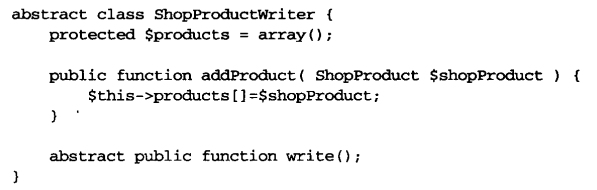
1. 静态方法：static（工厂模式用）

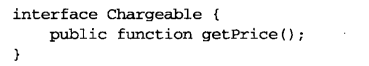
1：静态方法不能访问类中的普通属性

2：静态方法不需要new对象就可以调用，因为在声明的时候就放到内存中了，但使用时这个类必须引用过来

1. 常量属性：const
   1. 开头不能又美元符号，只能用大写字母命名
   2. 只能通过类名调用
   3. 应用场景：类中都能访问某个属性，该属性值不需要改变
   4. 类常量必须通过 "const" 定义为类的成员，强烈不鼓励使用 "define" 定义的全局常量
2. 抽象类：abstract class
   1. 抽象类不能被实例化，只可以让子类继承进行实例化
   2. 子类必须实现抽象类中的全部方法
   3. 提供具体的实现方法



1. 接口：interface
   1. 只定义功能，不包含实现内容
   2. 实现方式：implements 方式继承



* 1. 接口对象声明使用

public function addChargeableItem( Chargeable $item){

//…….

}

1. 延迟静态绑定：static关键词

abstract class DomainObject {

public static function create(){

return new self()

}

}

1. 应用场景：解决修改父类self关键词的属性（这个功能只适用于静态属性或静态方法，self:: 不能改变static属性的值，可以使用static:: 属性改变）
2. 错误处理
   1. Exception异常信息处理
   2. getMessge() ：获得传递给构造方法的消息字符串
   3. getCode()：获得传递给构造方法的错误码
   4. getFIle()：获取产生的异常文件
   5. getLine()：获取生成异常行号
   6. getPrevious()：获取一个嵌套的异常对象
   7. getTrace()：获取一个多维数组，该数组追踪导致异常的方法调用，包含类、方法、文件和参数数据
   8. getTraceAsString()：获取getTrace()返回数据的字符串版本
   9. \_\_toString()：在字符串中使用Exception对象时自动调用。返回一个描述异常细节的字符串

抛出异常方式

1. throw new Exception()
2. try{}catch(Exception $e)

异常子类化 ----》继承异常进行自定义异常

Final类和方法

1. final 关键词不能有子类，继承final属性会报如下错误
2. 

使用拦截器：（interceptor）,java和c++ 中被称为重载

1. \_\_get($property)：访问未定义的属性
2. \_\_set($property, $value)：给未定义的属性定义值
3. \_\_isset($property)：对未定义属性调用isset()时被调用
4. \_\_unset($property)：对未定义属性调用unset()时被调用
5. \_\_call($method,$arg\_array)：调用未定义方法时被调用

析构方法：\_\_destruct()

复制对象：\_\_clone()

对象字符串：\_\_toString() ----》直接输出对象的值，有这个不会报错，输出这个函数中的值，返回值必须时字符串，以下错误时【5.53版本】

**Catchable fatal error**: Method Person::\_\_toString() must return a string value ----》 返回不是字符串报错

**Catchable fatal error**: Object of class Person could not be converted to string -----》没有这个函数报的错

回调、匿名函数和闭包：call\_user\_func()、array\_walk()等函数调用

string **create\_function** ( string $args , string $code ) ：创建函数，7.2版本弃用

1. 命名空间：模仿实现包
   1. 解决不同文件中创建相同函数冲突
   2. 命名规则最好以文件在的目录+方法名进行命名，命名出现冲突使用as 进行重命名
   3. Lister::helloWord() ----》访问本地 \Lister::helloWord() ----》访问全局
   4. 一个文件中定义多个命名空间

namespace {

class A{}

}

自动加载：\_\_autoload()

spl\_autoload\_register()：将函数注册到SPL \_\_autoload函数队列中。如果该队列中的函数尚未激活，则激活它们 ---》定义好加载函数，加载函数名称

bool **spl\_autoload\_register** ([ [callable](http://php.net/manual/zh/language.types.callable.php) $autoload\_function [, bool $throw = true [, bool $prepend = false ]]] )

DIRECTORY\_SEPARATOR：目录分割符

类函数和对象函数

1. class\_exists()：查找类 ---》bool **class\_exists** ( string $class\_name [, bool $autoload = true ] )
2. get\_declared\_classes() ---》返回由已定义类的名字所组成的数组
3. get\_class()：返回对象的类名 ----》string **get\_class** ([ object $object = **NULL** ] )
4. is\_a()：象属于该类或该类是此对象的父类则返回 **TRUE -----》**bool **is\_a** ( object $object , string $class\_name [, bool $allow\_string = **FALSE** ] )
5. instanceof：和is\_a()功能一样 -----》object instanceof string
6. **get\_class\_methods：**返回由类的方法名组成的数组 ---》array **get\_class\_methods** ( [mixed](http://php.net/manual/zh/language.pseudo-types.php#language.types.mixed) $class\_name )
7. **is\_callable()：**检测参数是否为合法的可调用结构 ----》bool **is\_callable** ( [callable](http://php.net/manual/zh/language.types.callable.php) $name [, bool $syntax\_only = false [, string &$callable\_name ]] )

$methodVariable = array($anObject, 'someMethod');

var\_dump(is\_callable($methodVariable, true, $callable\_name));  //  bool(true)

$functionVariable = 'someFunction';  
var\_dump(is\_callable($functionVariable, false, $callable\_name));  // bool(true)

1. method\_exists()：检查类的方法是否存在 ------》bool **method\_exists** ( [mixed](http://php.net/manual/zh/language.pseudo-types.php#language.types.mixed) $object , string $method\_name )
2. get\_class\_vars():返回由类的默认属性组成的数组 ----》array **get\_class\_vars** ( string $class\_name )
3. get\_parent\_class()：返回对象或类的父类名 ----》string **get\_parent\_class** (string $class\_name )
4. is\_subclass\_of()：如果此对象是该类的子类，则返回 **TRUE ----》**bool **is\_subclass\_of** ( object $object , string $class\_name )，如果对象 object 所属类是类 class\_name 的子类，则返回 **TRUE**，否则返回 **FALSE**。
5. call\_user\_func()：把第一个参数作为回调函数调用 -----》[mixed](http://nc1.php.net/manual/zh/language.pseudo-types.php#language.types.mixed) **call\_user\_func** ( [callable](http://nc1.php.net/manual/zh/language.types.callable.php) $callback [, [mixed](http://nc1.php.net/manual/zh/language.pseudo-types.php#language.types.mixed) $parameter [, [mixed](http://nc1.php.net/manual/zh/language.pseudo-types.php#language.types.mixed) $... ]] )

call\_user\_func('increment', 1)  
call\_user\_func(array($classname, 'say\_hello'),2);  
call\_user\_func($classname .'::say\_hello',2); // As of 5.2.3

反射api：如java中的java.lang.reflect，获取类中的信息，用于判断类中的一些信息（具体怎么用有点懵） ----》Reflection.php 中有详细的方法

1. Reflection：为类的摘要信息提供静态函数export()
2. ReflectionClass：类信息和工具
3. ReflectionMethod：类方法信息和工具
4. ReflectionParameter：方法参数信息
5. ReflectionProperty：类属性信息
6. ReflectionFunction：函数信息和工具
7. ReflectionExtension：PHP扩展信息
8. ReflectionException：错误类
9. 对象和设计
   1. 面向对象设计和过程式编程：核心区别是如何分配职责
   2. 职责：面向对象每个方法的职责必须明确，不能多个职责混淆
   3. 内聚：一个模块内部各个成分之间相关联程度的度量 ----》职责清晰、分工明确 ---》高聚合
   4. 耦合：系统各个模块紧密绑定一起，参数高耦合 ---》低耦合
   5. 正交：将职责相关的组件紧紧组合在一起，与外部系统环境隔开，保持独立，修改只影响自身
   6. 多态：一个公共接口后面维护多个实现 ----》强制定义抽象接口，子类继承实现方法 ---》无法强制类返回数据类型
   7. 封装：对客户端代码隐藏数据和功能 --》private和protected
10. 设计模式：解决共性的问题
    1. 核心：命名、问题、解决方案、效果
    2. 设计模式与语言无关
    3. 模式定义了一组词汇
    4. 模式是经过测试的
    5. 模式为协作而设计的
    6. 针对接口编程，不是针对实现编程
11. 单例模式
    1. 全局变量破坏封装性，全局变量不会进行报错。。。

实现方式

1. 构造方法定义为私有，外部无法访问： private function \_\_construct(){}
2. 定义静态方法，初始化调用：

public static function getInstance(){

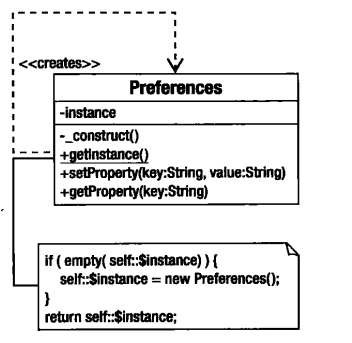
if(empty(self::$instance)){

self::$instance = new Per()

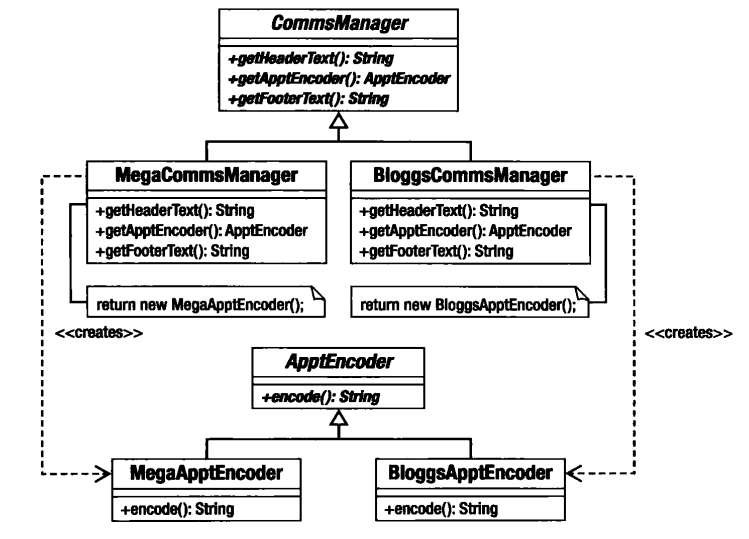
}

return self::$instance;

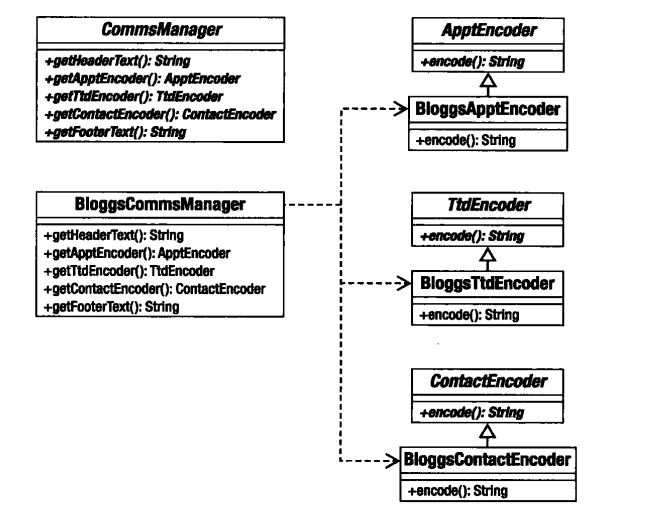
}

1. 只初始化一次就可以了
2. 
3. 单例在系统任何地方都可以被访问，很难调试依赖关系，用了，在系统中依赖关系很难追踪

十一、工厂模式

* 1. 解决代码一般化而不是特殊化，用特定的类来处理实例化
  2. 用继承和多态来封装具体产品并创建
  3. 把创建者类（工厂---> 提供生产产品的类方法【具体实现步骤实现或未实现】 ）与生产的产品类分开
  4. 
  5. 将系统与实现细节分开
  6. 对系统中功能强制进行组合
  7. 添加新方法有问题，必须对继承的进行全部实现，没用的方法也得时间 ----》抽象是父类

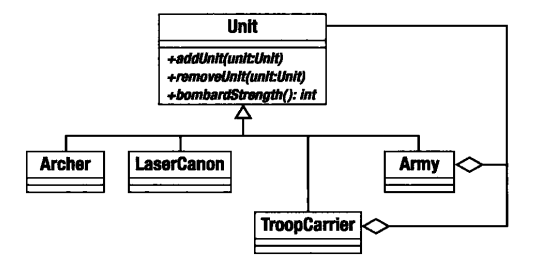
抽象工厂方法



原型模式：创造多个父类，组合多个父类实现子类的方法

function \_\_construct(Sea $sea, Plains $plains, Forest $forest)

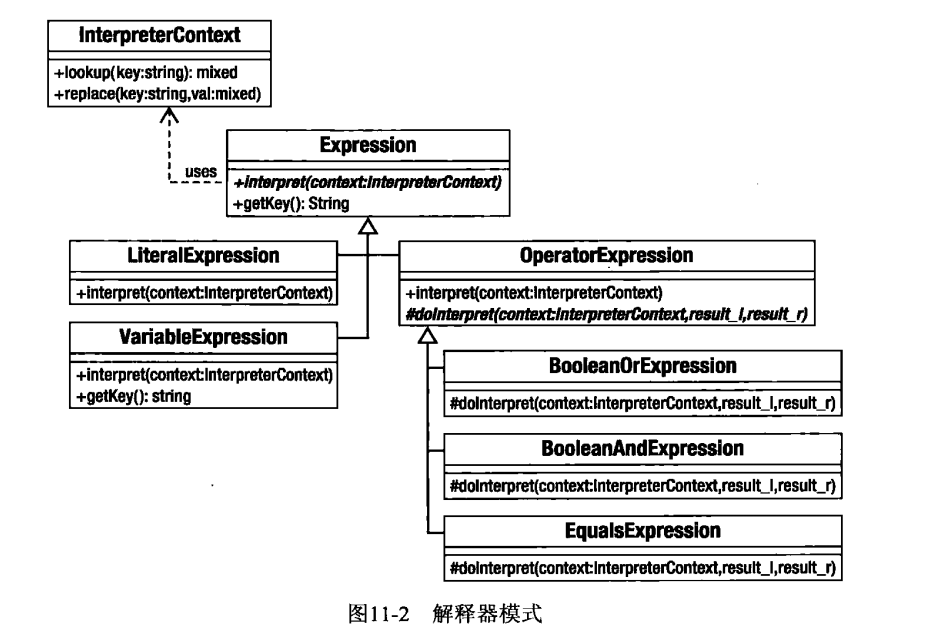
组合模式：将一组对象组合为可像单个对象一样被使用的结构，子类继承父类进行不通的操作或功能



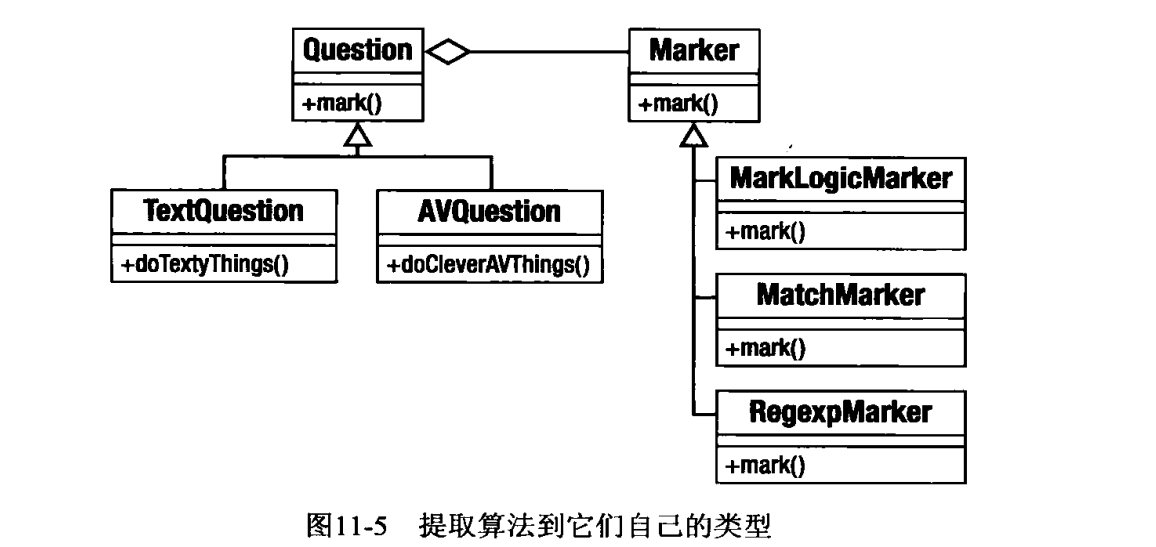
装饰模式：通过在运行时合并对象来扩展功能

外观模式：为复杂或多变的系统创建一个简单的接口，只为一个分层或一个子系统创建单一入口

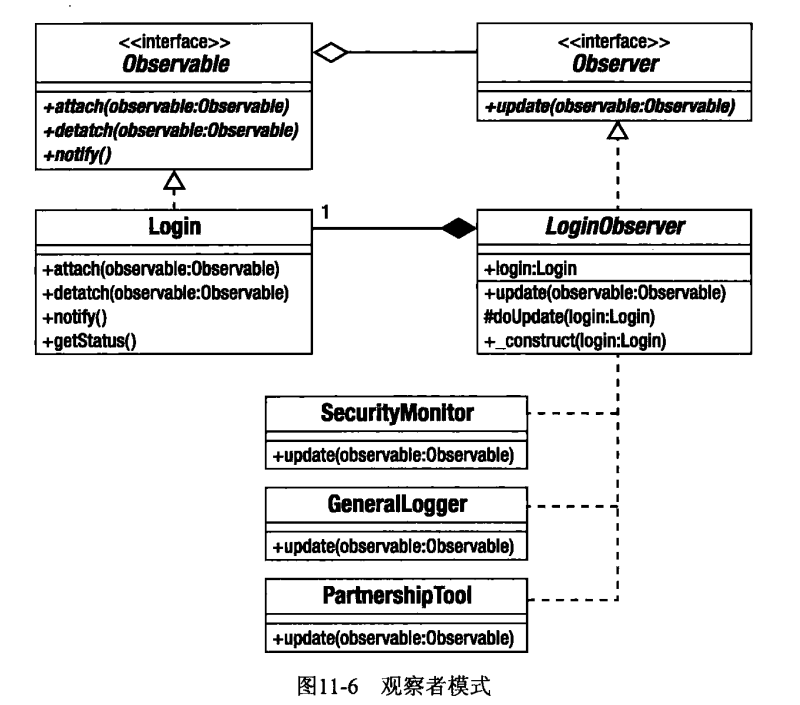
解释器模式：构造一个可用于创建脚本化应用的迷你语言解释器 ----》还以为是直接执行c了。。。，实际执行的是和组合模式差不多。。。



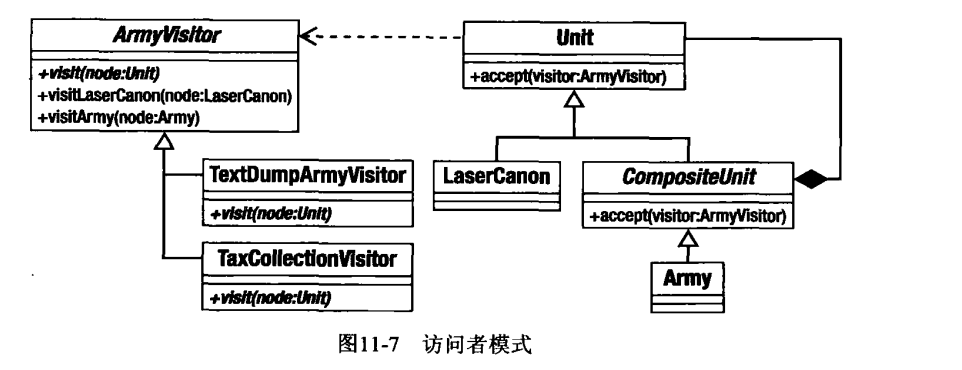
策略模式：使用组合模式不是继承模式，传入实例化的类进行处理，和依赖注入有点像，就是依赖注册。。。



观察者模式：正交性【改动和转移时对其他组件影响最小的组件】，把客户元素（观察者）从一个中心类（主体）中分离开，修改一个其他的监听，保存对象操作，这个可以使用spl内置函数操作

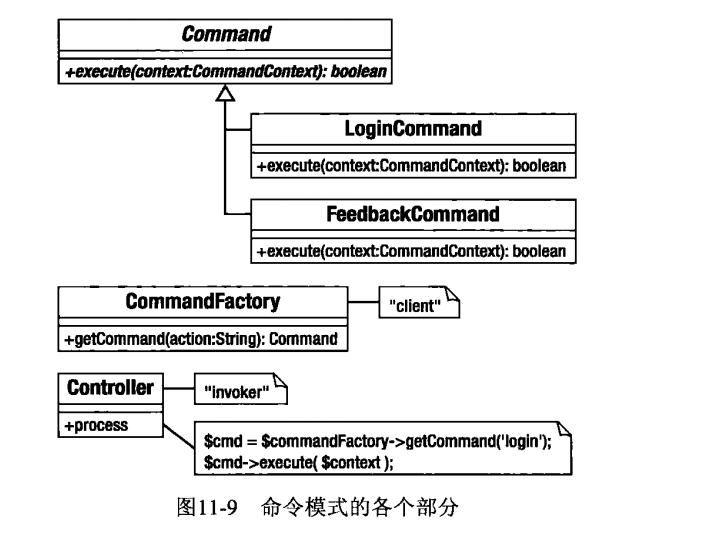


访问者模式：解决类越来越大的问题 ---》层层继承。。。。，实现相同的方法

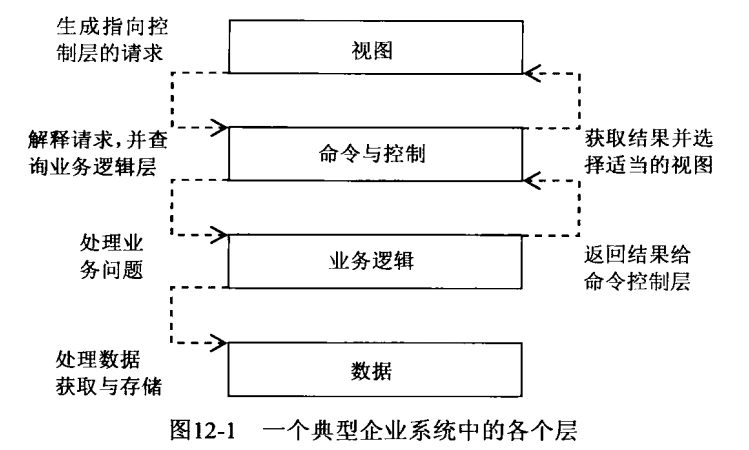


命令模式：封装类，公共调用

组成类：客户端（client）、部署命令对象的调用者（invoker）、接收命令的接收者（receiver），可以进行文件的引用方式或命名空间，实例化类



企业模式：系统中个层分工明确，进行解耦、web应用程序很难测试



视图层：用户实际看到的交互界面

命令与控制：处理用户的请求

业务处理：根据请求执行业务操作

数据层：保存和获取系统中的持久信息

企业架构之外的基础模式

1、注册表：提供系统级别的对象访问功能，静态函数调用，初始化内存中

2、作用域：应用程序作用域

3、php的shm：进程中共享变量

表现层：

1、前端控制器：用一个中心处理所有的请求，调用视图将结果呈现给用户（感觉是多态）

2、应用控制器：将命令和视图分离开

3、页面控制器：处理请求、领域逻辑和表现之间的关系

4、模板视图和视图助手

业务逻辑层

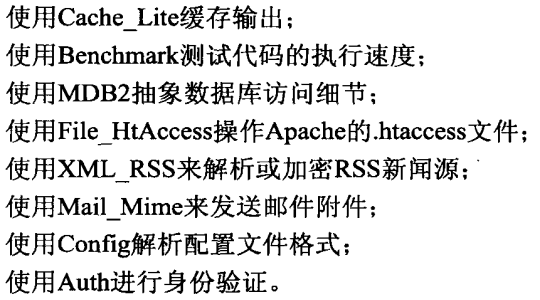
1、事务脚本模式：自己处理请求，不是委托给特定的对象完成

2、领域模型：是原始的逻辑引擎，被描述为一组属性及附加代理，是做某些事的某些东西

十三、实践

避免重复写代码

pear官方站点：<http://pear.php.net/>



合作愉快：使用svn或git版本控制器，发布代码使用安装器（Ant、Phing），写文档（phpDocumentor: http://demo.phpdoc.org）,测试（测试用具：一系列自动化测试的集合），持续集成

十四、pear和pyrus

pear

pear：许多包的集合

查看pear安装路劲：[root@localhost ~]# pear config-get php\_dir



查看pear所有配置及值：[root@localhost ~]# pear config-show

pyrus：下一代的pear应用:

安装pear包：以Log为参考

[root@localhost ~]# pear install pear/Log

install ok: channel://pear.php.net/Log-1.13.1

pear频道：【channel】，可以同时使用多个pear代码库中的包

[root@localhost ~]# pear channel-discover pear.phpunit.de

pear安装参数：

-o：确保pear安装程序会自动安装任何必要的代码包

-a：下载所有依赖包

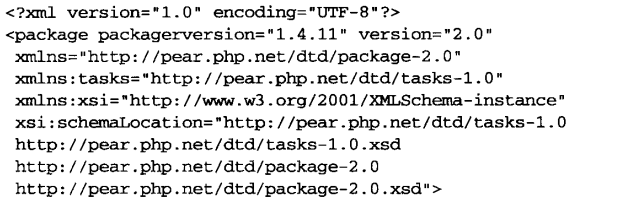
使用pear包：require\_once(文件路劲或名称)

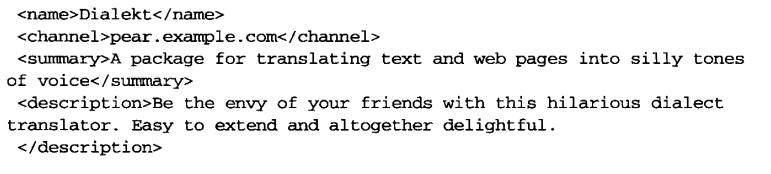
pear错误处理：PEAR::isError()检查返回值

创建pear包：

package.xml：核心文件，提供包信息，觉得该包如何被安装，安装在哪，定义该包依赖的类库，没有这个包会安装失败

package.xml组成：

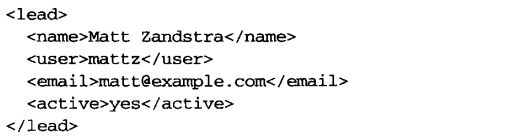




name：包名称

summary：对包的描述

description：提供更详细的说明

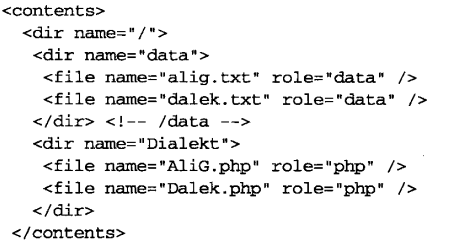


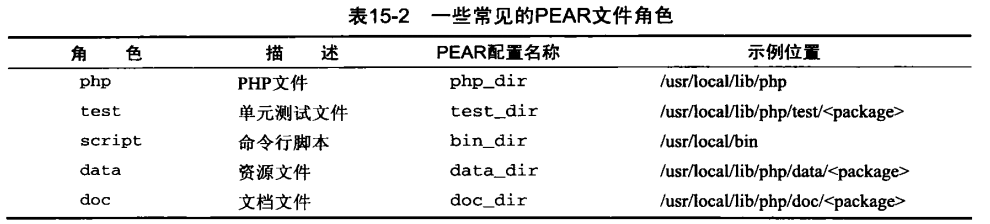
lead：至少包含一个

还需要一些细节内容

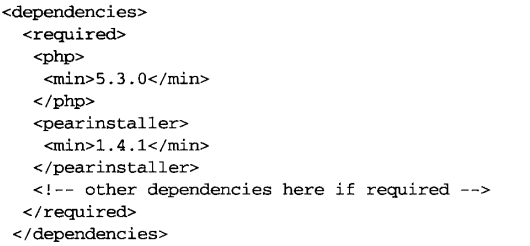


contents元素：定义了要包含的压缩包，有时被称为tarball，用tar或gzip工具压缩生成的，可以用dir和file描述结构



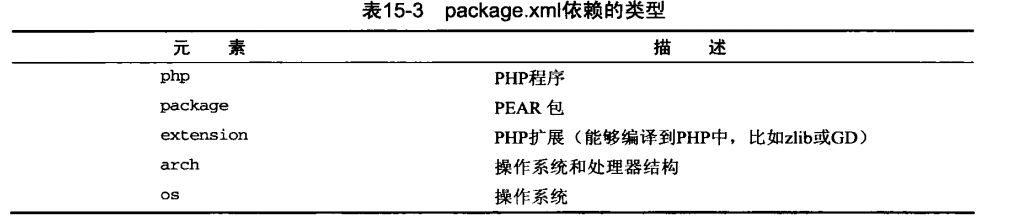


依赖：dependencies，必要，至少指定php和pear安装器的版本



php和pearinstall都可以包含min、max、recommended【设置首选安装器】、exclude【定义不能和包兼容的软件】元素

除package外，还可以定义其他的：extension、os、arch



phprelease灵活自定义安装



installconditions：决定phprelease元素是否执行

十五、用phpDocumentor生成文档

安装：[root@localhost ~]# pear upgrade PhpDocumentor

php支持zlib，可以直接使用pear安装 pear install 文件名