	条件句
target="_blank"|cond="$val1['open_window']=='Y'" target="_blank"\|cond="$val2['open_window']=='Y'" target="_blank"|cond="$val3['open_window']=='Y'"	连接的菜单有新窗口属性，则显示 target="_blank"	条件句
{$val1['href']} {$val2['href']} {$val3['href']}	菜单链接 URL	变量
{$val1['link']} {$val2['link']} {$val3['link']}	菜单链接文本	变量

输出综合搜索

网站的综合搜索是指不以特定模块作为搜索对象，而可以把已生成的全部模块作为搜索对象搜索的功能。如果不提供综合搜索样式，则不能使用综合搜索功能。综合搜索样式通常由布局皮肤提供，其样式的代码由 **扩张功能 > 已安装的模块 > 综合搜索**页面提供。

测试用布局皮肤中，在 ... 元素内部添加了综合搜索样式代码。

```
<div class="user_layout">
    <div class="header">
        <h1>Site Logo</h1>
        <form action="{getUrl()}" method="get" class="search">
            <input type="hidden" name="vid" value="{$vid}" />
            <input type="hidden" name="mid" value="{$mid}" />
            <input type="hidden" name="act" value="IS" />
```

```
            <input type="text" name="is_keyword" value="{$is_keyword}" title="{$lang->cmd_search}" class="iText" />
            <input type="submit" value="{$lang->cmd_search}" class="btn" />
        </form>
        <hr />
        <div class="gnb">
            .gnb
            ...
        </div>
    </div>
    <hr />
    <div class="body">
        <div class="lnb">
            .lnb
            ...
        </div>
        <hr />
        <div class="content">{$content}</div>
    </div>
    <hr />
    <div class="footer">.footer</div>
</div>
```

上述代码中使用到的变量如下：

变量	说明
{getUrl()}	处理用户提交的搜索关键词的页面 URL
{$vid}	虚拟站点 ID
{$mid}	模块 ID
{$is_keyword}	在搜索结果页面中再度显示用户提交的关键词。
{$lang->cmd_search}	语言变量。当前分配了 '搜索' 一词给该变量。

输出会员登录样式

如果网站需要以会员制运营，那么需要提供会员注册和会员登录功能。XE core已提供有会员注册与会员登录页面样式。将登录页面样式嵌入到布局皮肤，则可以在任何位置显示登录页面。

测试用布局皮肤中，在 ... 元素内部区域了会员登录控件(Widget)代码。

```
...
<div class="lnb">
    .lnb
    <div class="account">
        <img widget="login_info" skin="xe_official" />
    </div>
    <h2>...</h2>
```

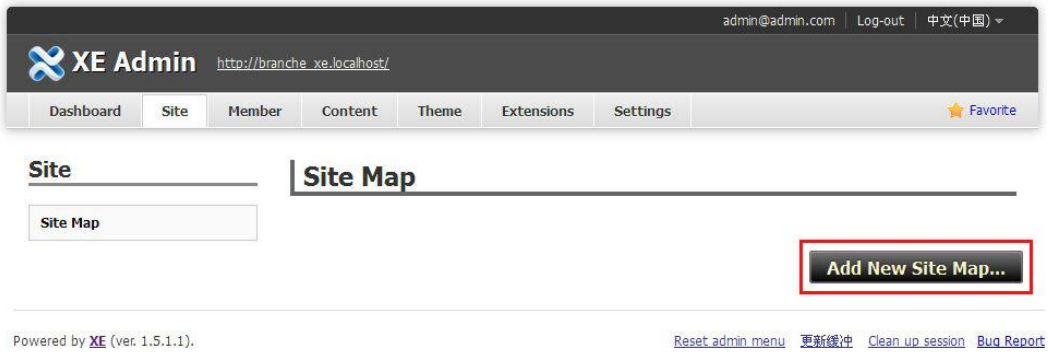
```
    <ul>...</ul>
</div>
...
```

3.7 生成站点地图

站点地图是指站点菜单。在XE管理员后台生成站点地图并连接到布局副本后，可以在输出布局时一并输出该站点地图。

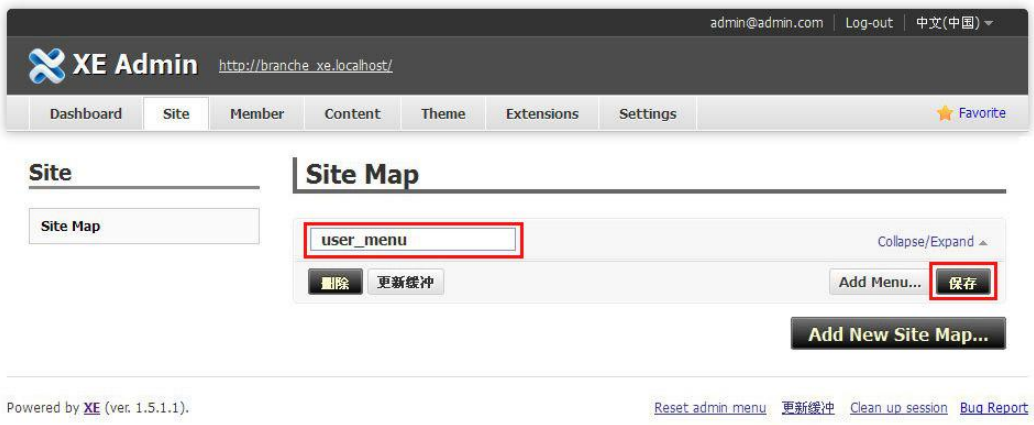
生成站点地图的方法如下。

1. 在XE 管理员后台选择 **站点 > 站点地图。**
2. 在站点地图页面中点击**添加新的站点地图。**

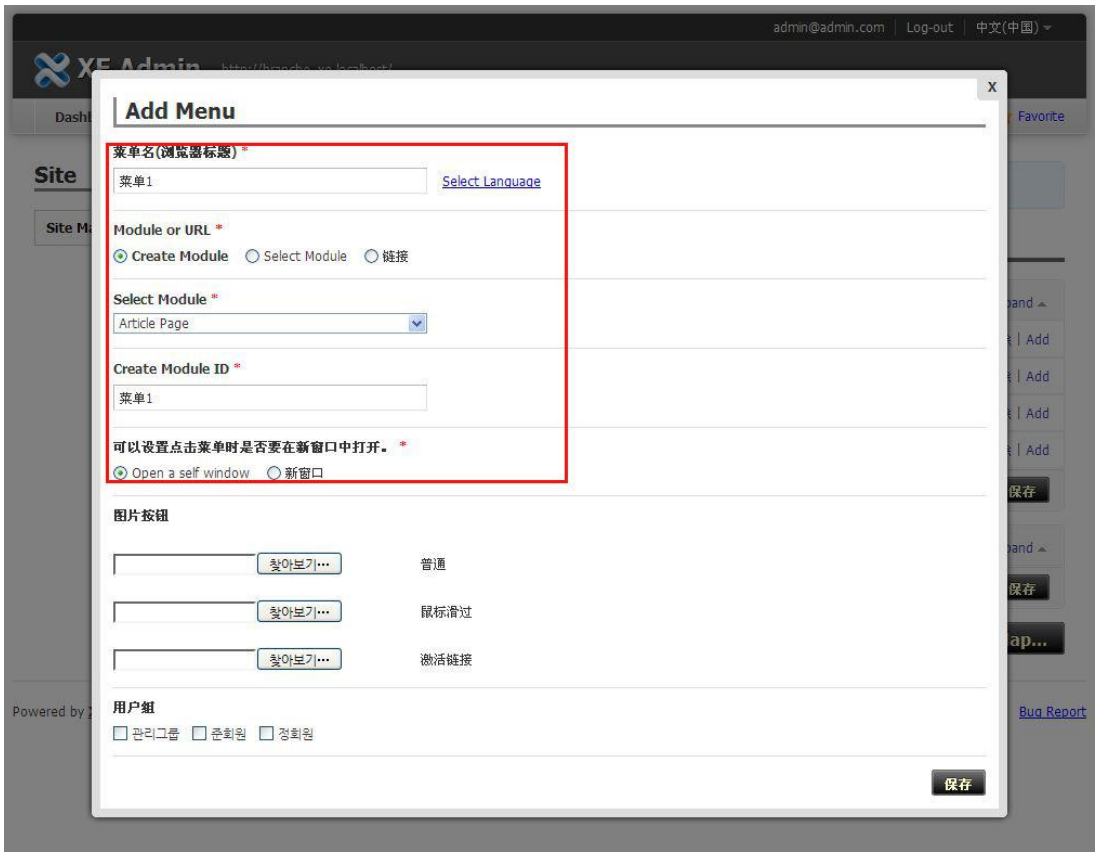


可以生成多套站点地图，因此当连接到布局时应提供与其他站点题图易于区分的名称。

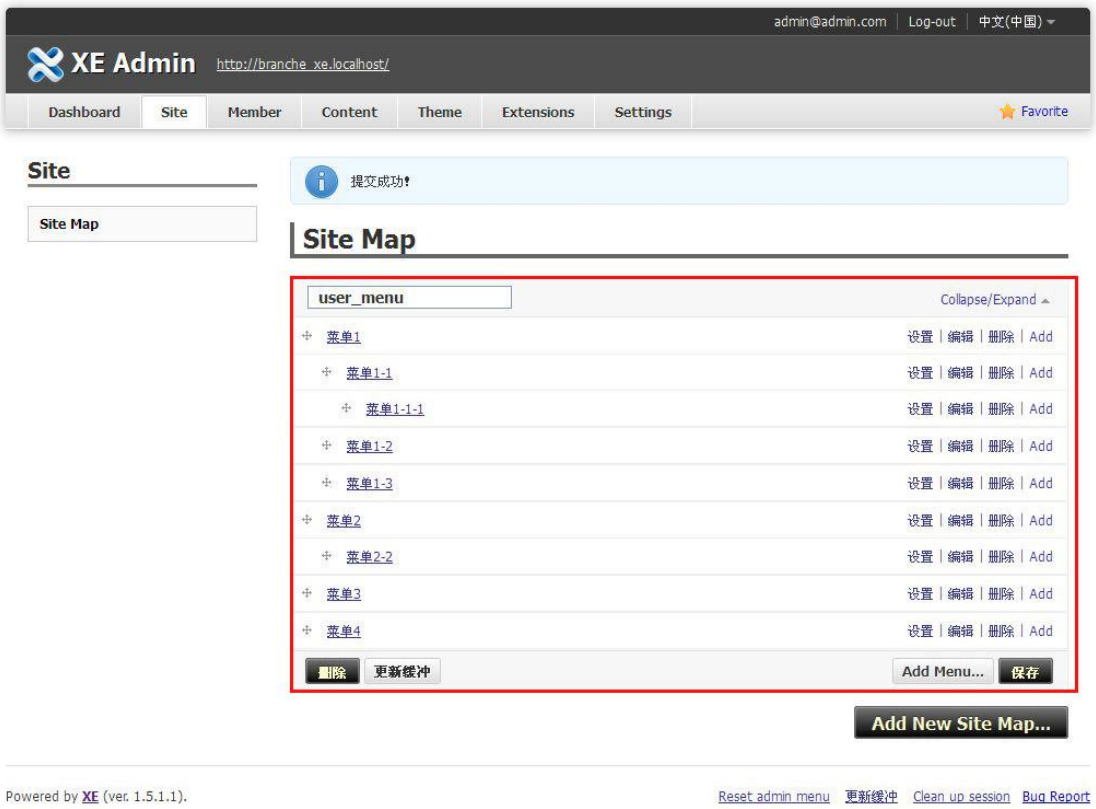
3. 在**标题**栏填写例如 *user_menu*后，点击**保存。**



4. **user_menu** 页面中点击**添加新菜单项**，完成站点的全局菜单。 更详细的生成菜单方法请参照 "XE 用户指南"。



5. 在生成了名为user_menu的站点地图后，点击**保存。**



现在还不可以在用户界面中看到 'user_menu'菜单。当布局调用了该菜单后才会显示在用户界面。

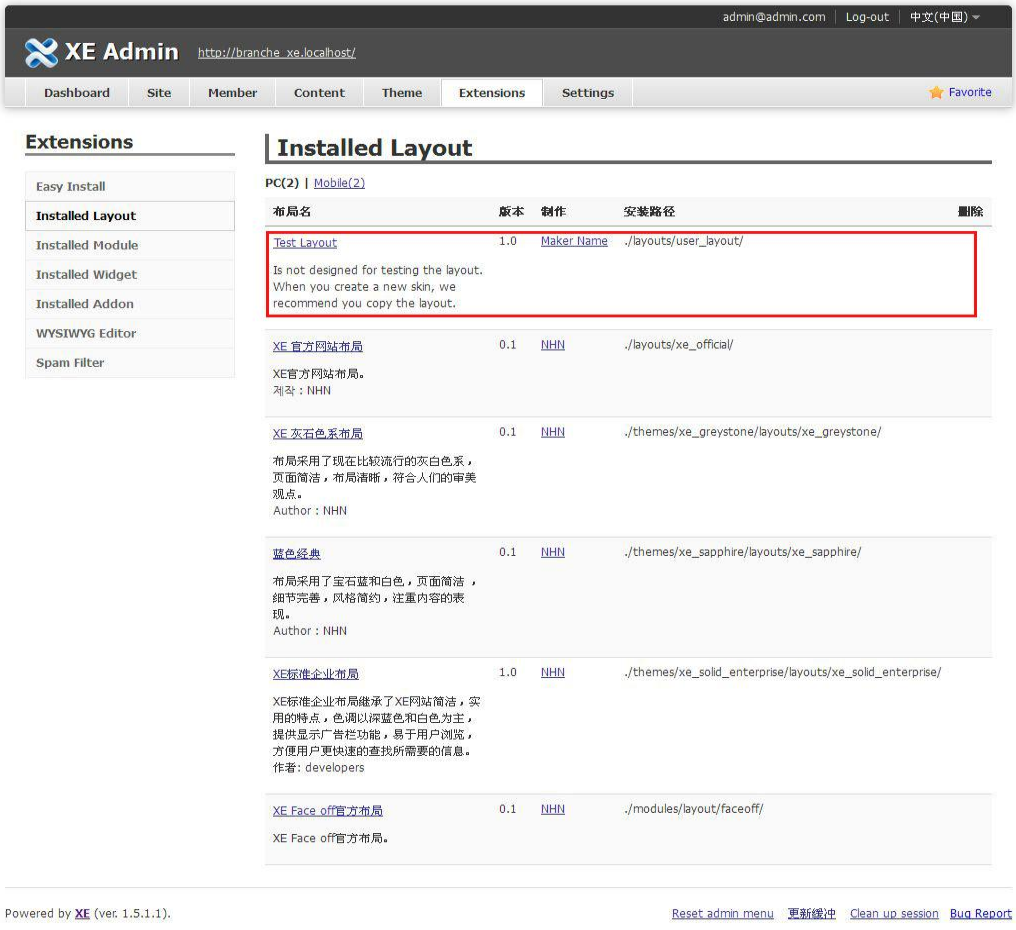
参考

在生成菜单的步骤中必须在**模块或URL**栏填写有效的模块ID或URL链接。如果填写了无效的模块ID或URL链接，则有可能在布局中非正常显示该菜单。.

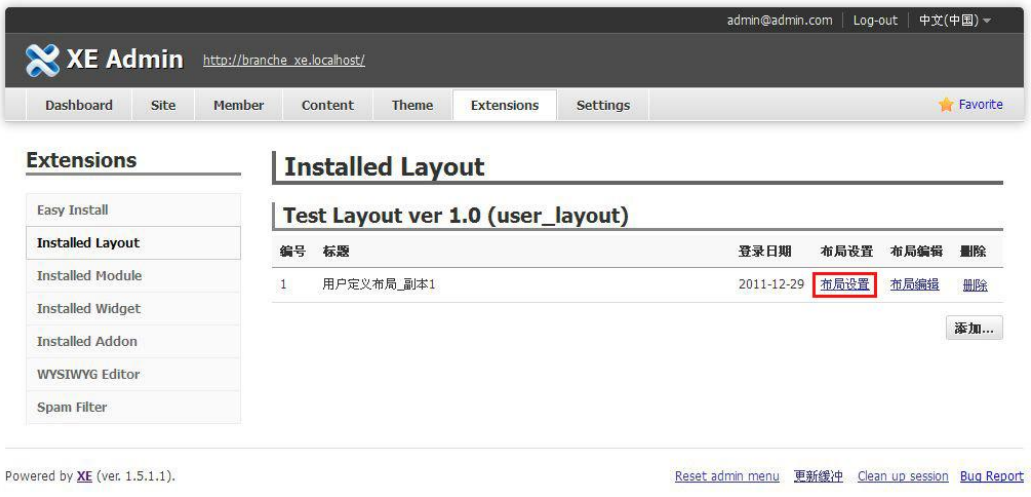
3.8 布局与站点地图的连接

生成完名为'user_menu'的站点地图后，现在布局可以调用该站点地图。给布局连接该'user_menu'站点地图的方法如下：

1. 在XE管理员后台选择 **扩展功能 > 已安装的布局。**



2. 打开 **测试用布局**，点击**用户定义布局_副本1**的**设置**按钮。



3. 在**菜单 > 全局菜单(global_menu)**中选择 **user_menu**量，并给布局**布局共享**打勾，点击**保存。**

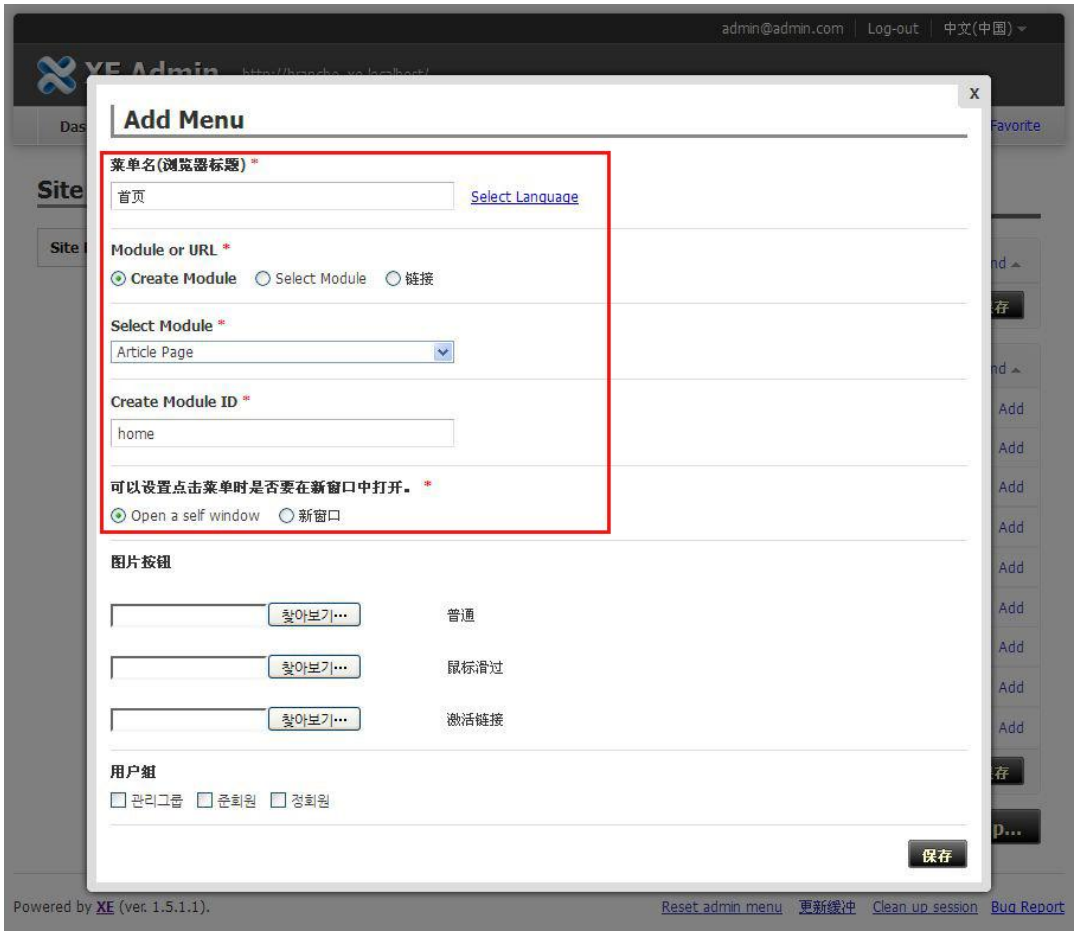
   选择**布局共享**后，可以将连接到该菜单的所有模块适用当前布局。此后 **user_menu**菜单将保持与该布局同时输出的状态。

3.9 页面模块与布局的连接

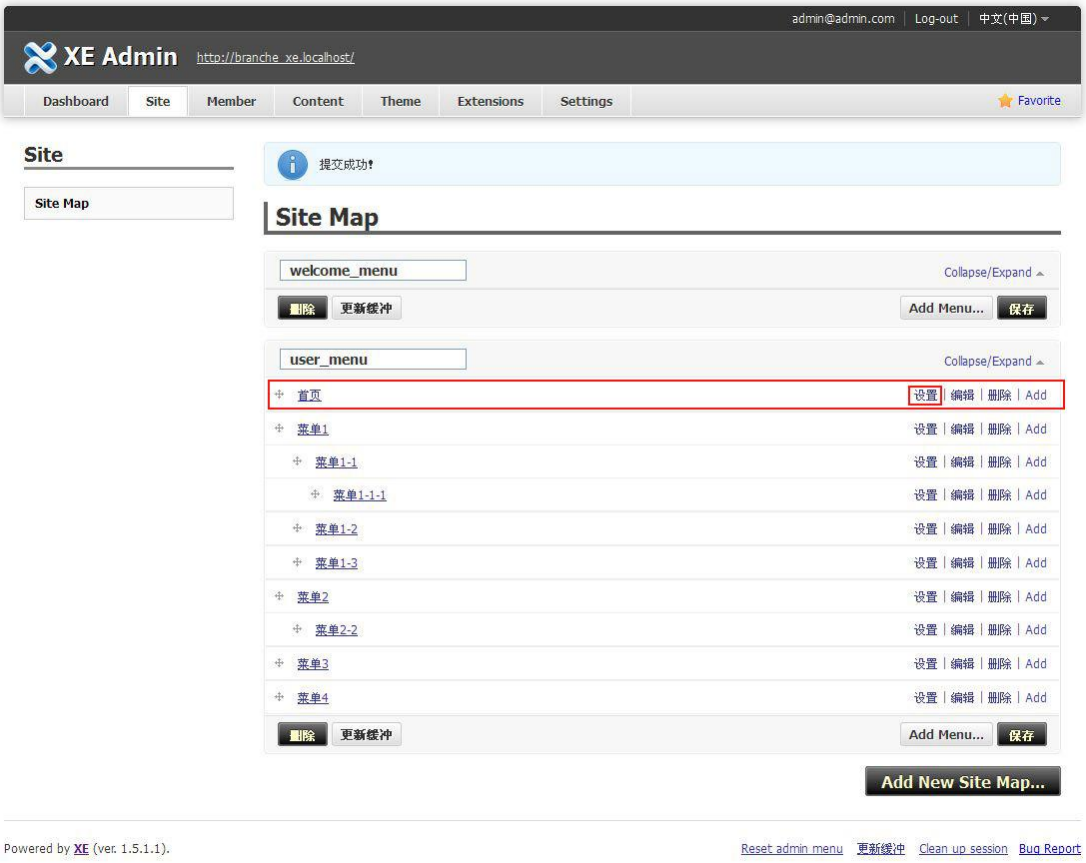
接下来，将会给页面模块套用测试用布局皮肤，并确认测试用布局皮肤是否可以正确地显示。

页面生成

1. 在XE管理员后台选择 **站点 > 站点地图。**
2. 点击**user_menu** 下方的**添加新菜单项。**
3. 如下图，弹出**添加新菜单项**页面后，依次填写或选择**菜单名称(浏览器标题栏)**、**模块或URL链接**、**生成模块ID名**、**在当前页打开。**在本实践中设置为下列值，点击**保存。**
   - **菜单名称：** *首页*
   - **选择模块：**：*文档页面*
   - **生成模块ID名**: *home*



4. 在**站点地图 > user_menu** 确认成功生成‘首页’后，可以点击‘首页’的**设置**，进入‘**页面管理**’页面。



5. 在**页面管理**中的‘**布局**’一栏选择 **用户定义布局_副本1 (user_layout)**后，点击‘保存’。

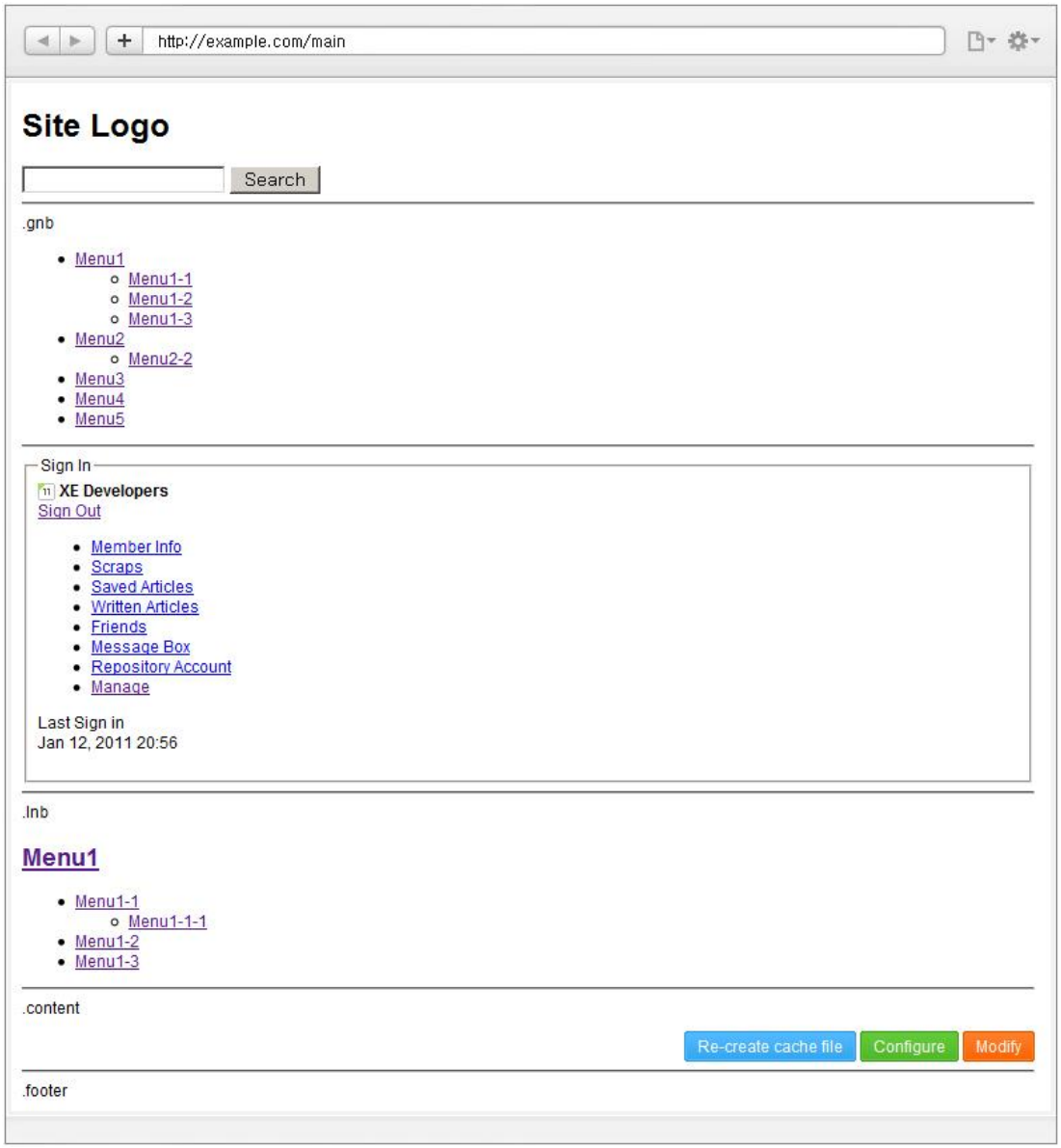
如果在‘**布局**’栏中找不到 **用户定义布局_副本1 (user_layout)** 项，则意味着还未生成'user_layout'的副本。 关于生成 'user_layout' 副本的方法请参考"生成布局副本"。

页面确认

在用户界面打开刚刚生成的页面可以确认适用了布局皮肤的效果。因为模块名为 'home'，因此该页面的地址如下列所示。下列地址中 'example.com'是指用户设置站点时使用的域名。

- 使用mode_rewrite时: http://example.com/home/
- 不使用mode_rewrite时: http://example.com/?mid=home

打开home 页面后，效果如下：



图示 3-3 应用布局皮肤的页面

因为还没有填写页面正文，因此无内容可显示，当前只显示正文以外的布局部分。'Site Logo, gnb, lnb, content, footer'等这些文本是编写在皮肤文件 'layout.html' 里的。除此之外，可以看到包含在布局皮肤里的综合搜索样式、全局菜单、会员登录控件、局部菜单等部分可以正常显示。在 content 区域有一个编辑页面正文的按钮。这个按钮只有管理员才可以看到。如果没有看到该按钮，那有可能当前没有以管理员身份登录，或者没有编写{$content}变量。

> **参考**
>
> 生成菜单的过程中，在**模块或URL**一栏填写了无效的**模块ID**或**链接URL**，则有可能无法正常显示局部菜单。

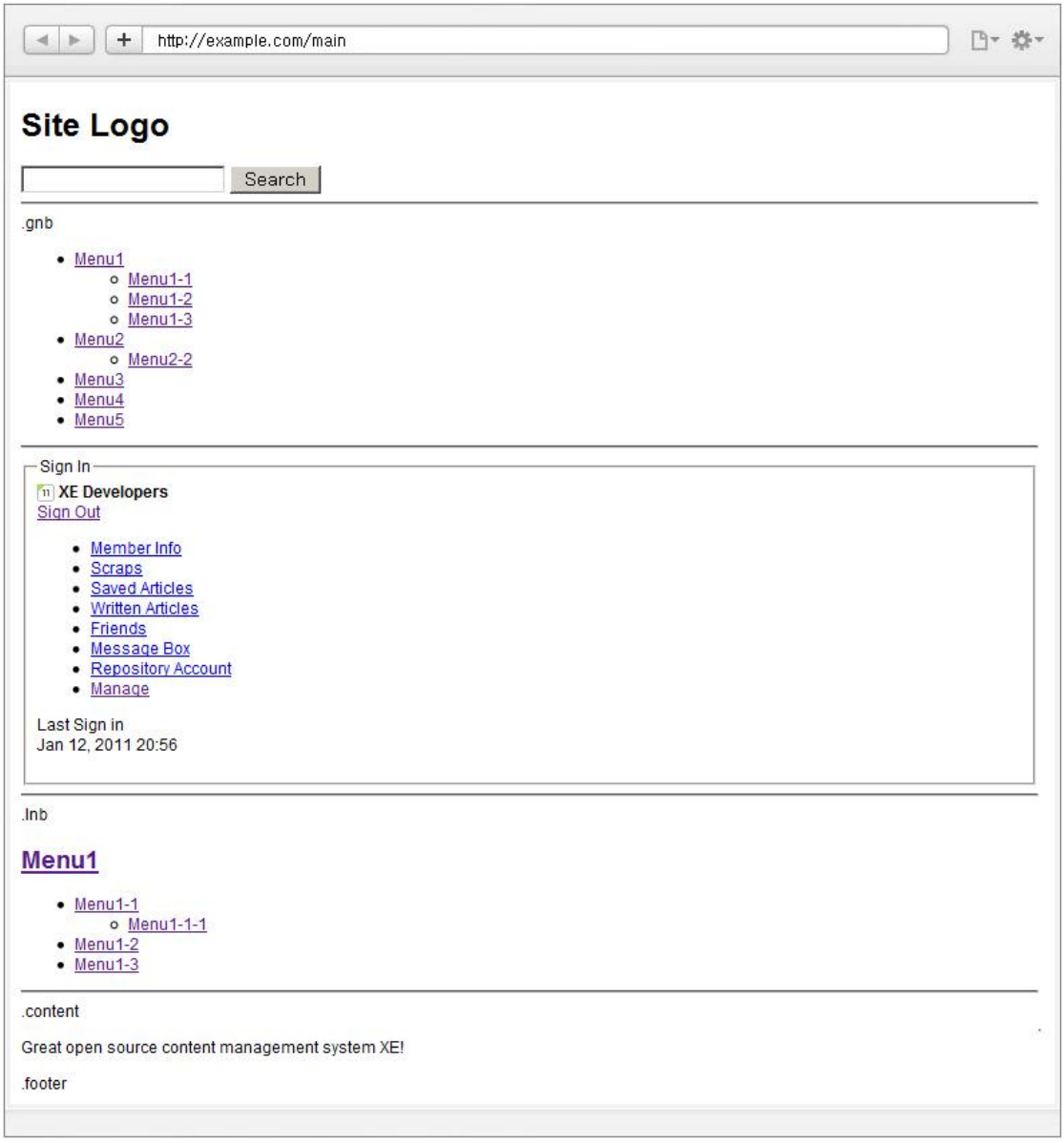
页面修改

如图示 3-3 所示，可以给页面提交新的内容。修改页面的方法如下：

1. 在已经适用布局皮肤的页面中点击位于右下角的**页面修改**按钮。
2. 添加一些简单的标题与正文后，点击**提交**。

添加的内容会在页面中得到反映。

3. 在用户界面最终显示的效果如下图所示。位于 content 区域右下角的‘**设置**’、‘**页面修改**’按钮只会显示给有权限编辑该页面模块的管理员，而用户的页面中并不会显示。



目前页面的状态还不可以称之为布局，应使用CSS代码来具体实现各种页面布置。

3.10 应用 CSS

这里将对给布局皮肤适用CSS代码的方法进行说明，但并不会说明CSS代码的编写方法。皮肤制作者可以按照自己的意愿来修改并使用CSS代码。

1. 测试用布局皮肤中的 user_layout.css 文件里将 .lnb区域与 .content 领域用分栏的形式左右分开。

```
@charset "utf-8";
/* Layout */
hr{ display:none;}
form, fieldset{ border:0; margin:0; padding:0;}
.user_layout{ width:960px; margin:0 auto;}
.header{ zoom:1; background:#ddd;}
.header:after{ content:""; display:block; clear:both;}
.header .search{ float:right;}
.gnb{ float:left;}
.body{ margin:20px 0; zoom:1; background:#eee;}
.body:after{ content:""; display:block; clear:both;}
.lnb{ float:left; width:200px; background:#ddd;}
.account{}
.content{ float:right; width:740px; background:#ddd;}
.footer{ background:#ddd;}
```

2. layout.html 里按下例代码调用 user_layout.css文件。

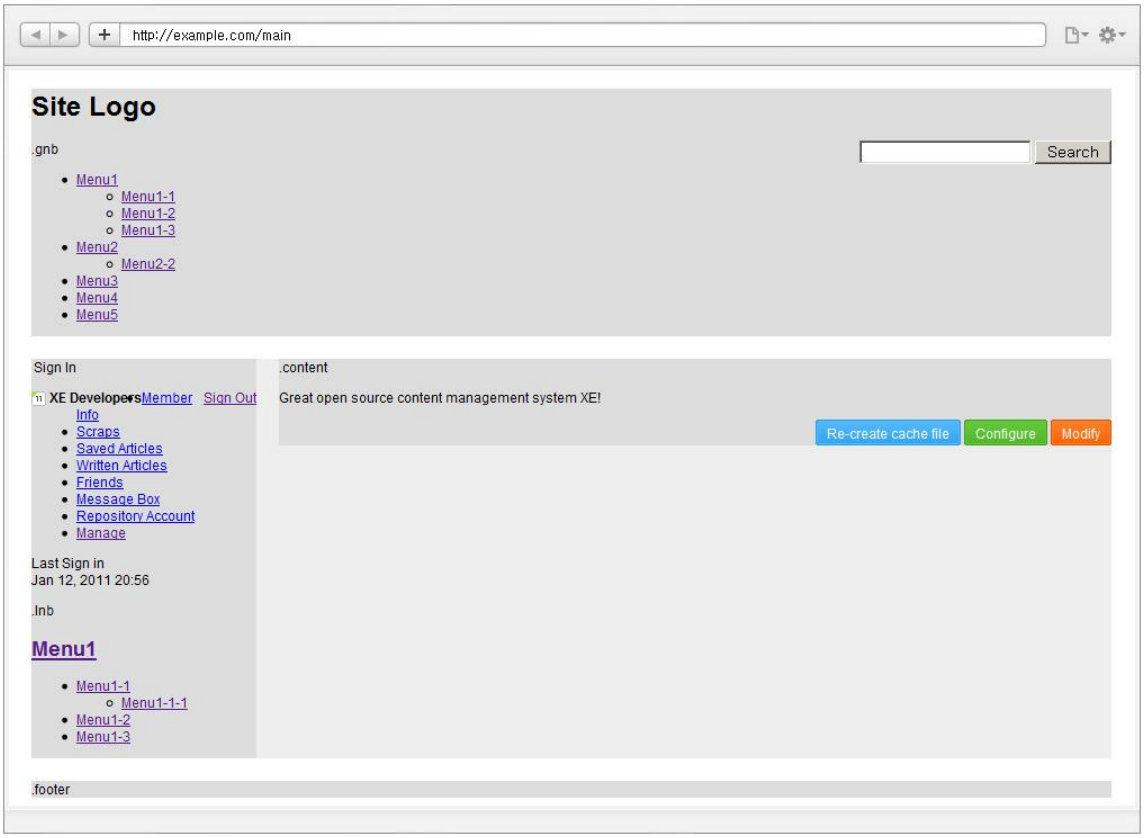
```
<load target="user_layout.css" />
```

详细说明请参考"调用CSS 文件"。

3. 请从以下地址查看页面是否正确调用了CSS文件。 下列地址中 'example.com'是指用户设置站点时使用的域名。

- 使用mode_rewrite时: http://example.com/home/
- 不使用mode_rewrite时: http://example.com/?mid=home

下图是正常使用 user_layout.css 配置画面布局时的效果图。



图示 3-4 正确应用CSS的页面

如果画面没有反映编写的代码，请确认调用CSS文件的<load />语法或CSS文件路径是否正确。

3.11 应用 Javascript

这里将阐述将Javascript代码添加到布局皮肤的方法，但并不会说明jQuery代码的编写方法。皮肤制作者可以按照自己的意图来编写代码。

1. 在user_layout.js文件中编写 jQuery 代码。

```
// <![CDATA[
jQuery(function($){
    // 在这里编写jQuery 代码。
});
// ]]>
```

2. layout.html中调用 user_layout.js 文件，代码如下：

```
<load target="user_layout.js" type="body" />
```

详细说明请参考"通过使用 index 属性可以改变CSS文件调用顺序。<load /> 与index属性可以在XE core 1.4.4 以上版本使用。

index属性值可以为正•负整数。使用负整数可以优先调用，而使用正整数可以将载入优先度滞后。指定 index="-1" 时，在其他CSS文件的上一行载入。

当CSS 中出现属性值冲突时，只有最后定义的值才会生效。因此推荐将优先度高的文件放到最后去调用。

"。

3. 请从以下地址确认Javascript代码是否正常地执行。下列地址中 'example.com' 是指用户设置站点时使用的域名。
   - 使用mod_rewrite时: http://example.com/home/
   - 不使用mod_rewrite时: http://example.com/?mid=home

如果画面没有反映编写的代码，请确认调用user_layout.js文件的<load />语法或JS文件路径(target)是否正确。

参考

在XE使用jQuery的基本方法请参考 "使用Javascript与 jQuery"。
要想通过使用jQuery来实现多种效果，请访问 http://jquery.com/

4. 制作版面(board)皮肤

这一章主要介绍如何使用版面皮肤样本来制作版面皮肤并使用。

4.1 什么是版面(board)皮肤

版面皮肤作为一种界面负责把版面模块显示在用户页面。版面皮肤以用户端作为参照主要由列表、阅读、撰写、删除、写评论、删除评论、删除引用、权限提示、密码输入等页面构成。

4.2 安装版面(board)模块

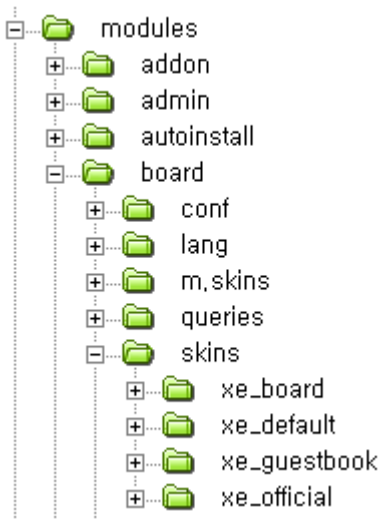
要想在XE制作版面就必须另行安装版面模块。版面模块可以通过 XE 管理员后台的 '安装·更新' 功能或直接将版面模块的代码文件上传到服务器上来安装。

安装□更新

在 XE 管理员后台页面中选择 **扩展功能 > 安装•更新 > 模块** 来把版面模块安装到服务器中。

通过**安装•更新**功能来安装版面模块后，需要制作版面皮肤，就首先要用FTP定位到版面模块安装路径把该部分下载到本地电脑上。

版面模块的路径结构如下。请确认模块路径是否包含 'skins' 文件夹，版面皮肤就保存在 'skins' 文件夹里。



图示 4-1 版面模块的路径与结构

上传代码文件

从XE 官方网站下载版面模块的代码文件到本地电脑之后，就需要通过FTP软件把这些文件上传到服务器的 '/modules/' 文件夹里。如果XE是安装在用户自定义的/xe/路径里，那么版面模块的安装路径将是 '/xe/modules/'. 如果上传过程正常，那么版面模块的位置应该是'/modules/board/'或 '/xe/modules/board/'

4.3 下载版面皮肤样本

要制作版面皮肤就要生成一系列版面皮肤必需的几个文件。本文档为方便皮肤制作者提供了包含必要文件在内的版面皮肤样本。制作皮肤时，可以很方便地改造样本皮肤来制作自己需要的样式

版面皮肤样本的下载路径为：

- http://doc.xpressengine.com/manual/user_board.zip

4.4 版面皮肤的位置与必要文件

4.4.1 确认版面皮肤的位置

版面皮肤的路径位置如下：

```
/modules/board/skins/
```

将下载的版面皮肤样本解压缩，并复制到 '/modules/board/skins/' 路径下。

```
/modules/board/skins/user_board
```

> **参考**
>
> 在本地电脑修改版面皮肤后，需要把这些内容上传到站点的版面皮肤路径上。.

4.4.2 版面皮肤必要文件

下列文件是制作版面皮肤所必需的。

表 4-1 版面皮肤必要文件

文件	说明	备注
skin.xml	版面皮肤信息	不可变更文件名
list.html	文章列表	不可变更文件名
write_form.html	撰写文章页面	不可变更文件名
delete_form.html	删除文章页面	不可变更文件名
comment_form.html	撰写评论页面	不可变更文件名
delete_comment_form.html	删除评论页面	不可变更文件名
delete_trackback_form.html	删除引用页面	不可变更文件名
input_password_form.html	输入密码页面	不可变更文件名
message.html	提示页面	不可变更文件名
_header.html	显示版面头部	被引用(include)于其他文件
_footer.html	显示版面尾部	被引用(include)于其他文件
_read.html	显示文章阅读页面	被引用(include)于其他文件
_comment.html	显示评论页面	被引用(include)于其他文件
_trackback.html	显示引用页面	被引用(include)于其他文件
user_board.css	版面皮肤样式表	

4.5 编辑版面皮肤信息

skin.xml文件包含版面皮肤的基本信息并提供给管理员后台需要显示的说明与选项。skin.xml文件名不可变更。

skin.xml的基本结构如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<skin version="0.2">
    <title xml:lang="ko">user_board</title>
    <title xml:lang="zh-CN">user_board</title>
    <title xml:lang="en">user_board</title>
    <title xml:lang="jp">user_board</title>
    <description xml:lang="ko">게시판 스킨 제작 실습을 위한 user_board입니다.</description>
    <description xml:lang="zh-CN"> user_board 供实习制作版民皮肤用。</description>
    <description xml:lang="en">This is user_board for creating a board skin.</description>
    <description xml:lang="jp">掲示板スキン製作実習のためのuser_boardです。</description>
    <version>1.0</version>
    <date>2010-12-24</date>
    <author email_address="user@user.com" link="http://user-define.com/">
        <name xml:lang="ko">제작자 이름</name>
        <name xml:lang="zh-CN">制作者名</name>
        <name xml:lang="en">Author Name</name>
        <name xml:lang="jp">製作者名</name>
    </author>
    <license>LGPL v2</license>
    <extra_vars>
        <var name="title" type="text">
            <title xml:lang="ko">게시판 제목</title>
            <title xml:lang="zh-CN">版面标题</title>
            <title xml:lang="en">Board Title</title>
            <title xml:lang="jp">掲示板タイトル</title>
                    <description xml:lang="ko">작성하면 화면에 표시됨</description>
                    <description xml:lang="zh-CN">提交后显示在页面</description>
                    <description xml:lang="en">This will be displayed on the screen as you
write.</description>
                    <description xml:lang="jp">作成すると画面に表示される</description>
        </var>
        <var name="comment" type="textarea">
            <title xml:lang="ko">게시판 설명</title>
            <title xml:lang="zh-CN">版面说明</title>
            <title xml:lang="en">Board Details</title>
            <title xml:lang="jp">掲示板詳細説明</title>
                    <description xml:lang="ko">작성하면 화면에 표시됨</description>
                    <description xml:lang="zh-CN">提交后显示在页面</description>
                    <description xml:lang="en">This will be displayed on the screen as you
write.</description>
                    <description xml:lang="jp">作成すると画面に表示される</description>
```

```
        </var>
    </extra_vars>
</skin>
```

上述代码的内容如下：

代码	说明
<?xml version="1.0" encoding="UTF-8"?>	XML 文档格式声明
<skin version="0.2">	声明该文档为皮肤信息文档。version 里应表示 XE core 所支持的版本。支持以 XE core 1.4.4.2 为标准的版本是 0.2 版。
<title xml:lang="zh-CN">...</title>	版面皮肤名称
<description xml:lang="zh-CN">...</description>	版面皮肤说明
<version>...</version>	版面皮肤版本
<date>YYYY-MM-DD</date>	版面皮肤制作日期。以年-月-日(YYYY-MM-DD)形式来标识
<author email_address="..." link="..."> <name xml:lang="zh-CN">...</name> </author>	版面皮肤制作者信息。如电子邮件、个人主页、制作者名等。
<license>LGPL v2</license>	版面皮肤使用许可信息。推荐采用与 XE core 相同的 LGPL v2 版本。
<extra_vars>...</extra_vars>	如果需要在版面模块使用扩展变量，请在改元素内部定义扩展变量内容。
<var name="title" type="text"> <title xml:lang="zh-CN">版面标题</title> <description xml:lang="zh-CN"> 提交后显示在页面 </description> </var>	将提交的版面标题显示在页面上。
<var name="title" type="textarea"> <title xml:lang="zh-CN">版面说明</title> <description xml:lang="zh-CN"> 提交后显示在页面 </description> </var>	将提交的版面说明显示在页面上。

要想确认'user_board' 版面皮肤的 skin.xml 文档是否编写正确，须生成一个版面在选择使用该皮肤来确认。

除此之外，skin.xml还可以把用户自定义的各种类型变量包含在<extra_vars>元素内部。以下就是皮肤制作者把接收到的如 select, image, text, textarea等类型的数据扩展为可使用变量的实例。

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<layout version="0.2">
    ...
    <extra_vars>
        <var name="colorset" type="select">
            <title xml:lang="ko">컬러셋</title>
            <title xml:lang="zh-CN">色彩方案</title>
            <description xml:lang="ko">원하는 컬러셋을 선택해주세요.</description>
            <description xml:lang=" zh-CN ">请选择色彩方案</description>
            <options value="black">
                <title xml:lang="ko">Black (기본)</title>
                <title xml:lang=" zh-CN ">Black (默认)</title>
            </options>
            <options value="white">
                <title xml:lang="ko">White</title>
                <title xml:lang=" zh-CN ">Black</title>
            </options>
        </var>
        <var name="board_image" type="image">
            <title xml:lang="ko">版面头部图片 </title>
            <description xml:lang="ko">请上传版面头部显示的的图片。</description>
        </var>
    </extra_vars>
    ...
</layout>
```

上述代码所使用的扩展变量如下：

变量	说明
<var name="colorset" type="select">...</var>	select 类型扩展变量. 以{$module_info->colorset} 形式来使用变量。
<var name="board_image" type="image">...</var>	image 类型的扩展变量. 以{$modul_info->image} 形式来使用变量。

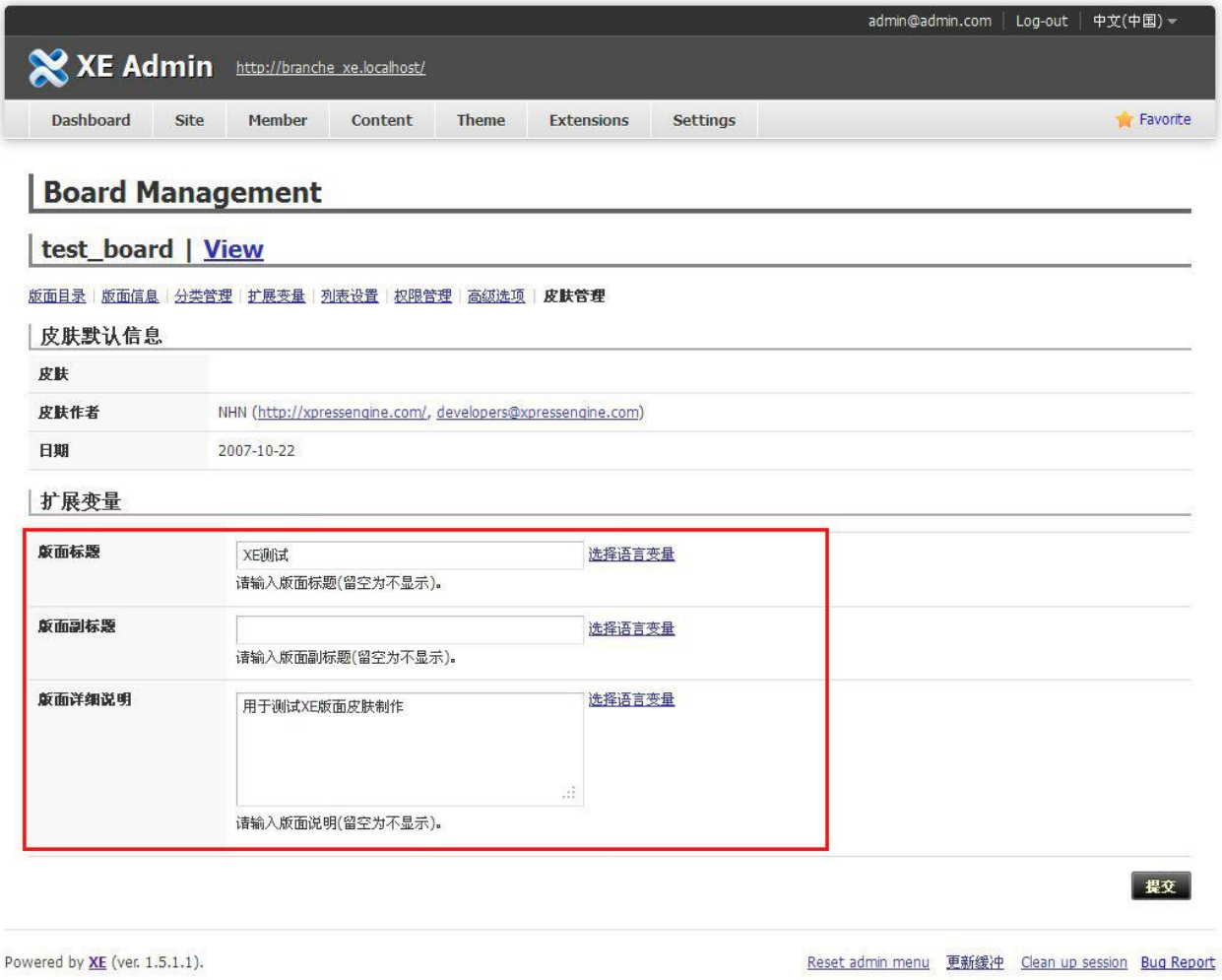
skin.xml里新添加的扩展变量显示在 XE 管理员后台的 **扩展功能 > 已安装的模块 > 版面 > 设置 > 皮肤管理** 页面。. 版面管理员给该变量赋值后可以在皮肤中显示出来。

4.6 新建版面并应用皮肤

新建版面并应用皮肤的方法如下：

1. 在 XE 管理员后台选择 **扩展功能 > 已安装的模块 > 版面** 。
2. 打开 **版面管理 > 版面列表**， 点击**新建。**.

3. 按下列给出的内容填写完后，点击**提交。**实际上，还有更多的选项供填写，但并属于必填项。因此在这里可以省略不填。
   - **模块名称**: *test_board*
   - **浏览器标题栏:** *测试用版面皮肤*
   - **皮肤: user_board**
4. 打开已生成的 test_board 版面的**皮肤管理**， 填写**版面标题**、**版面详细说明**后点击**提交。**
   - **版面标题**: *XE 测试*
   - **版面详细说明**: *用于测试XE版面皮肤制作。.*


admin@admin.com | Log-out | 中文(中国)
XE Admin
http://branche_xe.localhost/
Dashboard
Site
Member
Content
Theme
Extensions
Settings
Favorite
Board Management
test_board | View
版面目录 | 版面信息 | 分类管理 | 扩展变量 | 列表设置 | 权限管理 | 高级选项 | 皮肤管理
皮肤默认信息
皮肤
皮肤作者
NHN (http://xpressengine.com/, developers@xpressengine.com)
日期
2007-10-22
扩展变量
版面标题
XE测试
选择语言变量
请输入版面标题(留空为不显示)。
版面副标题
选择语言变量
请输入版面副标题(留空为不显示)。
版面详细说明
用于测试XE版面皮肤制作
选择语言变量
请输入版面说明(留空为不显示)。
提交
Powered by XE (ver. 1.5.1.1).
Reset admin menu
更新缓冲
Clean up session
Bug Report

4.7 制作版面的头部与尾部

4.7.1 制作头部

版面头部会将相同内容显示在版面模块所有页面的头部。测试用版面皮肤中为了始终显示已在管理页面中填写的**版面标题**、**版面详细说明**等内容和调用CSS文件，按下列代码编写了_header.html文件。这样，版面的所有页面调用_header.html文件即可。

```
<div class="user_board">
    <div class="board_header" cond="$module_info->title || $module_info->comment">
        <h2 cond="$module_info->title">
            <a href="{getUrl('','mid',$mid)}">{$module_info->title}</a>
        </h2>
        <p cond="$module_info->comment">{$module_info->comment}</p>
    </div>
```

请留意， 并没有关闭标签。这一元素的关闭标签会写在_footer.html文件里。 元素可以囊括版面里的所有元素，以方便管理CSS代码。

上述代码所使用的模板语法与变量如下：

模板语法/变量	**说明**
cond="$module_info->title || $module_info->comment"	如果包含**版面标题**或**版面详细说明**其中之一，就显示该内容。
cond="$module_info->title"	如果包含**版面标题，**显示该内容。
{$module_info->title}	显示**版面标题**。
cond="$module_info->comment"	如果包含**版面详细说明**，显示该内容。
{$module_info->comment}	**显示**版面详细说明。
{getUrl('','mid',$mid)}	版面 URL

从XE core 1.4.4 开始，采用了新的模板语法。关于新模板语法请参考 "XE 模板语言"。

4.7.2 制作尾部

版面尾部会将相同内容显示在版面模块所有页面的尾部。在测试用版面皮肤中只是关闭了在 _header.html里未关闭的 元素标签，并没有添加别的内容。

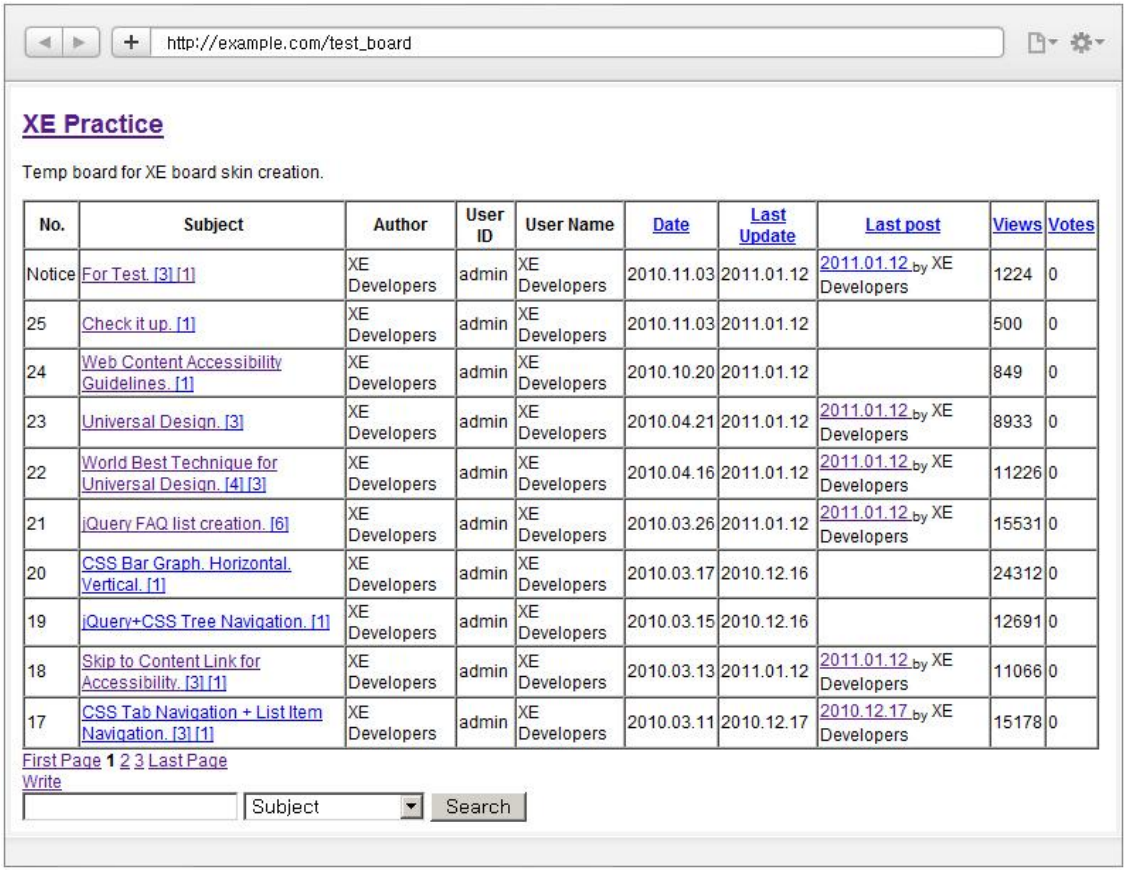
```
</div>
```

_footer.html将与 _header.html一起，在所有页面被调用。

4.8 制作列表页面

打开版面时首先看到的就是列表页面，该页面的代码将编写在list.html文件。

下图就是测试用版面皮肤的列表页面。在管理页面编辑过的版面标题‘XE测试’与 ‘用于测试XE版面皮肤制作’的详细说明始终会显示在版面头部。



图示 4-2 版面列表页面制作完成

上面的列表把所有的项显示了出来。因为不知道版面管理员将会选择显示哪些项，因此制作皮肤时要设置显示全部的显示项。

制作list.html 的方法如下

引用(include) _header.html, _footer.html

测试用皮肤中的list.html里如下列代码调用(include)了_header.html和_footer.html文件。

```
<include target="_header.html" />
        这里将显示版面目录
<include target="_footer.html" />
```

上述代码使用到的模板语法如下：

XE 模板语法	说明

<include target="_header.html" />	引用(include) _header.html
<include target="_footer.html" />	调用(include) _footer.html

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

版面无文章时显示内容

使用条件句可以在没有提交的文章时显示提示信息。测试用版面皮肤中的 list.html在无文章时会显示："无发布文章"。该段条件句代码如下：

```
<include target="_header.html" />
<p cond="!$document_list && !$notice_list" class="no_document">{$lang->no_documents}</p>
<include target="_footer.html" />
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
cond="!$document_list && !$notice_list"	无文章或公告时显示。
{$lang->no_documents}	"无发布文章"语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

使用表格<table>来编写版面目录

一般情况下版面目录是用表格<table>来编写的。 为了使用各种条件句与变量，按如下代码来编写目录的整体结构。表格大体上用目录头部(LIST HEADER)，公告（NOTICE)，目录(LIST) 行来区分。为了提高代码可读性，利用注释来表示。

```
<include target="_header.html" />
<p cond="!$document_list && !$notice_list">{$lang->no_documents}</p>
<table width="100%" border="1" cellspacing="0" summary="List of Articles" id="board_list" class="board_list"
cond="$document_list || $notice_list">

    <thead>
        <!-- LIST HEADER -->
        <tr>
            <th scope="col">序号</th>
            <th scope="col">标题</th>
            <th scope="col">作者</th>
            <th scope="col">ID</th>
            <th scope="col">姓名</th>
            <th scope="col">日期</th>
            <th scope="col">最近更新 </th>
            <th scope="col">最新文章 </th>
            <th scope="col">浏览数</th>
            <th scope="col">推荐数</th>
        </tr>
```

```
        <!-- /LIST HEADER -->
    </thead>
    <tbody>
        <!-- NOTICE -->
        <tr>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <!-- /NOTICE -->
        <!-- LIST -->
        <tr>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <!-- /LIST -->
    </tbody>
</table>
<include target="_footer.html" />
```

上述代码使用的模板语法与变量如下：

XE 模板语法/变量	说明
cond="$document_list || $notice_list"	无文章或公告时，显示表格及其内容。

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

显示版面目录头部

版面目录头部是指例如序号、作者、日期、浏览数等数据列的标题部分。因为不知道版面管理员将会选择显示哪些项，因此皮肤制作者应该尽量把所有可支持的内容用条件句来处理。

比如版面管理员可以在 XE 管理员后台点击 **扩展功能 > 已安装的模块 > 版面**中的 **设定**， 打开 **列表设置** 来设置显示全部的选项。此时如果皮肤不支持显示全部的选项，版面管理员可能会误认该皮肤有缺陷。

图示 4-3 版面列表设置

XE的版面模块支持并显示如上图所示的12种信息.**缩略图**与**简要**部分通常不作为独立的内容来显示，而是与文章标题一并显示的。因此并不在版面目录的总列数考虑范围内。因此为了显示全部项，只需编写10个显示项的条件句即可。如果版面管理员设置为显示所有列，表格就会有10列显示项。

测试用版面皮肤中的 list.html里如下列代码，在 <thead> 元素内部编写了关于这十个列的条件句。

```
...
<thead>
    <!-- LIST HEADER -->
    <tr>
        <block loop="$list_config=>$key,$val">
        <th scope="col" cond="$val->type=='no'">{$lang->no}</th>
        <th scope="col" class="title" cond="$val->type=='title'">{$lang->title}</th>
        <th scope="col" cond="$val->type=='nick_name'">{$lang->writer}</th>
        <th scope="col" cond="$val->type=='user_id'">{$lang->user_id}</th>
        <th scope="col" cond="$val->type=='user_name'">{$lang->user_name}</th>
        <th scope="col" cond="$val->type=='regdate'"><a
href="{getUrl('sort_index','regdate','order_type',$order_type)}">{$lang->date}</a></th>
```

```
            <th scope="col" cond="$val->type=='last_update'"><a
href="{getUrl('sort_index','last_update','order_type',$order_type)}">{$lang->last_update}</a></th>
            <th scope="col" cond="$val->type=='last_post'"><a
href="{getUrl('sort_index','last_update','order_type',$order_type)}">{$lang->last_post}</a></th>
            <th scope="col" cond="$val->type=='readed_count'"><a
href="{getUrl('sort_index','readed_count','order_type',$order_type)}">{$lang->readed_count}</a></th>
            <th scope="col" cond="$val->type=='voted_count'"><a
href="{getUrl('sort_index','voted_count','order_type',$order_type)}">{$lang->voted_count}</a></th>
            </block>
        </tr>
        <!-- /LIST HEADER -->
</thead>
...
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
<block loop="$list_config=>$key,$val">...</block>	遍历显示项目
<th scope="col" cond="$val->type=='no'">{$lang->no}</th>	文章序号
<th scope="col" class="title" cond="$val->type=='title'"> {$lang->title} </th>	文章标题
<th scope="col" cond="$val->type=='nick_name'"> {$lang->writer} </th>	作者
<th scope="col" cond="$val->type=='user_id'"> {$lang->user_id} </th>	ID
<th scope="col" cond="$val->type=='user_name'"> {$lang->user_name} </th>	用户名
<th scope="col" cond="$val->type=='regdate'"> {$lang->date} </th>	日期(可以变更排列顺序)
<th scope="col" cond="$val->type=='last_update'"> {$lang->last_update} </th>	最近更新（可以变更排列顺序)

XE 模板语法 / 变量	说明
<th scope="col" cond="$val->type=='last_post'"> {$lang->last_post} </th>	最新文章 (可以变更排列顺序)
<th scope="col" cond="$val->type=='readed_count'"> {$lang->readed_count} </th>	浏览数(可以变更排列顺序)
<th scope="col" cond="$val->type=='voted_count'"> {$lang->voted_count} </th>	推荐数(可以变更排列顺序)

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

显示文章列表

列表有公告列表与一般文章列表两个部分。公告列表始终显示在表格上部。

测试用版面皮肤里的 list.html文件按下列代码，在<tbody>元素内部输出公告列表与普通文章列表。

```
...
<tbody>
    <!-- NOTICE -->
    <tr class="notice" loop="$notice_list=>$no,$document">
        <block loop="$list_config=>$key,$val">
        <td class="notice" cond="$val->type=='no'">
            <block cond="$document_srl==$document->document_srl">&raquo;</block>
            <block cond="$document_srl!=$document->document_srl">{$lang->notice}</block>
        </td>
        <td class="title" cond="$val->type=='title'">
            <a href="{getUrl('document_srl',$document->document_srl, 'listStyle', $listStyle, 'cpage','')}">
                {$document->getTitle()}
            </a>
            <a cond="$document->getCommentCount()" href="{getUrl('document_srl', $document->document_srl)}#comment" class="replyNum" title="Replies">
                [{$document->getCommentCount()}]
            </a>
            <a cond="$document->getTrackbackCount()" href="{getUrl('document_srl', $document->document_srl)}#trackback" class="trackbackNum" title="Trackbacks">
                [{$document->getTrackbackCount()}]
```

```
                </a>
            </td>
            <td class="author" cond="$val->type=='nick_name'">{$document->getNickName()}</td>
            <td class="author" cond="$val->type=='user_id'">{$document->getUserID()}</td>
            <td class="author" cond="$val->type=='user_name'">{$document->getUserName()}</td>
            <td class="time" cond="$val->type=='regdate'">{$document->getRegdate('Y.m.d')}</td>
            <td class="time" cond="$val->type=='last_update'">{zdate($document-
>get('last_update'),'Y.m.d')}</td>
            <td class="lastReply" cond="$val->type=='last_post'">
                <block cond="(int)($document->get('comment_count'))>0">
                    <a href="{$document->getPermanentUrl()}#comment" title="Last Reply">
                        {zdate($document->get('last_update'),'Y.m.d')}
                    </a>
                    <span cond="$document->get('last_updater')">
                        <sub>by</sub>
                        {htmlspecialchars($document->get('last_updater'))}
                    </span>
                </block>
                <block cond="(int)($document->get('comment_count'))==0"> </block>
            </td>
            <td class="readNum" cond="$val->type=='readed_count'">{$document-
>get('readed_count')>0?$document->get('readed_count'):'0'}</td>
            <td class="voteNum" cond="$val->type=='voted_count'">{$document-
>get('voted_count')!=0?$document->get('voted_count'):'0'}</td>
            </block>
        </tr>
        <!-- /NOTICE -->
        <!-- LIST -->
        <tr loop="$document_list=>$no,$document">
            <block loop="$list_config=>$key,$val">
            <td class="no" cond="$val->type=='no'">
                <block cond="$document_srl==$document->document_srl">&raquo;</block>
                <block cond="$document_srl!=$document->document_srl">{$no}</block>
            </td>
            <td class="title" cond="$val->type=='title'">
                <a href="{getUrl('document_srl',$document->document_srl, 'listStyle', $listStyle, 'cpage','')}">
                    {$document->getTitle()}
                </a>
                <a cond="$document->getCommentCount()" href="{getUrl('document_srl', $document-
>document_srl)}#comment" class="replyNum" title="Replies">
                    [{$document->getCommentCount()}]
                </a>
                <a cond="$document->getTrackbackCount()" href="{getUrl('document_srl', $document-
>document_srl)}#trackback" class="trackbackNum" title="Trackbacks">
                    [{$document->getTrackbackCount()}]
                </a>
            </td>
            <td class="author" cond="$val->type=='nick_name'">{$document->getNickName()}</td>
```

```
        <td class="author" cond="$val->type=='user_id'">{$document->getUserID()}</td>
        <td class="author" cond="$val->type=='user_name'">{$document->getUserName()}</td>
        <td class="time" cond="$val->type=='regdate'">{$document->getRegdate('Y.m.d')}</td>
        <td class="time" cond="$val->type=='last_update'">{zdate($document->get('last_update'),'Y.m.d')}</td>
        <td class="lastReply" cond="$val->type=='last_post'">
            <block cond="(int)($document->get('comment_count'))>0">
                <a href="{$document->getPermanentUrl()}#comment" title="Last Reply">
                    {zdate($document->get('last_update'),'Y.m.d')}
                </a>
                <span cond="$document->get('last_updater')">
                    <sub>by</sub>
                    {htmlspecialchars($document->get('last_updater'))}
                </span>
            </block>
            <block cond="(int)($document->get('comment_count'))==0"> </block>
        </td>
        <td class="readNum" cond="$val->type=='readed_count'">{$document->get('readed_count')>0?$document->get('readed_count'):'0'}</td>
        <td class="voteNum" cond="$val->type=='voted_count'">{$document->get('voted_count')!=0?$document->get('voted_count'):'0'}</td>
        </block>
    </tr>
    <!-- /LIST -->
</tbody>
...
```

上述代码使用的模板语法与变量如下。

XE 模板语法 /变量	说明
<tr class="notice" loop="$notice_list=>$no,$document">	循环公告列表
<tr loop="$document_list=>$no,$document">	循环普通文章列表
<block loop="$list_config=>$key,$val">	循环文章显示项列表
<block cond="$document_srl==$document->document_srl">»</block>	如果文章固有号与当前页文章的固有号一致，则显示右书名号>>
<block cond="$document_srl!=$document->document_srl">{$lang->notice}</block>	如果文章固有号与当前页文章的固有号不一致，则显示‘公告’。
<block cond="$document_srl!=$document->document_srl">{$no}</block>	如果文章固有号与当前页文章固有号不一致，则显示‘文章序号’。
<td class="title" cond="$val->type=='title'">	文章标题单元格
document_srl, 'listStyle', $listStyle, 'cpage','')}">...	文章链接
{$document->getTitle()}	显示文章标题

XE 模板语法 /变量	说明
getCommentCount()" href="{getUrl('document_srl', $document->document_srl)}#comment" class="replyNum" title="Replies"> [...] 	文章评论链接
{$document->getCommentCount()}	文章评论数
getTrackbackCount()" href="{getUrl('document_srl', $document->document_srl)}#trackback" class="trackbackNum" title="Trackbacks"> [...] 	文章引用链接
{$document->getTrackbackCount()}	文章引用数
<td class="author" cond="$val->type=='nick_name'">{$document->getNickName()}</td>	昵称
<td class="author" cond="$val->type=='user_id'">{$document->getUserID()}</td>	ID
<td class="author" cond="$val->type=='user_name'">{$document->getUserName()}</td>	会员名
<td class="time" cond="$val->type=='regdate'">{$document->getRegdate('Y.m.d')}</td>	日期
<td class="time" cond="$val->type=='last_update'">{zdate($document->get('last_update'),'Y.m.d')}</td>	最终修改
<td class="lastReply" cond="$val->type=='last_post'">...</td>	最终文章
<block cond="(int)($document->get('comment_count'))>0">...</block>	有一个以上评论时显示
<block cond="(int)($document->get('comment_count'))==0">...</block>	无评论时显示
getPermanentUrl()}#comment" title="Last Reply"> {zdate($document->get('last_update'),'Y.m.d')} 	最后评论日期 + 链接
<span cond="$document->get('last_updater')"> _{by} {htmlspecialchars($document->get('last_updater'))}	最终评论人

XE 模板语法 /变量	说明
<td class="readNum" cond="$val->type=='readed_count'"> {$document->get('readed_count')>0?$document->get('readed_count'):'0'} </td>	浏览数
<td class="voteNum" cond="$val->type=='voted_count'"> {$document->get('voted_count')!=0?$document->get('voted_count'):'0'} </td>	推荐数

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

> 普通文章目录显示数可以在XE管理员后台的 **扩展功能 > 已安装的模块 > 版面**中，点击需要修改模块右方的**设置**，并在**版面信息**标签栏页面中设置**列表数**来变更，默认值为20..

显示分页链接

当文章数量超过了每页显示数量，则为了浏览以前的文章须提供页面移动链接。

测试用版面皮肤中的 list.html文件按下列代码，输出分页链接。之所以用 ... 元素来包含分页链接是因为还要把 ‘撰写’按钮与‘搜索’提交样式添加到该元素内部 。

```
<table summary="List of Articles" id="board_list" class="board_list">
    <!-- LIST HEADER -->
    <!-- /LIST HEADER -->
    <!-- NOTICE -->
    <!-- /NOTICE -->
    <!-- LIST -->
    <!-- /LIST -->
</table>
<div class="list_footer">
    <!-- PAGINATION -->
    <div class="pagination" cond="$document_list || $notice_list">
        <a href="{getUrl('page','','document_srl','','division',$division,'last_division',$last_division)}"
class="prevEnd">{$lang->first_page}</a>
        <block loop="$page_no=$page_navigation->getNextPage()">
            <strong cond="$page==$page_no">{$page_no}</strong>
            <a cond="$page!=$page_no"
href="{getUrl('page',$page_no,'document_srl','','division',$division,'last_division',$last_division)}">{$page_no}</a
>
        </block>
        <a href="{getUrl('page',$page_navigation-
>last_page,'document_srl','','division',$division,'last_division',$last_division)}" class="nextEnd">{$lang-
>last_page}</a>
    </div>
```

```
    <!-- /PAGINATION -->
</div>
<include target="_footer.html" />
```

上述代码中使用到的模板语法与变量如下：

XE 模板语法 / 变量	说明
...	如果有文章或公告，则显示分页号。
{$lang->first_page}	‘第一页’ 链接
last_page,'document_srl','','division',$division,'last_division',$last_division)}" class="nextEnd">{$lang->last_page}	‘最后一页’ 链接
<block loop="$page_no=$page_navigation->getNextPage()">...</block>	页面链接循环
{$page_no}	页面序号与当前页一致时显示。
{$page_no}	页面序号与当前页不一致时显示。

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

显示撰写按钮

list.html不仅在目录页面显示，阅读页面也会引用。点击**撰写**按钮后，将会移动到 write_form.html 页面。

测试用版面皮肤中，移动到撰写页面的链接代码如下：

```
<div class="list_footer">
    <!-- PAGINATION -->
    <div class="pagination">
        ...
    </div>
    <!-- /PAGINATION -->
    <div class="btnArea">
        <a href="{getUrl('act','dispBoardWrite','document_srl','')}" class="btn">{$lang->cmd_write}</a>
    </div>
</div>
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
{$lang->cmd_write}	撰写页面链接按钮，显示在目录页与阅读页面。

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

显示版面搜索部分样式

显示可以搜索版面内容的搜索项样式。测试用版面皮肤中搜索样式部分的代码如下：

```
<div class="list_footer">
    <!-- PAGINATION -->
    <div class="pagination">
        ...
    </div>
    <!-- /PAGINATION -->
    <a ...>{$lang->cmd_write}</a>
    <!-- SEARCH -->
    <form cond="$grant->view" action="{getUrl()}" method="get" onsubmit="return procFilter(this, search)"
class="board_search">
        <input type="hidden" name="vid" value="{$vid}" />
        <input type="hidden" name="mid" value="{$mid}" />
        <input type="hidden" name="category" value="{$category}" />
        <input type="text" name="search_keyword" value="{htmlspecialchars($search_keyword)}"
accesskey="S" title="{$lang->cmd_search}" class="iText" />
        <select name="search_target">
            <option loop="$search_option=>$key,$val" value="{$key}"
selected="selected"|cond="$search_target==$key">{$val}</option>
        </select>
        <input type="submit" onclick="xGetElementById('fo_search').submit();return false;" value="{$lang-
>cmd_search}" class="btn" />
    </form>
    <!-- /SEARCH -->
</div>
```

上述代码中使用到的模板语法与代码如下：

XE 模板语法 / 变量	说明
<form cond="$grant->view" action="{getUrl()}" method="get" onsubmit="return procFilter(this, search)" id="board_search" class="board_search">	如果拥有阅读权限，则显示搜索样式部分
<input type="hidden" name="vid" value="{$vid}" />	提交虚拟站点 ID (hidden type)
<input type="hidden" name="mid" value="{$mid}" />	提交模块 ID (hidden type)
<input type="hidden" name="category" value="{$category}" />	提交分类信息 (hidden type)
<input type="text" name="search_keyword"	검색 搜索关键字的填写与显示元素. 匹配快捷键

XE 模板语法 / 变量	说明
value="{htmlspecialchars($search_keyword)}" accesskey="S" title="{$lang->cmd_search}" class="iText" />	'S'.
<select name="search_target">	选择搜索范围
<option loop="$search_option=>$key,$val" value="{$key}" selected="selected"|cond="$search_target==$key">{$val}</option>	显示搜索范围选项。
<input type="submit" onclick="xGetElementById('board_search').submit();return false;" value="{$lang->cmd_search}" class="btn" />	提交搜索内容按钮

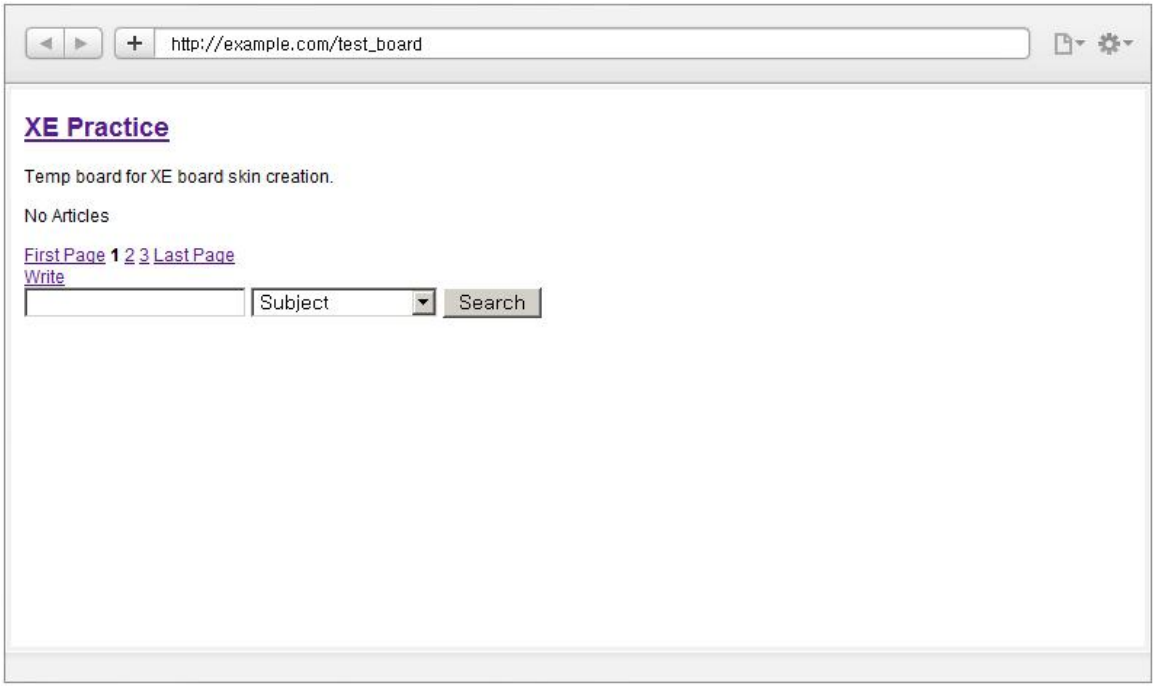
这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

确认版面目录页面显示结果

请从以下地址确认版面目录页面 。下列地址中 'example.com' 是指用户设置站点时使用的域名。

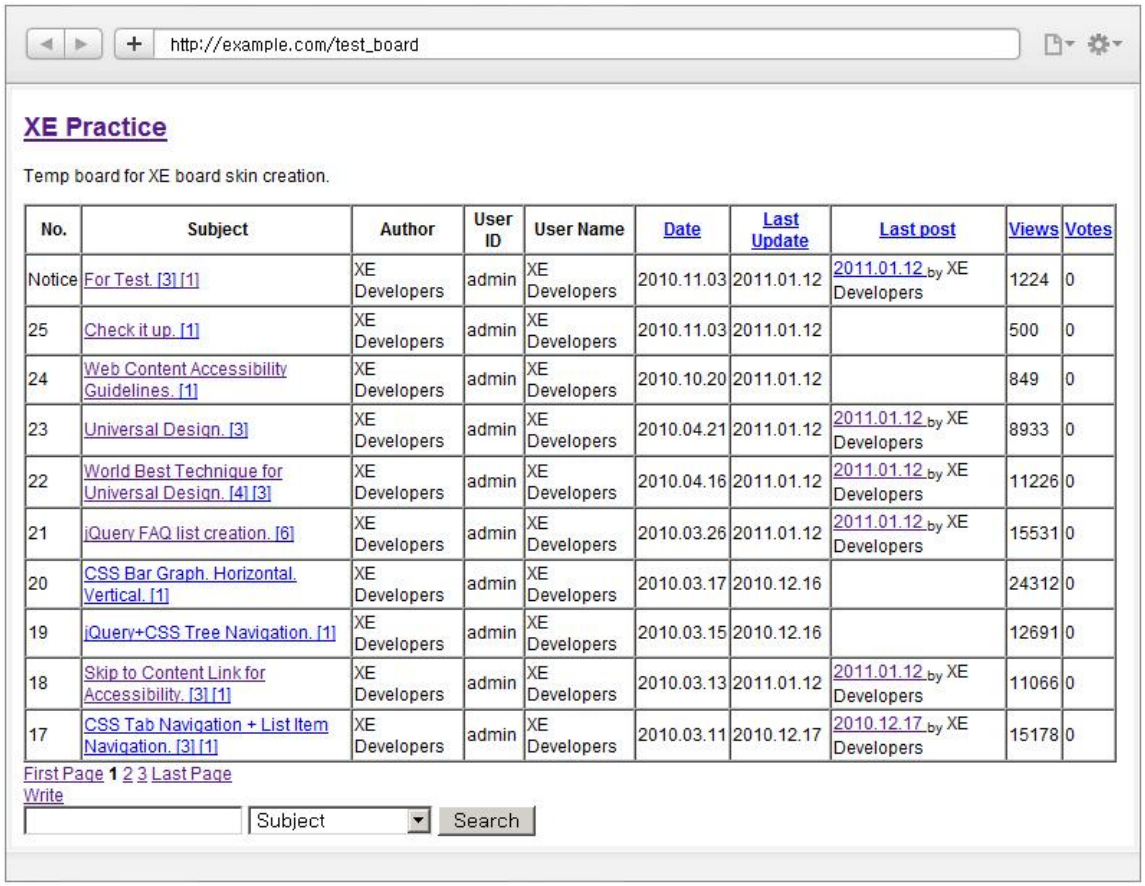
- 使用mod_rewrite时: http://example.com/test_board/
- 不使用mod_rewrite时: http://example.com/?mid=test_board

如果没有提交的文章，则显示“无发布文章” ，按下图所示：



图示 4-4 版面目录页面 – 无文章状态

如果有发布的文章则显示下图中页面。



图示 4-5 版面目录页面 – 有文章状态

因为还没有 write_form.html 页面，目前还不能直接在 'test_board' 版面发布文章。

但是可以把测试用文章数据复制到'test_ board' 版面中。上图中的页面就是从别的版面将一些测试用数据复制到 'test_board' 版面的。

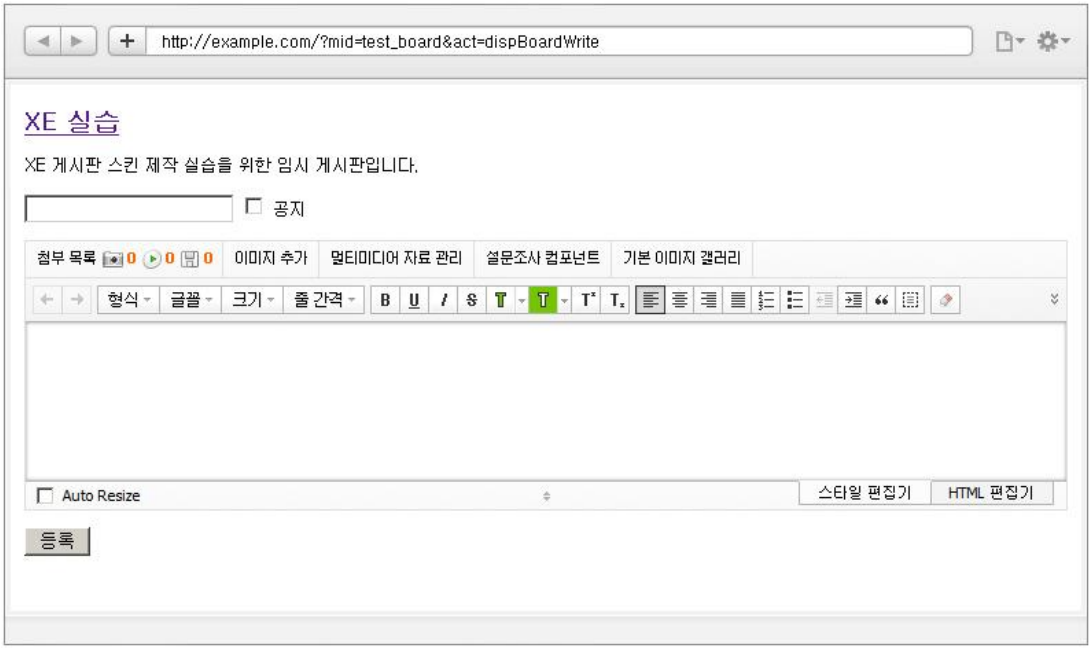
在XE管理员后台页面中选择 **内容 > 文章**页面，并选择任意文章后，点击**管理已选择文章**来把这些文章复制到 'test_board' 版面中。

4.9 制作撰写页面

撰写页面用来发布新的文章或修改已有的文章。撰写页面. 该页面的代码将编写在list.html文件。

撰写页面由 _header.html, _footer.html, 文章标题文本框、文章内容编辑区、作者信息文本输入框(姓名、密码、个人主页)、以及发布按钮构成。XE core已经包含该可视化编辑器，撰写页面只需调用该编辑器模块即可。

下图是用测试用版面皮肤中的撰写页面。



图示 4-8 撰写页面制作完成

制作write_form.html的方法如下：

引用(include) _header.html, _footer.html 文件

测试用版面皮肤的 write_form.html文件中，调用(include)了_header.html与 _footer.html, 代码如下：

```
<include target="_header.html" />
    这里编写文章撰写样式部分。
<include target="_footer.html" />
```

上述代码中使用到的模板语法与变量如下：

XE 模板语法 / 变量	说明
<include target="_header.html" />	调用(include) _header.html
<include target="_footer.html" />	调用(include) _footer.html

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

撰写页面的结构

撰写页面的 HTML 结构如下：

```
<include target="_header.html" />
<form action="" class="board_write">
    <div class="write_header">标题文本框</div>
    <div class="write_editor"> 内容编辑区 </div>
    <div class="write_author">作者信息文本框 (姓名、密码、个人主页)</div>
    <div class="write_footer">发布按钮</div>
</form>
<include target="_footer.html" />
```

为了将 ‘标题、内容、作者信息’ 提交到服务器上，需要用 form元素来编写。点击**发布**按钮就可以将form里的内容发送出去。

制作撰写页面样式

可以通过onsubmit 属性，在用户端检查填写的内容是否符合要求。利用撰写页面来修改文章时，系统将调出用户发布过的内容数据。

测试用版面皮肤的 write_form.html文件里，使用如下代码编写了撰写内容部分样式。

```
<include target="_header.html" />
<form action="./" method="post" onsubmit="return procFilter(this, window.insert)" class="board_write">
    <input type="hidden" name="mid" value="{$mid}" />
    <input type="hidden" name="content" value="{$oDocument->getContentText()}" />
    <input type="hidden" name="document_srl" value="{$document_srl}" />
    <input type="hidden" name="allow_comment" value="Y" />
    <input type="hidden" name="allow_trackback" value="Y" />
    ...
</form>
<include target="_footer.html" />
```

上述代码使用的模板语法与变量如下：

XE 模板语发 / 变量	说明
onsubmit="return procFilter(this, window.insert)"	检查用户填写内容的有效性并传送 form.
<input type="hidden" name="mid" value="{$mid}" />	提交模块 ID (hidden type) .
<input type="hidden" name="content" value="{$oDocument->getContentText()}" />	提交内容 (hidden type) .
<input type="hidden" name="document_srl" value="{$document_srl}" />	提交文档固有序号 (hidden type) .
<input type="hidden" name="allow_comment" value="Y" />	提交是否允许评论 (hidden type). 不允许时选择 value="N".
<input type="hidden" name="allow_trackback"	提交是否允许引用 (hidden type). 不允许时选择

XE 模板语发 / 变量	说明
value="Y" />	value="N".

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

制作标题文本输入框

关于标题文本框需要考虑 '撰写新文章' 和 '修改已有文章' 两种情况，还要考虑是否选择发布为公告。

测试用版面皮肤的 write_form.html文件中，按下列代码来编写了标题文本框。

```
...
<div class="write_header">
    <input cond="$oDocument->getTitleText()" type="text" name="title" class="iText" title="{$lang->title}"
value="{htmlspecialchars($oDocument->getTitleText())}" />
    <input cond="!$oDocument->getTitleText()" type="text" name="title" class="iText" title="{$lang->title}" />
    <input cond="$grant->manager" type="checkbox" name="is_notice" value="Y"
checked="checked"|cond="$oDocument->isNotice()" id="is_notice" />
    <label cond="$grant->manager" for="is_notice">{$lang->notice}</label>
</div>
...
```

上述代码使用到的模板语法与变量。

XE 模板语法 / 变量	说明
cond="$oDocument->getTitleText()"	已有标题的修改页面中显示。
cond="!$oDocument->getTitleText()"	无标题的新撰写页面中显示。
{htmlspecialchars($oDocument->getTitleText())}	修改文章状态中显示文档标题
{$lang->title}	'标题' 语言变量
cond="$grant->manager"	身份为管理员时显示是否设为公告的选项和提示语。
checked="checked"|cond="$oDocument->isNotice()"	修改页面中，如果该文章为公告，则显示 checked 属性与值。
{$lang->notice}	'公告' 语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

制作内容编辑区

有关内容编辑区的代码只需使用 {$oDocument->getEditor()} 变量即可。使用该变量只需到 **扩展功能 > 已安装的模块 > 版面**，在需要编辑的模块右方点击 **设置**，并在**附加设置**标签页中的可视化编辑器中将代码复制过来即可。

测试用版面皮肤的 write_form.html文件中，按如下代码调用了可视化编辑器

```
...
```

```
<div class="write_editor">
    {$oDocument->getEditor()}
</div>
...
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
{$oDocument->getEditor()}	编辑器（可视化编辑器）变量

编写作者信息输入框

作者信息输入框只显示给没有登录的用户。输入框有 '姓名，密码，个人主页'，输入的密码在以后修改文章或删除时使用。

测试用版面皮肤的 write_form.html 文件中，按如下代码编写了作者信息输入框。

```
...
<div class="write_author" cond="!$is_logged">
    <label for="userName">{$lang->writer}</label>
    <input type="text" name="nick_name" id="userName" class="iText userName"
value="{htmlspecialchars($oDocument->get('nick_name'))}" />
    <label for="userPw">{$lang->password}</label>
    <input type="password" name="password" id="userPw" class="iText userPw" />
    <label for="homePage">{$lang->homepage}</label>
    <input type="text" name="homepage" id="homePage" class="iText homePage"
value="{htmlspecialchars($oDocument->get('homepage'))}" />
</div>
...
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
	没有登录时显示作者信息输入框
{$lang->writer}	'姓名' 语言变量
{htmlspecialchars($oDocument->get('nick_name'))}	当撰写页面用于修改已有文章时，显示该文章作者名。
{$lang->password}	'密码' 语言变量
{$lang->homepage}	'个人主页' 语言变量
{htmlspecialchars($oDocument->get('homepage'))}	当撰写页面用于修改已有文章时，显示已填的个人主页地址

制作发布按钮

要将用户填写的form内容提交到服务器时，需要使用”发布”按钮

测试用版面皮肤的 write_form.html 文件中，按如下代码编写了”发布”按钮。

```
...
<div class="btnArea">
    <input type="submit" value="{$lang->cmd_registration}" class="btn" />
</div>
...
```

上述代码中使用到的模板语法与变量如下：

XE 模板语法 / 变量	说明
{$lang->cmd_registration}	'发布' 语言变量

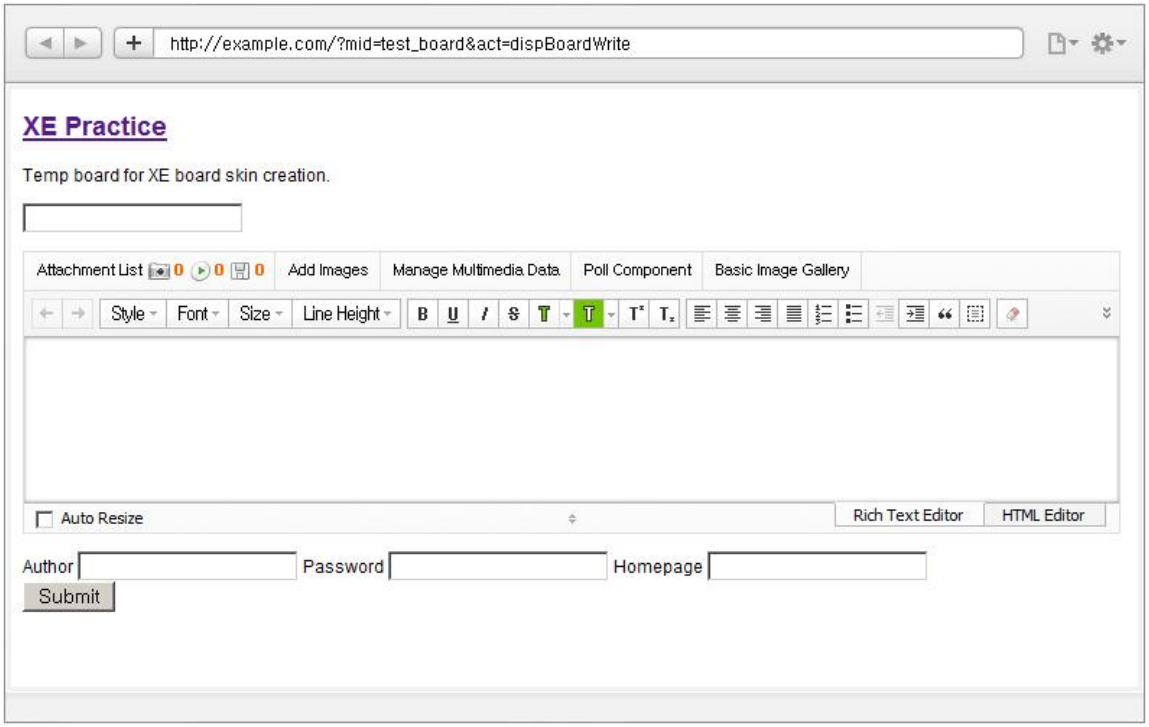
这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

确认撰写页面显示效果

在版面目录页点击**撰写**按钮，或通过以下地址访问版面撰写页面。地址中 'example.com' 是指用户设置站点时使用的域名。

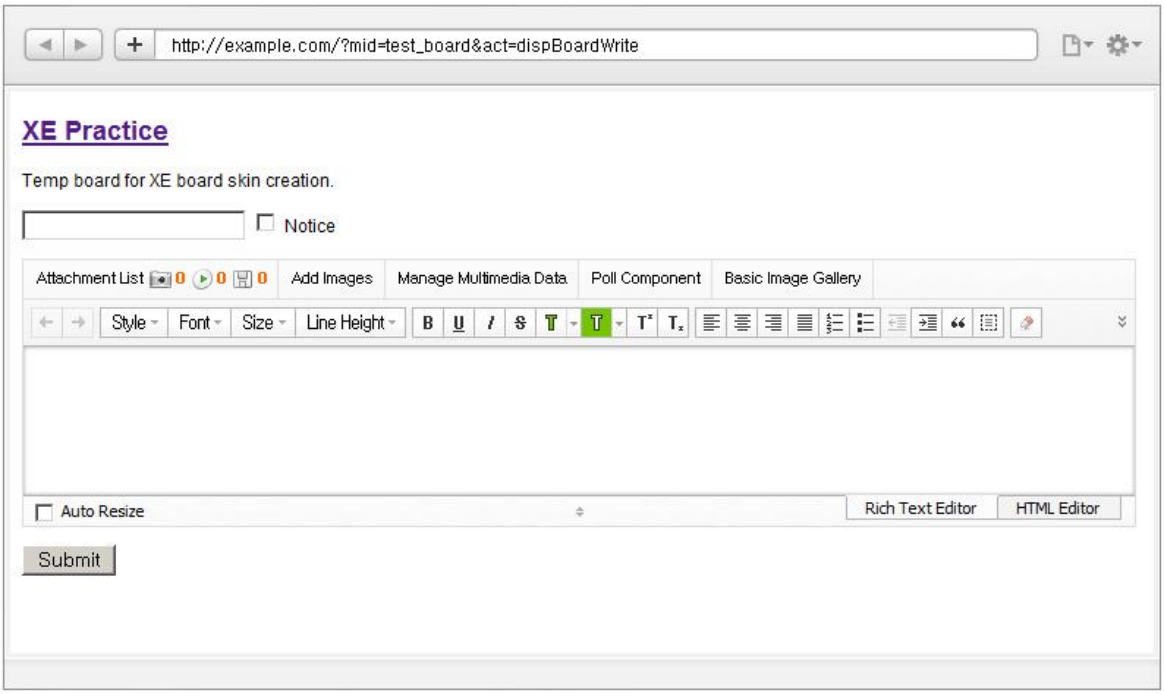
- http://example.com/?mid=test_board&act=dispBoardWrite

在未登录状态下打开该页面会显示作者信息输入文本框部分，如下图：



图示 4-6 未登录状态下的撰写页面

登录状态下该页面显示如下。页面不会显示作者信息输入框，并且如果当前用户拥有管理员权限，则出现是否设为公告的选项。



图示 4-7 登录状态下的撰写页面

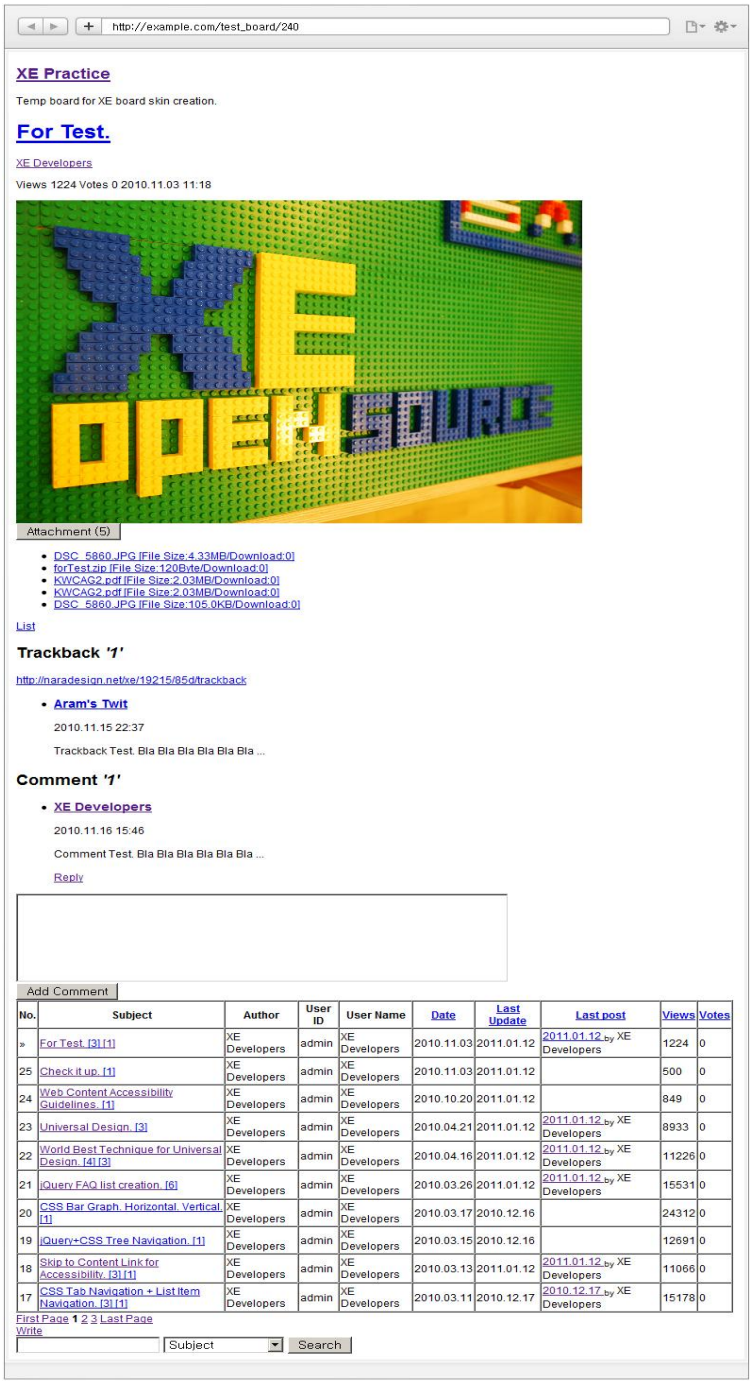
填写标题与内容，并点击”发布”后，在版面目录页面确认刚发布的文章是否显示在版面目录中。已登录时的必填项为标题和内容，未登录则是标题与内容，再加上姓名、密码。

4.10 制作阅读页面

阅读页面除了文章正文部分外，还包括引用列表，评论列表，提交评论部分，文章目录等部分，这些内容在,_read.html文件中编写。

目录页面（list.html)中引用(include）阅读页面，是为了用户在阅读页面下方便浏览其它文章。

以下为测试用版面皮肤中的阅读页面：



图示 4-8 阅读页面制作完成

制作_read.html文件方法如下：

list.html 中 引用(include) _read.html 文件

list.html要按条件来引用 _read.html文件。用户打开阅读页面时，list.html将引用(include) _read.html文件。

예测试用版面皮肤中的 list.html在下列代码中的位置引用了包括条件句的 include 语句。当用户访问阅读页面时，在阅读区域下方一并显示文章目录。

```
<include target="_header.html" />
<include cond="$oDocument->isExists()" target="_read.html" />
<p ...>{$lang->no_documents}</p>
<table ...  summary="List of Articles" id="board_list" class="board_list">
    ...
</table>
<div class="list_footer">
    ...
</div>
<include target="_footer.html" />
```

上述代码使用到的模板语法与代码如下：

XE 模板语法 / 变量	说明
<include cond="$oDocument->isExists()" target="_read.html" />	访问阅读页面时，引用(include) _read.html 文件

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

阅读页面结构

阅读页面如下列代码，由头部(标题、作者、浏览数、推荐数、日期)和内容正文，以及尾部(上传的附件、目录按钮、修改按钮、删除按钮)、引用列表、评论列表、 提交评论部分组成。

```
<div class="board_read">
    <!-- READ HEADER -->
    <div class="read_header">
        标题, 作者, 浏览数, 推荐数, 日期
    </div>
    <!-- /READ HEADER -->
    <!-- READ BODY -->
    <div class="read_body">
        文章正文
    </div>
    <!-- /READ BODY -->
    <!-- READ FOOTER -->
    <div class="read_footer">
        上传的附件、目录按钮、修改按钮、删除按钮
    </div>
    <!-- /READ FOOTER -->
```

```
</div>
调用(include) 引用列表
调用(include)评论列表
<!-- WRITE COMMENT -->
<form class="write_comment">
    评论填写部分
</form>
<!-- /WRITE COMMENT -->
```

显示标题与作者

测试用版面皮肤的_read.html 文件中，如下列代码显示标题与作者。

```
...
<!-- READ HEADER -->
<div class="read_header">
    <h1><a href="{$oDocument->getPermanentUrl()}">{$oDocument->getTitle()}</a></h1>
    <a cond="$oDocument->getHomepageUrl()" href="{$oDocument->getHomepageUrl()}"
class="author">{$oDocument->getNickName()}</a>
    <strong cond="!$oDocument->getHomepageUrl()" class="author">{$oDocument-
>getNickName()}</strong>
</div>
<!-- /READ HEADER -->
...
```

上述代码中使用到的语法与变量如下：

XE 模板语法 / 变量	说明
<a href="{$oDocument->getPermanentUrl()}">	文章永久 URL
{$oDocument->getTitle()}	文章标题
getHomepageUrl()" href="{$oDocument->getHomepageUrl()}" class="author">...	有个人主页时， 显示链接与昵称
<strong cond="!$oDocument->getHomepageUrl()" class="author">...	无个人主页时，只显示用户名
{$oDocument->getNickName()}	显示作者昵称

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

浏览数、推荐数、日期

测试用版面皮肤的 _read.html文件中，如下列代码显示浏览数、推荐数、日期。

```
...
<!-- READ HEADER -->
<div class="read_header">
    ...
```

```
    <p class="sum">
        <span class="read">{$lang->readed_count}
            <span class="num">{$oDocument->get('readed_count')}</span>
        </span>
        <span class="vote">{$lang->voted_count}
            <span class="num">{$oDocument->get('voted_count')}</span>
        </span>
        <span class="time">{$oDocument->getRegdate('Y.m.d H:i')}</span>
    </p>
</div>
<!-- /READ HEADER -->
...
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
{$lang->readed_count}	'浏览数' 语言变量
{$oDocument->get('readed_count')}	显示浏览数
{$lang->voted_count}	'推荐数' 语言变量
{$oDocument->get('voted_count')}	显示推荐数
{$oDocument->getRegdate('Y.m.d H:i')}	显示发布文章的日期与时间 (如果需要显示秒单位，则 H:i:s)

显示文章正文

测试用版面皮肤的 _read.html中，按下列代码显示文章正文。

```
...
<!-- READ BODY -->
<div class="read_body">
    {$oDocument->getContent(false)}
</div>
<!-- /READ BODY -->
...
```

上述代码中使用到的模板语法与变量如下：

XE 模板语法 / 变量	说明
{$oDocument->getContent(false)}	显示文章正文内容

显示上传的附件

测试用版面皮肤的 _read.html文件中，按下列代码显示上传的附件。

```
...
<!-- READ FOOTER -->
<div class="read_footer">
```

```
    <div cond="$oDocument->hasUploadedFiles()" class="fileList">
        <button type="button" class="toggleFile" onclick="jQuery(this).next('ul.files').toggle();">{$lang->uploaded_file} ({$oDocument->get('uploaded_count')})</button>
        <ul class="files">
            <li loop="$oDocument->getUploadedFiles()=>$key,$file"><a href="{getUrl('')}{$file->download_url}">{$file->source_filename} <span class="fileSize">[File Size:{FileHandler::filesize($file->file_size)}/Download:{number_format($file->download_count)}]</span></a></li>
        </ul>
    </div>
</div>
<!-- /READ FOOTER -->
...
```

上述代码中使用到的模板语法与变量如下：

XE 模板语法 / 变量	说明
<div cond="$oDocument->hasUploadedFiles()" class="fileList">	有上传附件时显示
{$lang->uploaded_file}	'附件' 语言变量
{$oDocument->get('uploaded_count')}	上传附件数
<li loop="$oDocument->getUploadedFiles()=>$key,$file">	有附件目录时输出的循环句
<a href="{getUrl('')}{$file->download_url}">	附件下载链接
{$file->source_filename}	附件名
{FileHandler::filesize($file->file_size)}	附件大小
{number_format($file->download_count)}	附件下载次数

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

显示目录、修改、删除按钮

测试用版面皮肤的 _read.html 文件中，按下列代码显示目录、修改、删除按钮。

```
...
<!-- READ FOOTER -->
<div class="read_footer">
    ...
    <div class="btnArea">
        <span class="goList"><a href="#board_list" class="btn">{$lang->cmd_list}</a></span>
        <span class="goEdit">
            <a cond="$oDocument->isEditable()" class="btn" href="{getUrl('act','dispBoardWrite','document_srl',$oDocument->document_srl,'comment_srl','')}">{$lang->cmd_modify}</a>
```

```
                <a cond="$oDocument->isEditable()" class="btn"
href="{getUrl('act','dispBoardDelete','document_srl',$oDocument->document_srl,'comment_srl','')}">{$lang->cmd_delete}</a>
            </span>
        </div>
</div>
<!-- /READ FOOTER -->
...
```

위上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
...	移动到下方的文章目录
{$lang->cmd_list}	'目录' 语言变量
isEditable()" ...>...	有编辑权限时显示
{getUrl('act','dispBoardWrite','document_srl',$oDocument->document_srl,'comment_srl','')}	显示修改页面 URL
{$lang->cmd_modify}	'修改' 语言变量
{getUrl('act','dispBoardDelete','document_srl',$oDocument->document_srl,'comment_srl','')}	显示删除页面 URL
{$lang->cmd_delete}	'删除' 语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

调用(include) 引用目录、评论目录

测试用版面皮肤的 _read.html文件中，按下列代码调用(include)了 _trackback.html与 _comment.html.

```
<div class="board_read">
    <!-- READ HEADER -->
    ...
    <!-- /READ HEADER -->
    <!-- READ BODY -->
    ...
    <!-- /READ BODY -->
    <!-- READ FOOTER -->
    ...
    <!-- /READ FOOTER -->
</div>
<include cond="$oDocument->allowTrackback()" target="_trackback.html" />
<include cond="$oDocument->allowComment()" target="_comment.html" />
<!-- WRITE COMMENT -->
...
<!-- /WRITE COMMENT -->
```

上述代码中使用到的模板语法与变量如下：

XE 模板语法 / 变量	说明
<include target="..." />	调用(include) 文件
cond="$oDocument->allowTrackback()"	允许引用(Trackback)则调出(include)引用(Trackback)目录文件 (_trackback.html)
cond="$oDocument->allowComment()"	允许发表评论则调用(include)评论目录文件 (_comment.html)

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

显示文章的评论发表样式

测试用版面皮肤的 _read.html文件中，按下列代码显示文章评论的发表样式。关于评论的发表与修改部分代码写在 comment_form.html文件。

```
<div class="board_read">
    <!-- READ HEADER -->
    ...
    <!-- /READ HEADER -->
    <!-- READ BODY -->
    ...
    <!-- /READ BODY -->
    <!-- READ FOOTER -->
    ...
    <!-- /READ FOOTER -->
</div>
<include cond="$oDocument->allowTrackback()" target="_trackback.html" />
<include cond="$oDocument->allowComment()" target="_comment.html" />
<!-- WRITE COMMENT -->
<form cond="$grant->write_comment && $oDocument->isEnableComment()" action="./" method="post"
onsubmit="return procFilter(this, insert_comment)" class="write_comment">
    <input type="hidden" name="mid" value="{$mid}" />
    <input type="hidden" name="document_srl" value="{$oDocument->document_srl}" />
    <input type="hidden" name="comment_srl" value="" />
    <textarea name="content" rows="5" cols="50"></textarea>
    <div class="write_author" cond="!$is_logged">
        <label for="userName">{$lang->writer}</label>
        <input type="text" name="nick_name" id="userName" class="iText userName" />
        <label for="userPw">{$lang->password}</label>
        <input type="password" name="password" id="userPw" class="iText userPw" />
        <label for="homePage">{$lang->homepage}</label>
        <input type="text" name="homepage" id="homePage" class="iText homePage" />
    </div>
    <div class="btnArea">
        <input type="submit" value="{$lang->cmd_comment_registration}" class="btn" />
    </div>
</form>
```

```
<!-- /WRITE COMMENT -->
```

上述代码中使用到的模板语法与变量如下：

XE 模板语法 / 变量	说明
<form cond="$grant->write_comment && $oDocument->isEnableComment()"> ...</form>	有发表评论权限并且允许评论时显示评论发表样式
onsubmit="return procFilter(this, insert_comment)"	检查用户填写内容的有效性以及 form 提交
<input type="hidden" name="mid" value="{$mid}" />	提交模块 ID (hidden type)
<input type="hidden" name="document_srl" value="{$oDocument->document_srl}" />	提交文章固有序号 (hidden type)
<input type="hidden" name="comment_srl" value="" />	提交评论固有序号 (hidden type)
<textarea name="content" rows="5" cols="50"></textarea>	评论编辑区
...	没有登录时显示包含的内容 (姓名输入框、密码输入框、个人主页输入框)
{$lang->writer}	'作者' 语言变量
{$lang->password}	'密码'语言变量
{$lang->homepage}	'个人主页' 语言变量
{$lang->cmd_comment_registration}	'发表评论' 语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

4.11 制作引用/评论页面

4.11.1 制作引用目录

引用目录在 _trackback.html编写。引用目录会被调用（include）到_read.html文件并显示。

测试用版面皮肤的 _trackback.html文件中，按下列代码显示引用目录。

```
<!-- TRACKBACK -->
<div class="feedback" id="trackback">
    <div class="fbHeader">
        <h2>{$lang->trackback} <em>'{$oDocument->getTrackbackCount()}'</em></h2>
        <p class="trackbackURL"><a href="{$oDocument->getTrackbackUrl()}" onclick="return
false;">{$oDocument->getTrackbackUrl()}</a></p>
    </div>
    <ul cond="$oDocument->getTrackbackCount()" class="fbList">
        <li class="fbItem" loop="$oDocument->getTrackbacks()=>$key,$val" id="trackback_{$val-
>trackback_srl}">
            <h3 class="author"><a href="{$val->url}" title="{htmlspecialchars($val-
>blog_name)}">{htmlspecialchars($val->title)}</a></h3>
            <p class="time">{zdate($val->regdate, "Y.m.d H:i")}</p>
            <p class="xe_content">{$val->excerpt}</p>
            <p class="action" cond="$grant->manager"><a
href="{getUrl('act','dispBoardDeleteTrackback','trackback_srl',$val->trackback_srl)}">{$lang-
>cmd_delete}</a></p>
        </li>
    </ul>
</div>
<!-- /TRACKBACK -->
```

上述代码使用到的模板语法与变量下：

XE 模板语法 / 变量	说明
{$lang->trackback}	'引用' 语言变量
{$oDocument->getTrackbackCount()}	输出引用数
{$oDocument->getTrackbackUrl()}	显示用来引用的 URL
cond="$oDocument->getTrackbackCount()"	有引用时显示包含的内容（条件）
loop="$oDocument->getTrackbacks()=>$key,$val"	有引用时显示包含的内容（循环）
{$val->trackback_srl}	引用固有编号
{$val->url}	引用页面的 URL
{htmlspecialchars($val->blog_name)}	引用网站名
{htmlspecialchars($val->title)}	引用文标题
{zdate($val->regdate, "Y.m.d H:i")}	引用时间

XE 模板语法 / 变量	说明
{$val->excerpt}	引用文内容
cond="$grant->manager"	如果是管理员，则显示包含的内容
{getUrl('act','dispBoardDeleteTrackback','trackback_srl',$val->trackback_srl)}	删除引用页面 URL
{$lang->cmd_delete}	'删除' 语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

4.11.2 制作评论目录

评论目录在 _comment.html编写。评论目录会被调用 (include) 到_read.html文件并显示。

测试用版面皮肤的 _comment.html文件中，按下列代码显示评论目录。

```
<!-- COMMENT -->
<div class="feedback" id="comment">
    <div class="fbHeader">
        <h2>{$lang->comment} <em>'{$oDocument->getCommentcount()}'</em></h2>
    </div>
    <ul cond="$oDocument->getCommentcount()" class="fbList">
        <li loop="$oDocument->getComments()=>$key,$comment" class="fbItem" style="padding-
left:{($comment->get('depth'))*15}px"|cond="$comment->get('depth')" id="comment_{$comment-
>comment_srl}">
            <h3 class="author">
                <a cond="$comment->homepage" href="{$comment->homepage}">{$comment-
>getNickName()}</a>
                <strong cond="!$comment->homepage">{$comment->getNickName()}</strong>
            </h3>
            <p class="time">{$comment->getRegdate('Y.m.d H:i')}</p>
            {$comment->getContent(false)}
            <p class="action">
                <a href="{getUrl('act','dispBoardReplyComment','comment_srl',$comment-
>comment_srl)}">{$lang->cmd_reply}</a>
                <a cond="$comment->isGranted()||!$comment->get('member_srl')"
href="{getUrl('act','dispBoardModifyComment','comment_srl',$comment->comment_srl)}">{$lang-
>cmd_modify}</a>
                <a cond="$comment->isGranted()||!$comment->get('member_srl')"
href="{getUrl('act','dispBoardDeleteComment','comment_srl',$comment->comment_srl)}">{$lang-
>cmd_delete}</a>
            </p>
        </li>
    </ul>
    <div cond="$oDocument->comment_page_navigation" class="pagination">
        <a href="{getUrl('cpage',1)}#comment" class="prevEnd">{$lang->first_page}</a>
        <block loop="$page_no=$oDocument->comment_page_navigation->getNextPage()">
```

```
            <strong cond="$cpage==$page_no">{$page_no}</strong>
            <a cond="$cpage!=$page_no" href="{getUrl('cpage',$page_no)}#comment">{$page_no}</a>
        </block>
        <a href="{getUrl('cpage',$oDocument->comment_page_navigation->last_page)}#comment"
class="nextEnd">{$lang->last_page}</a>
    </div>
</div>
<!-- /COMMENT -->
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
{$lang->comment}	'评论' 语言变量
{$oDocument->getCommentcount()}	显示评论数
cond="$oDocument->getCommentcount()"	有评论时显示包含的内容 (条件)
loop="$oDocument->getComments()=>$key,$comment"	有评论时显示包含的内容 (循环)
{($comment->get('depth'))*15}	评论的层级乘于 15
|cond="$comment->get('depth')"	评论有层级时显示属性
{$comment->comment_srl}	评论固有序号
cond="$comment->homepage"	有个人主页时显示包含的内容
{$comment->homepage}	个人主页 URL
{$comment->getNickName()}	显示作者姓名
cond="!$comment->homepage"	无个人主页时显示包含的内容
{$comment->getRegdate('Y.m.d H:i')}	显示评论发表日期与时间
{$comment->getContent(false)}	显示评论内容
{getUrl('act','dispBoardReplyComment','comment_srl',$comment->comment_srl)}	关于评论的评论编辑页面 URL
{$lang->cmd_reply}	'评论' 语言变量
cond="$comment->isGranted()\|\|!$comment->get('member_srl')"	有评论编辑权限或不是会员时显示包含的内容
{getUrl('act','dispBoardModifyComment','comment_srl',$comment->comment_srl)}	修改评论页面 URL
{$lang->cmd_modify}	'修改' 语言变量
{getUrl('act','dispBoardDeleteComment','comment_srl',$comment->comment_srl)}	删除评论页面 URL
{$lang->cmd_delete}	'删除' 语言变量
cond="$oDocument->comment_page_navigation"	评论过多需要分页时显示

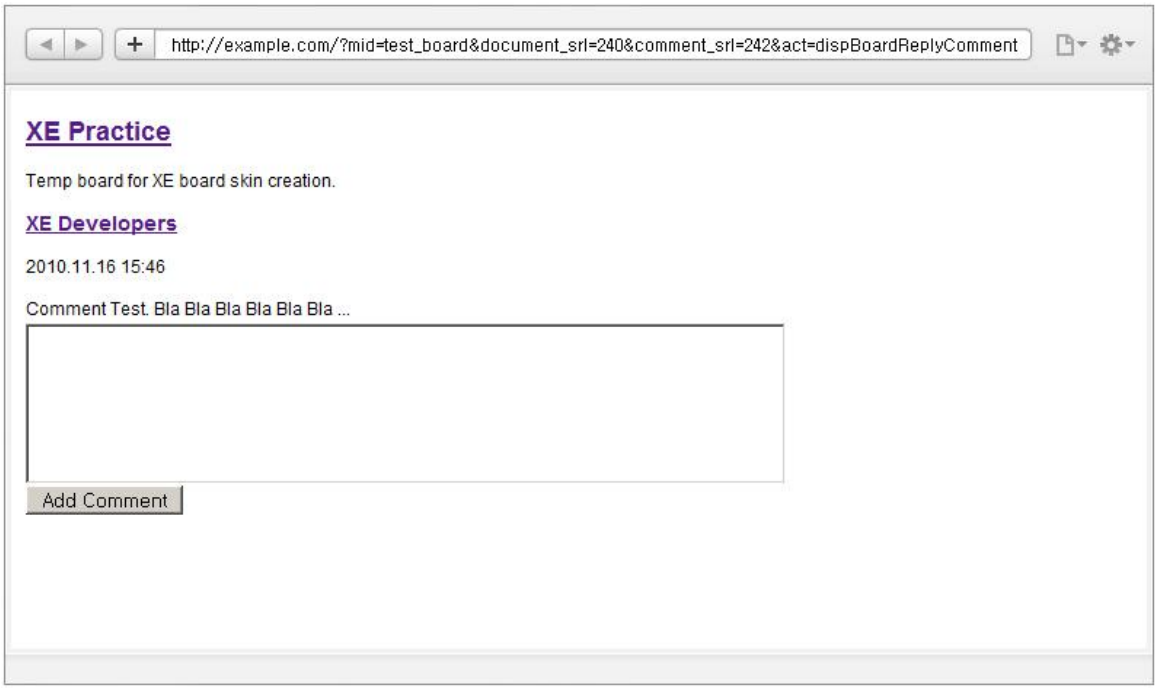
XE 模板语法 / 变量	说明
{getUrl('cpage',1)}	评论首页 URL
{getUrl('cpage',$oDocument->comment_page_navigation->last_page)}	评论最后一页 URL
{$lang->first_page}	评论 '首页' 语言变量
{$lang->last_page}	评论 '最后一页' 语言变量
<block loop="$page_no=$oDocument->comment_page_navigation->getNextPage()">...</block>	显示评论页面分页(循环)
cond="$cpage==$page_no"	当前评论页与分页编号一致时显示包含的内容
cond="$cpage!=$page_no"	当前评论也与分页编号不一致时显示
{getUrl('cpage',$page_no)}	评论页面 URL
{$page_no}	评论页面编号

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

4.11.3 关于撰写/修改关于评论的评论页面

用户就发表关于评论的二次评论或修改这些二次评论时使用的页面。 关于文章正文的评论编辑样式是被包含在阅读页面（_read.html)里的。关于评论的二次评论的编辑与修改部分在 comment_form.html文件编写。

다下图为测试用版面皮肤来制作的关于评论的二次评论编辑页面。关于原评论的作者名、评论时间、 评论原文显示在上方。至于修改自己的评论时，把本人的原评论显示在textarea区域。



图示 4-9 撰写关于评论的二次评论页面制作完成

예测试用版面皮肤的 comment_form.html文件中， 撰写/修改关于评论的二次评论部分代码如下：

```
<include target="_header.html" />
<div cond="$oSourceComment->isExists()" class="context_data">
    <h3 class="author">
        <a cond="$oSourceComment->homepage" href="{$oSourceComment-
>homepage}">{$oSourceComment->getNickName()}</a>
        <strong cond="!$oSourceComment->homepage">{$oSourceComment->getNickName()}</strong>
    </h3>
    <p class="time">{$oSourceComment->getRegdate('Y.m.d H:i')}</p>
    {$oSourceComment->getContent(false)}
</div>
<!-- WRITE COMMENT -->
<form action="./" method="post" onsubmit="return procFilter(this, insert_comment)" class="write_comment">
    <input type="hidden" name="mid" value="{$mid}" />
    <input type="hidden" name="document_srl" value="{$oComment->get('document_srl')}" />
    <input type="hidden" name="comment_srl" value="{$oComment->get('comment_srl')}" />
    <input type="hidden" name="parent_srl" value="{$oComment->get('parent_srl')}" />
    <textarea name="content" rows="5" cols="50">{htmlspecialchars($oComment-
>get('content'))}</textarea>
    <div class="write_author" cond="!$is_logged">
        <label for="userName">{$lang->writer}</label>
        <input type="text" name="nick_name" id="userName" class="iText userName"
value="{htmlspecialchars($oComment->get('nick_name'))}" />
        <label for="userPw">{$lang->password}</label>
        <input type="password" name="password" id="userPw" class="iText userPw" />
        <label for="homePage">{$lang->homepage}</label>
```

```
        <input type="text" name="homepage" id="homePage" class="iText homePage"
value="{htmlspecialchars($oComment->get('homepage'))}" />
    </div>
    <div class="btnArea">
        <input type="submit" value="{$lang->cmd_comment_registration}" class="btn" />
    </div>
</form>
<!-- /WRITE COMMENT -->
<include target="_footer.html" />
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
<include target="_header.html" />	引用(include) _header.html
<include target="_footer.html" />	引用(include) _footer.html
cond="$oSourceComment->isExists()"	有原评论时显示
cond="$oSourceComment->homepage"	有个人主页地址时显示包含的内容
cond="!$oSourceComment->homepage"	无个人主页地址时显示包含的内容
{$oSourceComment->homepage}	个人主页地址
{$oSourceComment->getNickName()}	原评论的作者名
{$oSourceComment->getRegdate('Y.m.d H:i')}	原评论发表时间
{$oSourceComment->getContent(false)}	显示评论原文
onsubmit="return procFilter(this, insert_comment)"	检查用户填写内容的有效性以及 form 提交
<input type="hidden" name="mid" value="{$mid}" />	提交版面模块 ID (hidden type)
<input type="hidden" name="document_srl" value="{$oComment->get('document_srl')}" />	提交文章固有编号 (hidden type)
<input type="hidden" name="comment_srl" value="{$oComment->get('comment_srl')}" />	提交评论固有编号 (hidden type)
<input type="hidden" name="parent_srl" value="{$oComment->get('parent_srl')}" />	提交原评论固有编号(hidden type)
<textarea name="content" rows="5" cols="50">{htmlspecialchars($oComment->get('content'))}</textarea>	评论的撰写/编辑区域，修改评论时显示 textarea 内部的变量
cond="!$is_logged"	没有登录时显示包含的内容
{$lang->writer}	'作者名' 语言变量
{$lang->password}	'密码' 语言变量
{$lang->homepage}	'个人主页' 语言变量

XE 模板语法 / 变量	说明
{htmlspecialchars($oComment->get('nick_name'))}	作者姓名
{htmlspecialchars($oComment->get('homepage'))}	个人主页 URL
{$lang->cmd_comment_registration}	'发表评论' 语言变量

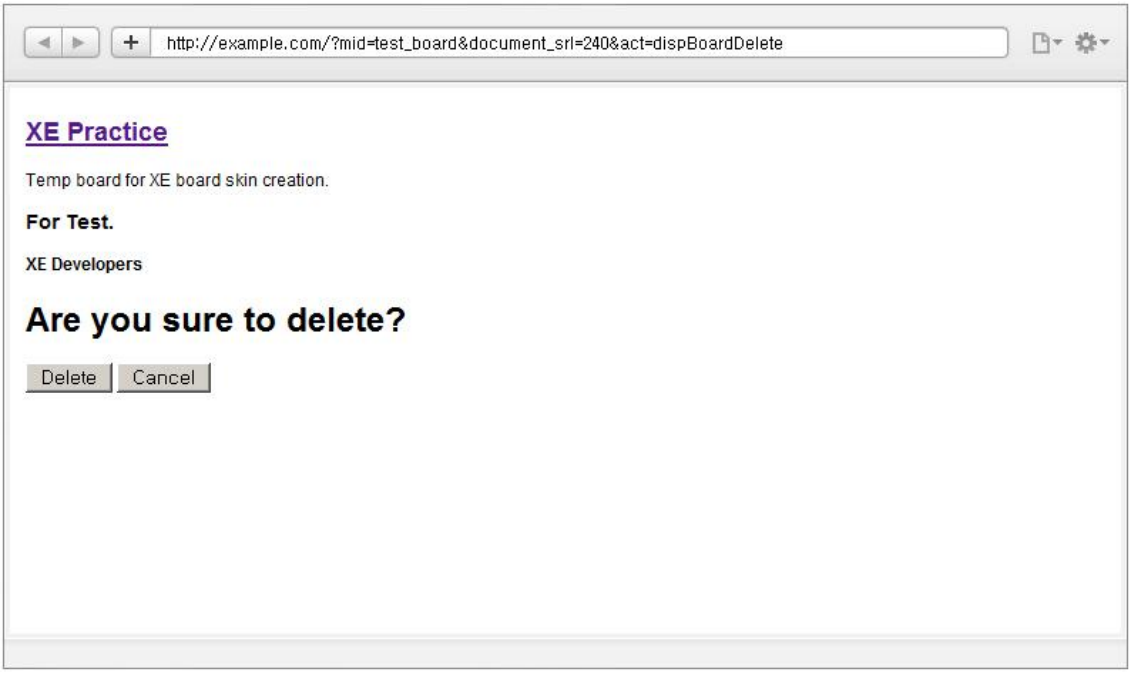
这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

4.12 制作删除页面

4.12.1 制作删除帖子页面

有权限的用户想要删除某一帖子时进行确认的页面。删帖部分在 delete_form.html文件中编写。

下图为测试用版面皮肤中的删除帖子页面效果图。



图示 4-10 删除帖子页面制作完成

测试用版面皮肤的 delete_form.html文件中，按如下代码编写了删帖页面。

```
<include target="_header.html" />
<div cond="$oDocument->isExists()" class="context_data">
    <h3 class="title">{$oDocument->getTitle()}</h3>
    <p class="author">
        <strong>{$oDocument->getNickName()}</strong>
    </p>
</div>
<form action="./" method="get" onsubmit="return procFilter(this, delete_document)"
class="context_message">
    <input type="hidden" name="mid" value="{$mid}" />
    <input type="hidden" name="page" value="{$page}" />
    <input type="hidden" name="document_srl" value="{$document_srl}" />
    <h1>{$lang->confirm_delete}</h1>
    <div class="btnArea">
        <input type="submit" value="{$lang->cmd_delete}" class="btn" />
        <button type="button" onclick="history.back()" class="btn">{$lang->cmd_cancel}</button>
    </div>
</form>
```

```
<include target="_footer.html" />
```

上述代码使用的模板语法与变量如下：

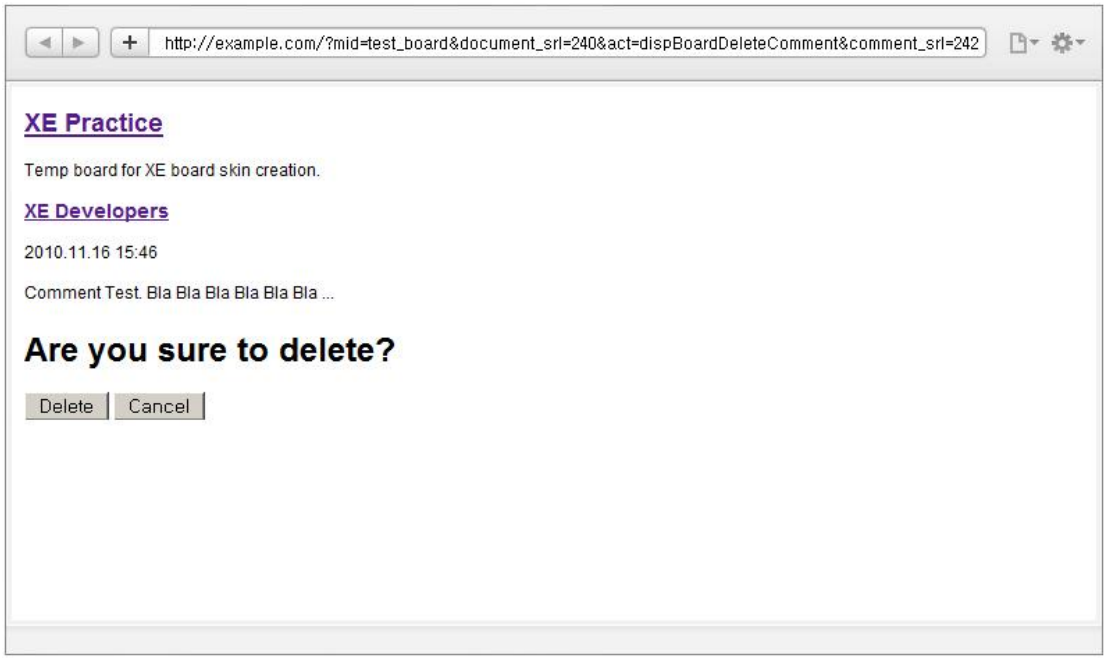
XE 模板语法 / 变量	说明
<include target="_header.html" />	引用(include) _header.html
<include target="_footer.html" />	引用(include) _footer.html
cond="$oDocument->isExists()"	有帖子原文时显示。
{$oDocument->getTitle()}	原帖标题
{$oDocument->getNickName()}	原帖作者
onsubmit="return procFilter(this, delete_document)"	检查填写内容有效性以及 form 提交
<input type="hidden" name="mid" value="{$mid}" />	提交模块 ID (hidden type)
<input type="hidden" name="page" value="{$page}" />	提交页面编号 (hidden type)
<input type="hidden" name="document_srl" value="{$document_srl}" />	提交帖子固有序号 (hidden type)
{$lang->confirm_delete}	'确定要删除吗?' 语言变量
{$lang->cmd_delete}	'删除' 语言变量
{$lang->cmd_cancel}	'权限' 语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

4.12.2 制作删除评论页面

当用户点击‘**删除**’按钮时进行确定的页面。删除评论页面在 delete_comment_form.html文件编写。

下图为测试用版面皮肤中的删除评论页面。要删除的评论包括作者、发表时间、以及评论原文显示在上方，以及**删除**与**取消**按钮在下方。



图示 4-11 删除评论页面制作完成

测试用版面皮肤的 delete_comment_form.html文件中，按如下代码编写删除评论部分。

```
<include target="_header.html" />
<div cond="$oComment->isExists()" class="context_data">
    <h3 class="author">
        <a cond="$oComment->homepage" href="{$oComment->homepage}">{$oComment-
>getNickName()}</a>
        <strong cond="!$oComment->homepage">{$oComment->getNickName()}</strong>
    </h3>
    <p class="time">{$oComment->getRegdate('Y.m.d H:i')}</p>
    {$oComment->getContent(false)}
</div>
<form action="./" method="get" onsubmit="return procFilter(this, delete_comment)"
class="context_message">
    <input type="hidden" name="mid" value="{$mid}" />
    <input type="hidden" name="page" value="{$page}" />
    <input type="hidden" name="document_srl" value="{$oComment->get('document_srl')}" />
    <input type="hidden" name="comment_srl" value="{$oComment->get('comment_srl')}" />
    <h1>{$lang->confirm_delete}</h1>
    <div class="btnArea">
        <input type="submit" value="{$lang->cmd_delete}" class="btn" />
        <button type="button" onclick="history.back()" class="btn">{$lang->cmd_cancel}</button>
    </div>
</form>
<include target="_footer.html" />
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明

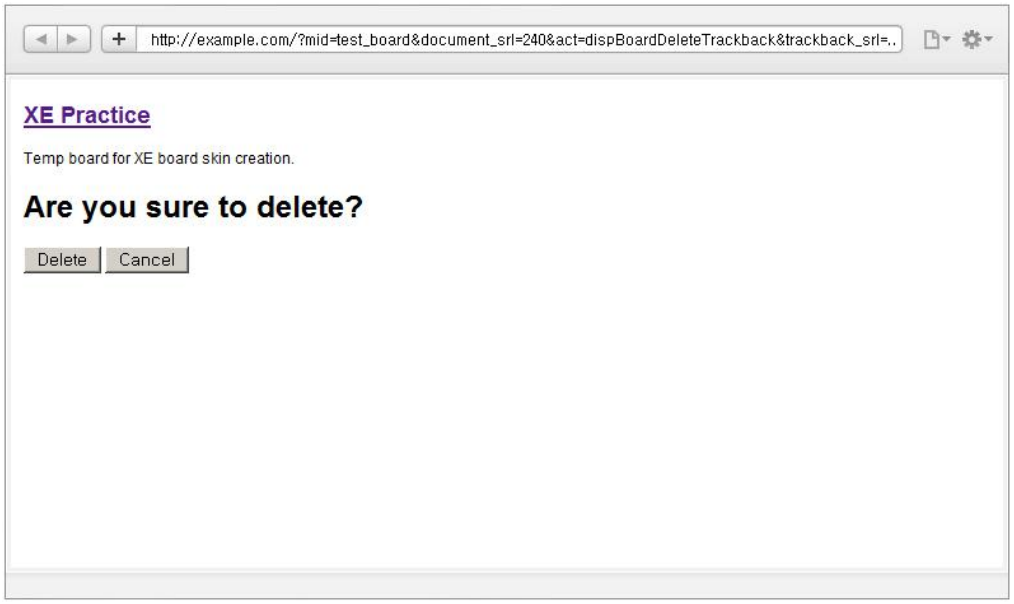
XE 模板语法 / 变量	说明
<include target="_header.html" />	引用(include) _header.html
<include target="_footer.html" />	引用(include) _footer.html
cond="$oComment->isExists()"	有原评论时显示
cond="$oComment->homepage"	原评论有评论人个人主页时显示
cond="!$oComment->homepage"	原评论无评论人个人主页时显示
{$oComment->homepage}	原评论作者的个人主页 URL
{$oComment->getNickName()}	评论人姓名
{$oComment->getRegdate('Y.m.d H:i')}	原评论发表时间
{$oComment->getContent(false)}	显示原评论正文
onsubmit="return procFilter(this, delete_comment)"	检查填写内容的有效性以及 form 提交
<input type="hidden" name="mid" value="{$mid}" />	提交模块 ID (hidden type)
<input type="hidden" name="page" value="{$page}" />	提交页面标号 (hidden type)
<input type="hidden" name="document_srl" value="{$oComment->get('document_srl')}" />	提交原评论隶属的帖子固有编号 (hidden type)
<input type="hidden" name="comment_srl" value="{$oComment->get('comment_srl')}" />	提交原评论固有编号 (hidden type)
{$lang->confirm_delete}	'确定要删除吗?' 语言变量
{$lang->cmd_delete}	'删除' 语言变量
{$lang->cmd_cancel}	'取消' 语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

4.12.3 制作删除引用页面

用户点击**删除引用**按钮时，进行确认的页面。删除引用页面编写在 delete_trackback_form.html文件。

下图为测试用版面皮肤中的删除引用页面。



图示 4-12 删除引用页面制作完成

测试用版面皮肤的 delete_trackback_form.html文件中，按如下代码编写了删除引用部分。

```
<include target="_header.html" />
<form action="./" method="get" onsubmit="return procFilter(this, delete_trackback)"
class="context_message">
    <input type="hidden" name="mid" value="{$mid}" />
    <input type="hidden" name="page" value="{$page}" />
    <input type="hidden" name="document_srl" value="{document_srl}" />
    <input type="hidden" name="trackback_srl" value="{$trackback_srl}" />
    <h1>{$lang->confirm_delete}</h1>
    <div class="btnArea">
        <input type="submit" value="{$lang->cmd_delete}" class="btn" />
        <button type="button" onclick="history.back()" class="btn">{$lang->cmd_cancel}</button>
    </div>
</form>
<include target="_footer.html" />
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
<include target="_header.html" />	引用(include) _header.html
<include target="_footer.html" />	引用(include) _footer.html
onsubmit="return procFilter(this, delete_trackback)"	检查填写内容的有效性以及 form 提交
<input type="hidden" name="mid" value="{$mid}" />	提交模块 ID (hidden type)
<input type="hidden" name="page" value="{$page}" />	提交页面编号 (hidden type)
<input type="hidden" name="document_srl" value="{document_srl}" />	提交帖子固有编号 (hidden type)

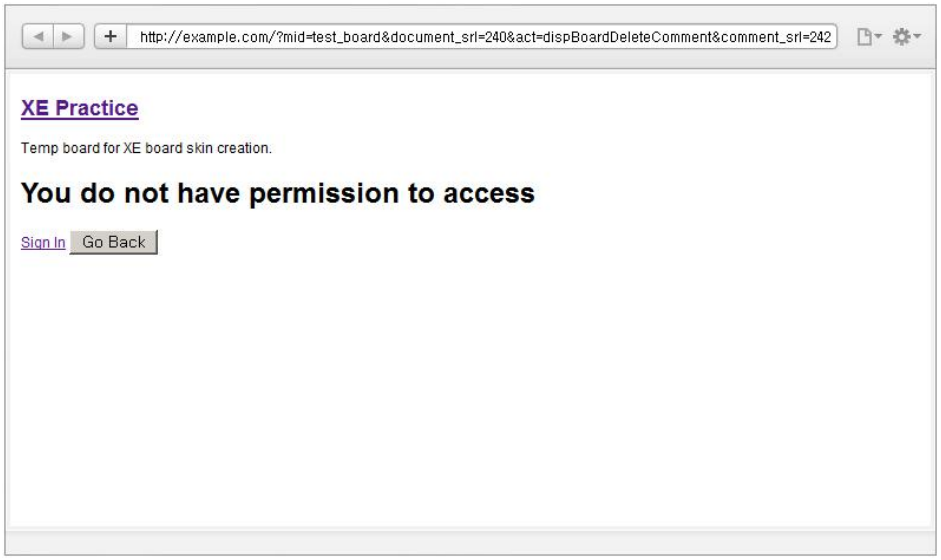
XE 模板语法 / 变量	说明
<input type="hidden" name="trackback_srl" value="{$trackback_srl}" />	提交引用的固有编号 (hidden type)
{$lang->confirm_delete}	'确定要删除吗?' 语言变量
{$lang->cmd_delete}	'删除' 语言变量
{$lang->cmd_cancel}	'取消' 语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

4.13 制作权限提示页面

权限提示页面是指当用户打开某一页面时提示用户无访问权限页面，并移动到登录页面或后退的页面。权限提示页面在 message.html 文件编写。.

下图为测试用版面皮肤中的权限提示页面效果图。



图示 4-13 权限提示页面效果图

测试用版面皮肤的 message.html文件中，按如下代码编写了权限提示页面。

```
<include target="_header.html" />
<div class="context_message">
    <h1>{$message}</h1>
    <div class="btnArea">
        <a cond="!$is_logged" href="{getUrl('act','dispMemberLoginForm')}" class="btn">{$lang->cmd_login}</a>
        <button type="button" onclick="history.back()" class="btn">{$lang->cmd_back}</button>
    </div>
</div>
<include target="_footer.html" />
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
<include target="_header.html" />	引用(include) _header.html
<include target="_footer.html" />	引用(include) _footer.html
{$message}	无访问权限时提示 ‘无访问权限’ 信息
cond="!$is_logged"	未登录时显示包含的内容
{getUrl('act','dispMemberLoginForm')}	登录页面 URL

XE 模板语法 / 变量	说明
{$lang->cmd_login}	'登录' 语言变量
{$lang->cmd_back}	'返回' 语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

4.14 制作密码输入页面

密码输入页面是指当用户打开需要用密码确认的页面时询问密码的页面。 非会员用户需要修改或删除自己发表的帖子时需要密码输入页面。 密码输入页面在 input_passowrd_form.html文件编写。

下图为测试用面部皮肤中的密码输入页面效果图

图示 4-14 输入密码页面制作完成

测试用版面皮肤的 input_passowrd_form.html文件中，按如下代码编写了密码输入页面

```
<include target="_header.html" />
<form action="./" method="get" onsubmit="return procFilter(this, input_password)" class="context_message">
    <input type="hidden" name="mid" value="{$mid}" />
    <input type="hidden" name="page" value="{$page}" />
    <input type="hidden" name="document_srl" value="{$document_srl}" />
    <input type="hidden" name="comment_srl" value="{$comment_srl}" />
    <h1>{$lang->msg_input_password}</h1>
    <input type="password" name="password" title="{$lang->password}" class="iText" />
    <input type="submit" value="{$lang->cmd_input}" class="btn" />
</form>
<include target="_footer.html" />
```

上述代码使用的模板语法与变量如下：

XE 模板语法 / 变量	说明
<include target="_header.html" />	引用(include) _header.html
<include target="_footer.html" />	引用(include) _footer.html
onsubmit="return procFilter(this, input_password)"	检查用户填写内容的有效性并传送 form.
{$lang->msg_input_password}	'请输入密码' 语言变量

XE 模板语法 / 变量	说明
<input type="hidden" name="mid" value="{$mid}" />	提交模块 ID (hidden type)
<input type="hidden" name="page" value="{$page}" />	提交页面编号 (hidden type)
<input type="hidden" name="document_srl" value="{$document_srl}" />	提交帖子固有编号 (hidden type)
<input type="hidden" name="comment_srl" value="{$comment_srl}" />	提交评论固有编号 (hidden type)
{$lang->password}	'密码' 语言变量
{$lang->cmd_input}	'输入' 语言变量

这里使用了 XE core 1.4.4 新增的模板语法。关于新模板语法的详细说明请参考 "XE 模板语言"。

4.15 应用 CSS

这里将对给版面皮肤适用CSS代码的方法进行说明，但并不会说明CSS代码的编写方法。皮肤制作者可以按照自己的意愿来修改并使用CSS代码。

1. 编写测试用版面皮肤的CSS文件。

下列代码是user_board.css文件中测试用版面皮肤使用到的类列表。 皮肤制作者可以按照自己的意愿来修改并使用CSS代码。

```
@charset "utf-8";

/* User Board */
.user_board{}

/* Board Header */
.board_header{}
.board_header h2{}
.board_header p{}

/* Form Control */
/* list.html | _read.html | write_form.html | comment_form.html */
.user_board .iText{}
.user_board .iCheck{}
.user_board textarea{}

/* Button Area */
/* list.html | write_form.html | comment_form.html | _read.html | delete_form.html |
delete_comment_form.html | delete_trackback_form.html | message.html */
.user_board .btnArea{}
.user_board .btnArea .goList{}
.user_board .btnArea .goEdit{}
.user_board .btn{}

/* Board List */
/* list.html */
.no_document{}
.board_list{}
.board_list th{}
.board_list th.title{}
.board_list tr.notice{}
.board_list td{}
.board_list td.notice{}
.board_list td.no{}
.board_list td.title{}
.board_list td.title a{}
.board_list td.title a.replyNum{}
.board_list td.title a.trackbackNum{}
```

```
.board_list td.author{}
.board_list td.time{}
.board_list td.lastReply{}
.board_list td.lastReply a{}
.board_list td.lastReply span{}
.board_list td.lastReply sub{}
.board_list td.readNum{}
.board_list td.voteNum{}
.list_footer{}
.list_footer .pagination{}
.list_footer .btnArea{}
.list_footer .board_search{}
.list_footer .board_search select{}
.list_footer .board_search option{}

/* Board Write */
/* write_form.html */
.board_write{}
.write_header{}
.write_header .iText{}
.write_header .iCheck{}
.write_header label{}
.write_editor{}
.board_write .write_author{}
.board_write .btnArea{}

/* Board Read */
/* _read.html */
.board_read{}
.read_header{}
.read_header h1{}
.read_header h1 a{}
.read_header a.author{}
.read_header strong.author{}
.read_header .sum{}
.read_header .sum .read{}
.read_header .sum .vote{}
.read_header .sum .time{}
.read_body{}
.read_body .xe_content{}
.read_footer{}
.read_footer .fileList{}
.read_footer .toggleFile{}
.read_footer .files{}
.read_footer .files li{}
.read_footer .btnArea{}

/* Feedback (Trackback+Comment) */
```

```
/* _trackback.html | _comment.html */
.feedback{}
.feedback .fbHeader{}
.feedback .fbHeader h2{}
.feedback .fbHeader .trackbackURL{}
.feedback .fbList{}
.feedback .fbItem{}
.feedback .author{}
.feedback .author a{}
.feedback .author strong{}
.feedback .time{}
.feedback .xe_content{}
.feedback .action{}
.feedback .action a{}
.feedback .pagination{}

/* Pagination */
/* list.html | _comment.html */
.user_board .pagination{}
.user_board .pagination a{}
.user_board .pagination a.prevEnd{}
.user_board .pagination a.nextEnd{}
.user_board .pagination strong{}

/* Write Author */
/* _read.html | write_form.html | comment_form.html */
.write_author{}
.write_author label{}
.write_author .iText{}
.write_author .userName{}
.write_author .userPw{}
.write_author .homePage{}

/* Write Comment */
/* _read.html | comment_form.html */
.write_comment{}
.write_comment textarea{}
.write_comment .write_author{}
.write_comment .btnArea{}

/* Context Data | Context Message */
/* comment_form.html | delete_form.html | delete_comment_form.html | input_password_form.html |
message.html */
.context_data{}
.context_data h3.author{}
.context_data h3.author a{}
.context_data h3.author strong{}
.context_data h3.title{}
```

```
.context_data p.author{}
.context_data p.author strong{}
.context_data .time{}
.context_data .xe_content{}
.context_message{}
.context_message h1{}
.context_message .iText{}
.context_message .btnArea{}
```

2. _header.html里按下例代码调用 user_board.css文件。

```
<load target="user_board.css" />
```

详细说明请参考"调用CSS 文件"。

3. 请从以下地址查看页面是否正确调用了CSS文件。下列地址中 'example.com' 是指用户设置站点时使用的域名。
   - 使用mod_rewrite时: http://example.com/test_board/
   - 未使用mod_rewrite时: http://example.com/?mid=test_board

如果画面没有反映编写的代码，请确认调用CSS文件的<load />语法或CSS文件路径是否正确。

4.16 应用 Javascript

这里将阐述将Javascript代码添加到面部皮肤的方法，但并不会说明jQuery代码的编写方法。皮肤制作者可以按照自己的意图来编写代码。

1. user_board.js文件中编写jQuery代码。

```
// <![CDATA[
jQuery(function($){
    // 在这里编写jQuery代码。
});
// ]]>
```

2. _header.html中调用 user_board.js文件，代码如下：

```
<load target="user_board.js" type="body" />
```

详细说明请参考"通过使用 index 属性可以改变CSS文件调用顺序。<load /> 与index属性可以在XE core 1.4.4 以上版本使用。

index属性值可以为正•负整数。使用负整数可以优先调用，而使用正整数可以将载入优先度滞后。指定 index="-1" 时，在其他CSS文件的上一行载入。

当CSS 中出现属性值冲突时，只有最后定义的值才会生效。因此推荐将优先度高的文件放到最后去调用。

"。

3. 请从以下地址确认Javascript代码是否正常地执行。下列地址中 'example.com' 是指用户设置站点时使用的域名。
   - 使用mod_rewrite时: http://example.com/test_board
   - 未使用mod_rewrite时: http://example.com/?mid=test_board

如果画面没用反映编写的代码，请确认调用user_board.js文件的 <load /> 语法或JS文件路径 (target)是否正确。

参考

在XE使用jQuery的基本方法请参考 "使用Javascript与 jQuery"。
要想通过使用jQuery来实现多种效果，请访问 http://jquery.com/