上海众森计算机科技有限公司

eZ Publish 基础

- -

提交至:
所有者:
作者: 上海众森计算机科技有限公司　www.zerustech.com
编辑者: Michael Lee
版本: 1.1
日期: 2007／09／11

免责 条款

本文件只用于信息沟通目的，不能作为上海众森计算机科技有限公司对于商品，服务的质量承诺书。上海众森计算机科技有限公司对本文件的一切内容拥有最终解释权。 本文件内容如有任何改动，将不做特别通知。

文件 修改 履历

版本	修改 日	修改 人	注释
1.0	2007/04/17	Michael Lee	初稿
1.1	2007/07/31	Michael Lee	修改版权声明

目录

1 eZ Publish 基础知识

本章内容主要是介绍 eZ Publish 的基本范围、模型、结构以及建模。只有掌握了这些基础知识，才能正确的理解和运用 eZ Publish 系统。我们需要认真阅读这些章节。对术语和技术进行笔记是个良好的学习方法。你将会更加容易掌握 eZ Publish 令人吃惊的强大功能。

1.1 eZ Publish 内部结构

这部分介绍 eZ Publish 根据系统不同软件层所呈现出内部结构的概述，另外包括目录结构和注释。

1.1.1 内核、开发库和模块

eZ Publish 是使用 PHP 语言开发，针对面向对象的应用混合的开发构架。系统基本上包括三大部分。

- Modules （模块）
- Kernel (内核）
- Libraries （开发库）

1.1.1.1 开发库

（libraries）开发库可以说是系统主要的构架块，它高度支持创建的 php 类，它可以完全不依靠内核。我们在建立 eZ Publish 系统的目录中就可以很容易在根目录下找到 lib/这个文件夹。附录 A 包含了（libraries）开发库的一张列表和简介。

1.1.1.2 内核

eZ Publish 的(kernel)内核被描述成系统的核心，它兼顾着所有低层功能性函数，比如像(content handling)内容控制、(workflows)工作流、(version control)版本管理、(access control)存取控制等等。基本上(kernel)内核可以看作是一个引擎的集合，它建立于(libraries)开发库之上，并且调用它。

1.1.1.3 模块

(modules) 模块可以被看作为功能性函数的集合体。它被描述作为一个对(kernel)内核里面的引擎的接口。每个(modules)模块都包含了具有完成各个不同任务的集合。附录 B 包含包含了（modules）所有模块的一张列表和简介。

1.1.2 目录结构

eZ Publish 根目录下包含了多个子目录，每个子目录都专注管理和控制系统具体的一个部分。每个子目录都包括了所有逻辑上相关的文件。结构上十分清晰，应该很容易理解。

1.1.2.1 bin 目录

包含各种各样的 Perl 和 shell script 程序。这些脚本主要被使用作为维护目的。

1.1.2.2 design 目录

包含所有设计相关的文件比如(templates)模板、(banner/site) 图片、(stylesheets)样式表, 等。

1.1.2.3 doc 目录

包含了文档和变动日志。

1.1.2.4 kernel 目录

包含了所有核心文件，比如(classes)类、(views)视图、(dataTypes)数据类型, 等。这是整个系统的核心的地方。只有具有专业的水平才能修改这部分。

1.1.2.5 lib 目录

包含了通用(libraries)开发库。这些(libraries)开发库是执行各种各样的任务类的集合。这些(libraries)开发库被(kernel)内核所使用。

1.1.2.6 settings 目录

包含动态的, 站点的具体配置文件。

1.1.2.7 share 目录

包含静态配置文件比如代码表、位置信息和翻译文件。

1.1.2.8 var 目录

包含了(cache files)缓存文件和(logs)日志。另外, 它并且包含(imaages)图象和文件(站点具体内容) 。很明显, 这个目录(size)大小将不断增长。

1.2 内容和设计

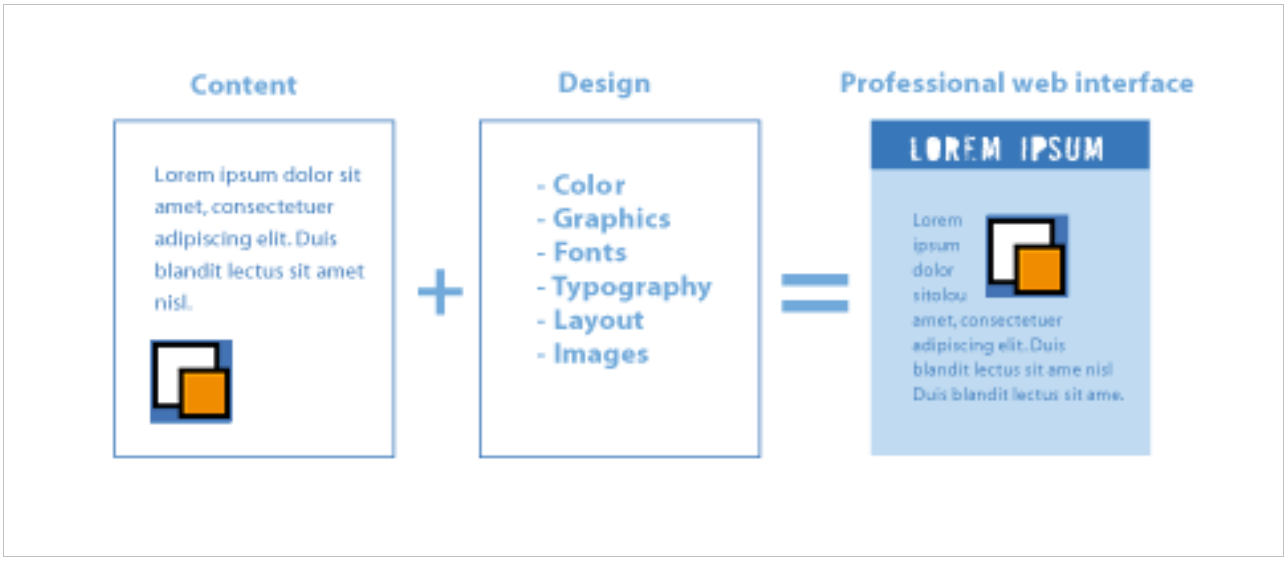
在 eZ Publish 框架当中，(content)内容和(design)设计是分离开的。

通过(content)内容我们可以将信息组织和存放在一些结构里(设为最新检索) 。然后我们可以用来建立一篇新的文章(有标题、介绍、内容, 图片), 或一个人的基本信息(名字、生日、地址, 头像) 等等。

很显然, 信息被存储在一个(content)内容的结构中必须以某种方法被显示出来, 而且最好是被人容易理解的方法。当内容包含了"(raw data)新的数据", (design)设计就是将所有的这些数据内容被显示在前端的方法。通常所说到的(design)设计, 其实的组成部分包括:(style sheets) 样式表、(banner images)图片、(layouts)布局, 等。

1.2.1 内容与设计分离

以下图例显示出一个内容(content)和设计(design)完美的分离- 和如何组合形成一个专业的界面样例。



进行分离使系统能够处理是 eZ Publish 框架当中的最重要的一个关键部分。内容的和设计分离和展开不是简单地实现。所以有如下一些要求:

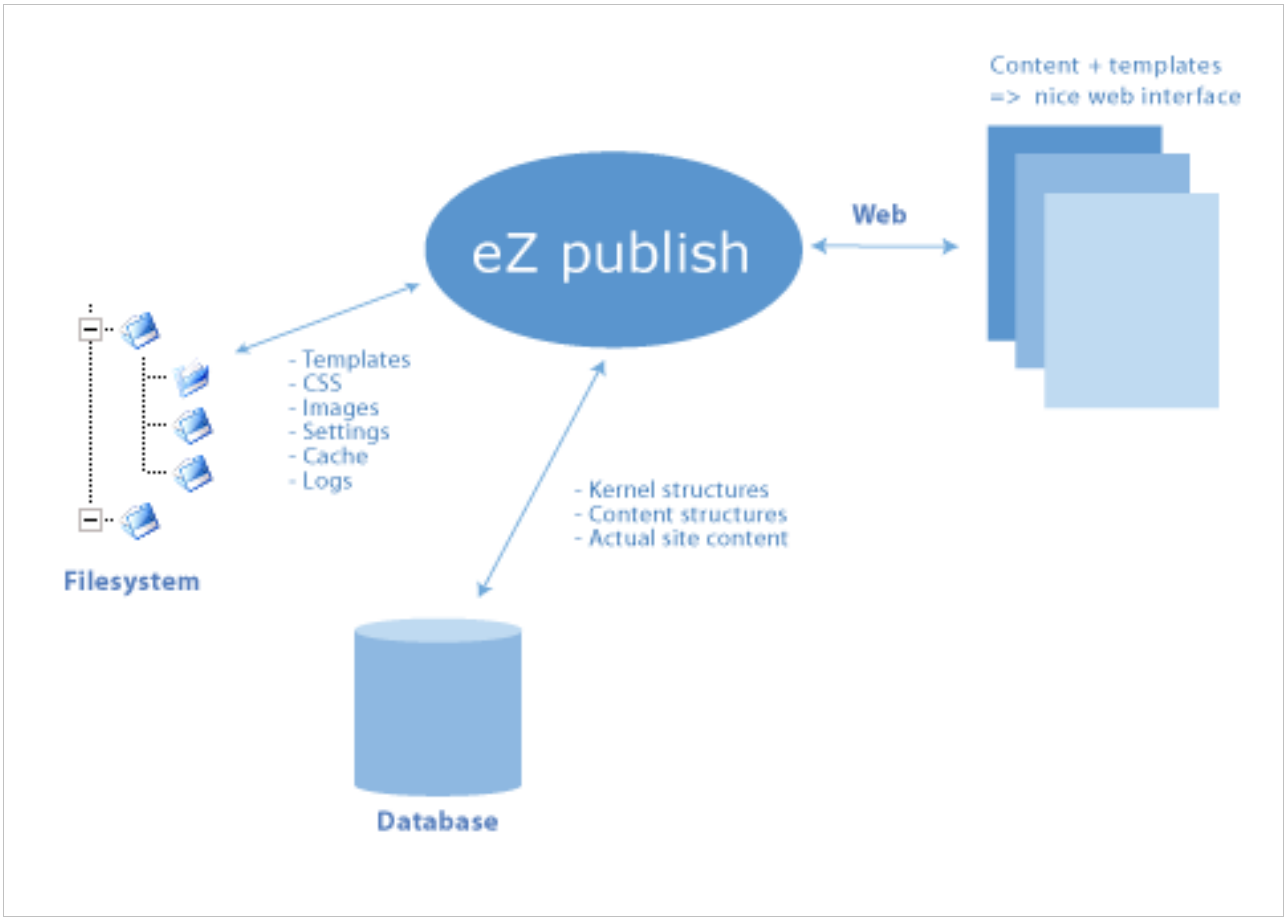
- (content)内容的开发人员和(design)设计人员能分开地工作并且没有冲突。
- (content)内容可以很容易地被发布在不同的样式上。
- (content)内容可以很容易的转移和重定向。
- 全局性的(redesigns/changes)重设计/改变，可以允许进行简单的修改。

1.2.2 模版

eZ Publish 使用(templates)模版作为站点设计的基本单位。一个模版文件是一个扩展的 HTML 文件，它可以实现特殊类型内容的外观显示。(extension)扩展是 eZ Publish 系统一个强大的功能设置，可以建立特殊的(template functions)模版函数和(operators)操作器。例如：模版可以指令页面背景是蓝色边框，网站标题条位于顶部，然后控制内容元素在页面中间。当我们访问该页面后，内容管理系统将按照(template)的要求将提取的内容放入模版指定的位置。

1.2.3 存储

eZ Publish 的结构和保存内容存储在数据库(database)里面。这是指对于所有的(content)内容除了(images)图片和（raw/binary）二进制文件。而这些则被存放在(filesystem)文件系统(在"var" 目录下) 用以增加运行速度。所有一切与设计有关(模板文件(template files), (CSS files)样式文件, 非内容(content)里的图片, 等。) 也被存放在(filesystem)文件系统中。



1.3 eZ Publish 内容管理

当创建一个文件时，无论该文件是(document)文档，(financial transaction)财务交易单，(report)报告，(memo)备忘录，(video file)影音文件，(image)图片，(purchase order)购买订单，(data back-up)数据备份或者是(customer contact)客户联系表－都被分类作为'content' 内容。一个(content management system)内容管理系统所扮演的角色就是组织每个这个'content'内容里的元素（不用去管文件的类型和复杂程度）。(content management system)内容管理系统提供了一个灵活的，自动化，结构化的解决方案。它允许信息被自由的发布和及时的更新在各种各样的通讯平台上。比如，万维网，内部网和混合的前后端布局系统。

在 eZ Publish 开发系统内，所有'content'内容的管理都通过一个图形用户界面来处理，叫作后台管理界面(administration interface)。这个界面的详细内容参看” The administration interface”（后台管理界面）章节。

这个部分描述 eZ Publish 系统怎样的发布结构并且怎样管理内容(content)的。

1.3.1 面向对象技术介绍

实际上，所谓面向对象就是参照现实事物当中的对象作为对象。比如现实生活中，人们由几个对象所围拢：家具、宠物、汽车、人等等。每个这样的对象都可以使我们有清楚辨认他们的特征。通过抽象这些特征将这些内容放入 eZ Publish 当中进行管理。

1.3.2 The content structure

不同于其他(content management system)对象内容管理系统，eZ Publish 不能使用预定义的(content model)内容模块即'one-size-fits-all'一物多用。反之，它允许站点管理员可以使用针对面向对象的方法去创建自己的独特的(content structrue)对象结构，使它更加容易构造, 存放, 检索和提供自定义数据。而不是试图设定一个适合所有数据的预定义类型和刚性结构, 假如我们要简单地创造适合我们数据的一个结构，未必需要我们每个人去自定义这个(content structrue)对象结构。这就是为什么 eZ Publish 发布了一套立即可用的结构和数据类型(datatypes)，并且还可以在结构里进行修改和扩展。换句话说, eZ Publish 提供了一个立即可以使用的对象内容结构(content structrue)。这是 eZ Publish 系统当中一个极端灵活和成功的系统特点。

1.3.3 数据类型

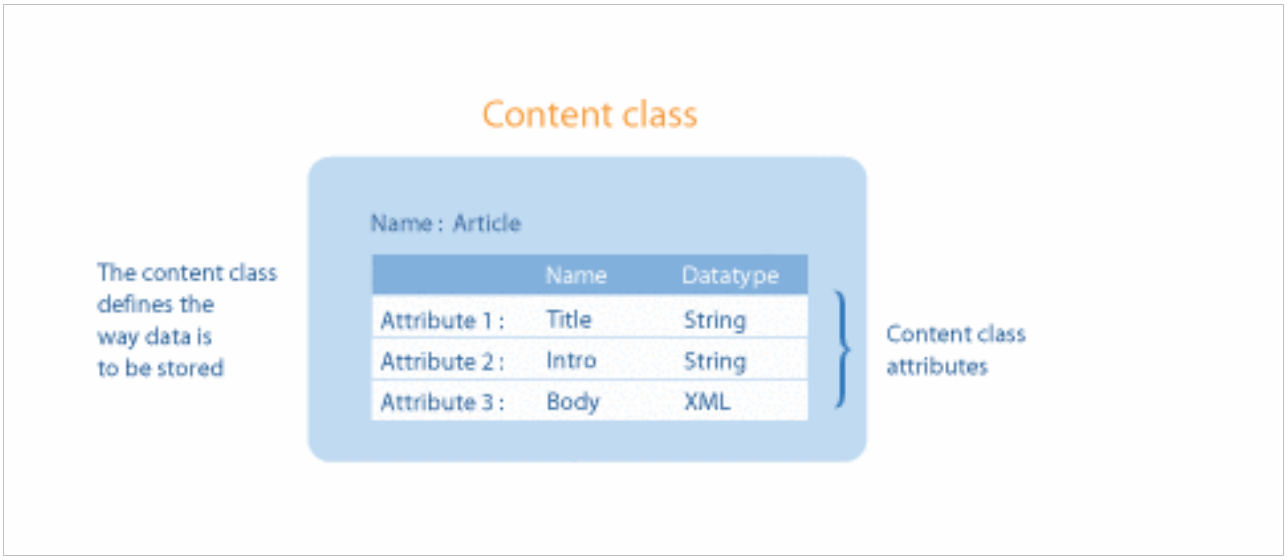
一个数据类型(datatype)是一个可以存储的最小实体。eZ Publish 正是由最基本的数据类型集合在一起，从而构建了一个强大而灵活的(content structrue)对象内容结构。另外，还可根据特殊需要创建自定义数

据类型。默认的数据类型有：字符串(text string)、XML 文本(XML text)、图片(image)、二进制文件(binary file)、价格(price)，等等。附录 C 包含了通用数据类型(datatypes) 的一张完整列表。

1.3.4 The content class

内容对象类(content class)简单地说是一种任意数据结构的定义。一个内容对象类(content class)不能存储任何实际的数据。按我们前面提到的，eZ Publish 提供了一套预定义，立即可用的内容对象类(content class)，比如：“folder“、“article”、“user account”、“image”等等类。这些(content class)内容对象类被设计用来满足一般通用/日常(generic/everyday)**CMS** 的工作任务。另外，可以通过使用后台管理界面(administration interface)创建自定义的(content class)内容对象类。(content class)内容对象类中无论是自定义还是内置(both custom and built-in)都可以很容易地随时进行修改和扩展。附录 D 包含了内容对象类(content class)的一张完整列表。

1.3.4.1 Content class attributes



每个(content class)内容对象类都是由一到多个(attribute)属性构成，因此属性的集合可以定义一个内容对象类(content class)的实际数据结构。一个(attribute)属性可能是(text string),(integer),(float),(image)等等数据类型，属性的类型由被选择为那个具体属性的(datatype)数据类型来确定。换句话说：每个(content class)内容对象类都是由一个到多个(attribute)属性组成并且这些(attribute)属性都有指定的数据类型(datatype)。内容对象类(content class)可以通过后台管理界面(administration interface)很容易的进行创建和修改。

例子：

假设，我们想存储关于水果的一些信息。我们希望存储的信息是：水果的名称、重量和颜色。我们可以很轻松的创建一个叫“Fruit”的内容对象类(content class)，它有三个属性：名称(name)，重量(weight)和颜色(color)。把 name 和 color 属性(attribute)设置成(string)字符串，weight 属性(attribute)设置成整数(integer)：

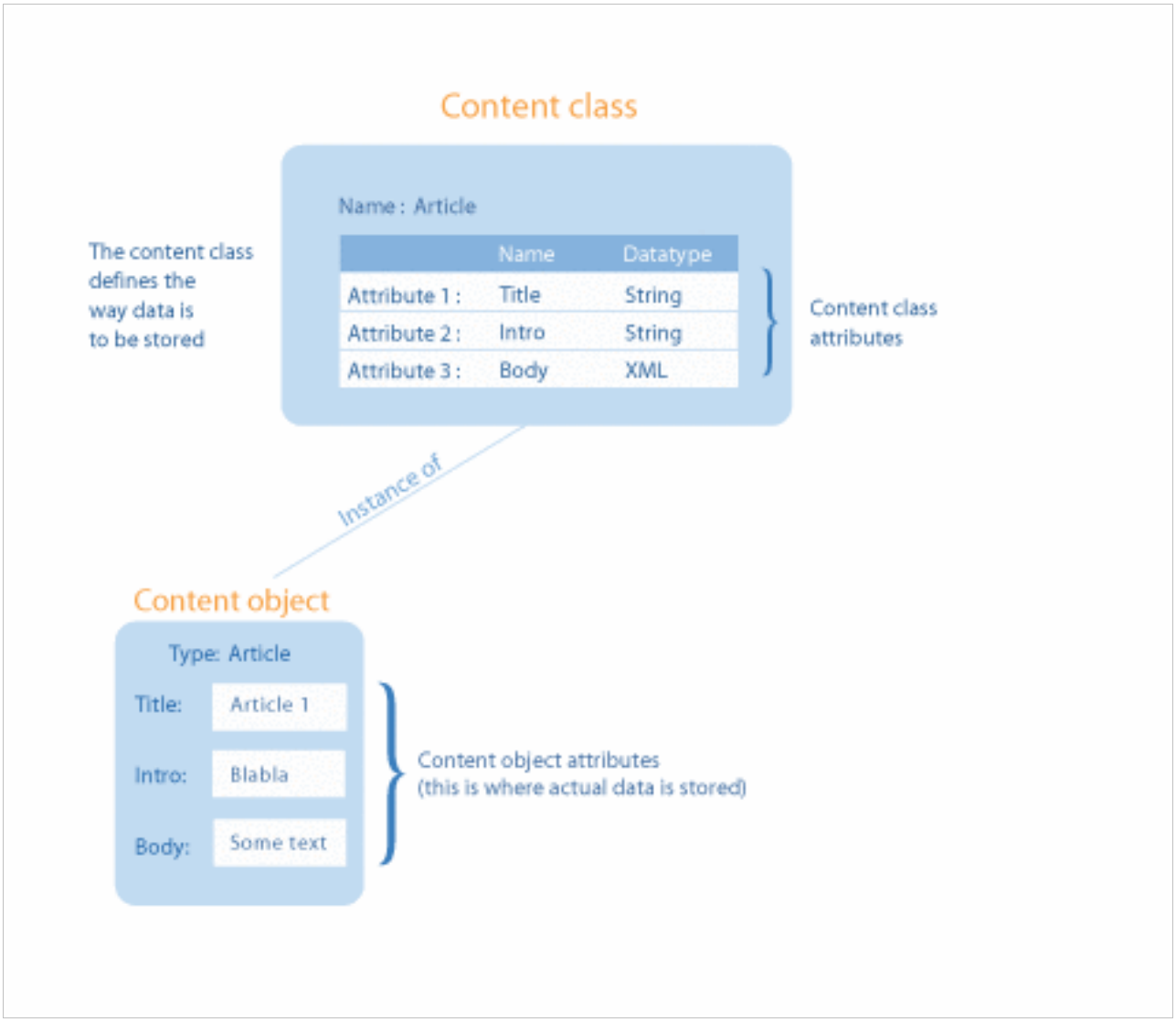
```
Content class: Fruit
Attribute name Datatype
---------------------------------------
Attribute 1: Name    Text
Attribute 2: Weight  Integer
Attribute 3: Color   Text
```

1.3.5 The content object

每一个(content object)内容对象就是一个(content class)内容对象类的实例化对象。(content class)内容对象类简单的定义了(content objects)内容对象的结构和使用的类属性(content class attribute)。每个内容对象(content object)都包含有实际的数据/信息(data/information)。一旦内容对象类(content class)被定义好后，就可以创建出几个这样类型的内容对象(content objects)。同一类型的多个对象都用相同的数据/信息(data/informaiion)类型进行编辑和存储，例如：使用多个文章对象(atricle object)存放不同的新闻文章(news atricles)。

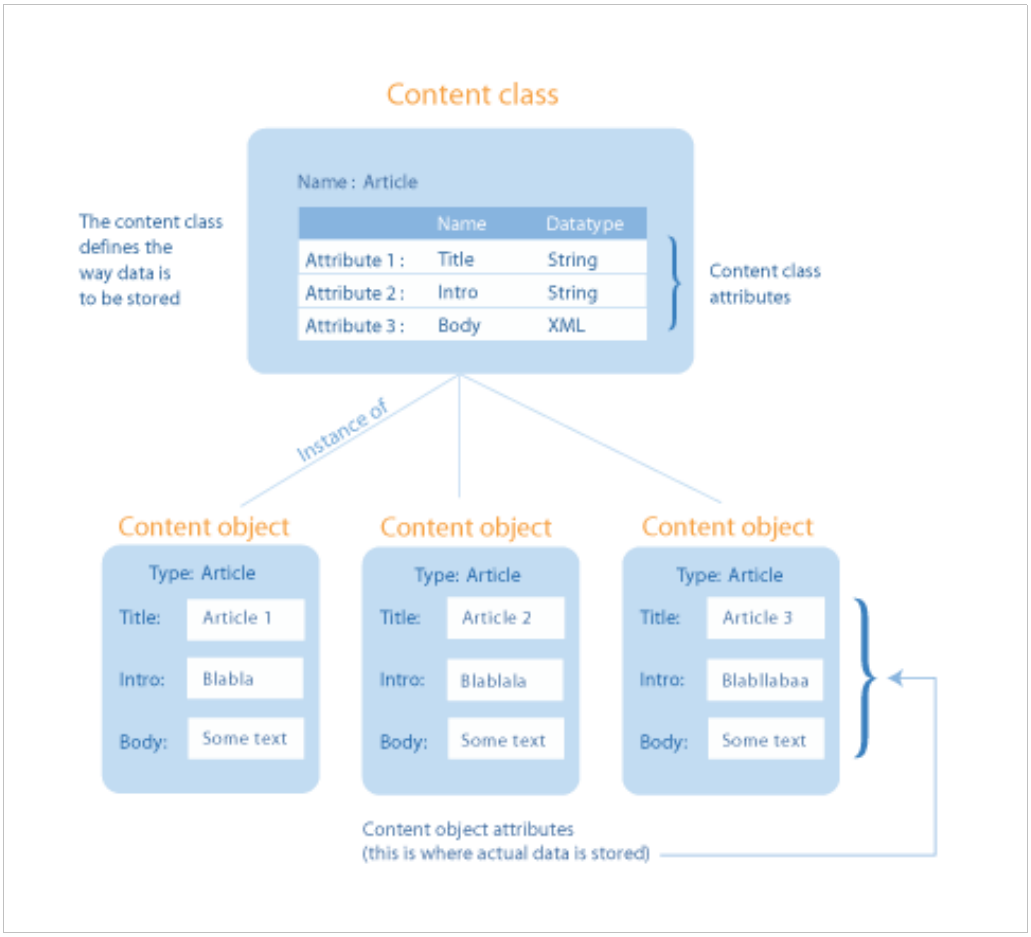
1.3.5.1 Content object attributes

每个内容对象(content object)都包含几个独立的属性(attribute)。这些属性(attribute)由(content class attribute)内容对象类属性来定义。在一个(content object)内容对象中，所有数据都被浓缩在一个或多个对象属性(content object attributes)当中。



1.3.6 数据类型，类和对象之间的关系

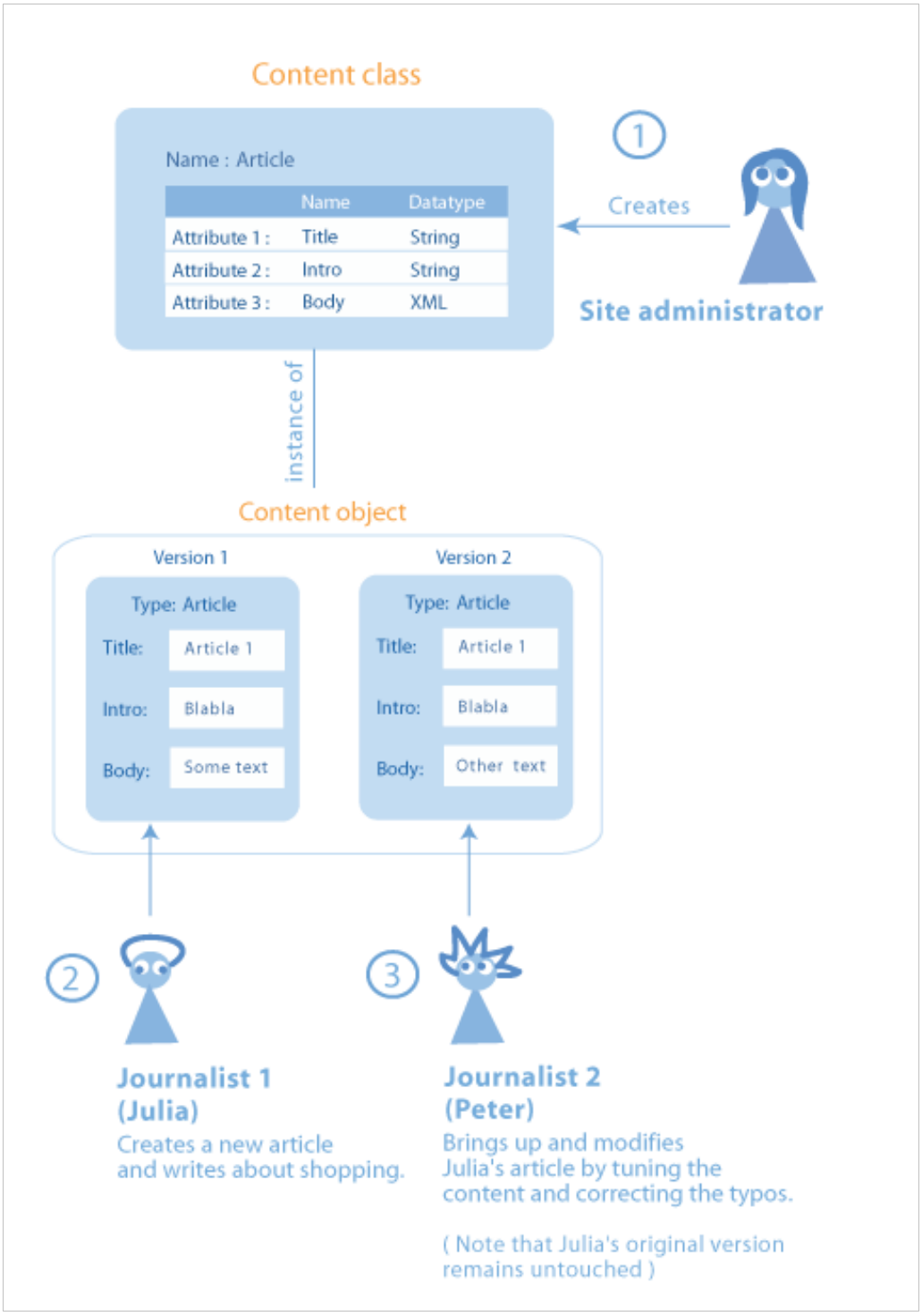
以下图表总结和说明了数据类型，类和对象之间的关系(datatypes,classes,object)。(content objects)内容对象是(content class)内容对象类的实例；那基本上意味着他们是相同类型的，但他们包含各自的内容/数据/信息(conent /data/information)。



1.3.7 Content object versioning

被存放在(content object)内容对象中的实际内容可能存在一个或多个版本。每次对内容的修改，都会创建一个新的版本，而老的版本依然保留。eZ Publish 系统就是通过这样的方法保留不同用户修改的记录。版本系统允许用户使用恢复/取消(revert/undo)功能等等。为了预防数据库充满老的和未使用的版本(old/non-used versions)，管理员可以对版本进行限制，就是基于(content class)内容对象类设置一个最大的版本号数字。

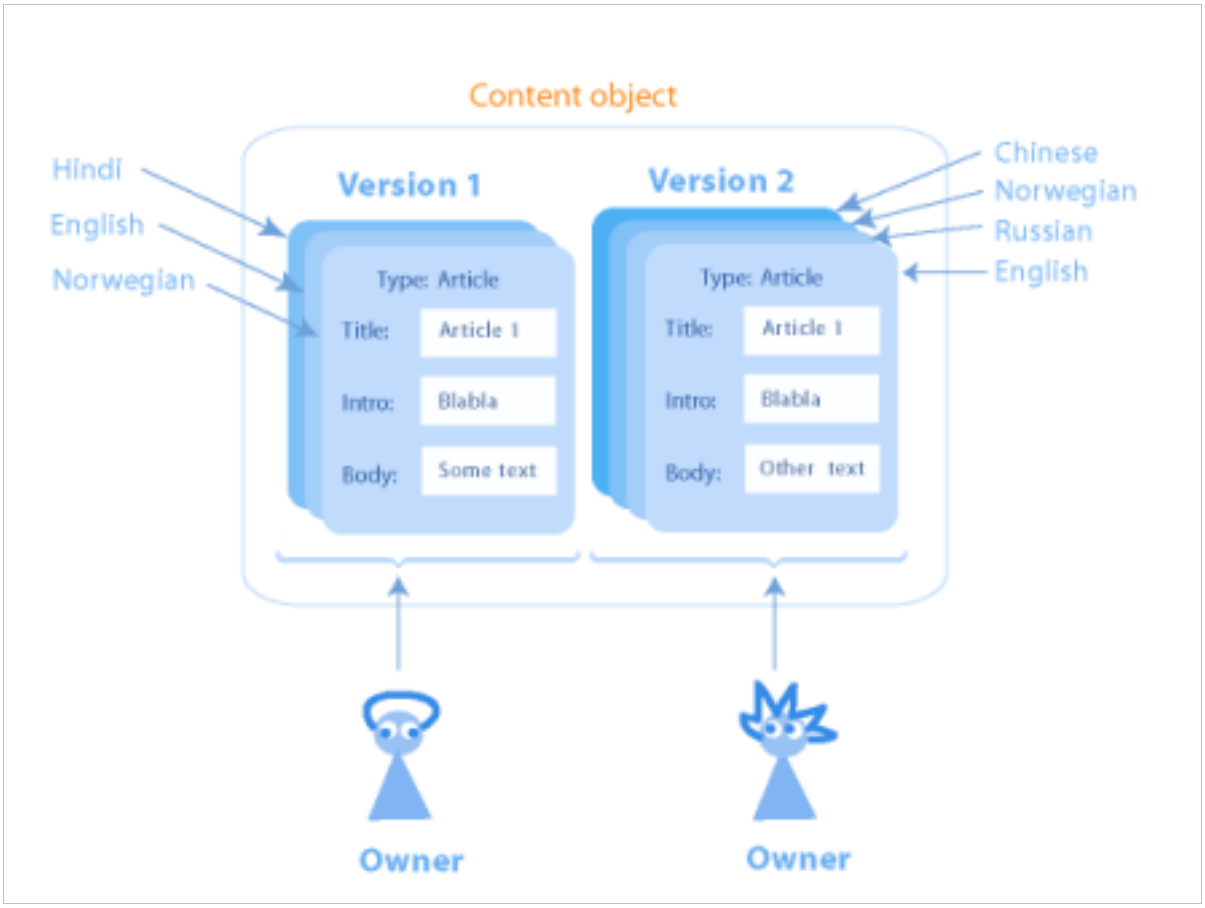
每个内容对象(content object)的版本可以属于不同的用户。但只能有一个版本是已经被成功发布的(published)，这被称为当前版本。使用新闻编辑人员和发布文章为例，以下图表提供了一个概略的例证说明(versioning) 版本系统是如何运作的。


Content class
Name : Article
Name
Datatype
Attribute 1 :
Title
String
Attribute 2 :
Intro
String
Attribute 3 :
Body
XML
1
Creates
Site administrator
instance of
Content object
Version 1
Type: Article
Title:
Article 1
Intro:
Blabla
Body:
Some text
Version 2
Type: Article
Title:
Article 1
Intro:
Blabla
Body:
Other text
2
Journalist 1
(Julia)
Creates a new article
and writes about shopping.
3
Journalist 2
(Peter)
Brings up and modifies
Julia's article by tuning the
content and correcting the typos.
(Note that Julia's original version
remains untouched)

内置的(versioning system)版本管理系统是个十分方便的工具，可以用来对版本进行管理任何不同的内容(不管类型/外观/结构/等等) 。它基本上提供了一种通用的,现成的版本控制框架。

1.3.8 支持多种语言系统

除了版本管理系统(versioning system)之外，eZ Publish 系统还提供了一种强大的支持多种语言的解决方案。你可以把(content object)内容对象想像成为一个三维空间，而每个任意内容的版本可能存在几个不同语言。这是一个十分巧妙的比喻。它基本上提供了一个通用的，现成的支持多语言的框架，可以能够被各种(content)内容所使用。如下图说明建立在(versioning system)版本管理系统中怎样解决多语言系统支持的方案。



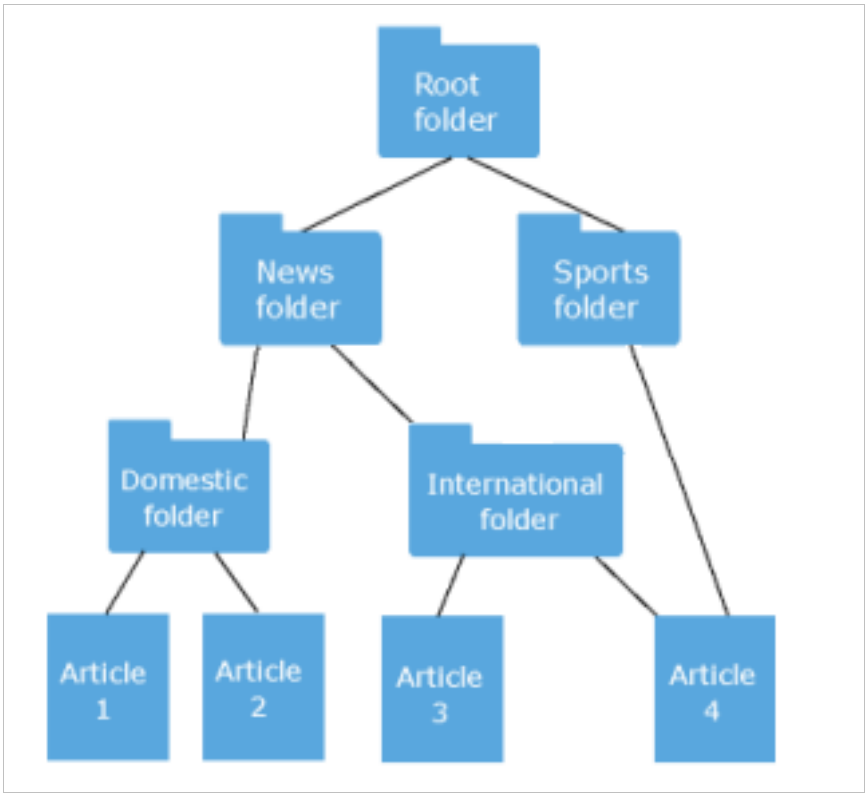
1.3.9 Objects ,node and the content node tree

一个(content object)内容对象就是一个(content class)内容对象类的实例化对象。当系统运行时，新的

(content object)内容对象同时也被创建。例如，构成一篇新的文章，需要建立一个新的(content object)内容对象用以存储实际的文章内容(这个对象同时还要兼顾到该文章不同的版本和语言的支持)。显然地，内容对象(content objects)也不是随便地就放置在一个地方，它们必须通过某种方式进行组织和构造。这就引入了节点(nodes)和节点树(content node tree)的概念。

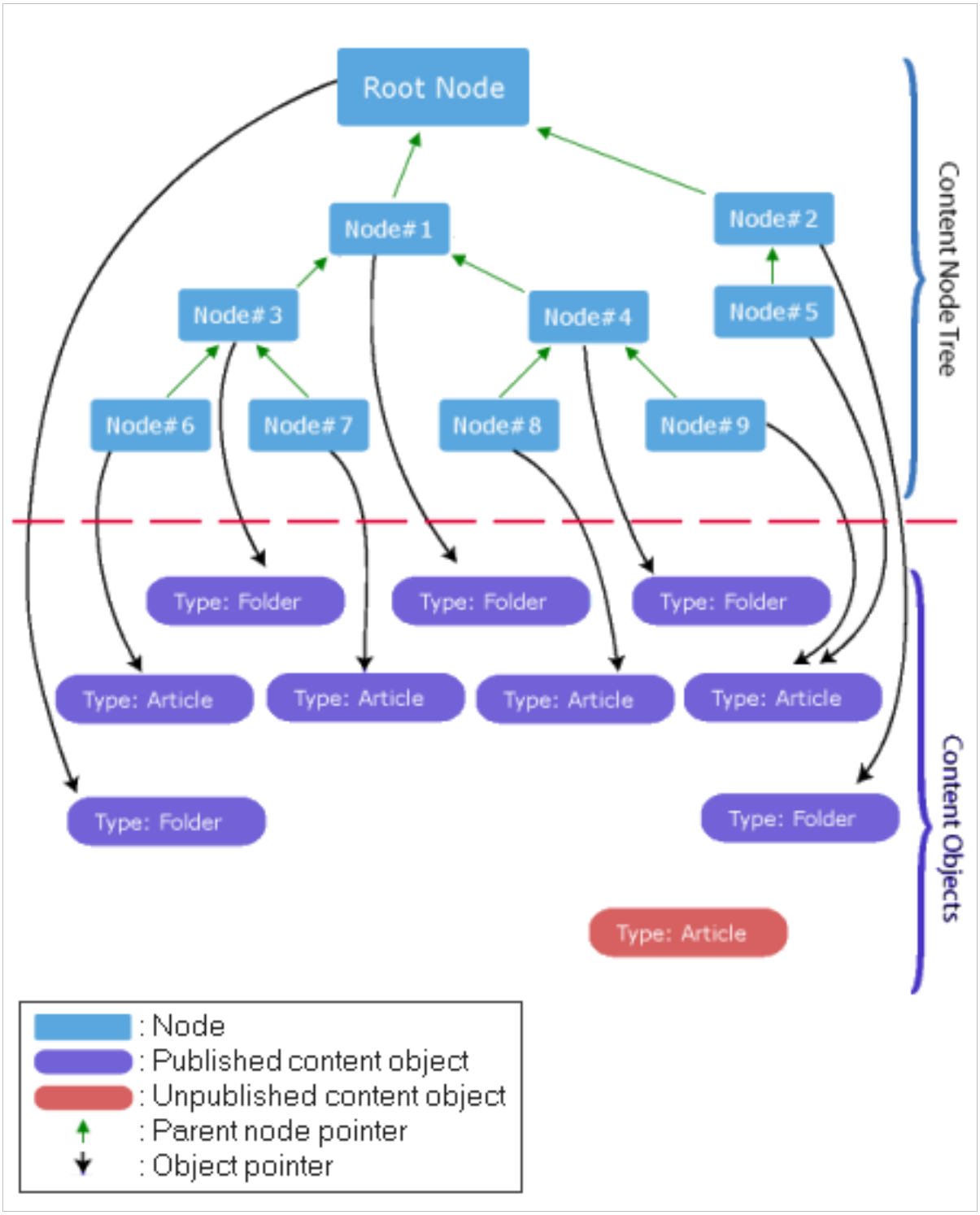
节点(node)可以认为是对内容对象(content object)的封装。节点树(content node tree)由节点(node)组成。这是一个将属于站点(site)内所有的内容按照分层次的树状结构进行表示的方法。换句话说，这是一个用来组织每一个内容对象(content object)并使之呈现在系统中的机制。对树状结构中的内容进行分类的一般做法是利用容器(例如文件夹 folder)将相关的内容对象(content object)放置在一起（大致与文件系统 filesystem 的方式相类似）。

当使用后台管理界面(administration interface)来管理文件夹、文章、文件、图片等等，用户以节点树(content node tree)的方式进行，可以很容易地产生一个站点图表(sitemap)。以下图表说明了内容对象(content object)和节点(node)之间的关系。



每片节点树(content tree)中的叶子就是一个节点(node)。一个节点树(content tree)最少包括一个节点(node)，被称为根节点(root node)。这就是将所有的内容对象(content object)放置在实际的根文件夹下。

每个(node) 节点(根节点除外)都有两个指向，一个指向父节点(parent node)，一个指向内容对象(content object)。一个节点(node)只能有一个父节点(parent node)和一个内容对象(content object)。但是，一个内容对象(content object)可能被几个节点(node)同时指向。这就意味着同一对象可以出现在节点树的不同位置。一个未发布的(content object)内容对象(草稿)是不会被一个节点(node)所关联的。换句话说，只有被发布过的内容对象(content object)才能出现在节点树(content node tree)当中。被归档的内容对象(content objects)将不会再有节点(node)指向它并且会从节点树中消失。由于对象的节点封装(node-encapsulation)，任一类型的内容对象(content object)都可以被放置在节点树的任何地方(但是要一直在系统当中)。节点树的层次数量是没有限制的。显而易见，这是一种构造任意自定义类型内容的极其灵活和通用的解决方案。


Root Node
Node#1
Node#2
Node#3
Node#4
Node#5
Node#6
Node#7
Node#8
Node#9
Content Node Tree
Type: Folder
Type: Folder
Type: Folder
Type: Article
Type: Article
Type: Article
Type: Article
Type: Folder
Type: Folder
Type: Article
Content Objects
: Node
: Published content object
: Unpublished content object
: Parent node pointer
: Object pointer

1.3.10 Locations

当要描述一个对象的所处位置时，我们可以利用前面章节提到的术语“location”(位置)。

一个对象(object)可以被几个节点所分配，从而出现在节点树(content node tree)的多个地方。换句话说，一个对象(object)在节点树(content node tree)当中可以有几个不同的位置。在这里位置(location)和节点(node)基本上是同义的（一个封装的对象）。

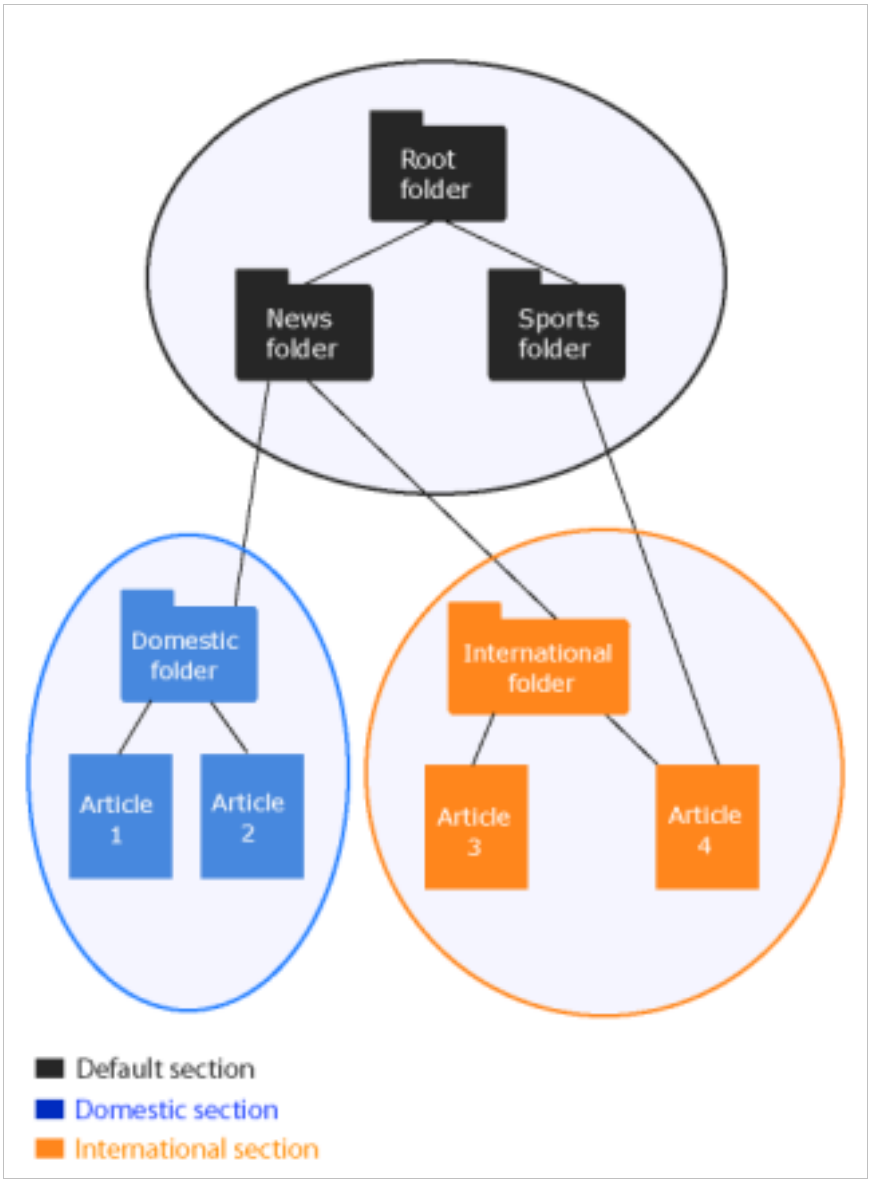
1.3.11 Sections

依照前面部分提到的，节点树是(content node tree)一个将属于站点(site)内所有的内容按照分层次的树状结构进行表示的方法。对树状结构中的内容进行分类的一般做法是利用容器(例如文件夹 folder)将相关的内容对象(content object)放置在一起（大致与文件系统 filesystem 的方式相类似）。除了具有层次的结构之外，节点树(content node tree)还可以按照逻辑块来划分。块(sections)基本上被用来分割节点树(content node tree)。

利用块(sections)可以很容易的实现：

- 使用自定义模版(custom templates)
- 控制/限制(control/limit)对内容(content)的访问

对块(section)的使用是应用于对象(object)之上的。每个(object)对象只能属于一个块(section)。缺省的，一个对象(object)是属于(起始/默认)块的(first/default section)。后台管理界面(administration interface)利用块(section)可以很容易地匹配上基于允许在节点水平(node level)上的划分。每次块(section)所分配的节点(node)是变化的，随之而来，基于块(section)所有的子节点地分配也是变化的。它们将自动继承属于块(section)的对象的主要位置(object's main location)。



1.4 eZpbulish 的站点管理

eZpbulsih 系统具有在同一安装环境下运行多个站点(site)的能力。一个站点(site)可以有不同的方法去实现。各种不同的设计通过站点(site)不同的部分联系在一起。这段章节包括了关于 eZpbulsih 如何对站点(site)进行管理和站点界面(site interface)的简介。

1.4.1 站点

术语'site'(站点)是指将一切内容压缩成属于一个独有的站点：

- 配置设置
- 数据库包括的数据结构和内容
- 与内容相关的文件
- 与设计相关的文件

1.4.2 站点界面

网站的内容可以以各种各样的方式显示(和修改)。这可能将用到对多个站点的界面(site interface)的使用，一个站点界面(site interface)基本上确定了当通过某种方式访问该站点(site)时，那些特别的设计被使用。每个站点(site)最少有两个界面：管理员界面(administration interface)和用户界面(user interface)。一个典型的例子是，站点管理员使用管理员界面(administration interface)对站点(site)进行创建和修改，而用户只能对唯一的用户界面(user interface)进行访问。使用漂亮的样式仅仅只是显示网页内容，通常一个站点(site)可能有几个不同的界面。

1.4.3 站点的访问

辨别站点(site)和站点界面(site interface)，eZpbulish 使用了一般的方式叫(site access)站点访问，站点访问设置基本上定义了两件事情：

- eZ Publish 怎样辨认出被访问站点/界面
- 多方面的配置设置将复写缺省的设置

最重要的配置是复写(database setting)数据库配置文件和(design setting)设计配置文件，由增加的站点访问(site access)，可以创建一个新的站点(和/或)在已有的站点上添加新的界面。

1.5 ezpbulsih URLs

eZ Publish 具有两种类型的 URL：

- System URLs 系统 URLs
- Virtual URLs 虚拟的 URLs

系统 URLs

一个系统的 URL 是一个未处理的 eZ Publish URL，它包含了一个被访问的关于 content/functionality 内

容/功能函数的混和信息。(system URL)系统 URL 看起来像如下这个样子：

http://www.example.com/content/view/full/44

虚拟 URLs

虚拟 URL(virtual URL)是任意一个系统 URL(System URL)的简化版本，使用虚拟 URL 更好，它更加便于记忆并且通常比对应的(system URL)系统 URL 要短。虚拟 URL(virtual URL)经常被归类为“nice URLs“和”URL aliases“。一个虚拟 URL(virtual URL)看起来像如下这个样子：

http://www.example.com/recent_news

样例

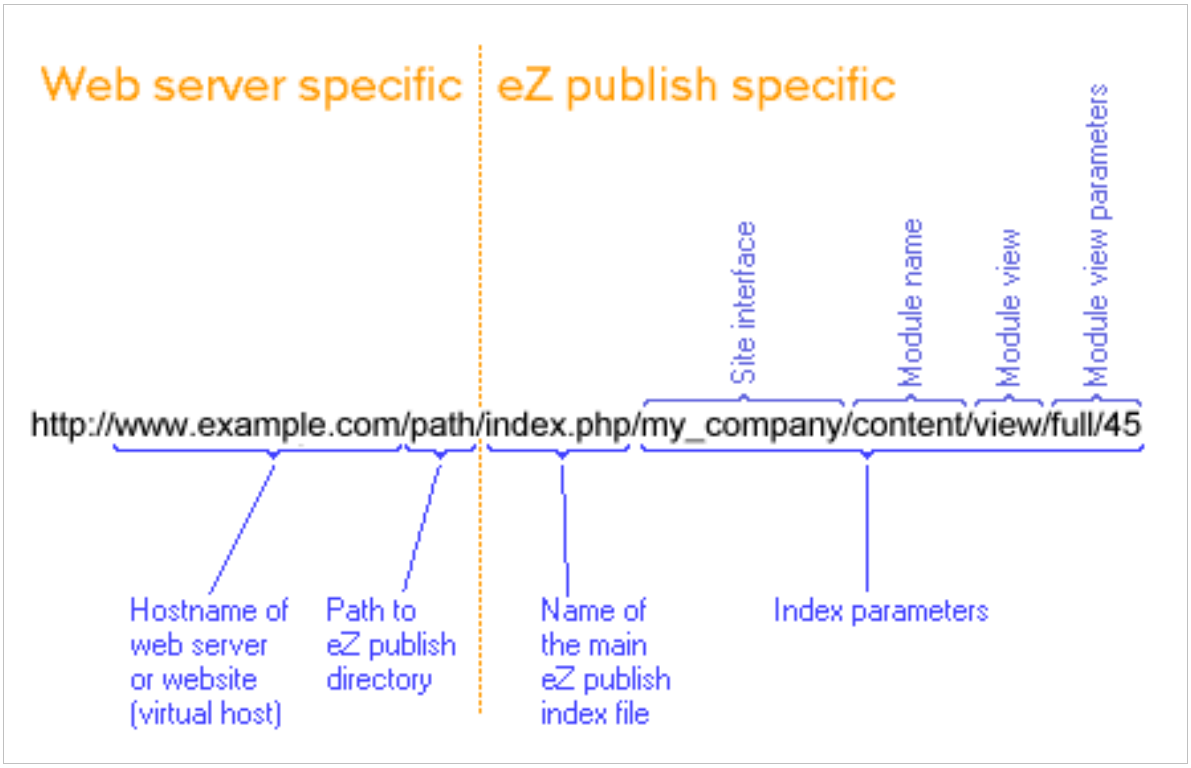
访问一个站点(site)的页面链接有两种方式：

- 使用系统 URL(System URL)：http://www.example.com/content/view/full/46
- 使用虚拟 URL(Virtual URL)： http://www.example.com/links

下面对这两种 URL 类型进一步解释。

1.5.1 系统 URLs

一开始，系统 URL(system URL) 可能猛得看起来挺混乱的。但是，实际上并非如此，所有的系统 URL 都具有一个完美，清晰的定义结构。下面的插图详细的解释了一个典型的 eZ Publish 系统 URL 每个不同的部分：

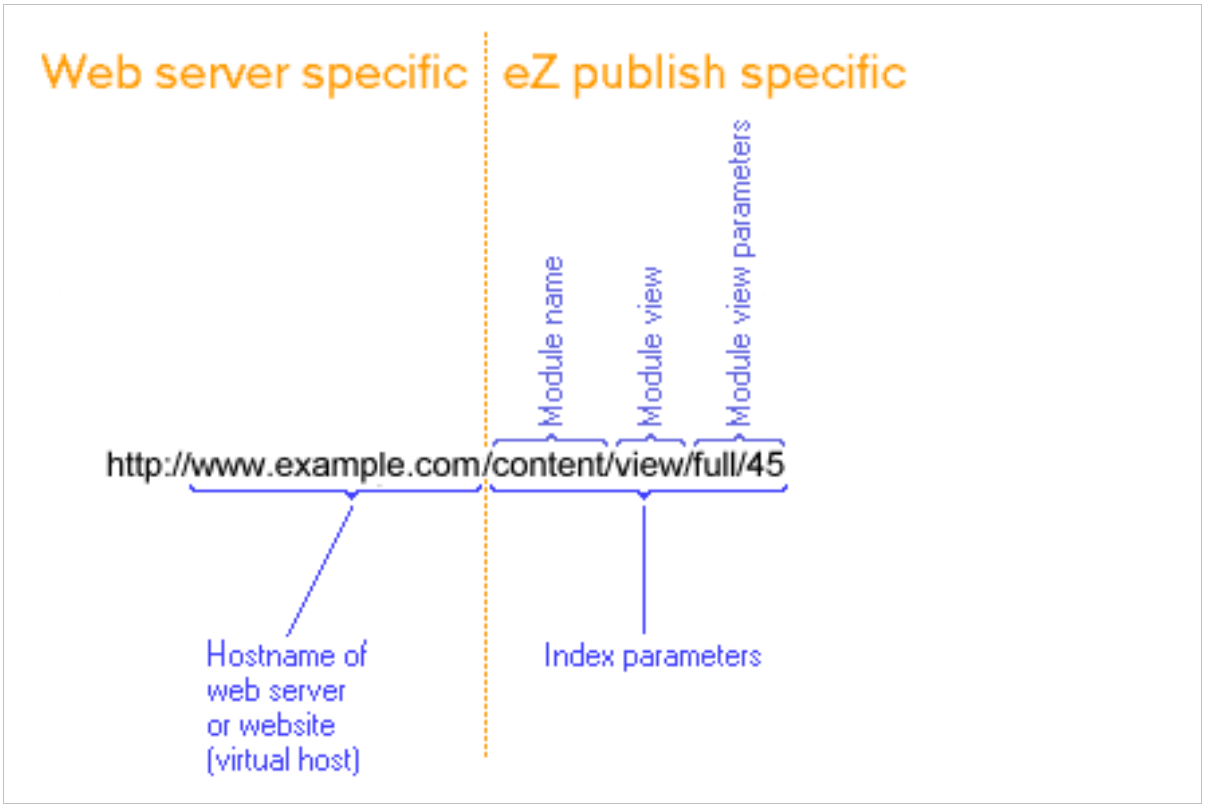


这个插图中的 URL 反映了较多基本/缺省的 web 服务和 eZ Publish 的配置。

web 服务器可以配置不显示索引文件；如果那样， “index.php“将会所有的 URL 请求中消失。

缺省 eZ publish 站点的访问配置是 URL，这就意味者站点界面(site interface)的名称被合并在自己的 URL 请求中，如果站点访问配置是 hostname，站点名字很可能然后将被合并在 hostname(或者是子域的形式)。如果站点访问配置是端口(port)，那么就是在 hostname 后添加一个冒号和端口(port)号。在这两个例子中(hostname 和 port)，站点界面(site interface) 的名字将不会被包括在 URL 中。

下面的图表显示了 URL 早期例子中一个被剥离的版本(隐藏索引文件，非 URL 站点访问模式)。

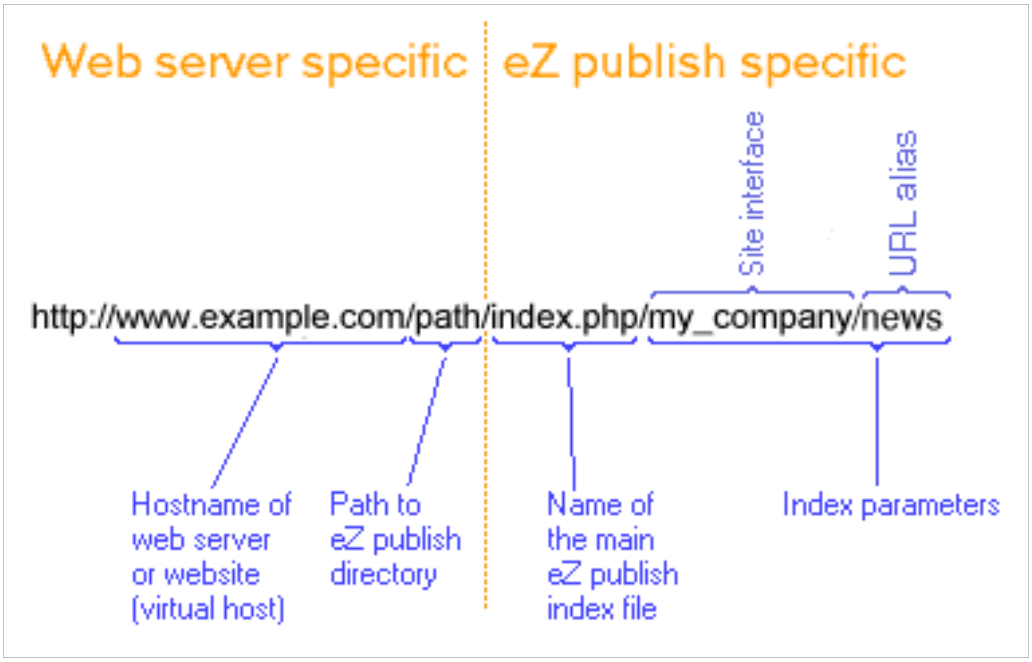


1.5.2 虚拟 URLs

虚拟 URL(virtual URLs) 简单地说是一个任意系统 URL(system URL)的别名，使用虚拟 URL 更好，它更加便于记忆并且通常比对应的(system URL)系统 URL 要短。虚拟 URL(virtual URL)经常被归类为“nice URLs“和”URL aliases“。

各个节点(content node)可以通过虚拟 URL 获得(eZ Publish 自动生成被发布的 content 对象 URL 别名)。节点的虚拟 URL(一个发布的 content 对象)是一个节点名(node's name)被简化的版本。节点名(node's name)都被转化为小写，空格之间用下划线联接等等。虚拟 URL(包括重定向，等。)可以通过使用管理员界面(administration interface)手动的添加。

下面插图详细说明了一个典型的 eZ Publish 虚拟 URL 各个不同的部分：

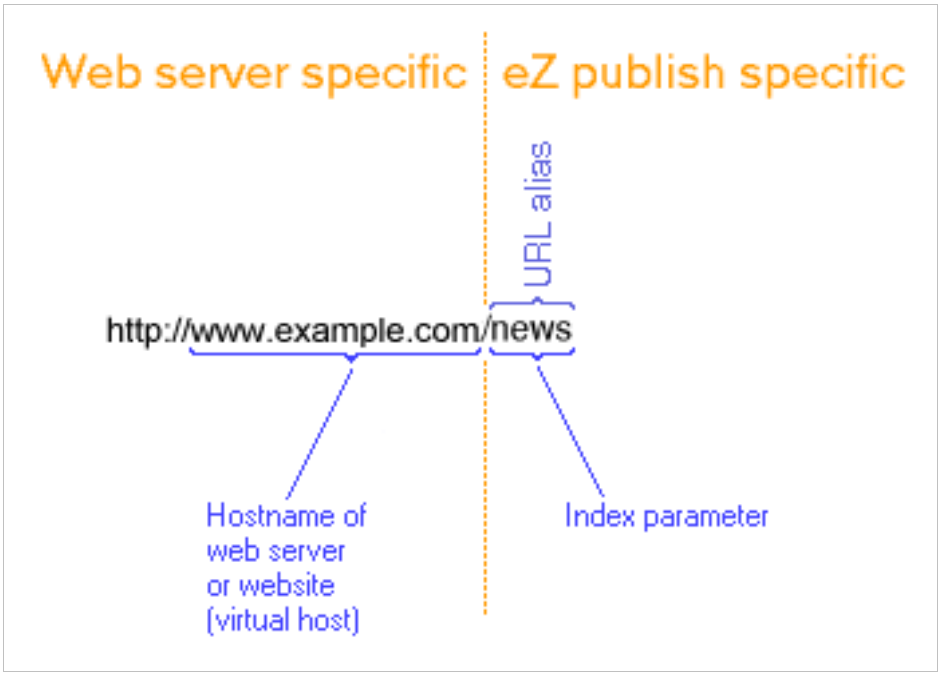


这个插图中的 URL 反映了较多基本/缺省的 web 服务和 eZ Publish 的配置。

web 服务器可以配置不显示索引文件；如果那样， “index.php“将会所有的 URL 请求中消失。

缺省 eZ publish 站点的访问配置是 URL，这就意味者站点界面(site interface)的名称被合并在自己的 URL 请求中，如果站点访问配置是 hostname，站点名字很可能然后将被合并在 hostname(或者是子域的形式)。如果站点访问配置是端口(port)，那么就是在 hostname 后添加一个冒号和端口(port)号。在这两个例子中(hostname 和 port)，站点界面(site interface) 的名字将不会被包括在 URL 中。

下面的图表显示了(virtual/aliased/nice)URL 早期例子中一个被剥离的版本(隐藏索引文件，非 URL 站点访问模式)。



1.6 总结

这个章节对... __ FIX_ME __进行了简要的介绍。

2 附录

2.1 词汇表

词汇	解释

附录 1: 词汇表

2.2 引用

引用	注释
上海众森计算机科技有限公司	www.zerustech.com
eZ Publish basics	http://ez.no/eZ Publish/documentation/ez_publish_basics

附录 2: 引用