|  |
| --- |
| XE_표지이미지.pngasdXE 개발자 매뉴얼**开发人员指南** |
|  |  |

|  |
| --- |
| 知识产权 |

Copyright © 2011 NHN Corp. All Rights Reserved.

本文档属于NHN公司的知识财产。在任何时候，没有得到NHN(株)明示授权，不得复制、传播、发布以及变更使用本文档或其部分内容。

本文档仅以提供信息为目的。 NHN(株)已竭尽全力确保本文档所收录内容的完整性、准确性，但不保证其内容可能存在遗漏、错误，且无需对此承担责任。因此，用户就使用本文档或使用本文档所产生的后果承担全部责任，NHN(株)

对此不承担任何明示或默示保证。

包括对关联URL信息，或在本文档所提及的特定软件商品/产品的使用受当前用户所在地区或国内外相关法律法规管辖。用户对其违反法律法规所产生的一切后果负全部责任。

NHN(株)可不时对本文档作出变更而不另行通知。

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，但更加限制适用范围。

与GPL相同，遵循LGPL许可的所有软件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许可的程序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

此文档介绍了怎样开发XE 模块、插件和控件等扩展功能。其内容以XE Core 1.5.x 为基准。.

目标读者

此文档是针对想要开发XE扩展功能的开发人员而编写的。 在这里不对 web 服务器以及PHP 技术进行详细说明。Web服务器和PHP技术的信息请参考相关书籍。

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，请通过下列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 | 2011.07.29 | 发布1.0  |
| 1.1 | 2011.12.15 | 基于XE core 1.5修订 |

|  |
| --- |
| 标注规则 |

参考

|  |  |
| --- | --- |
| 참고_big.png | 参考描述读者应该参考的内容。. |

注意

|  |  |
| --- | --- |
| 주의_big.png | 注意描述读者必须了解的事项，可引发系统错误的事项，未执行时带来财产性损害的事项。. |

窗口名称 / 站点名称 / 菜单名称/ 字符段名称/选择值, 使用者输入值及 符号标记

本文档中对窗口名称、站点名称、菜单名称、字符段名称、选择值、使用者输入值 做如下标记。.

* 窗口名称: **窗口名称**
* 网站名称:**‘ Naver桌面下载’站点**
* 菜单名称: 菜单 > 子菜单
* 所选值 : **选择**NBoard 1.0.
* 使用者输入值: **输入** localhost.

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 |

[1. 理解XE core 9](#_Toc318623369)

[1.1 概要 10](#_Toc318623370)

[1.2 XE 请求周期 11](#_Toc318623371)

[1.2.1 Context 初始化 12](#_Toc318623372)

[1.2.2 模块(module)初始化 12](#_Toc318623373)

[1.2.3 执行被请求模块的操作 12](#_Toc318623374)

[1.2.4 生成回应结果 12](#_Toc318623375)

[1.3 XE 文件夹结构 13](#_Toc318623376)

[1.3.1 addons 文件夹 13](#_Toc318623377)

[1.3.2 classes 文件夹 14](#_Toc318623378)

[1.3.3 common 文件夹 15](#_Toc318623379)

[1.3.4 config 文件夹 15](#_Toc318623380)

[1.3.5 files 文件夹 16](#_Toc318623381)

[1.3.6 layouts 文件夹 18](#_Toc318623382)

[1.3.7 modules 文件夹 18](#_Toc318623383)

[1.3.8 themes 文件夹 19](#_Toc318623384)

[1.3.9 widgets 文件夹 20](#_Toc318623385)

[1.3.10 widgetstyles 文件夹 21](#_Toc318623386)

[2. XE扩展功能 23](#_Toc318623387)

[2.1 模块 24](#_Toc318623388)

[2.1.1 编写config/info.xml 24](#_Toc318623389)

[2.1.2 编写action 24](#_Toc318623390)

[2.1.3 Action Forward 的使用 26](#_Toc318623391)

[2.1.4 Trigger 的使用 27](#_Toc318623392)

[2.1.5 规则集(ruleset)的使用 27](#_Toc318623393)

[2.1.6 Form filters的使用 28](#_Toc318623394)

[2.1.7 DB query 的定义 29](#_Toc318623395)

[2.2 Add-on 31](#_Toc318623396)

[2.2.1 调用add-on的时间 31](#_Toc318623397)

[2.2.2 调用add-on时所传递的变量 32](#_Toc318623398)

[2.2.3 编写add-on文件 32](#_Toc318623399)

[2.2.4 XE XML query的使用方法 33](#_Toc318623400)

[2.2.5 生成add-on时的考虑事项 33](#_Toc318623401)

[2.3 Widget 34](#_Toc318623402)

[2.3.1 config/info.xml 的编写 34](#_Toc318623403)

[2.3.2 widget class开发 34](#_Toc318623404)

[2.3.3 扩展变量的使用 35](#_Toc318623405)

[3. 与DB联系 37](#_Toc318623406)

[3.1 概要 38](#_Toc318623407)

[3.2 XML schema language reference 39](#_Toc318623408)

[3.3 XML query 语句 41](#_Toc318623409)

[3.3.1 使用方法 41](#_Toc318623410)

[3.3.2 XML 要素 41](#_Toc318623411)

[3.3.3 XML subquery使用示例 44](#_Toc318623412)

[3.4 </query>数据类型映射 47](#_Toc318623413)

[3.5 XML Query Parser 48](#_Toc318623414)

[3.6 XE DB Class 50](#_Toc318623415)

[4. Form的使用 51](#_Toc318623416)

[4.1 概要 52](#_Toc318623417)

[4.2 编写 XE form 53](#_Toc318623418)

[4.2.1 生成 Form view 53](#_Toc318623419)

[4.2.2 添加XML ruleset文件与controller action 54](#_Toc318623420)

[4.2.3 输出问好消息 54](#_Toc318623421)

[5. Document 模块的使用 57](#_Toc318623422)

[5.1 概要 58](#_Toc318623423)

[5.2 document 模块的创建 59](#_Toc318623424)

[5.2.1 生成文档 59](#_Toc318623425)

[5.2.2 文档属性 59](#_Toc318623426)

[5.2.3 文档 URL 60](#_Toc318623427)

[5.2.4 文档的category 60](#_Toc318623428)

[5.2.5 文档修订历史 61](#_Toc318623429)

[5.2.6 文档查询 61](#_Toc318623430)

[6. API reference 63](#_Toc318623431)

[6.1 XE 全局变量 64](#_Toc318623432)

[6.2 Context Class 68](#_Toc318623433)

[6.3 Extravar Class 69](#_Toc318623434)

[6.4 Mail Class 70](#_Toc318623435)

[6.5 Object Class 72](#_Toc318623436)

[6.6 FileHandler Class 73](#_Toc318623437)

|  |
| --- |
| 表及图片目录 |

表目录

[表 1‑1 XE的文件和文件夹 13](#_Toc317081636)

[表 1‑2 addons 文件夹的结构 13](#_Toc317081637)

[表 1‑3 各classes 文件 class 14](#_Toc317081638)

[表 1‑4 common 文件夹结构 15](#_Toc317081639)

[表 1‑5 config 文件夹结构 15](#_Toc317081640)

[표 1‑6 files 文件夹结构 16](#_Toc317081641)

[表 1‑7 layouts 文件夹结构 18](#_Toc317081642)

[表 1‑8 modules 文件夹结构 18](#_Toc317081643)

[表 1‑9 themes 文件夹结构 19](#_Toc317081644)

[表 1‑10 widgets 文件夹结构 20](#_Toc317081645)

[表 1‑11 widgetstyles 文件夹结构 21](#_Toc317081646)

[表 2‑1在编写action时使用的属性 25](#_Toc317081647)

[表 2‑2 编写ruleset时 使用的元素和属性 27](#_Toc317081648)

[表 2‑3 在Form fielder 使用的属性 29](#_Toc317081649)

[表 3‑1 <table> 要素的属性 39](#_Toc317081650)

[表 3‑2 <column> 要素的属性 39](#_Toc317081651)

[表 3‑3 在XML query上使用的XML 的要素和属性 42](#_Toc317081652)

[表 3‑4 XE-DBMS 之间数据类型映射 47](#_Toc317081653)

[表 5‑1 文档属性 59](#_Toc317081654)

图目录

[图 1‑1 XE 请求周期 11](#_Toc317081655)

[图 4‑1 输入名称form 54](#_Toc317081656)

[图 4‑2 输出问好消息 55](#_Toc317081657)

# 理解XE core

本章主要介绍XEcore 的基本信息 以及XE 的请求周期和文件夹的结构。

## 概要

XE core 是 开发人员在开发指定的web 程序时的一个基准框架。 XE core 不仅可以管理会员、文档/评论， 而且提供了基于不同种类DBMS的数据管理功能。 XE 采用了MVC(Model-View-Controller) 框架， 以便呈现出完美的 SoC (Separation of Concerns )。

XE 程序的所有请求都在index.php中进行处理。这个页面担当 将请求内容初始化 并找到合适的模块 向客户（browser）回复的作用。.

XE的功能大部分都是由模块组成。当XE 收到请求时 会根据模块的名字和action的名字（没有的话使用默认值）来决定使用哪个模块。 比如显示管理者页面URL为 <root\_url>?module=admin&act=dispBoardAdminContent 。

本章节主要说明了怎样以XE core为基准， 添加模块、插件、控件等扩展功能， 以及XE的基本结构和请求周期。

## XE 请求周期

XE请求周期表现的是从访问URL 的瞬间开始到向客户发送回复的一系列的过程。以下为实现XE请求周期的图示。



图 1‑1 XE 请求周期

XE请求周期的主要过程如下.

* + - * 1. Context 初始化
				2. 模块初始化
				3. 执行被请求的模块操作
				4. 生成回应结果

开发人员可以使用插件（addon） 在这个周期的特定瞬间 执行自定义代码。 addon是XE上扩展功能中的一种利用PHP include原理的操作。. 因为其代码直接包含在核心method里 所以可以实现操作lifecycle的功能。关于addon 的详细说明 请参考"**错误！未找到引用源。** addon".

### Context 初始化

Context 初始化在 Context class 上处理。这个class 可对XE 的action 环境进行封装。主要作用如下。

* 设定$GLOBALS的 context 变量 (在display handler上使用)
* 根据语言的类型加载其语言文件
* 设置context 和session的认证(authentication)信息
* 确认在服务器上是否使用 rewrite模块
* 设置javascript的使用位置

Context class的位置为 ./classes/context/Context.class.php。

### 模块(module)初始化

模块初始化是在 ModuleHandler class的 init() method 里进行处理。 init() method 的作用如下。

* 在初始化模块之前使用addon (before\_module\_init hook)
* 根据请求的参数设置变量
* 防止XSS 的变量认证
* 以module\_srl, mid, document\_srl 的基准 搜索请求模块。
* 将当前的模块信息设定为context

ModuleHandler class 位置为 ./classes/module/ModuleHandler.class.php.

### 执行被请求模块的操作

所有的模块都是通过 ModuleHandler class的 procModule() method 来执行。这个method 的作用如下.

* 执行在模块运行之前被hook的addon (before\_module\_proc hook)
* 执行目前模块的action

### 生成回应结果

由DisplayHandler class 担任生成结果的作用。它可以根据请求的种类 输出 HTML或 XML/JSON 内容。HTML的情况，此class 会对指定的模板文件进行搜索和分析 创造出完成的HTML形式

. XML/JSON的情况 ModuleObject 属性无需其他的格式化直接串联(serialize)为 XML/JSON。

## XE 文件夹结构

安装完XE后root上会生成如下的文件和文件夹。

表 1‑1 XE的文件和文件夹

|  |  |
| --- | --- |
| 文件夹/文件 | 说明 |
| addons | 包含所有 XE里的addon |
| classes | 包含XE core基础类（class） |
| common | 包含 在所有的XE 模块上共同使用的指定文件和模版. Global 语言文件也包含在此文件夹中。 |
| config | 包含默认设置和共同的功能 |
| files | 此文件夹在安装过程中生成，用来保存上传的文件和内部cache文件，DB/设置环境文件。 |
| layouts | 包含所有默认、自定义的XE layout |
| libs | 包含在XEcore 上使用的所有library（例：ftp，tar） |
| m.layouts | Mobile layout |
| modules | 包含所有模块(XE core 模块, custom模块) |
| themes | 包含了主题（theme ）包含所有layout 和各种模块的skin) |
| widgets | 包含所有的XE控件（widget） |
| widgetstyles | 创建widget所需的所有widget style |
| index.php | 包含用于XE 所有输入输出的gateway功能 |
| .htaccess | 使用Apache Web Server 的 rewrite mod 的配置文件 |
| LICENSE | 包含XE的 license |

### addons 文件夹

addon 即可以单纯的设置为应用/非应用，需要添加设置时也可以与模块进行连接。

addons 文件夹结构如下。

表 1‑2 addons 文件夹的结构

| 文件夹/文件 | 说明 |
| --- | --- |
| addons | Addon顶层文件夹 |
|  | addon\_name | 含有addon名称的文件夹的文件夹 |
|  |  | conf | addon配置文件的文件夹 |
|  |  |  | info.xml | 编写addon 的说明和作者，版本及生成日期信息 |
|  |  | addon\_name.addon.php | addon执行代码在执行addon时插入 |
|  |  | queries | 在addon使用的queries  |
|  |  |  | queryID.xml | query文件queryschema 与在模块上使用的相同。 |

详细的内容请参考 "**错误！未找到引用源。** addon"。

### classes 文件夹

classes 文件夹中包含了XE模块、addon和widget 等组件共同使用的library class。

各文件夹所提供的class如下。

表 1‑3 各classes 文件 class

| 文件夹 | 说明 |
| --- | --- |
| cache | 包含了在XE core上可以使用的所有cache class (CacheAPC, CacheMemcache, CacheHandler) .默认class为 CacheHandler. |
| context | 包含了 Context class – 管理例如请求参数或者环境变量的context |
| db | 包含了CUBRID, MySQL, Firebird, MySQL Innodb, MySQLi, PostgreSQL, SQLite2, SQLite3 with PDO 等在 XE core可支持的所有DB. |
| display | 包含了用来输出执行结果的class (根据请求类型而不同: HTML, JSON 还有XMLRPC). |
| editor  | 包含editor handler class |
| extravar | 包含了用来处理 公告 会员等方面使用的附加变量的class |
| file | 包含了用来处理 文件和文件夹时使用的class |
| handler | 包含了(\*)Handler的抽象类 |
| httprequest | 包含了用外部服务器发送HTTP请求和接受应答时使用的class |
| mail | 包含了关于mailing的class  |
| mobile | 包含了用于mobile 优化的class |
| module | 包含了模块handler 和模块object class |
| object  | 包含了XE模块之间传递object instance的默认class |
| page | 包含了处理page搜索(navigation)的默认class |
| template | 包含了利用正规表达式将模板文件转变为PHP代码用于以后使用而创建XEcache的class |
| widget | 包含了为了执行widget的handler class |
| xml | 包含了解析和生成XML的class |

###

### common 文件夹

common 文件夹包含了XE上不可缺少的资源

表 1‑4 common 文件夹结构

| 文件夹/文件 | 说明 |
| --- | --- |
| common | 汇聚了XE上统一使用的的 JS, CSS 文件 |
|  | css | 汇聚了XE上统一使用的的 CSS文件 |
|  |  | default.css | 默认style 和在XE上特殊化style 的定义 |
|  |  | button.css | 在XE使用的默认按钮style定义 |
|  | js | 汇聚了XE上统一使用的JS文件 |
|  |  | common.js | 定义XE中多种类型的Javascript 函数 |
|  |  | jquery.js | 在XE上使用的JAVA script framework的 jQuery (http://jquery.com) 文件 |
|  |  | js\_app.js | XE上使用的JAVA script application framework的 JAF 文件 |
|  |  | x.js | cross-browser 的JAVA script library 文件 预定将会删除所以尽量不使用这个文件. |
|  |  | xml\_js\_filter.js | XE上使用的XML JS 过滤文件 |
|  | lang | 包含XE支持的所有语言文件的文件夹 |
|  | tpl | 汇聚了XE的共同的layout/template 文件 |
|  |  | common\_layout.html | 在XE上统一使用的layout |
|  |  | default\_layout.html | 没有正在使用的layout skin时 是只输出contents的空layout |
|  |  | mobile\_layout.html | 在XE mobile环境上使用的layout  |
|  |  | popup\_layout.html | 在XE上打开弹出窗口时使用的layout |
|  |  | redirect.html | 转移到其他页面时使用的template文件 |
|  |  | refresh.html | 重新修改时使用的template文件 |

### config 文件夹

config文件夹内包含的是保存默认设置内容和经常使用的各种函数的文件.

表 1‑5 config 文件夹结构

| 文件夹/文件 | 说明 |
| --- | --- |
| config.inc.php | 为了开发人员的XE版本和debug配置文件 |
| config.user.inc.php  | 为了开发人员的debug配置保存文件(开发人员自己制作使用) |
| func.inc.php | 包含XE上经常能使用的函数的文件 |

### files 文件夹

包含了cache文件、upload 文件、其他模块上需要的文件.

표 1‑6 files 文件夹结构

| 文件夹/文件 | 说明 |
| --- | --- |
| \_debug\_message.php | XE log 文件. 根据在./config/config.inc.php的\_DEBUG\_OUTPUT\_ 常数上设置的值来表示PHP error 和 DB error 消息等. 此文件基本上不存在，在发生debuglog的瞬间生成. |
|  | attach | 在附加文件（upload 文件）时使用的文件夹 |
|  |  | binaries | 保存以gif, jpg, jpeg, png, swf, mpeg以外的扩展名附加的文件(为了避免恶性的攻击 使用fpassthru()函数 不执行(execute)文件直接转达给contents client). |
|  |  | images  | 保存能够直接访问浏览器的图片和视频文件. 下属文件夹名称要与/$module\_srl/$document\_srl/$file\_name格式相同. |
|  | cache | Cache 文件夹 |
|  |  | addon | 汇聚了关于addon的cache文件 |
|  |  |  | mobileactivated\_addons.cache.php | 包含了执行活性化addon 的PHP 代码(mobile环境). |
|  |  |  | pcactivated\_addons.cache.php | 包含了执行活性化addon 的PHP 代码(PC 环境). |
|  |  | document\_category | 保存文档类型的XML， PHP cache 文件 |
|  |  | editor | 含有编辑器组件信息的cache 文件 |
|  |  | js\_filter\_compiled | XE的 XML JS filter cache文件 |
|  |  | lang\_defined | 保存用户自定义语言代码的cache文件 |
|  |  | layout | 包含了XE 布局信息的cache文件。 任何布局的修改都将保存在这里。. |
|  |  | menu | 包含了XE 菜单信息的XML、PHP cache 文件。 |
|  |  | module\_info | 保存各XE模块的信息cache文件 |
|  |  | opage | XE的外部页面模块应用的cache文件 |
|  |  | optimized | 为了将CSS 和JS 合并减少通信量并提高页面加载速度的 优化cache文件 |
|  |  | page | XE page模块的缓存文件 |
|  |  | queries | XE的 XML Query 编译缓存文件 |
|  |  | template\_compiled | XE模板缓存文件 |
|  |  | thumbnails | XE的文档缩略图 |
|  |  | widget | XE的widget信息缓存文件 |
|  |  | widget\_cache | 包含了所有已生成的widget的cache 文件。当widget的缓存时间指定时，相关的cache文件将被保存. |
|  |  | triggers | XE trigger函数的cache文件 |
|  |  | widgetstyles | 保存并应用widgetstyle 信息的cache文件 |
|  |  | newest\_news.language.cache.php | 管理者page上最新news的 临时保存文件 |
|  | config | DB, FTP等site的 最高管理者的设置信息 |
|  |  | db.config.php | DB 配置文件 |
|  |  | ftp.config.php | 保存XE安装服务器FTP信息 |
|  |  | lang\_selected.info | 保存管理者要操作的特定site的语言目录 |
|  | member\_extra\_info | 使用于会员信息附加变量的文件 |
|  |  | image\_mark | 显示在会员名称前面的标志图片 |
|  |  | image\_name | 会员图片名称的文件 |
|  |  | profile\_image | 会员登录的profile image 文件 |
|  |  | signature | 会员签名 |
|  |  | point | 会员积分 |
|  |  | new\_message\_flags | 向特定会员显示是否有新消息的临时文件位置 |
|  |  | agreement.txt | 在会员管理模块保存设置条款的文件 |
|  | ruleset | 保存动态ruleset 文件 |
|  | theme | 保存当前的主题(theme)信息. |

### layouts 文件夹

l布局（layout）是内容 （contents）和模块 （modules）的外壳。XE上的各个模块都可以设置需要应用的layout。用户在layout 管理菜单上可以通过 编辑widget 及layout的功能 来修改布局文件。

表 1‑7 layouts 文件夹结构

| 文件夹/文件 | 说明 |
| --- | --- |
| Layout Name | layout root 文件夹 |
|  | conf | 包含layout 信息的配置文件 |
|  |  | info.xml | layout制作者说明，附加变量，相关菜单的数量和名称的定义 |
|  | layout.html | layout 模板文件 |

### modules 文件夹

modules 文件夹的结构如下.

表 1‑8 modules 文件夹结构

| 文件夹/文件 | 说明 |
| --- | --- |
| module\_name | 以模块为名称的文件夹 |
|  | conf | 包括模块说明，action设置以及权限（permission）设置 |
|  |  | info.xml | 模块制作者信息和说明 |
|  |  | module.xml | 包含了关于模块运作信息的action 定义 |
|  | lang | 语言包文件 |
|  |  | en.lang.php | 英语包 |
|  | schemas | 安装模块时使用的DB table schema只有当前模块需要新建数据表时才使用该文件夹 |
|  |  | table.xml | table schema（用table名称生成文件） |
|  | queries | 用来定义使用于insert, select, update的query 的XML语法文件 |
|  | ruleset | 在模块上使用的ruleset XML文件 |
|  | tpl | 为了模块管理者视图(administrator view) 而使用的模板文件 |
|  |  | css | Style sheet |
|  |  | images | 保存模板图片 |
|  |  | js | 保存模板JS文件 |
|  |  | filter | 传达到处理文件的表格中节点和参数的声明 |
|  |  | template\_files.html | 使用XE模板语法来制作的skin文件(模块管理者画面等) |
|  | skins | 包含模块前端显示的皮肤（skin）文件 |
|  |  | Skin Name | Skin名称 |
|  |  | css | Stylesheet |
|  |  | images | Skin 图片保存 |
|  |  | js | skin JS 文件保存 |
|  |  | skin.xml | skin 制作者信息和skin的扩展变量声明 |
|  |  | template\_files.html | 用XE 模板语法 制作的skin文件 |
|  | module\_name.class.php | 包含安装、更新、删除等函数的模块基础类 |
|  | module\_name.view.php | 输出模块前端显示的view函数 |
|  | module\_name.model.php | 模块model类和函数的定义 |
|  | module\_name.controller.php | 用户界面的controller |
|  | module\_name.admin.view.php | 在显示模块后端时使用的view class和函数 |
|  | module\_name.admin.model.php | 管理者模块class和函数声明 |
|  | module\_name.admin.controller.php | 管理者函数的controller action |
|  | module\_name.api.php | 与View功能相似 – 准备画面显示数据，在输出结果中删除内部数据，能够返回JSON或XML数据，可以用在不同类型的应用上， 例如iPhone 应用 |
|  | module\_name.wap.php | 包含许多 |
|  | module\_name.smartphone.php | iphone 等智能手机的特殊class |

关于模块的详细内容请参考 "**错误！未找到引用源。** 模块".

### themes 文件夹

主题(theme)是方便的去管理layout 和模块skin的一种功能. 为了提高site设计的统一性而被使用.

表 1‑9 themes 文件夹结构

| 文件夹/文件 | 说明 |
| --- | --- |
| Theme Name | 主题root 文件夹 |
|  | conf | 有theme信息的配置文件 |
|  |  | info.xml | theme制作者和说明，对包含在theme的skin进行定义 |
|  | layouts | 布局皮肤(layout skin)的root文件夹 |
|  |  | Layout Name | layout的文件夹 |
|  |  |  | conf | 包含layout信息的配置文件 |
|  |  |  |  | info.xml | layout 制作者和说明，扩展变量，相关菜单的个数和名称的定义 |
|  |  |  | layout.html | layout 模板文件 |
|  | modules | 聚集模块skin的root文件夹 |
|  |  | Module Name | 应用于相关skin的模块名称 |
|  |  |  | css | Style sheet |
|  |  |  | images | 保存skin图片 |
|  |  |  | js | 保存skin JS 文件 |
|  |  |  | skin.xml | skin制作者信息和skin的扩展变量声明 |
|  |  |  | template\_files.html | 用XE 模板语法制作的skin文件 |

### widgets 文件夹

控件(widget)是在屏幕上显示的小程序. 其中有一些控件是与最新文章或者会员信息（登录表）相互配合使用，有一些控件与外部的 open API 相互使用。

widgets 文件夹的名称要与相应的widgets 名称相同. 文件夹结构如下.

表 1‑10 widgets 文件夹结构

| 文件夹/文件 | 说明 |
| --- | --- |
| Widget Name | 控件root 文件夹 |
|  | widget\_name.class.php | 处理widgets 的class文件数据 并指定模板文件 |
|  | conf | 配置文件夹 |
|  |  | info.xml | 保存widgets信息（名称，说明）和定义widgets class可用的变量 |
|  | skins | Skin文件夹 |
|  |  | Skin Name | 包含widgets文件.文件夹的名称要跟skin名称相同 |
|  |  |  | skin.xml | 包含skin的名称，说明，制作者，色彩方案信息的配置文件 |

### widgetstyles 文件夹

此文件夹中包含了Widgetstyle, 即控件样式。widget style 是装饰widget container 时使用的，使用者可以利用widget style 变更 widget 背景，范围，题目等widget的模样。

widget style 文件夹的结构如下.

表 1‑11 widgetstyles 文件夹结构

| 文件夹/文件 | 说明 |
| --- | --- |
| widgetstyles | widget style 文件夹 |
|  | Widgetstyle names | widget style名称 |
|  |  | widgetstyle.html | widget style 的模板文件 |
|  |  | skin.xml | 设置widget style的题目，说明，制作者，扩展变量等的文件 |
|  |  | preview.gif | 预览widgetstyle |

# XE扩展功能

本章介绍的是怎样开发XE扩展功能中的模块(Module), 插件(Addon), 控件（Widget）的方法

## 模块

XE 是一个可以通过各种各样的扩展功能来不断更新的CMS(内容管理系统). 扩展功能中最重要的是模块（即module）。 模块是添加新功能的文件的集合.

生成模块要遵守以下三种规则.

* 模块要保存在modules 文件夹下属的文件夹内。文件夹名称与模块名称相同。将生成的模块发布的话不要与其他开发人员的模块名称相冲突，模块要有唯一匹配的名称。
* 在info.xml 文件上编写 模块制作者和模块说明，版本，制作日期这样的基本信息 。
* 在module.xml 文件中保存设置的action定义等。

### 编写config/info.xml

info.xml 形式如下.

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

<module version="0.2">

 <title xml:lang="en">Module name</title>

 <description xml:lang="en">Module description </description>

 <version>1</version>

 <date>2011-05-01</date>

 <category>service</category>

 <author email\_address="author@authorland.com" link="http://www.authoria.com/">

 <name xml:lang="en">Author name</name>

 </author>

</module>

<category>标签涉及在管理者菜单上显示的模块分类。可输入的选项如下

* service: 服务管理
* member: 会员管理
* content: 信息管理
* construction: site设置
* utility: 功能设置
* accessory:附加功能设置
* system: 系统 管理/设置
* package: cafeXE, textyle 等去掉的package模块

### 编写action

XE上的所有输入输出都通过index.php来处理。action 请求参数由Module Handler决定，一般使用$act变量。模块的action在 conf/module.xml文件中声明。

<?xml version="1.0" encoding="utf-8"?>

<module>

 <grants>

 <grant name="post" default="guest">

 <title xml:lang="en">Post</title>

 </grants>

 <permissions>

 <permission action="dispForumAdminInsertForum" target="manager" />

 <permission action="dispForumAdminForumInfo" target="manager" />

 <permission action="procForumAdminInsertForum" target="manager" />

 <permission action="procForumAdminInsertListConfig" target="manager" />

 </permissions>

 <actions>

 <action name="dispForumIndex" type="view" />

 <action name="dispForumContent" type="view" index="true"/>

 <action name="dispForumNoticeList" type="view" />

 <action name="dispForumContentList" type="view" />

 <action name="dispForumContentView" type="view" />

 <action name="dispForumCatogoryList" type="view" />

 <action name="dispForumContentCommentList" type="view" />

 <action name="dispForumContentFileList" type="view" />

 <action name="procForumInsertDocument" type="controller" />

 <action name="procForumDeleteDocument" type="controller" />

 <action name="dispForumAdminContent" type="view" standalone="true" admin\_index="true" menu\_name="forum" menu\_index="true" />1

 <action name="dispForumAdminForumInfo" type="view" standalone="true" menu\_name="forum" />

 <action name="dispForumAdminExtraVars" type="view" standalone="true" menu\_name="forum" />

 <action name="dispForumAdminForumAdditionSetup" type="view" standalone="true" menu\_name="forum" />

 <action name="procForumAdminDeleteForum" type="controller" standalone="true" menu\_name="forum" ruleset="deleteForum" />

 <action name="procForumAdminInsertListConfig" type="controller" standalone="true" menu\_name="forum" ruleset="insertListConfig" />

 <action name="dispForumCategory" type="mobile" />

 <action name="getForumCommentPage" type="mobile" />

 </actions>

 <menus>

 <menu name="forum">

 <title xml:lang="en">Forum</title>

 <title xml:lang="ko">포럼</title>

 </menu>

 </menus>

</module>

<action>

在con/module.xml中使用的属性如下。

表 2‑1在编写action时使用的属性

| 属性 | 说明 |
| --- | --- |
| name | 需要包含模块名称的action名称管理者 权限的action 名称里 包含“admin” |
| type | 用来定义action是哪种类型，要保存在哪种文件（view，module，controller）中. 名字里包含“admin”的话要保存在admin view，module或者 controller PHP 文件中. |
| standalone | 这个属性设置为“true”的话 当前的操作在剩下的模块中是独立的。此属性要是设置为“false”的话 不能接受请求而且模块被执行时 输出error. 这个属性以后将被禁用. |
| index | 仅仅应用于一个操作(action)，作为模块的默认action. |
| admin\_index | 仅仅应用于一个操作(action)，作为模块后台的默认action |
| setup\_index | 使用为模块设置页面的话，只能在管理者action上进行设置。 |
| menu\_name | 包含相应操作的菜单名称 |
| menu\_index | 此属性设置为“true”的话 表示此操作为目前菜单的初期操作. |
| ruleset | 适用与相应操作的规则集(ruleset)名称 |
| action | 权限(permission)被声明的操作名称  |
| target | 可支持的权限如下.* member: 会员
* manager: 管理者
 |

### Action Forward 的使用

一般情况下action是属于XE模块的. 但是也有 一个action 被多个模块 使用的情况. 这就叫 Action Forward.

非常典型的例子是 RSS 模块. RSS action 虽然定义在帖子板块(board)模块上 但是可以在action forward 功能上调用来执行.

?mid=board&act=rss

可以使用Action Forward 和独立的功能 一起来处理模块

在上面的请求中查找叫做“board”的mid. 这个mid 要是不包含rss action的话, 在DB上通过Action Forward table 来查找已登录的rss。ss action 是以 rss 模块的 view type 登录在DB上。以由XE来 设置所有帖子版的mid 信息 生成rss模块的view 体系 来执行rss method。

这个action forward 在XE layout或者维持当前被请求的模块信息时将其他的method 作为需求时才需要。其他示例上是为了查看好友列表的communication模块的dispCommunicationFriend action. 此action 维持目前模块的layout 并用相应的contents 用好友目录来替换。

即，contents 上输出的内容可根据指定的action来改变，根据被请求的模块信息显示出不同的结果。

#### Action Forward 的登录

一般情况下Action Forward是在module.class.php上处理moduleInstall()时被保存。登录方法如下.

$oModuleController = &getController('module');

$oModuleController->insertActionForward('module', 'type(Ex:controller)', 'action\_name');

#### Action Forward 的验证

下面的方法可以确认是否登录 Action Forward。一般在module.class.php的checkUpdate()方法上使用.

$oModuleModel = &getModel('module');

if(!$oModuleModel->getActionForward('action\_name')) ...

#### Action Forward 的删除

Action Forward 不需要时，按照下面的方式删除.

$oModuleModel = &getModel('module');

$oModuleModel = &getController('module');

if($oModuleModel->getActionForward('Action Name'))

 $oModuleController->deleteActionForward('Module Name','Type','Action Name');

Action名称为 (disp|proc|get)+ModuleName+ActionName 的情况 不用登录 Action Forward.

### Trigger 的使用

#### 有些模块想做一些其他模块上的特征操作这时则使用trigger. 但是，相应的模块也要提供trigger才可以。例如我们想在论坛(forum)模块里使用已在document 模块triggerDisplayDocumentAdditionSetup上存在的admin view的时候。.

#### trigger 的使用方法如下。

#### 在DB上插入trigger

$oModuleController->insertTrigger('forum.dispForumCommentSetup', 'comment', 'view', 'triggerDispCommentAdditionSetup', 'before');

#### 获取trigger

if(!$oModuleModel->getTrigger('forum.dispForumAdditionSetup', 'document', 'view', 'triggerDispDocumentAdditionSetup', 'before')) return true;

#### 调用trigger

ModuleHandler:: triggerCall ('Trigger Name', 'call time (Called Position)', the trigger will be used as a parameter of the object);

#### 删除trigger

$ OModuleController-> deleteTrigger ('Trigger Name', 'module name', 'call the method belongs to the type of instance', 'call the method (Called Method)' + ',' call time (Called Position) ');

### 规则集(ruleset)的使用

规则集(ruleset)是将HTML表单信息传达到PHP的处理method 时,在客户端当然还有服务器端,验证其信息有效性而使用的。ruleset 保存在各模块文件夹的 ruleset 文件夹里的XML文件里。下面是ruleset 的示例。.

<?xml version="1.0" encoding="utf-8"?>

<ruleset version="1.5.0">

 <customrules>

 </customrules>

 <fields>

 <field name="user\_id" required="true" length="3:20" />

 <field name="user\_name" required="true" length="2:40" />

 <field name="nick\_name" required="true" length="2:40" />

 <field name="email\_address" required="true" length="1:200" rule="email" />

 </fields>

</ruleset>

在ruleset中使用的元素和属性如下.

表 2‑2 编写ruleset时 使用的元素和属性

| 元素 |  | 属性 | 说明 |
| --- | --- | --- | --- |
| customrules |  | 自定义规则. |
|  | rule |  | 规则 |
|  |  | name | 规则名称 |
|  |  | type | * 规则的类型可使用 "regex", "enum", "expression" 中的一种.
* 编写"regex":正规表达式时
* 在"enum":给出的值中只能选择一个时
* 当需要expression公式时
 |
|  |  | test | 自定义规则的测试代码 |
| fields |  | 用来检查有效性的fields |
|  | field |  | 检查有效性的field |
|  |  | name | Form元素名称 |
|  |  | rule | 应用的规则 |
|  |  | required="true" | 必须输入. |
|  |  | length | 长度限制，可编写“最小：最大” |
|  |  | default | 默认值. |
|  |  | equalto | 表示放入equalto 的元素值要应与当前元素值相同(密码,密码确认等). |
|  |  | modifier | 在使用规则之前 可以改变输入值或者 检查结束后可以改变结果的功能. |

详细的内容请参考 "**错误！未找到引用源。** Form的使用 ".

### Form filters的使用

过滤器(filters)是为了在HTML 内使用PHP里的处理method 来传递信息并且制定 javascript callback 函数而使用的，filters是保存在tpl/filter 文件夹内的XML文件里.下面是 form filters的示例. 从XE1.5开始 比起form filters更佳提倡使用ruleset.

<filter name="insert\_contest" module="contest" act="procContestAdminInsertContest" confirm\_msg\_code="confirm\_submit">

 <form>

 <node target="mid" required="true" maxlength="40" filter="alpha\_number" />

 <node target="browser\_title" required="true" maxlength="250" />

 </form>

 <parameter>

 <param name="contest\_name" target="mid" />

 <param name="module\_srl" target="module\_srl" />

 <param name="module\_category\_srl" target="module\_category\_srl" />

 <param name="layout\_srl" target="layout\_srl" />

 <param name="skin" target="skin" />

 <param name="browser\_title" target="browser\_title" />

 <param name="header\_text" target="header\_text" />

 <param name="footer\_text" target="footer\_text" />

 </parameter>

 <response callback\_func="completeInsertContest">

 <tag name="error" />

 <tag name="message" />

 <tag name="module" />

 <tag name="act" />

 <tag name="page" />

 <tag name="module\_srl" />

 </response>

</filter>

在 Form filters 使用的要素和属性如下.

表 2‑3 在Form fielder 使用的属性

| 要素 |  | 属性 | 说明 |
| --- | --- | --- | --- |
| form |  | 用来确认输入值是否有效的最上级元素 |
|  | node |  | 确认HTML form |
|  |  | required | 设定所输入元素是否为必需的值.required 属性值设置为true时，相应元素内不能输入值的话 发生警告. |
|  |  | filter = "filter type" | 可在filter上使用的type是 email(email\_address), userid(user\_id), url(homepage), korean, korean\_number, alpha, number, alpha\_number. |
|  |  | equalto = "target person" | equalto里的值要与当前元素值一致(密码,确认密码等). |
|  |  | maxlength | 最大长度 |
|  |  | minlength | 最小长度 |
| parameter |  | 向服务器传送时 改变form 元素的名称或者 只将 form 元素中的开发者在参数上编写的值 传送给服务器时 不使用默认使用的参数的话 则将所有form 元素 都传送给服务器. |
|  | param |  | 编写要重定义或者要传送到服务器的form元素的信息 |
|  |  | name | form元素的名称 |
|  |  | target | 重定义的元素名称 |
| response |  |  |
|  |  | callback\_func | JavaScript Callback函数必写项. |
|  | tag |  | 由callback 函数传递的变量定义 |
|  |  | name | 要传达给回调函数的变量名称。这些变量是在 controller上执行完action后 将传递给回调函数的值用$this->add('变量名', '值')来呈现。 |

更详细的内容请参考"**错误！未找到引用源。** Form的使用.

### DB query 的定义

因为XE使用的是自定义查询（Custom query）语言 所以需要对query 进行定义。 XML代码在 ./classes/xml 文件夹里的 XmlQueryParser.class.php 上解析。使用示例如下。

<query id="getCounterStatus" action="select">

<tables>

<table name="counter\_status" />

</tables>

<columns>

<column name="sum(unique\_visitor)" alias="unique\_visitor" />

<column name="sum(pageview)" alias="pageview" />

</columns>

<conditions>

<condition operation="more" column="regdate" var="start\_date" notnull="notnull"pipe="and" />

<condition operation="less" column="regdate" var="end\_date" notnull="notnull"pipe="and" />

</conditions>

</query>

## Add-on

在XE插件(add-on)要执行hooking. Hooking 是表示获取其他的正常action的行为。和其他以解释器为基准的语言一样（例如PHP）， Hooking使用‘include’ XE将add-on 作为XE Context 的内部代码来插入所以不编写成函数或者类的形式。所以从调用XE的add-on 瞬间开始 会发挥很强的的效果。但是 add-on 会给整体的XE 运行增加负载 所以要小心使用。

要生成add-on的话要遵守以下规则。

* Add-on 要保存在 addons 文件夹下属的 addon\_name 文件夹里。
* Add-on的执行文件名称必须是 addon\_name.addon.php
* info.xml 文件里要保存作者信息，add-on的说明，管理者（如果需要）还要有得到add-on的变量。

### 调用add-on的时间

调用add-on的时间如下。

* before\_module\_init: 创建模块对象之前：找到使用者请求的模块后，在创建相应模块对象之前
* before\_module\_proc: 执行模块之前：初始化模块对象后，执行相应模块之前
* after\_module\_proc: 执行完模块以后：执行创建的模块得到结果后马上
* before\_display\_content: 结果输出之前：在输出应用layout的模块结果前，为了理解各hook有哪些作用，而且为什么在XE 控制路径(control path)上 在特定时间才使用部分的add-on ，下面将举几个例子。

#### Tag目录 - After module proc

假设有一个add-on能将所有文档的tag目录在一个页面上显示。想要生成这样的tag目录，首先要获得包含当前页面的module\_srl的文档，而且之前要知道module\_srl。为此要选择一个after\_module\_proc的呼叫位置。模块信息被处理以后才能执行之前被定义的所有操作。

#### Meta tag - Before module proc

这个add-on能够将meta说明和keyword 等meta tag 插入到所有页面里。contents 在处理模块时 要在生成之前插入tag 所以将before\_module\_proc 地点作为hook来使用。

#### Point leave icon - before display content

根据特定会员积攒的积分，需要一个add-on来给各个等级的用户表示不同的图标(icon)。这个add-on 要根据已经处理的部分参数修改contents里的HTML 代码 所以将 before\_display\_content作为hook 来使用。

#### counter- Before Module Init

这个add-on是为了统计XE网站的访问流量而开发的。此add-on使用的是counter 模块。counter add-on使用$is\_logged 变量来计算访问次数。从处理模块开始不再需要获取信息，所以这个模块在时间上是使用第一次hook的before\_module\_init。

### 调用add-on时所传递的变量

以下的变量可以在4次调用时间传递给add-on

* $called\_position: 包括了调用时间信息。 值是 before\_module\_init, before\_module\_proc, after\_module\_proc, before\_display\_content 这四种中的一个。
* $addon\_path: 包括了被调用的add-on的路径。
* $addon\_info: XE的add-on可独立进行设置，相应的add-on可以指定要操作的对象模块。$addon\_info变量包含了在add-on声明的extra\_vars(info.xml 内)的信息，这个根据每个add-on而不同。

### 编写add-on文件

在addons文件夹里可以保存多样名称的文件，在文件夹内可以使用类(class)。但是由于使用的结构是用native 代码操作的include 结构 所以函数声明是不可以的。

#### config/Info.xml

info.xml 文件如下编写。

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

<addon version="0.2">

 <title xml:lang="en">Addon title</title>

 <description xml:lang="en">Addon description</description>

 <version>Addon version</version>

 <date>Year-Month-Date</date>

 <author email\_address="The email address of an author" link="The homepage address of an author">

 <name xml:lang="en">Author name</name>

 </author>

 <extra\_vars>

 <var name="Variable name" type="textarea">

 <title xml:lang="en">Variable name (for output)</title>

 <description xml:lang="en">Variable description</description>

 </var>

 </extra\_vars>

</addon>

需要时生成extra\_vars。没有细节内容时使用 "<extra\_vars />"命令语句来省略。如上所示编写的文件用 info.xml名字保存在conf 文件夹下。

#### addon\_name.addon.php

add-on要执行某种action时 用PHP形式编写add-on文件。但是， 一个add-on通常只在一个类的方法中调用，所以不能声明其内部函数。在add-on内部可以对class进行定义并使用。

Add-on文件的开头部分要如下编写

<?php

 /\*\*

 \* @file addon name.addon.php

 \* @author author name (email address)

 \* @brief description

 \*\*/

 if(!defined('\_\_ZBXE\_\_')) exit();

XE的所有功能都是通过index.php来执行，而index.php 在\_\_ZBXE\_\_常量设置为true时会开始执行。而且， XE扩展功能在执行之前检查 \_ZBXE\_常量是否设置为true。add-on的执行时间可用called\_position来确认，这个操作必须在相应的add-on上手动来执行。

举个例子，假设有个add-on在页面下端输出所有文档的标签目录。首先想要调用add-on的话要确认哪些hook是适当的。想得到文档的目录时应该像下面的代码一样对页面模块进行处理而使用'after\_module\_proc'。

<?php

if(!defined("\_\_ZBXE\_\_")) exit();

/\*\*

\* @file tag\_list.addon.php

\* @author Author (author@authorland.com)

\* @brief Description of the addon

\*\*/

if($called\_position != 'after\_module\_proc' || Context::getResponseMethod()!=='HTML') return;

$obj->module\_srl=Context::get('module\_srl');

$document\_list=executeQueryArray('addons.tag\_list.getModuleDocumentTags',$obj);

$tags='';

foreach ($document\_list->data as $val) {

 $tags=$tags.','.$val->tags;

}

$tags=explode(',', $tags);

for($i=1;$i<count($tags);$i++) {

 $tags[$i]='<a href="'.getUrl('act','TS','is\_keyword',$tags[$i]).'">'.$tags[$i].'</a>';

}

$tags=implode(' ', $tags);

$tags='<div class="tags" align="center">'.$tags.'</div>';

$content=Context::get('page\_content');

$content=$content.$tags;

Context::set('page\_content',$content);

?>

在上面的代码中，HTML 页面里插入了标签目录HTML代码.

### XE XML query的使用方法

在XE插件中其他模块生成的DB上的数据 可通过XML query来使用。这种情况下，在add-on文件夹下生成一个queries 文件夹后将定义XML query语句的XML 文件保存起来。执行query的方法如下。

$document\_list=executeQueryArray('addons.tag\_list.getModuleDocumentTags',$obj);

### 生成add-on时的考虑事项

生成add-on时应该考虑的事项如下。

* XE 的add-on会插入在所有模块的多个部分中, 所以<?php ... ?> 前后不能有空格。

有空格的话就算能够调用before\_display\_content 也不能正常运作。

* XE Core不能分别处理执行add-ons时发生的错误(exception)。因此要建立良好的检查机制以防止这种冲突发生。
* 由于add-on的代码错误导致web site上发生严重的错误时 要修改files/cache/activated\_addons.cache.php文件后重新上传。

XE add-on 可以实现很强的功能。但是编写代码不适当的话会有意想不到的结果发生或者XE会被中断。所以生成add-on时提倡参照默认的add-on。

## Widget

控件(widget)是将数据显示在画面上时使用的XE 组件。widget可以与最新帖子和会员信息这样的现有模块或者由外部API输出的数据联系起来。widget可以在所有种类的页面添加也可以在layout上直接添加。通过widget输出的内容也可以轻松定制。.

widget是管理者手动在页面模块进行输入并保存在<img/>要素里。调用要输出的web 页面时widgetController::triggerWidgetCompile() 的trigger 会使用widgetproc()编译<img/>里的代码，并转换为相应的HTML代码。

### config/info.xml 的编写

info.xml 文件是用来保存widget作者和版本，以及关于其他设置变量的信息。如下进行编写。

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

<widget version="0.2">

 <title xml:lang="en">Widget title</title>

 <description xml:lang="en">Widget description</description>

 <version>Widget version</version>

 <date>Widget creation date</date>

 <author email\_address="..." link="...">

 <name xml:lang="en">Author name</name>

 </author>

 <extra\_vars>

 <var id="extensionVariableName">

 <name xml:lang="en">Extension variable name</name>

 <type>Type of extension variable: text | textarea | select | select-multi-order | mid | mid-list | menu </type>

 </var>

 </extra\_vars>

</widget>

### widget class开发

Widget 执行的功能定义在widgetName.class.php 的类文件里。Widget的执行是由Widget 继承类WidgetHandler中的proc() method来实现的。

<?php

class myWidget extends WidgetHandler {

 function proc($args) {

 // .. Widget implementation ..

 // Template, specify the path of the skin (skin, colorset according to the value)

 $tpl\_path = sprintf('%sskins/%s', $this->widget\_path, $args-> skin);

 Context::set ('colorset', $args->colorset);

 // Template file name

 $tpl\_file = 'HTML template file except the extension ';

 // Template compilation

 $oTemplate = &TemplateHandler::getInstance();

 return $oTemplate->compile($tpl\_path, $tpl\_file);

 }

}

?>

### 扩展变量的使用

扩展变量是为了在插入widget page 之前从widget管理部分获取数据而使用的。为了在page上获得自动生成的各变量的值 可以按变量来设置输入类型。widget的扩展变量如下。

* text: 一般字符串类型
* textarea: 包含段落的字符串类型
* select: 从多个内容中选择一个
* select-multi-order: 如下所示,决定选择要素后改变顺序时使用

* mid: 只选择一个模块
* mid\_list: 选择多个模块
* menu: 在site菜单中选择一个

# 与DB联系

本章说明的是XE和DB的联系方法。

## 概要

XE中有个与数据无关的(database -agnostic)DB抽象层。意思就是XE可以与多种DBMS一起使用也可以轻松进行DBMS之间的转换.XE可以支持MySQL, MS SQL, CUBRID, PostgreSQL, SQLite3, Firebird。

为此XE使用XML schema语言(XML Schema Language)和XML query语言(XML Query Language)将所有DBschema和query 用XML来编写。

下面是 XMLschema文件的示例。

# Excerpt from ./modules/member/schemas/member.xml

<table name="member">

 <column name="member\_srl" type="number" size="11" notnull="notnull" primary\_key="primary\_key" />

 <column name="user\_id" type="varchar" size="80" notnull="notnull" unique="unique\_user\_id" />

 <column name="find\_account\_question" type="number" size="11" />

 <column name="allow\_mailing" type="char" size="1" default="Y" notnull="notnull" index="idx\_allow\_mailing" />

 <column name="limit\_date" type="date" />

 <column name="regdate" type="date" index="idx\_regdate" />

 <column name="description" type="text" />

 <column name="list\_order" type="number" size="11" notnull="notnull" index="idx\_list\_order" />

</table>

像XE一样刚开始安装时 包含的模块里有table.xml时会自动生成table。安装完XE后，安装扩展模块时要是有table.XML的话，在管理者画面上会显示**模块安装**按钮。通过XML文件可生成关于此table的query。

#./modules/member/queries/getMemberInfo.xml

<query id="getMemberInfo" action="select">

 <tables>

 <table name="member" />

 </tables>

 <columns>

 <column name="\*" />

 </columns>

 <conditions>

 <condition operation="equal" column="user\_id" var="user\_id" notnull="notnull" />

 </conditions>

</query>

将此query在PHP中调用 (如下):

 $args->user\_id = $user\_id;

 $output = executeQuery('member.getMemberInfo', $args);

## XML schema language reference

XE的DB table 模式定义为XML文件。DB table的schema保存在各模块的schemas文件夹中。

XML schema 文件是由一个root<table>要素和一个以上的子<column>要素组成。各要素的属性如下。

表 3‑1 <table> 要素的属性

|  |  |
| --- | --- |
| 属性 | 说明 |
| name | 要生成的table名称为前缀xe\_会自动添加上去不用另外保存。要与XML文件名称一致。 |

表 3‑2 <column> 要素的属性

| 属性 | 说明 |
| --- | --- |
| Name | 列(column) 名称 |
| Type | 列所要保存的数据类型值为下列中的一种。* number
* bignumber
* varchar
* char
* text
* bigtext
* date
* float

parser会将此数据类型自动映射到各个DB的数据类型上。例如，bignumber是对应于 MySQL的 bigint。对于各数据类型映射到各DB数据类型的方法详细内容请参考 "表 3‑4 XE-DBMS 之间数据类型对应 "。 |
| size | 列的大小。在数字或者字符类型上使用。* 数字类型：表示正确度。
* 字符类型：表示相应的字符串可包含的字符个数。
 |
| default | 指定列的默认值  |
| notnull | 指定该列是否能够使用null值。要是可以用列的null值可省略此属性，若不可用则如下所示添加此属性。示例) notnull = "notnull" |
| primary\_key | 指定将用为table primary key的列（column）。在各个列上指定primary\_key="primary\_key"属性的话 两个属性会捆绑在一起变为primary\_key。 |
| index | 生成列的索引。此属性的值将表示要生成的索引的名字。将一个索引名称在一个以上的列上重复使用的话会生成出结合的索引。示例) index="idx\_list\_order" |
| unique | 生成列固有的索引。此属性的值将表示要生成的索引名称。 |
| auto\_increment | 指定列值是否自动增加。示例) auto\_increment="auto\_increment" |

## XML query 语句

XE为了支持多种DB 所以将SQL query编写为XML来使用。

### 使用方法

XML query 在module和add-on，widget等方面可如下进行使用。

$args->name = "zero";

$output = executeQuery("member.getMemberInfo", $args);

executeQuery() 函数是./classes/db/DB.classs.php的 DB::executeQuery() 函数的别称去掉(alias)。此函数操作实际DB数据并根据被使用的DB 在XML query 被native SQL解析后接收其结果。

function executeQuery($xml\_query\_name, $args = null);

* 第一个参数是将要执行的XML query的名称。值为“模块名称.queryID”。
* 第二个参数是以stdClass 类型，将数据传递给query时被使用。此参数可以是null。
* 其结果会返还到Object class的结构中。
	+ $output->toBool()为 FALSE时表示请求失败， $output->toBool()为 TRUE时表示请求正常被执行。
	+ Select语句的结果 会放在 $output->data 变量里返还。

### XML 要素

<query id="query\_id" action="select|update|delete|insert">

 <tables>

 <table name="tableName" alias="alias" />

 </tables>

 <columns>

 <column name="columnName" alias="alias" />

 </columns>

 <conditions>

 <condition operation="doSomething" column="column1" var="variable" filter="filterType" default="default" notnull="notnull" minlength="minimumLength" maxlength="maximumLength" pipe="TheConcatenationOperator "/>

 <group pipe="pipe">

 <condition operation="anotherOperation" column="column" var="variable" filter="filterType" default="default" notnull="notnull" minlength ="minimumLength" maxlength="maximumLength" pipe="TheConcatenationOperator"/>

 </group>

 </conditions>

 <navigation>

 <index var="var" default="default" order="desc|asc" />

 <list\_count var="var" default="default" />

 <page\_count var="var" default="default" />

 <page var="var" default="default" />

 </navigation>

 <groups>

 <group column="GroupBy daesang" />

 </groups>

 </query>

在XML query中使用的XML 的要素和属性如下。

表 3‑3 在XML query上使用的XML 的要素和属性

| 要素 | 属性 | 说明 |
| --- | --- | --- |
| <query>  |  | query XML的顶层要素 |
|  | id | 用于query搜索的ID。使用 module.query\_id搜索并使用query XML 文件。 |
|   | action | Action有 select, update, delete, insert这四种类型。 |
|  | alias | 是使用subquery时 query语句的alias名. |
| <tables> |  | 用于query的table集合 |
| <table> |  | Table要素 |
|  | name | 源table的名称(XE的前缀可省略) |
|   | alias | 在指定列或者搜索时被使用的table别称 |
| <columns> |   | 用于query的列集合 |
| <column> |  | 列的要素 |
|   | name | 列的名称 |
|   | alias | 列别名 |
| <conditions> |   | 创建条件语句时使用，使用<group>要素可将条件语句区分为多个group.  |
| <group> ... </group> |  | 讲条件语句按group区分时使用pipe="and|or"来指定group之间的条件 |
| <condition> |  | 条件语句 |
|   | operation | 可用以下的运算符来处理。* equal : column = (var|default)
* more : column >= (var|default)
* excess : column > (var|default)
* less : column <= (var|default)
* below : column < (var|default)
* notequal : column != (var|default)
* notnull : column is not null
* null : column is null
* like\_prefix : column like '%var|default'
* like\_tail : column like 'var|default%'
* like : column like '%var|default%'
* in : column in (var|default)
* notin : column not in (var|default)
 |
|  | column | 指定列的名称。 |
|  | var | 指定executeQuery(Array)() 函数的第二个参数stdClass的key值。 |
|  | filter | Var值的条件过滤. 所支持的filter如下。* email, email\_address: mail形式
* homepage: http|https:// website 地址形式
* userid, user\_id: XE使用者id的形式(头两个字必须为字母。从第三个字符开始必须为 number+alphabet+ \_ 这样的形式。)
* number: 允许使用数字
* alpha: 允许使用字母
* alpha\_number: 数字和文字都可以使用
 |
|  | default | Var值为null时用默认值来代替。默认值可以使用一般的字符串，数字，也可以使用以下函数。* ipaddress(): IP 地址
* unixtime(): unix时间戳(php 内 time() 函数)
* curdate(): YYYYMMDDHHIISS
* plus(int count): column = column + count
* minus(int count): column = column – count
* multiply(int arg): column = column \* arg
* sequence(): 执行XE的 getNextSequence()
 |
|  | notnull | 确认不为null。指定时必须有 var值。 |
|  | minlength | 确认最小的长度 |
|  | maxlength | 确认最大的长度 |
|  | pipe | 指定and|or 条件 |
| <navigation> |   | 支持排列顺序及paging |
| <index> |   | 指定要排序的列和排序方式 |
|  | var | 将列的名称作为值的变量名称 |
|  | default  | 不能指定var值时可使用的默认排序的列名称。 |
|  | order  | 排列名称。使用非asc|desc的变量名称，按着变量值而被排列。但是，变量的值要以 asc|desc来传递(升序排列 "asc", 降序排列 "desc") |
| <list\_count> |  | 启用paging结果保存于变量中 |
|  | var | 指定页数变量 |
|  | default  | 指定默认页数 |
| <page\_count> |  | 指定显示在分页底部的页数的个数 |
|  | var  | 列表行数 |
|  | default  | 默认页数 |
| <page> |   | 指定当前页面编号 |
|  | var  | 指定当前页数 |
|  | default  | 未指定var值时使用的默认页面编号 |
| <groups> |   | 启用Group by 语句 |
|  | column | Group by的列名 |

### XML subquery使用示例

从XE 1.5 版本开始可以编写subquery。下面是subquery的各种类型的编写示例。

#### Select子句的使用

SQL 使用示例

select \*,

(select count(\*) as "count"

 from "xe\_documents" as "documents"

 where "documents"."user\_id" = "member"."user\_id"

 ) as "totaldocumentcount"

 from "xe\_member" as "member"

 where "user\_id" = 7

XML subquery编写示例

<query id="getStatistics" action="select">

 <tables>

 <table name="member" alias="member" />

 </tables>

 <columns>

 <column name="\*" />

 <query id="getMemberDocumentCount" alias="totalDocumentCount">

 <tables>

 <table name="documents" alias="documents" />

 </tables>

 <columns>

 <column name="count(\*)" alias="count" />

 </columns>

 <conditions>

 <condition operation="equal" column="documents.user\_id" default="member.user\_id" />

 </conditions>

 </query>

 </columns>

 <conditions>

 <condition operation="equal" column="user\_id" var="user\_id" notnull="notnull" />

 </conditions>

</query>

#### Where子句的使用

SQL 使用示例

SELECT \*

FROM xe\_member as member

WHERE regdate = (SELECT MAX(regdate) as regdate

 FROM xe\_documents as documents

 WHERE documents.user\_id = member.user\_id)

XML subquery编写示例

<query id="getMemberInfo" action="select">

 <tables>

 <table name="member" alias="member" />

 </tables>

 <columns>

 <column name="\*" />

 </columns>

 <conditions>

 <condition operation="equal" column="regdate" notnull="notnull">

 <query alias="documentMaxRegdate">

 <table>

 <table name="documents" alias="documents" />

 </table>

 <columns>

 <column name="max(regdate)" alias="maxregdate" />

 </columns>

 <conditions>

 <condition operation="equal" column="documents.user\_id" var="member.user\_id" notnull="notnull" />

 </conditions>

 </query>

 </condition>

 </conditions>

</query>

#### From子句的使用

SQL 使用示例

SELECT m.member\_srl, m.nickname, m.regdate, a.count

FROM (

 SELECT documents.member\_srl as member\_srl, count(\*) as count

 FROM xe\_documents as documents

 GROUP BY documents.member\_srl) a

 INNER JOIN xe\_members m on m.member\_srl = a.member\_srl

XML subquery编写示例

<query id="getMemberInfo" action="select">

 <tables>

 <table query=nfo" action="selec <table>

 <table name="documents" alias="documents" />

 </table>

 <columns>

 <column name="member\_srl" alias="member\_srl" />

 <column name="count(\*)" alias="count" />

 </columns>

 <groups>

 <group column="member\_srl" />

 </groups>

 </table>

 <table name="member" alias="m" type="inner join">

 <conditions>

 <condition operation="equal" column="m.member" default="a.member\_srl" />

 </conditions>

 </table>

 </tables>

 <columns>

 <column name="m.member\_srl" />

 <column name="m.nickname" />

 <column name="m.regdate" />

 <column name="a.count" />

 </columns>

## </query>数据类型映射

XE与各 DBMS的数据类型是如下进行对应。

表 3‑4 XE-DBMS 之间数据类型对应

| XE | MySQL | CUBRID | MS SQL |
| --- | --- | --- | --- |
| number | Bigint | integer | int |
| bignumber | Bigint | numeric(20) | bigint |
| varchar | varchar | character varying | varchar |
| char | char | character | char |
| text | text | character varying(1073741823) | text |
| bigtext | longtext | character varying(1073741823) | text |
| date | varchar(14) | character varying(14) | varchar(14) |
| float | float | float | float |
| tinytext |  | character varying(256) |  |

## XML Query Parser

XML Query Parser class接受XML query文件并parsing后生成将生成SQLquery（Query的类型 - select, update, insert, delete -, 使用的语言表达式, join以及filtering 条件，group by以及order by 语句）所有需要的信息以相关class object 的形态来包含在一个PHP文件。

这个PHP 文件作为各DB Class 的输入值来使用，各DB Class 是按着各DBMS 使用适当的转义字符和自定义的语言结构而生成SQL。

例如，假设有如下的XML query。

# ./modules/document/queries/getCategory.xml

<query id="getCategory" action="select">

 <tables>

 <table name="document\_categories" />

 </tables>

 <conditions>

 <condition operation="equal" column="category\_srl" var="category\_srl" filter="number" notnull="notnull" />

 </conditions>

</query>

使用executeQuery 函数来调用此query的话 XE会确认是否生成包含parsing结果的PHP形式的cache文件。 未调用XML Query Parser Class 则生成PHP 文件保存在./files/cache/queries 里。

# ./files/cache/queries/document.getCategory.1.5.0.8.cache.php

<?php if(!defined('\_\_ZBXE\_\_')) exit();

$query = new Query();

$query->setQueryId("getCategory");

$query->setAction("select");

$query->setPriority("");

$category\_srl1\_argument = new ConditionArgument('category\_srl', $args->category\_srl, 'equal');

$category\_srl1\_argument->checkFilter('number');

$category\_srl1\_argument->checkNotNull();

$category\_srl1\_argument->createConditionValue();

if(!$category\_srl1\_argument->isValid()) return $category\_srl1\_argument->getErrorMessage();

if($category\_srl1\_argument !== null) $category\_srl1\_argument->setColumnType('number');

$query->setColumns(array(

new StarExpression()

));

$query->setTables(array(

new Table('`testtesttest\_document\_categories`', '`document\_categories`')

));

$query->setConditions(array(

new ConditionGroup(array(

new ConditionWithArgument('`category\_srl`',$category\_srl1\_argument,"equal")))

));

$query->setGroups(array());

$query->setOrder(array());

$query->setLimit();

return $query; ?>

然后DB executeQuery method被调用并且上面文件被输出到这个方法中调用。 DB Class 是生成SQL query后再执行的。例如以上query会变成下面的SQL query。

select \* from "xe\_document\_categories" as "document\_categories" where ("category\_srl" = 15)

cache.php 文件依然包含列(column) 类型相关的信息。此信息在table schema文件中抽取。 XE首先在 ./modules/<module\_name>/schemas/<table\_name> 里找到相应的schema文件。在未找到该文件之前一直会搜索各个模块直到找到<table\_name>文件为止 。

## XE DB Class

XE对所支持的所有DBMS都提供一个自定义类(custom class)。 custom class按照各个DBMS生成custom SQL语法。

例如 与 XE core一起使用的class如下。

DB.class.php

DBMysql.class.php

DBCubrid.class.php

DBMssql.class.php

….

所有的custom class都保存在./classes/db里

所有的custom DB class都是继承共同的DB class。编写代码时使用一般的 DBclass的话，可在运行时决定 XE会使用哪种DBclass去掉‘呈现体’。

# Form的使用

本章介绍的是使用form的方法。

## 概要

Form用于将用户填写的值传送到服务器。XE在传送form时为了校验其输入值的有效性而提供ruleset功能。使用 XE ruleset 功能 无需再另编写校验数据有效性的script。

每个form通常需要下列条件

* Form markup与设计
* 传送form时调用的服务器端method

传送此类form时连接的XE组成要素如下：

* Form模板(template)文件 ：form的layout和filed 定义
* 负责传送form的controller method(controller文件)
* 校验form有效性的ruleset XML 文件

## 编写 XE form

这里我们试着做一个提交用户名并显示问候语的页面。 模块只有一个view. 该view在用户提交用户名后会显示Hello信息或重新显示填写用户名的form.

该模块的示例版本可以点击[hello.zip](http://doc.xpressengine.com/manual/hello.zip)下载。如果要按本小节说明独立制作，只需下载起步(Starter)文件([hello-tutorial.zip](http://doc.xpressengine.com/manual/hello-tutorial.zip)

### 生成 Form view

首先做一个只有输入框和提交按钮的form形式。定好文件名后 (name.html)保存在 ./modules/hello/tpl/里。

 <h1>Enter your name:</h1>

 <form id="name\_form" action="./" method="post" ruleset="say\_hello">

 <input type="hidden" name="module" value="hello" />

 <input type="hidden" name="act" value="procHelloGreet" />

 <input type="text" name="name" id="name" value="" />

 <br />

 <input type="submit" value="OK" />

 </form>

XE中提交Form表单首先要选择指定模块的指定动作(Action), 其次指定Ruleset来执行数据校验。上例中数据会被提交到hello模块的procHelloGreet动作(Action)里，并使用名为say\_hello的ruleset来进行数据校验。

|  |  |
| --- | --- |
| 참고_big.png | 参考form中ruleset属性值为ruleset文件名。相应属性前加@符号，则表示要使用动态生成的ruleset. 例如在XE1.5中可以用user\_id或email\_address两种方式登录。为了在两种登录方式分别使用数据校验，必须采用动态生成的ruleset.动态生成的ruleset文件位于files/ruleset |

生成一个输出template文件的view 方法(method). 在 ./modules/hello/hello.view.php文件中添加以下方法(method).

 /\*\*

 \* @brief Display form for entering a name

 \*\*/

 function dispHelloName() {

 $this->setTemplateFile('name');

 }

在./modules/hello/conf/module.xml文件中添加如下方法(method)，最终可以让用户来使用。

<?xml version="1.0" encoding="utf-8"?>

<module>

 <grants />

 <permissions />

 <actions>

 <action name="dispHelloName" type="view" standalone="true" index="true" />

 </actions>

</module>

现在可以访问/?module=hello来确认form如图所示:



图 4‑1 输入名称form

### 添加XML ruleset文件与controller action

目前form还不具备任何功能，以下将给form添加方法来使其可以查询用户名，并输出Hello消息。

 /\*\*

 \* Action for handling the name input form submission

 \* Retrieves the name given by the user and passes it on for displaying the greeting screen

 \*/

 function procHelloGreet(){

 $name = Context::get('name');

 $this->setRedirectUrl(getNotEncodedUrl('', 'module', 'hello', 'act', 'dispHelloName', 'name', $name));

 }

在./modules/hello/conf/module.xml的 <actions> 要素中添加如下代码。

<action name="procHelloGreet" type="controller" standalone="true" />

要检查form 内容的有效性则要添加 XML ruleset 文件。文件名取为say\_hello.xml,保存到 ./modules/hello/ruleset/ 下。

<?xml version="1.0" encoding="utf-8"?>

<ruleset version="1.5.0">

 <fields>

 <field name="name" required="true" />

 </fields>

</ruleset>

关于ruleset文件中的元素与属性说明，请参考"**错误！未找到引用源。** **错误！未找到引用源。**"。

### 输出问好消息

将./modules/hello/hello.view.php里的 dispHelloName方法(method)修改如下。

 /\*\*

 \* @brief Display form for entering a name

 \*\*/

 function dispHelloName() {

 $name = Context::get('name');

 if(isset($name)){

 $hello\_message = "Hello " . $name;

 Context::set('hello\_message', $hello\_message);

 }

 $this->setTemplateFile('name');

 }

Template文件(./modules/hello/tpl/name.html)也进行如下修改。

<h1 cond="isset($hello\_message)">{$hello\_message}</h1>

<block cond="!isset($hello\_message)">

 <h1>Enter your name:</h1>

 <form id="name\_form" action="./" method="post" ruleset="say\_hello">

 <input type="hidden" name="module" value="hello" />

 <input type="hidden" name="act" value="procHelloGreet" />

 <input type="text" name="name" id="name" value="" />

 <br />

 <input type="submit" value="OK" />

 </form>

</block>在浏览器重新加载相应页面的话可看到如下的问好消息。



图 4‑2 输出问好消息

现在form就完成了。

# Document 模块的使用

本章主要说明XE提供的document模块的使用方法。

## 概要

因为XE是模块式结构，因此可以方便地利用已做好的模块扩展XE core功能。在实现与内容(Contents)相关的功能时，document模块起着非常重要的作用。

document模块提供的功能如下：

* 生成、查询contents功能
* 评论、浏览以及其他有用的统计信息
* 修改历史
* 通过category或者tag轻松组成contents的功能
* Batch的编辑
* 与XE的其他模块容易集成

想进一步了解document模块的使用方法，请参考使用document模块来存储内容(contents)的论坛(Forum),Wiki, textile, issue tracker等模块。

## document 模块的创建

### 生成文档

文档的生成方法(method)定义在 documentController - ./modules/document/document.controller.php上。示例如下。

$obj->title = "My sample document";

$obj->content = "Hello World!";

$obj->tags = "demo, hello";

$document\_srl = getNextSequence();

$obj->document\_srl = $document\_srl;

$obj->module\_srl = $this->module\_srl;

$obj->allow\_comment = 'Y';

$obj->allow\_trackback = 'Y';

$oDocumentController = &getController('document');

$output = $oDocumentController->insertDocument($obj);

所有文档 都保存在[DBflag]\_documents table上。 除了字段以外，使用extra\_var 很容易添加所需要的custom字段。 extra\_vars是以模块instance为基准而生成。 而且，包含在此模块instance的所有文档可使用用 extra\_vars来定义的字段。

Custom 字段的名称和类型信息保存在 [DBflag]\_document\_extra\_keys table。可使用在 documentController的 insertDocumentExtraKey method 添加新的key。新key的值会保存在 [DBflag]\_document\_extra\_vars table。 可使用documentController class的 insertDocumentExtraVar方法(method)来添加新key的值。

### 文档属性

使用的文档属性如下。

表 5‑1 文档属性

| 属性 | 说明 |
| --- | --- |
| document\_srl  | 文档固有的 ID |
| module\_srl  | 文档所连接的模块实例(instance) |
| category\_srl  | 文档分类(category)的 ID. 文档category是被保存在 [DBflag]\_document\_categories table里。 |
| lang\_code  | 文档的语言代码。将同一个文档用其他语言做成多个版本时使用。 |
| is\_notice  | 在文档中做重要标记时使用的属性。例如文档目录的最上面标示公示时可使用此属性。 |
| title  | 文档标题 |
| content  | 文档内容 |
| readed\_count  | 文档阅览次数 |
| voted\_count  | 文档推荐次数。此属性是与point模块集成来呈现。 |
| blamed\_count  | 文档举报次数  |
| comment\_count  | 文档的评论个数 |
| trackback\_count  | 文档的引用个数 |
| uploaded\_count  | 文档的附加次数 |
| password  | 保密文档的使用。非会员每次要写帖子时要保存密码，在修改或删除的时候使用。保密帖子可在查看帖子时使用。 |
| user\_id, user\_name, nick\_name, member\_srl  | 关于文档所有者的信息 |
| tags  | 文档tag. 值用逗号(,)来区分保存 |
| regdate  | 文档生成日期 |
| last\_updated  | 文档最终修改的日期 |
| ipaddress  | 生成文档的使用者IP 地址 |
| comment\_status | 是否许可文档评论(ALLOW: 许可, DENY: 限制) |
| status | 文档的状态值(PRIVATE: 非公开, PUBLIC: 公开, SECRET: 保密帖子, TEMP: 临时保存) |

这些属相都表示 [DBflag]\_documents table的字段。 文档项的模块class为document.item.php。

### 文档 URL

文档可以用多种方法来访问。

首先， 用下面的结构来表示永久地址(permalink)。

http://<xe\_name>/<document\_srl>

XE的所有文档可如下所示用使用者友好的名字来访问。

http://<xe\_name>/entry/<document\_title>

文档标题太长或者包含空格的话可在管理者控制板的 信息管理 > 文档内定义文档的别称。一个文档可加一个以上的别称。用别称来访问文档时其URL结构如下。

http://<xe\_name>/entry/<alias>

除了上面内置的访问文档的方法以外，在custom模块上可以定义自己的view method。

|  |  |
| --- | --- |
| 참고_big.png | 参考以上示例需要mod\_rewrite支持。 |

### 文档的category

每个文档可以包含在类别里。 类别被包含在 [DBflag]\_document\_categories table里，可做成层次结构。基本上是非层次的。

Category是用 documentController和 documentModel class来进行管理的。documentController class 包含了与 管理 category 相关的以下method。

* insertCategory
* deleteCategory
* moveCategoryUp
* moveCategoryDown
* procDocumentMoveCategory
* updateCategory
* updateCategoryCount

documentModel class 包含了与 管理 category 相关的以下method。

* getCategory
* getCategoryChildCount
* getCategoryDocumentCount
* getCategoryHTML
* getCategoryList
* getDocumentCategories
* getCategoryTplInfo

### 文档修订历史

document 模块拥有维持文档修改历史的算法。每次当文档被修改时documentController class的 updateDocument method 会自动添加记录。

**修改历史**默认没有被开启。要使用修改历史的话要在文档部分设置页面选择**history 使用**选项。

修改历史保存在 [DBflag]\_document\_histories table里。可使用 documentModel class里的以下method来查询文档的log。

* getHistories
* getHistory

### 文档查询

在查询文档时使用的method是 documentModel的 getDocumentList。使用这个method可用下面的基准来过滤文档。

* 模块 srl
* Category
* 生成文档的会员
* 标题
* 内容
* Tag
* 类型 – 公示, 保密
* 查询次数，推荐次数等
* 生成日期，修改日期

# API reference

本章主要说明的时XE的全局变量和各类的函数。

## XE 全局变量

XE的全局变量是在XE\_ROOT/config/func.inc.php 文件里定义的。

#### debugPrint(mixed OBJECT)

debugging 函数

\_\_DEBUG\_\_的值要在 [XE\_ROOT]/config/config.inc.php 文件内定义为 1以上。

可根据 \_\_DEBUG\_OUTPUT\_\_的值来选择获得结果值的方法。

* 0: 与要输出的文件/\_debug\_message.php连接
* 1: 在HTML 用评论来说输出(回复类型为 HTML的情况)
* 2: 在Firebug console输出(PHP >= 5.2.0. Firebug/FirePHP plugin)

#### instance getController(string MODULE\_NAME)

获取模块的 Controller instance。

// If you want to get the document.controller.class instance

$oDocumentController = &getController('document');

#### instance getAdminController(string MODULE\_NAME)

获取模块的 Admin Controller instance。.

// If you want to get the documentAdminController instance

$oDocumentAdminController = &getAdminController('document');

#### instance getView(string MODULE\_NAME)

获取模块的 View instance。.

// If you want to get the rssView instance

$oRssView = &getView('rss');

#### instance getAdminView(string MODULE\_NAME)

获取模块的 Admin View instance。

// If you want to get the adminAdminView instance

$oAdminAdminView = &getAdminView('admin');

#### instance getModel(string MODULE\_NAME)

获取模块的 Model instance。

// If you want to get the documentModel instance

$oDocumentModel = &getModel('document');

#### instance getAdminModel(string MODULE\_NAME)

获取模块的 Admin Model instance。

// If you want to get the documentAdminModel instance

$oDocumentAdminModel = &getAdminModel('document');

#### instance getAPI(string MODULE\_NAME)

获取模块的 API instance。

// If you want to get the boardAPI instance

$oBoardAPI = &getAPI('board');

#### instance getWAP(string MODULE\_NAME)

获取模块的 WAP instance。

// If you want to get the boardWAP instance

$oBoardWAP = &getWAP('board');

#### instance getClass(string MODULE\_NAME)

获取模块的 class instance。

// If you want to get the documentClass instance

$oDocumentClass = &getClass('document');

#### Object executeQuery(string QUERY\_ID, stdClass PARAM)

执行XML query。结果数据返还给 Object class的 instance。 Object::toBool()为 FALSE时表示query 失败， TRUE时则表示query正常执行。

Select语句的结果数据由 Object::data 变量返还给体系。

#### Object executeQueryArray(string QUERY\_ID, stdClass PARAM)

虽然是执行与executeQuery()一样的功能，但是 Object:data 变量的结果就算是一行也用数组来返还。

#### int getNextSequence()

获取以下的sequence编号。

XE内部使用的是一个sequence ， 像member\_srl, module\_srl, document\_srl 的所有 primary\_key都是使用这个函数来设置的。即， 不是在[DBflag]\_documents table上一个一个的增加 document\_srl (auto increment)，而是使用这个sequence编号。

#### string getUrl(['',] string KEY, string VALUE [,string KEY, string VALUE ...])

生成URL

XE在当前请求的URL上将所得到的参数值更改后会返回新的URL。如果第一个参数为‘’ (空值)，则XE按给定的参数来生成新的URL。

// domain : www.example.com

// xe install path : /xe

// request url : www.example.com/xe/index.php?module=sample&act=dispSampleAct

$reset\_url = getUrl('', 'module', 'reset');

print\_r($reset\_url);

// result : /xe/index.php?module=reset

$update\_url = getUrl('module', 'update');

print\_r($update\_url);

// result : /xe/index.php?module=update&amp;act=dispSampleAct

#### string getFullUrl(['',] string KEY, string VALUE [,string KEY, string VALUE ...])

生成以http://开头的URL。

// domain : www.example.com

// xe install path : /xe

// request url : www.example.com/xe/index.php?module=sample&act=dispSampleAct

$reset\_url = getFullUrl('', 'module', 'reset', 'mid', 'samplemid');

print\_r($reset\_url);

// result : http://www.example.com/xe/index.php?module=reset&amp;mid=samplemid

#### string getNotEncodedFullUrl(['',] string KEY, string VALUE [,string KEY, string VALUE ...])

生成不可编码(Encode)的URL。与getFullUrl()的功能相同。

// domain : www.example.com

// xe install path : /xe

// request url : www.example.com/xe/index.php?module=sample&act=dispSampleAct

$reset\_url = getNotEncodedFullUrl('', 'module', 'reset', 'mid', 'samplemid');

print\_r($reset\_url);

// result : http://www.example.com/xe/index.php?module=reset&mid=samplemid

#### string getAutoEncodedUrl([‘’,], string KEY, string VALUE [,string KEY, string VALUE …])

假如遇到已编码的URL,则避免重复编码

// domain : www.example.com

// xe install path : /xe

// request url : www.example.com/xe/index.php?module=sample&act=dispSampleAct

$reset\_url = getAutoEncodedUrl('', 'name', '<script>', ‘title’, ‘&lt;title’);

print\_r($reset\_url);

// result : http://www.example.com/xe/index.php?name=&lt;script&rt;&amp;title=&lt;title

#### string getSiteUrl(string DOMAIN, ['',] string KEY, string VALUE [,string KEY, string VALUE ...])

生成虚拟站点URL. 第一个参数用于域名(Domain)或vid.。

// domain : www.example.com

// xe install path : /xe

// request url : www.example.com/xe/index.php?module=sample&act=dispSampleAct

$reset\_url = getSiteUrl('site\_id', '', 'module', 'reset');

print\_r($reset\_url);

// result : http://www.example.com/xe/index.php?module=reset&vid=site\_id

#### string getNotEncodedSiteUrl(string DOMAIN, ['',] string KEY, string VALUE[,string KEY, string VALUE...])

生成未编码的URL。与getSiteUrl()的功能相同 。

#### string getFullSiteUrl(string DOMAIN, ['',] string KEY, string VALUE [,string KEY, string VALUE ...])

在虚拟站点生成以http://开头的URL。

#### int ztime(string STR)

将YYYYMMDDHHIISS 形式的时间值修改为unix时间。

#### string getTimeGap(string DATE, string FORMAT)

将YYYYMMDDHHIISS 形式的时间值显示为与当前时间的差异（分/时）。时间差距为一天以上时则表示为在 FORMAT设置的形式。

#### string getMonthName(int MONTH, bool SHORT)

表示月份。

print\_r(getMonthName(3, true));

// result : Mar

print\_r(getMonthName(10, false));

// result : October

#### string zdate(string STR, string FORMAT, bool CONVERSION)

将YYYYMMDDHHIISS 形式的时间值更改为所要的形式。

print\_r(zdate('19830310123644', 'Y-m-d H:i:s'));

// result : 1983-03-10 12:36:44

#### string cut\_str(string STRING, int CUT\_SIZE, string TAIL)

将字符串减到特定的大小并在字符串后面添加后缀 (tail)。

print\_r(cut\_str('All roads lead to XE', 3, '...'));

// result : All…

#### string removeHackTag(string CONTENT)

将疑似恶意代码删除。

#### bool isCrawler(string AGENT)

检查登录的user agent和IP确认是不是crawler。

## Context Class

Context时接受 GET/POST的值然后将变量和多种信息转达给template 。并识别为XMLRPC, JSON, GET/POST中的对应格式。

#### Context::set(string KEY, mixed VALUE)

设置传达给template的变量。

Context::set('user\_id','user');

在template可将 {$user\_id}传达的值输出。

#### mixed Context::get(string KEY)

对传达到请求的(Request)变量或设定值进行查询。

$user\_id = Context::get('user\_id');

#### stdClass Context::gets(string KEY1 [ , string KEY2 ...])

将多个值一次性查询以后返还给 stdClass。

#### stdClass Context::getRequestVars()

由请求传递过来的变量用 stdClass来返还。

#### Context::addJsFile(string FILE\_PATH, bool OPTIMIZED ,string TARGETIE, int INDEX)

将JS文件添加到template 。只添加扩展名为js的文件。

#### Context::addCSSFile(string FILE\_PATH, bool OPTIMIZED ,string TARGETIE, int INDEX)

将CSS文件添加到template。

#### Context::addJsFliter(string FILTER\_NAME)

将XML格式的filter载入到template。

#### Context::setBrowserTitle(string TITLE)

指定HTML的标题。

#### Context::loadJavascriptPlugin(string PLUGIN\_NAME)

将Javascript plug-in 上传到template。

#### Context::addHtmlHeader(string HEAD)

在HTML的 <head>和 </head> 之间添加字符串。

## Extravar Class

Extravar class使用于一般的扩展变量和帖子版类似的模块上。

#### ExtraItem::setValue(string VALUE)

指定扩展变量的值。

#### ExtraItem::getValueHTML()

根据扩展变量的类型返回HTML形式的扩展变量值。

#### ExtraItem::getFormHTML()

根据扩展变量的类型来输出 HTML结果文件的输入form。

## Mail Class

Mail class在 XE负责邮件的传送。 XE只有在服务器设置为邮件可传送时才可以传送邮件。

#### Mail::setSender(string NAME, string EMAIL)

指定邮件的发件人。

#### Mail::getSender()

返回在Mail::setSender() 函数指定的发件人。

* 发件人用 base64来编译，发件人名称存在时返还。
* 无发件人返回空字符串(' ')。

#### Mail::setReceiptor(string NAME, string EMAIL)

指定邮件的收件人。

#### Mail::getReceiptor()

返回在Mail::setReceiptor()函数指定的收件人。

* 收件人用 base64来编译，发件人名称存在时返还。
* 无收件人返回空字符串(' ')。

#### Mail::setTitle(string TITLE)

指定邮件的标题。

#### Mail::getTitle()

返回base64编码后的邮件标题。

#### Mail::setContent(string CONTENT)

指定邮件的原文。

#### Mail::replaceResourceRealPath(mixed MATCHES)

返回邮件正文中途品德绝对路径。

#### Mail::getPlainContent()

返回文本(text)格式的邮件正文。

#### Mail::getHTMLContent()

返回HTML格式的邮件正文。

#### Mail::setContentType(string MODE)

指定邮件的原文形式。默认值为 HTML格式。

#### Mail::send()

传送邮件。在传送邮件之前要用 Mail::setSender(), Mail::setReceiptor(), Mail::setContent() 函数指定发件人，收件人，邮件原文。

#### Mail::checkMailMX(string EMAIL\_ADDRESS)

检查邮件地址是否有效。邮件地址不正确时返回false。

#### Mail::isVaildMailAddress(string EMAIL\_ADDRESS)

用正则表达式快速检查邮件地址是否有效。邮件地址有效则直接返回。

## Object Class

Object class在模块之间交换数据时使用。 模块继承自Object Class，并利用error和 message, variables 变量来交换值和状态。

#### Object::Object([int ERROR, string MESSAGE])

Object构造函数.

* ERROR: 错误代码(该值为0时出现error)
* MESSAGE: 错误消息(该值为 success时出现error)

#### bool Object::toBool()

确认Object是否错误。如果值为true，则表示该对象没有出错。

$output = executeQuery('document.insertDocument', $obj);

if(!$output->toBool()) {

 $oDB->rollback();

 return $output;

}

#### Object::add(string KEY, mixed VALUE)

将KEY为键值，将变量添加到 Object中。

#### Object::adds(stdClass OBJECT)

将收到的所有属于 stdClass的变量添加到Object中。

$oObj = new Object();

$params->key1 = "value1";

$params->key2 = "value1";

$oObj->adds($obj);

#### mixed Object::get(string KEY)

返回Object 中键值为Key的变量。

#### stdClass Object::gets(string KEY[ , string KEY , ...])

在Object变量中将key值声明为KEY的变量用 stdClass绑在一起来返还。

$obj = $oObj->gets('key1','key2','key3');

// $obj->key1, $obj->key2, $obj->key3

## FileHandler Class

该类(class)包含了处理文件夹和文件的method。

#### FileHandler::copyDir(string SOURCE\_DIR, string TARGET\_DIR [ , string FILTER] [ , string TYPE])

复制文件夹从SOURCE\_DIR到TARGET\_DIR.。

* FILTER: 使用正则表达式复制文件夹以及子文件夹和文件时，不复制重复的文件。
* TYPE: 参数值为’force’,则覆盖该文件夹所有文件。

#### FileHandler::copyFile(string SOURCE \_FILE, string TARGET\_FILE [ , string FORCE])

复制文件从SOURCE\_FILE到TARGET\_FILE。

* FORCE: 参数值为’force’,则覆盖所有重复文件。

#### string FileHandler::readFile(string FILE\_NAME)

返回文件内容。

#### FileHandler::writeFile(string FILE\_NAME, string BUFFER [, string MODE])

将BUFFER的内容写在文件内。

* FILE\_NAME: 要保存的文件
* BUFFER: 要保存的内容
* MODE: '覆写原文件。 'a'是在现有文件末尾添加内容。

#### FileHandler::makeDir(string PATH)

将PATH的文件夹和子文件夹用递归的方式来生成 。

FileHandler::makeDir(\_XE\_PATH\_ . 'files/cache/nhn/openuitech/sol');

#### FileHandler::removeDir(string PATH)

将PATH的文件夹和子文件夹用递归的方式来删除。

FileHandler::removeDir(\_XE\_PATH\_ . 'files/cache/openiuthech');

#### bool FileHandler::getRemoteFile(string URL, string TARGET\_FILE)

下载远程文件到本地。

* URL: 输入以http:// 开始的路径。
* TARGET\_FILE: 目标文件

#### bool FileHandler::createImageFile(string SOURCE\_FILE, string TARGET\_FILE ,int WIDTH, int HEIGHT, string FILE\_TYPE, string THUMBNAIL\_TYPE)

利用现有的图片文件指定大小和生成方式(维持横竖比率，剪切 )来生成缩略图。

* SOURCE\_FILE: 源图片文件
* TARGET\_FILE: 要保存的图片文件
* WIDTH: 要保存的图片宽度
* HEIGHT: 要保存的图片高度
* FILE\_TYPE: 要保存的图片类型
* THUMBNAIL\_TYPE: ratio, crop, 或者 thumbnail