



Phing 用户指南

本篇大部分由 `httpnet` 翻译, 参见<http://blog.csdn.net/httpnet/archive/2005/04/20/354864.aspx>, 英文版原文地址 <http://phing.info/docs/guide/2.2.0/>。本文档由我(Hick <www.HickWu.com>) 整理, 最新的订正版信息可以访问 <http://www.hickwu.com/?p=12> 查看 (下载 pdf)。由于原译者并未全部翻译完成, 因此我会在学习过程中进一步完善, 但不一定会全部翻译, 欢迎提出错漏之处。如有兴趣一起学习并翻译的, 欢迎在上面的页面联系本人。

简介

Phing 是什么?

Phing 是一个基于 Apache Ant 的项目代码构建系统.你可以用她做传统的构建系统能做的任何事情,比如 GNU make,并且 Phing 使用简单的 XML 构建文件和可扩展的"任务"使之成为易于使用和极具可扩展性的框架。

因为 Phing 是基于 Ant 的,所以本手册部分内容是摘自 Ant 手册。

Phing & Binarycloud :历史

Phing 源自 Binarycloud 的一个子项目.Binarycloud 是一个高度工程化的框架,为了在企业环境中使用而设计.Binarycloud 广泛使用 XML 来存储关于项目的元数据(配置,节点,窗口小部件,站点结构,等等)。

因为 Binarycloud 是为 PHP 构建的,在每一个页面请求上执行 XML 处理和转换是不切实际的.Phing 用于"编译"XML 元数据为可被 PHP 引擎处理的数组形式。

当然,XML"编译"只是 Binarycloud 使用 Phing 构建系统的许多方法中的一个.Phing 构建系统使你能够:

1. 从单一的源代码树构建多语言页面。
2. 在单个 XML 文件中聚集元数据(Metadata)并且用多个不同的 XSLT 生成几个文件。(译者注:在 XSLT2.0 规范中,消除了 1.0 规范只能有一个输入一个输出的限制,现在我们可以做到单一输入来生成多个输出文件)

最初,Binarycloud 使用 GNU make 构建系统,但是,这样有些缺点. 在 makefile 中的"Space before tab problem"问题.实际上,它仅在 UNIX 系统上没有问题.因此需要寻找一个更好的构架系统.Apache Ant 是一个不错的选择—它使用 XML 构建文件和模块化的设计.但问题是 Ant 用 java 写成,要使用它你必须得在你得计算机上安装 Java 虚拟机。

Besides the need for yet another interpreter (i.e. besides PHP), there was also legal/ideological conflict in requiring a commercial JVM (there were problems with Ant on JVMs other than Sun's) for an LGPL'd Binarycloud.

因此,Phing 开始开发了,Phing 是一个借助于 Ant 的思想用 PHP 写成的构建系统.第一版同时设计和开发,因此不是十分稳定.系统很快暴露出了它的限制并且需要一个更好的 Phing 系统.因此衍生了 Phing2 的雏形。

Phing 当前的开发集中于 Phing2,它涉及许多功能增强,bug 修正,并且最值得注意的转变是用 PHP5 的抽象类,接口,和 try/catch/throw 异常处理来重写了原来 Phing 的基础代码。



Phing 怎样工作

Phing 使用包含一组项目构建描述的 XML 构建文件.构建文件由一些运行实际的命令的目标组成(比如复制文件,删除目录,执行数据库查询,等等).因此使用 Phing,第一步要编写构建文件,然后运行 Phing,在构建文件中定义的要执行的目标.

```
phing -f mybuildfile.xml mytarget
```

如图 1,键入 `phing -h` 可以查看命令参数的描述

默认情况下,Phing 将会查找一个叫 *build.xml* 的文件.(除非构建文件的名称不是 *build.xml*,否则不必为 Phing 指定构建文件的名称) 并且,如果没有指定任何目标那么 Phing 将执行在<project>标签中设置的默认目标 (default 属性所指定的).

Cool,so how can i help?

Phing 当前正在积极开发之中并且有许多事情要完成.如果要参与其中,可以通过下面的方法:

1. 阅读本手册以了家 Phing ;-)
2. 到<http://phing.tigris.org>订阅“dev”邮件列表

安装 Phing

(Hick 注: 原译者此处为“配置 Phing”, setting-up 应翻为“安装”)

本章的目标是帮助你在你的操作系统上正确的设置和执行 Phing.一旦你正确的设置了 phing 你就不需要再回过头来看这章了,除非你要重新安装,或者是迁移到其他的平台.

系统要求

操作系统

Phing 具有很好的可移植性,所有 PHP 能够跑起来的系统上都能够运行它.不过,某些比较高级的功能,可能是跟操作系统平台相关的,比如 `chmod` 来改变目录属性,这样的操作在 windows 平台会被忽略.

为了充分发挥 Phing 的作用,我们建议使用 Linux, FreeBSD, OpenBSD 为系统平台.

相关软件

(Hick 注: 下面两段在最新英文文档中已经没有,暂时保留!)

要使用 Phing 你必须安装 PHP5.0.0b1 和以上的版本并且加上 `-with-libxml2`,和`—with-xsl` 选项.

if you want to make use of advanced functionality. At the time of writing PHP5.0.0b2-dev is currently unable to run Phing due to segmentation faults arising somewhere in the XML parsing of the build file.

下面是一个 Phing 依赖的相关软件的清单:



Software	Required for	Source
PHP 5.0.1+	Execution	http://www.php.net
PHPUnit2 2.2.0+	Additional functionality	http://www.phpunit.de
Xdebug 2.0.0b2+	Additional functionality	http://www.xdebug.org
PhpDocumentor 1.3.0RC3+	Additional functionality	http://www.phpdoc.org

怎么获得 Phing

有几种方法来获得 Phing 的发行包.如果你不想参与当前的开发,推荐你获取最近的快照版本或稳定版本,如果你有兴趣参与我的开发,你可以从 CVS 获取正在开发中的文件.

获得发行包

获取 Phing 发行包的最容易的方法是访问 Phing 的主页<http://phing.info> 下载当前相应格式的你需要的分包包. (Hick 注: 目前的 <http://phing.info> 不容易找到下载包, 可以从 Phing 在 tigris 的主页 <http://phing.tigris.org/> 看到找到下载链接为 <http://phing.info/trac/wiki/Users/Download>)

自 2.0.0b1 版之后,你可以可以下载 PEAR 可安装包和完全的 Phing 分包包.如果你希望修改 Phing,我们建议你下载完整的 Phing 发行包,这样你可以创建你自己的 PEAR 包.如果你知识简单地使用 Phing 作为项目的需要或构建其他的包,下载并且安装 PEAR 包.

从 CVS 获取一个正在开发中的拷贝

(Hick 注: 英文文档并没有及时修改, 目前应该以 SVN 方式从 <http://svn.phing.info/trunk> check out 最新代码。)

鼓励你对 Phing 的开发作出贡献.如果你像参与到 Phing 的开发中或你知识对其中最新的功能感兴趣,你可以看看下面的叙述并从中从 CVS 获取一份拷贝.

The CVS revisions of Phing are not bullet-proof and may fail to execute properly on your machine. Only obtain the CVS versions if you are absolutely aware of limitations and constraints of such an action. Additionally you should sign up to the development mailinglist to report and notice errors and incompatibilities.

我们假设你正运行 UNIX 类型的操作系统

So we expect the CVS software is installed ant the cvs executable is in your system's search path.. However, the steps for a Windows based system are very similar.

这里有大量的关于如何使用 CVS 的资源可用,并且还有关于 tigris 项目管理平台的 CVS 说明.

首先你必须做的第一件事是登录到 CVS 服务器.在命令行下键入:

```
cvs -d :pserver:guest@cvs.tigris.org:/cvs login
```

如果你要作为开发者登录到 Tigris Web 站点上.用你自己的用户名和密码登录.你也可以以访客的身份登录到 Tigris Web 站点.



要从项目源代码资源库中检出个别的模块(如果你不需要整个项目资源库),键入:

```
cvs -d :pserver:guest@cvs.tigris.org/cvs checkout phing
```

PEAR 安装

安装 Phing 最简易的方法是使用 `pear` 安装工具.

```
$> pear install http://phing.info/pear/phing-current.tgz
```

PEAR 安装工具将检查包的依赖性,并把 Phing 的执行脚本放到 PHP 的安装目录下面(`pear` 命令脚本同时也在这个目录下面)

非 PEAR 安装(手动安装)

如果你不是用 `pear` 安装,你可以自己手动安装,但这样要复杂一些,你需要配置你的环境变量以使 `phing` 能够找到 `phing.bat` 执行脚本.你可以从<http://phing.tigris.org>下载

[phing-2.0.0.zip](#) 或 [phing-2.0.0.tar.gz](#),解压到任意一个目录下,产生三个目录 `bin`,`docs` 和 `classes`,然后设置环境变量 `PHING_HOME`,把它指向到 Phing 目录.

运行 Phing 之前,你还需要做一下配置.

1. 添加 `$PHING_HOME(*nix)`或 `%PHING_HOME%(windows)`到 `PATH` 环境变量中.
2. 设置 `PHING_HOME` 环境变量指向到 Phing 的安装目录.
3. 设置 `PHP_COMMAND` 变量,指向 `php` 命令所在的位置,例如 (`*nix /usr/bin/php`) 或 `windows(c:\php5\php.exe)`
4. 设置 `PHP_CLASSPATH` 环境变量以包含 Phing 需要额类库.至少应该包含 `%PHP_HOME%\classes`. 还有一种方法是把 `%PHP_HOME%\classes` 路径添加到 `php.ini` 文件中的 `include_path` 参数中去
5. 检查 `php.ini`,并确保有如下设置
 - a) `max_execution_time = 0 // 不限制`
 - b) `memory_limit = 32MB // 决于你的构建文件的大小,你可能需要更多的内存`

UNIX

如果你使用 UNIX,使用 `bash bourne shell`,并且 Phing 安装在 `/opt/phing` 目录下

通过下面的方法适当地设置环境.

```
export PHP_COMMAND=/usr/bin/php
```

```
export PHING_HOME=/opt/phing
```

```
export PHP_CLASSPATH=${PHING_HOME}/classes
```

```
export PATH=${PATH}:${PHING_HOME}/bin
```

Windows

在 Windows 平台生假设 Phing 安装在 c:\opt\phing 目录,那么配置入下:

```
set PHP_COMMAND=c:\opt\php\php.exe
set PHING_HOME=c:\opt\phing
set PHP_CLASSPATH=c:\opt\phing\classes
set PATH=%PATH%;%PHING_HOME%\bin
```

高级

有许多用于运行 Phing 的变量,你至少需要如下几个设置:

1. 如果你要 Phing 能够使用其他的包/类,那么你要把他们添加到 PHP_CLASSPATH 变量中 PHP.ini 文件的 include_path 变量中.

有些任务要求第三方库.

调用 Phing

现在,你准备在命令行或脚本中调用 Phing 了.下面的章节简要的描述了如何正确的执行 Phing

命令行

Phing 在命令行执行也很简单.仅仅切换到构建文件所在的目录,并键入:

```
phing [targetname]
```

Getting Started

phing 构建文件用 XML 标记语言编写,因此你应该有一些 XML 和 Ant 的基础知识才能很好的理解下面的章节,在 web 上有许多的资源可用.

XML 和 Phing

构建文件有如下基本结构:

1. 文档序言 (document prolog)
2. 根元素<project>
3. 几个类型元素 (<property>,<fileset>,<patternset>)
4. 包含一个或几个内建的或用户自定义的任务元素(例如:<javac>,<tar>)

编写一个简单的构建文件

```
<?xml version="1.0"?>
```



```

<project name="FooBar" default="dist" basedir=".>
<!--创建文件夹-->
<target name="prepare">
<echo msg="Preparing build..." />
<mkdir dir="./build" />
</target>
<!--复制文件-->
<target name="build" depends="prepare">
<echo>Building...</echo>
<copy file="./src/File.php" to="./build/File.php"/>
<copy file="./src/File2.php" to="./build/File2.php"/>
</target>
<!--打包-->
<target name="dist" depends="build">
<echo message="Creating archive..." />
<tar outfile="furbee.tar.gz" basedir="./build"/>
</target>
<!--删除, 清理-->
<target name="clean">
<echo msg="Cleaning up..." />
<delete file="./build"/>
</target>
</project>

```

project 元素

文件序言之后紧跟着的第一个元素的是<project>,此元素是一个包含其他元素的容器,并且有以下属性:

<project>的属性:

属性	含义	要求的?
name	项目的名称	No
basedir	项目的根目录,如果未指定将使用当前目录	No
default	在调用构建文件的时候,如果没有指定要执行的目标, 将执行在此定义的项目默认目标	Yes
description	项目的描述	Yes

target 元素

一个目标可依赖其他的目标.你可能有一个在构建树中安装文件的目标,例如,创建一个 tar.tgz 分发包的目標.

Target 元素的属性:

属性	含义	要求
name	目标的名称	Yes



depends	此目标依赖的逗号分隔列表	No
if	为了使此目标得以执行而必须设置的属性的名称	No
unless	为了使此目标得以执行而禁止设置的属性的名称	

任务元素

一个任务是一段可执行的 PHP 代码段.此代码段实现了特定的行为(例如,复制文件).因此你必须在构建文件中定义,才能是 Phing 真正的调用它.

附录有一组核心任务和许多可选的任务.编写你自己的任务也很容易(扩展 Phing 一节).

任务可以被分配一个 id 属性:

```
<taskname id="taskID" .../>
```

property 元素

属性是在构建文件使用的基本的变量,可以在构建文件中通过 **PropertyTask** 任务设置或在 Phing 外部通过命令行传递.通过命令行传递的属性总是会覆盖在构建文件中设置的属性,也就是说命令行是最后传递给 Phing 的,自然会覆盖先前在构建文件中设置的属性.

属性有一个名称和一个唯一的值.属性可以作为任务属性的值.这是通过在"\${"和"}"包含属性名称来设置的.例如:

```
<property name="javasourcedir" value="./src/java"/>
<javac srcdir="${javasourcedir}"/>
```

这里在运行时\${javasourcedir}将被展开为./src/java

内建属性

Phing 给你提供了访问系统属性的能力,好像它已经通过<property>定义过的一样.例如,{os.name}展开为操作系统的名称.附录有完整的内建属性的列表.

更复杂的构建文件

```
<?xml version="1.0" ?>
<project name="testsite" basedir="." default="main">
  <property environment="env"/>
  <property file="${env.BCHOME}/build.properties"/>
  <property name="package" value="${phing.project.name}" override="true" />
  <property name="builddir" value="${env.BCHOME}/build/testsite" override="true" />
  <property name="srcdir" value="${project.basedir}" override="true" />
  <!-- Fileset for all files -->
  <fileset dir="." id="allfiles">
    <include name="**">
  </fileset>
  <!-- Main Target -->
```

```
<target name="main" description="main target">
<copy todir="${builddir}">
<fileset refid="allfiles" />
</copy>
</target>
<!-- Rebuild -->
<target name="rebuild" description="rebuilds this package">
<delete dir="${builddir}" />
<phingcall target="main"/>
</target>
</project>
```

构建文件首先用<property>标记调用 PropertyTask 任务来定义属性.然后它定义一个文件集和两个目标.

前面 5 个元素定义了项目的属性.(按照访问方式不同分三种)

1. (环境变量) 第一个 property 元素,有属性 environment 属性,其值为 env,这样就可以通过 env.VARIABLE_NAME 的方式来访问环境变量.例如,我可以调用 echo 任务吧 env.PATH 变量打印在控制台输出上:

```
<echo messages="${env.PATH}"/>
```

这样就会输出 PATH 环境变量的字符串.

2. (文件) 第二个 property 元素仅包含 file 属性,这里 file 属性的值是一个相对的或绝对的指向属性文件的路径 (属性文件:后缀名为 .properties,其中包含的键值对 (key/value) 的集合,格式见附录 E) .

3. (直接在构建文件中定义属性变量) 通过如下方式定义:

```
<property name="propertyname" value="propertyvalue"/>
```

另外要注意的是构建文件中的<fileset>标签.它定义一个文件集合,例如一组多个文件组成的集合.你还可以用<fileset>元素的子元素<include>和<exclude>包含和排除<fileset>文件模式指定的文件集合

译者注:实际上<fileset>最终所包含的文件是<include>和<exclude>的差集)

关于文件集,见附录 C 有更详尽的描述.另外<fileset>标签有 id 属性,通过 id 属性可以在其他地方引用.就像直接包含一样,通过引用实现了代码的复用.

第一个任务仅仅包含通过<copy>标签调用的 CopyTask 任务.其中并没有用<include>和<exclude>标签来包含和排除文件,而是用了<fileset>的 refid 属性来引用前面定义的 fileset,这样我们就可以在一个地方定义,而在构建文件中的其他地方多次的使用.

在第二个目标中使用了<phingcall>标签来调用了 PhingTask 任务 (详尽见附录 B)

项目组件

这一章的目标是使你熟悉构建文件的基本组件.读完这一章,即使你不知道构建文件中;某些代码段的含义,你也能够读懂,和理解构建文件的基本结构.

对于补充的参考信息,你可以查看附录 A,附录 B,附录 C,附录 D.

Projects

在项目文件结构中,必须定义<project>元素,它是 Phing 构建文件的根元素,是必须的,意味着所有的项目相关的信息都是包含这<project>元素以内的.

```
<?xml version="1.0"?>
<project name="test" description="this is a simple test for buildfile" default="main">
<!--Everything else here-->
</project>
```

project 的三个属性,name—项目名称,这是必须的,description—项目的简单描述,default—如果在执行时没有指定目标,那么将使用项目的默认目标.

Project Components In General

项目组件是你能够在项目文件中找到的所有描述项目信息的标记,目标是项目组件,还有任务,类型,等等.项目组件可能有属性和嵌套的标签.属性仅仅包含简单的值,例如字符串,整数,等等.嵌套的元素可以是复杂的 Phing 类型(如 Filesets)或简单的值的封装类(wrapper classes)

Targets

Targets are collections of project components (but not other targets) that are assigned a unique name within their project. A target generally performs a specific task -- or calls other targets that perform specific tasks -- and therefore a target is a bit like a function (but a target has no return value).

Targets may depend on other targets. For example, if target A depends on a target B, then when target A is called to be executed, target B will be executed first. Phing automatically resolves these dependencies. You cannot have circular references like: "target A depends on target B that depends on target A".

The following code snippet shows an example of the use of targets.

```
<target name="othertask" depends="buildpage" description="Whatever">
  <!-- Task calls here -->
</target>
<target name="buildpage" description="Some description">
  <!-- Task calls here -->
</target>
```

When Phing is asked to execute the othertask target, it will see the dependency and execute buildpage first. Notice that the the dependency task can be defined after the dependent task.

Tasks

Tasks are responsible for doing the work in Phing. Basically, tasks are the individual actions that your buildfile can perform. For example, tasks exist to copy a file, create a directory, TAR files in a directory. Tasks may also be more complex such as XsltTask which copies a file and transforms the file using XSLT, SmartyTask which does something similar using Smarty templates, or CreoleTask which executes SQL statements against a specified DB. See Appendix B for descriptions of Phing tasks.

Tasks support parameters in the form of:

Simple parameters (i.e. strings) passed as XML attributes, or

More complex parameters that are passed by nested tags

Simple parameters are basically strings. For example, if you pass a value "A simple string." as a parameter, it is evaluated as a string and accessible as one. You can also reference properties as described in *Getting Started*.

Note: There are special values that are not mapped to strings, but to boolean values instead. The values true, false, yes, no, on and off are translated to true/false boolean values.

```
<property name="myprop" value="value" override="true"/>
```

However, some tasks support more complex data types as parameters. These are passed to the task with nested tags. Consider the following example:

```
<copy>
  <fileset dir=".">
    <include name="*" />
  </fileset>
</copy>
```

Here, CopyTask is passed a complex parameter, a Fileset. Tasks may support multiple complex types in addition to simple parameters. Note that the names of the nested tags used to create the complex types depend on the task implementation. Tasks may support default Phing types (see below) or may introduce other types, for example to wrap key/value pairs.

Refer to Appendix B for a list of system tasks and their parameters.

Types

Basics

Besides the simple types (strings, integer, booleans) you can use in the parameters of tasks, there are more complex Phing Types. As mentioned above, they are passed to a task by using nesting tags:

```
<task>
  <type />
</task>
<!-- or: -->
<task>
  <type1>
    <subtype1>
      <!-- etc. -->
    </subtype1>
  </type1>
</task>
```

Note that types may consist of multiple nested tags -- and multiple levels of nested tags, as you can see in the second task call above.

Referencing Types

An additional fact about types you should notice is the possibility of referencing type instances, i.e. you define your type somewhere in your build file and assign an id to it. Later, you can refer to that type by the id you assigned. Example:

```
<project>
  <fileset id="foo">
    <include name="*.php" />
  </fileset>
  <!-- Target that uses the type -->
  <target name="foo" >
    <copy todir="/tmp">
      <fileset refid="foo" />
    </copy>
  </target>
</project>
```

As you can see, the type instance is assigned an id with the id attribute and later on called by passing a plain fileset tag to CopyTask that only contains the refid attribute.

Basic Types

The following section gives you a quick introduction into the basic Phing types. For a complete reference see Appendix C.

FileSet

FileSets are groups of files. You can include or exclude specific files and patterns to/from a FileSet. The use of patterns is explained below. For a start, look at the following example:

```
<fileset dir="/tmp" id="fileset1">
  <include name="sometemp/file.txt" />
  <include name="othertemp/**" />
  <exclude name="othertemp/file.txt" />
</fileset>
<fileset dir="/home" id="fileset2">
  <include name="foo/**" />
  <include name="bar/**/*.*.php" />
  <exclude name="foo/tmp/**" />
</fileset>
```

The use of patterns is quite straightforward: If you simply want to match a part of a filename or dirname, you use *. If you want to include multiple directories and/or files, you use **. This way, filesets provide an easy but powerful way to include files.

FileList

FileLists, like FileSets, are collections of files; however, a FileList is an explicitly defined list of files -- and the files don't necessarily have to exist on the filesystem.

Besides being able to refer to nonexistent files, another thing that FileLists allow you to do is specify

files in a certain order. Files in FileSets are ordered based on the OS-level directory listing functions, in some cases you may want to specify a list of files to be processed in a certain order -- e.g. when concatenating files using the <append> task.

```
<filelist dir="base/" files="file1.txt,file2.txt,file3.txt"/>
<!-- OR: -->
<filelist dir="basedir/" listfile="files_to_process.txt"/>
```

FilterChains and Filters

FilterChains can be compared to Unix pipes. Unix pipes add a great deal of flexibility to command line operations; for example, if you wanted to copy just those lines that contained the string blee from the first 10 lines of a file called foo to a file called bar, you could do:

```
cat foo | head -n10 | grep blee > bar
```

Something like this is not possible with the tasks and types that we have learned about thus far, and this is where the incredible usefulness of FilterChains becomes apparent. They emulate Unix pipes and provide a powerful dimension of file/stream manipulation for the tasks that support them.

FilterChain usage is quite straightforward: you pass the complex Phing type filterchain to a task that supports FilterChains and add individual filters to the FilterChain. In the course of executing the task, the filters are applied (in the order in which they appear in the XML) to the contents of the files that are being manipulated by your task.

```
<filterchain>
  <replacetokens>
    <token key="BC_PATH" value="${top.builddir}"/>
    <token key="BC_PATH_USER" value="${top.builddir}/testsite/user/${lang}"/>
  </replacetokens>
  <filterreader classname="phing.filters.TailFilter">
    <param name="lines" value="10"/>
  </filterreader>
</filterchain>
```

The code listing above shows you some example of how to use filter chains. For a complete reference see Appendix C. This filter chain would replace all occurrences of BC_PATH and BC_PATH_USER with the values assigned to them in lines 4 and 5. Additionally, it will only return the last 10 lines of the files.

Notice above that FilterChain filters have a "shorthand" notation and a long, generic notation. Most filters can be described using both of these forms:

```
<replacetokens>
  <token key="BC_PATH" value="${top.builddir}"/>
  <token key="BC_PATH_USER" value="${top.builddir}/testsite/user/${lang}"/>
</replacetokens>
<!-- OR: -->
<filterreader classname="phing.filters.ReplaceTokens">
  <param type="token" name="BC_PATH" value="${top.builddir}"/>
  <param type="token" name="BC_PATH" value="${top.builddir}/testsite/user/${lang}"/>
</filterreader>
```

As the pipe concept in Unix, the filter concept is quite complex but powerful. To get a better understanding of different filters and how they can be used, take a look at any of the many uses of FilterChains in the build files for the binarycloud [bc] project.

File Mappers

With FilterChains and filters provide a powerful tool for changing contents of files, Mappers provide a powerful tool for changing the names of files.

To use a Mapper, you must specify a pattern to match on and a replacement pattern that describes how the matched pattern should be transformed. The simplest form is basically no different from the DOS copy command:

```
copy *.bat *.txt
```

In Phing this is the glob Mapper:

```
<mapper type="glob" from="*.bat" to="*.txt"/>
```

Phing also provides support for more complex mapping using regular expressions:

```
<mapper type="regexp" from="^(.*)\.conf\.xml$$" to="\1.php"/>
```

Consider the example below to see how Mappers can be used in a build file. This example includes some of the other concepts introduced in this chapter, such as FilterChains and FileSets. If you don't understand everything, don't worry. The important point is that Mappers are types too, which can be used in tasks that support them.

```
<copy>
  <fileset dir=".">
    <include name="*.ent.xml" />
  </fileset>
  <mapper type="regexp" from="^(.*)\.ent\.xml$" to="\1.php"/>
  <filterchain>
    <filterreader classname="phing.filters.XsltFilter">
      <param name="style" value="ent2php.xsl" />
    </filterreader>
  </filterchain>
</copy>
```

For a complete reference, see Appendix C.

Conditions

Conditions are nested elements of the condition and if tasks.

not

The <not> element expects exactly one other condition to be nested into this element, negating the result of the condition. It doesn't have any attributes and accepts all nested elements of the condition task as nested elements as well.

and

The <and> element doesn't have any attributes and accepts an arbitrary number of conditions as nested elements. This condition is true if all of its contained conditions are, conditions will be evaluated in the order they have been specified in the build file.

The <and> condition has the same shortcut semantics as the && operator in some programming languages, as soon as one of the nested conditions is false, no other condition will be evaluated.

or

The <or> element doesn't have any attributes and accepts an arbitrary number of conditions as nested elements. This condition is true if at least one of its contained conditions is, conditions will be evaluated in the order they have been specified in the build file.

The <or> condition has the same shortcut semantics as the || operator in some programming languages, as soon as one of the nested conditions is true, no other condition will be evaluated.

available

This condition is identical to the Available task, all attributes and nested elements of that task are supported, the property and value attributes are redundant and will be ignored.

uptodate

This condition is identical to the UpToDate task, all attributes and nested elements of that task are supported, the property and value attributes are redundant and will be ignored.

os

Test whether the current operating system is of a given type

Attribute	Description	Required
family	The name of the operating system family to expect.	Yes

Supported values for the family attribute are:

windows (for all versions of Microsoft Windows)

mac (for all Apple Macintosh systems)

unix (for all Unix and Unix-like operating systems)

equals

Tests whether the two given Strings are identical

Attribute	Description	Required
arg1	First string to test.	Yes
arg2	Second string to test.	Yes
casesensitive	Perform a case sensitive comparison. Default is true.	No
trim	Trim whitespace from arguments before comparing them. Default is false.	No



isset

Test whether a given property has been set in this project.

Attribute	Description	Required
property	The name of the property to test.	Yes

contains

Tests whether a string contains another one.

Attribute	Description	Required
string	The string to search in.	Yes
substring	The string to search for.	Yes
casesensitive	Perform a case sensitive comparison. Default is true.	No

istrue

Tests whether a string evals to true.

Attribute	Description	Required
value	value to test	Yes

```
<istrue value="{someproperty}"/>
```

```
<istrue value="false"/>
```

扩展 Phing

可扩展性

(Hick 注: 暂未翻译)

源代码结构

(Hick 注: 暂未翻译)

系统初始化

(Hick 注: 暂未翻译)

系统服务

(Hick 注: 暂未翻译)

Build Lifecycle

(Hick 注: 暂未翻译)



Writing Tasks

(Hick 注: 暂未翻译)

Writing Types

(Hick 注: 暂未翻译)

Writing Mappers

(Hick 注: 暂未翻译)

编写映射器

创建一个 Mapper

在 PHP5 中编写一个文件 Mapper 由 PHP5 支持的接口简化了.本质上,你的自定义文件名 mapper 必须实现 phing.mappers.FileNameMappers 接口.这里有一个创建 DOS 文件名风格的文件名 Mapper

附录 A 情况说明书

内建属性

属性	内容
application.startdir	
host.arch	主机名称.在 Windows 上不可用
host.doman	DNS 域名.例如 php.net.在 Windows 不可用
host.fstype	文件系统类型,可能的取值为 UNIX,WINNT,WIN32.
host.machine	系统架构.例如,i586.在 Windows 不可用
host.name	由 posix_name()返回的名称,在 Windows 上不可用
host.os.release	操作系统正是发行版本,例如 2.2.10,在 Windows 上不可用
host.os.version	操作系统版本, #4 Tue Jul 20 17:01:36 MEST 1999.在 Windows 上不可用
lince.separator	行分隔符,Unix 为"\n",Windows 为"\r\n",Macintosh 为"\r"
php.classpath	PHP_CLASSPATH 环境变量的值.
php.version	PHP 解释器的版本.
phing.buildfile	当前构建文件的完整路径
phing.id	
phing.version	当前的 Phing 版本



project.name	当前项目名称
project.basedir	当前项目的根路径
project.description	当前项目描述
user.home	环境变量 HOME 的值





附录 B 核心任务

AdhocTaskdefTask

AdhocTypedefTask

AppendTask

AvailableTask

CallTargetTask

CopyTask

DeleteTask

EchoTask

ExecTask

ForeachTask

InputTask

MkdirTask

MoveTask

PhingTask

PearPackageTask

PhpEvalTask

PropertyTask

ReflexievTask



ResolvePathTask

arTask

askdefTask

TouchTask

TypedefTask

UpToDateTask

XsltTask

附录 C Optional Tasks

CoverageMergerTask

CoverageReportTask

CoverageSetupTask

IoncubeEncoderTask

IoncubeLicenseTask

PearPackageTask

PHPDocumentorTask

PhpLintTask

PHPUnit2Task

PHPUnit2Report

SvnExportTask

SvnLastRevisionTask

TarTask

UntarTask

UnzipTask

XmlLintTask

ZipTask

附录 D 核心类型

这是一个 Phing 系统数据类型的参考

FileList 文件列表

FileList 数据类型，正如它的名字一样描述文件，不像 **FileSets**, **FileList** 可以包含还没有创建的文件，就是说它不在文件系统中。同样 **FileLists** 可以用指定的顺序来描述文件，然而从文件系统返回的 **FileSet** 确实无序的。

使用例子

```
<filelist dir="/etc" files="httpd/conf/httpd.conf,php.ini"/>
```

或者使用一个列表文件 (**listfile** 属性)，每行包含一个文件名

```
<filelist dir="conf/" listfile="ini_files.txt"/>
```

这将截取 **ini_files.txt** 文件中的每一行作为一个文件返回到 **FileList** 中

属性

Name	Type	Description	Default	Required
dir	String	files 和 listfile 指出的文件所在的目录	N/A	Yes
files	String	逗号或空格分隔的文件列表	N/A	Yes (或 listfile)
listfile	String	一个表示一个文件列表的文本文件， 每行表示一个文件	N/A	Yes(或 files)



FileSet

PATH/CLASSPATH

Core Filters

Phing Filter Reader

Expand Properties

HeadFilter

LineContains

LineContains RegExp

Prefix Lines

Replace Tokens

ReplaceRegexp

StripLineBreaks

StripLineComments

StripPHPComments

TabToSpace

TailFilter

XsltFilter

Core Mappers

FlattenMapper





GlobMapper

IdentityMapper

ergeMapper

RegexpMapper

附录 E 项目组件

Phing 项目

Projects 在构建文件中是项目最外面的容器元素,同时<project>也是项目构建文件中的根元素,它包含有名称,目录,一个简短的描述和默认目标等属性.

项目可包含任务调用和目标.(如下)

例子:

```
<project name="TestProject" basedir="." default="main"
Description="This is a test project to show how to use projects">
<!--Everthing else goes here -->
</project>
```

属性

名称	类型	描述	默认	要求否
basedir	字符串	项目的根目录	n/a	Yes
default	字符串	如果没有在命令行指定目标, 将使用这里定义的默认目标	all	No
description	字符串	项目简单描述	n/a	No
name	字符串	项目的名称	n/a	Yes

目标

例子:

```
<target if="lang" unless="lang.en" depends="foo1,foo2" name="main"
Description="This is a example target">
<!--everything else gose here -->
</target>
```





定义在上面例子中的这个目标仅在属性 `lang` 设置并且属性 `lang.en` 没有设置时执行,另外它依赖 `foo1` 和 `foo2` 两个目标,意思是在 `main` 目标执行前必须执行 `foo1` 和 `foo2` 两个目标,这个目标的名称是 `mian` 并且有一个简单的描述.

名称	类型	描述	默认	要求否
<code>depends</code>	String	当前目标所以来的目标, 如果有多个依赖目标,以逗号分隔	n/a	No
<code>description</code>	String	关于目标的简短的描述信息	n/a	No
<code>if</code>	String	必须设置的属性	n/a	No
<code>unless</code>	String		n/a	No
<code>name</code>	String	目标名称	n/a	Yes

附录 F 文件格式

构建文件格式

下面的 XML 文件显示了构建文件的一个骨架,它仅包含了一个 `project` 和一个 `target` 元素.关于如何使用他们,可查看有关 `Phing Types` 和 `Tasks` 的信息.

```
<?xml version='1.0' ?>
<!--
The root tag of each build file must be a "project" tag.
-->
<project name="(projectname)" [basedir="(projectbasedir)"]
[default="(targetname)"] [description="(projectdescription)"]>
<!--
Type and task calls here, i.e. filesets, patternsets, CopyTask calls etc.
-->
<target name="(targetname)" [depends="targetname1,targetname2"]
[if="(ifproperty)"] [unless="(unlessproperty)"]>
<!--
Type and task calls here, i.e. filesets, patternsets, CopyTask calls, etc.
-->
</target>
<!--
More targets here
-->
</project>
```

属性文件格式

属性文件定义属性.属性以键/值对的形式保存,并且只包含纯文本,属性文件的后缀名称一般是 `.properties`,构建文件的默认属性文件是 `build.properties`





```
# Property files contain key/value pairs
key=value
# Property keys may contain alphanumeric chars and colons, but
# not special chars. This way you can create pseudo-namespaces
myapp.window.hsize=300
myapp.window.vsize=200
myapp.window.xpos=10
myapp.window.ypos=100
# You can refer to values of other properties by enclosing their
# keys in "${}":
text.width=${myapp.window.hsize}
# Everything behind the equal sign is the value, you do
# not have to enclose strings:
text=This is some text, Your OS is ${php.os}
# I guess that is all there is to property files
```

参考资料

国际标准

[osi-model]

OSI (Open System Interconnect) 模型 <http://www.iso.org>
<http://www.instantweb.com/foldoc/foldoc.cgi?OSI>

[xml10-spec]

W3C XML 1.0 规范 <http://www.w3.org/XML/>

[unicode] Unicode <http://www.unicode.org>

许可

[gnu-lgpl]

The GPL (Gnu Lesser Public License) <http://www.gnu.org/licenses/lgpl.html>

[gnu-fdl]





The Gnu FDL (Free Documentation License), the license used for this documentation
<http://www.gnu.org/licenses/fdl.html>

开源项目

[bc]

Binarycloud <http://www.binarycloud.com> <http://binarycloud.tigris.org>

[w3c-tidy]

HTMLTidy, a W3C (x)HTML and XML syntax checker and code beautifier
<http://www.w3c.org/People/Ragget/tidy/>

[phpdoc]

PHPDoc 项目 <http://www.phpdoc.de>

[phpclasses]

Manuel Lemos 的 PHPClasses 资源库 <http://www.phpclasses.org>

[pear]

PEAR (Php Extension Archive Repository) PHP 扩展库 <http://pear.php.net>

[ant] Ant, a Java Build Tool, the main inspiration for Phing <http://ant.apache.org>

[gnumake] GNU make, an inspiration for Phing <http://www.gnu.org/software/make/make.html>

[pollo]

Pollo, a visual editor for XML files. A schema to edit phing build files is shipped with Phing.
<http://pollo.sourceforge.net>

[gingerall]

GingerAlliance - Home Of Sablotorn <http://www.gingerall.com>

[php]

PHP 官方主页 PHP Hypertext Preprocessor <http://www.php.net>

[gnu]

GNU (GNU's Not Unix)组织 <http://www.gnu.org>

[phing]

Phing (PHing Is Not Gnumake) 主页 <http://phing.info>





[cvs-howto]

Short manuals for CVS <http://www.ucolick.org/~de/CVSbeginner.html>

[cvs-tigris]

CVS and tigris.org

http://binarycloud.tigris.org/project/www/docs/ddUsingCVS_command-line.html

其他资源

[javadoc]

Sun Javadoc <http://java.sun.com/j2se/javadoc/>

[zend]

Zend Technologies, Ltd. <http://www.zend.com>

