PHP: План

- Массивы
- Работа с файлами
- Регулярные выражения
- Объекты
- Reflection

PHP: массивы

Массивы 2 видов:

- Индексные – ключ integer
- Ассоциативные – ключ string

Строки тоже массивы:

```
<?php
$str = “array”;
$str[1] = NULL;
$str[3] = 'm';

echo $str; //выведет 'army' (echo игнорирует NULL)
echo $str[0]; //выведет 'a'
?>
```

```
<?php
//массив может быть создан 2-мя
//способами:
// >>> 1
$arr[0] = “  0 ”;
$arr[3] = “3”; // можно не по порядку
$arr[] = “4”; //автоматическая
              //индексация
$arr[1] = “1”;
$arr[] = “5”; // автоматически в
              //конец
//>>> 2
$arr = array(); //пустой
                //массив
$filled = array(“one”, “two”);

$init = array();
$init[2] = “smth”;
```

PHP:массивы

Работа с ассоциативными массивами:

```
<?php
$arr[“tea”] = “Greenfield”;
$arr['big'] = “mac”;

$user = array(
    “name” => “student”,
    “pass” => “student”,
);
?>
```

Работа с многомерными массивами:

```
<?php
$users = array(
        0 => array(
            “name” => “A”,
            “email” => “a@a.com”
        ),
        1 => array(
            “name” => “B”,
            “email” => “b@b.com”
        }
);

echo $users[1][“email”] // выведет
                        //'b@b.com';
```

PHP:массивы

В PHP имеется конструкция *foreach()* позволяющая легко проитерироваться по массиву:

```
foreach(имя_массива as $value) {
    вывражение;
}
foreach(имя_массива as $key => $value) {
    выражение;
}
```

!!! foreach работает с копией массива

```
$a = array (1, 20, 300, 12);
echo “Array values: “
foreach ($a as $v) {
    echo $v.', ';
}

echo “Array values with keys: “
foreach ($a as $k => $v) {
    echo “[$k - $v ], ”;
}
```

```
// перебор многомерного массива
$a[0][0] = “zero – zero”;
$a[0][1] = “zero – one”;
$a[2][1] = “two – one”;
$a[2][0] = “two – zero”;
foreach($a as $k1 => $v1) {
    foreach($v1 as $k2 = > $v2) {
        echo “[$k1][$k2] - $v2\n”;
    }
}
```

PHP:массивы

Можно обойти массив самостоятельно, используя функции: *reset(), end(), prev(), next(), current()*.

```
$a = array(“one”, “two”,
“three”, “four”);
echo “Normal order:” ;
reset($a);
while ( current($a) ) {
   echo current($a).”, ”;
   next($a);
}
echo “<br/>Inverse order:” ;
end($a);
while ( current($a) ) {
   echo current($a).”, ”;
   prev($a);
}
```

```
/* Если надо просто просмотреть
структуру и значения элементов
массива, то можно
воспользоваться встроенной
функцией print_r()
*/
<?php
$a = array ('a' => 'apple',
     'b' => 'banana',
    'c' => array ('x','y','z')
);
print_r($a);
//!!!перемещает указатель массива
//в конец
```

При присваивании массива всегда происходит копирование значения. Чтобы копировать массив по ссылке, нужно использовать оператор ссылки:

```
$copy = $a; //создаём копию
$link = &$a; // $link  и $a – один и тот же массив
```

PHP: массивы

Существует множество других функций для работы с массивами:

```
bool sort(array &$arr [, int sort_flags] ) - сортирует элементы по
значению (ключи меняются)

bool asort(array &$arr [, int sort_flags] ) - сортирует элементы по
значению, но сохраняются связи между ключами и значениями

bool ksort(array &$arr [, int sort_flags] ) - сортирует элементы по
ключу, сохраняя связи между ключами и значениями

bool usort(array &$arr , cpm_function($a, $b)) - сортирует элементы,
используя пользовательскую функцию сравнения (1 = '>'; -1 = '<'; 0 =
'=')
```

… //много функций в интернете

PHP: работа с файлами

Работа с файломи в С стиле:
1) Открыть файл в определённом режиме
2) Записать и/или прочитать информацию из файла
3) Закрыть файл

```
resource fopen (string filename, string mode)-   mode : r, r+, w,
   w+, a, a+, x, x+;
bool fclose (resource handler);

integer fwrite (resource handler, string text [, integer length]) –
возвращает колличество записанных байт.

string fread (resources handler, integer length)

bool feof ( resource handle ) - проверяет достигнут ли конец файла

integer filesize (string filename)

string fgetc ( resource handle ) - функция возвращает строку,
содержащую один символ. Возвращает FALSE по достижению конца файла.

string fgets ( resource handle [, int length] ) - возвращает строку
размером в length - 1. Если длина не указана, по умолчанию ее
значение равно 1 килобайту или 1024 байтам.
```

PHP: работа с файлами

```
<?php
$file = fopen(“test.txt”, “wt”);

for($i = 0; $I < 10; ++$i){
    fwrite($file, “$i – test\n”);
}

fclose($file);

$file = fopen(“test.txt”, “rt”);

while( !feof ($file)) {
    echo fgets($file).”\n”;
}
fclose($file);
?>
```

PHP: регулярные выражения

За основу взяты регулярные выражения Perl: /шаблон/модификатор
Отличия : http://ru2.php.net/manual/en/reference.pcre.pattern.differences.php

Функции:
int preg_match (string pattern, string subject [, array &matches])
– ищет в тексте subject совпадения с шаблоном pattern. Если указан массив matches, то он будет заполнен результатами поиска. $matches[0] будет содержать часть строки, соответствующую вхождению всего шаблона, $matches[1] - часть строки, соответствующую первой подмаске, и так далее. Возвращает колличество совпадений (0, 1);

```
<?php
$str = "http://www.php.net/index.html";

preg_match("/^(http:\/\/)?([^\/]+)/i", $str, $matches);

$host = $matches[2]; // “www.php.net”

// получить два последних сегмента имени хоста
preg_match("/[^\.]+\.[^\.]+$/", $host, $matches);

echo "domain name is: ".$matches[0]."\n"; // ”php.net”
?>
```

PHP: регулярные выражения

int preg_match_all (string pattern, string subject, array matches [, int flags]) – ищет все совпадения с шаблоном.

flags:

PREG_PATTERN_ORDER - упорядочивает результаты таким образом, что $matches[0] это массив полных совпадений с патэрном, $matches[1] это массив строк, совпавших с первым субпатэрном в скобках, и так далее. Используется по умолчанию.

PREG_SET_ORDER - Упорядочивает результаты таким образом, что $matches[0] это массив первого набора совпадений, $matches[1] это массив второго набора совпадений, и так далее.

PREG_OFFSET_CAPTURE - Если этот флаг установлен, для каждого возникшего совпадения будет возвращено дополнительное строковое смещение.

```
<?php
$str = "<b>example: </b><div align=left>this is a test</div>";
//модификатор U означает “нежадный” поиск
// http://ru2.php.net/manual/en/reference.pcre.pattern.modifiers.php
preg_match_all ("/<[^>]+>(.*)<\/[^>]+>/U", $str ,  $out,
PREG_PATTERN_ORDER);
print $out[0][0].", ".$out[0][1]."\n";  //<b>example: </b>, <div align=left>this
                                                    //is a test</div>
print $out[1][0].", ".$out[1][1]."\n";  // example: , this is a test
```

PHP: регулярные выражения

`mixed preg_replace (mixed pattern, mixed replacement, mixed subject [, int limit])` – выполняет поиск и замену регулярного выражения, возвращает subject после обработки.

replacement может содержать ссылку $n на n'нный подшаблон в скобках, где n от 0 до 99. $0 – ссылка на строку совпавшую со всем шаблоном.

```
$string = 'December 11, 2009';
$pattern = '/(\w+) (\d+), (\d+)/i';
$replacement = '$0  <=>  $3 $2 $1' ;
echo preg_replace($pattern, $replacement, $string)."\n";
//December 11, 2009  <=>  2009 11 December
```

Если subject это массив, то поиск и замена выполняются в каждом элементе subject, return-значение также будет массивом.
Если pattern и replacement являются массивами, то preg_replace() принимает соответствующие значения из каждого массива и использует их для выполнения поиска и замены в subject.

Модификатор /e делает так, что preg_replace() рассматривает параметр replacement как PHP-код после выполнения соответствующей замены ссылок.

```
preg_replace("/(<\/?)(\w+)([^>]*>)/e", "'$1'.strtoupper('\\2').'$3'",$html);
```

PHP:регулярные выражения

`array preg_split (string pattern, string subject [, int limit [, int flags]])` – возвращает массив подстрок из строки subject, которая разбита по разделителю, соответствующему шаблону pattern.

flags:
PREG_SPLIT_NO_EMPTY – вернуть только непустые строки

PREG_SPLIT_DELIM_CAPTURE – извлечь и вернуть выражение,заключенное в круглые скобки в разделителе

PREG_SPLIT_OFFSET_CAPTURE -добавляет к каждой подстроке её позицию в исходной строке

```
$string = 'One123Two34345Three452345';
$pattern = "/[\d]+/";
print_r(preg_split($pattern, $string, -1,PREG_SPLIT_NO_EMPTY));
//Array
//(
//    [0] => One
//    [1] => Two
//    [2] => Three
//)
```

PHP:инкапсуляция

Инкапсуляция — свойство языка программирования, позволяющее объединить данные и код в объект и скрыть реализацию объекта от пользователя.

```
//simpleclass.php
<?php
class SimpleClass
{
// объявление и инициализация
public $var = 'default value';

// определение метода
function displayVar() {
    echo $this->var;
}

}
?>
```

```
<?php
require_once
'simpleclass.php;

//создание экземпляра класса
$myClass = new
SimpleClass();

//выведет: default value
$myClass->displayVar();

$myClass->var = 'new value';

//выведет: new value
$myClass->displayVar();
?>
```

Переменные разрешается инициализировать только константами.

PHP:инкапсуляция

Поля и методы могут быть объявлены как:

- `public` – доступны внутри и снаружи класса
- `private` – доступны только внутри класса в котором объявлены
- `Protected` – доступны внутри класса и в производных от него классах
- без объявления для методов и var для полей - эквивалентно public

```
<?php
class Test {
    private $myInt = 0;
    public function get(){
        return $myInt;
    }
    public function set($newVal) {
        $this->myInt = $newVal;
    }
}
?>
```

PHP:инкапсуляция

При создании экземпляра класса автоматически вызывается конструктор класса – метод __construct(). Когда освобождается последняя ссылка на объект, перед высвобождением памяти, занимаемой этим объектом, вызывается метод __destruct().

```
//constr_destr.php
class ConstructDestruct {
    private $text;
    function __destruct()  {
        echo "Деструктор:{$this->text}\n";
    }
    function __construct() {
            echo "Конструктор\n";
    }
    function __clone() {
        $this->text = "clone";
        echo "Конструктор копирования\n";
    }
}
```

```
require 'constr_destr.php';

$test = new ConstructDestruct;
$clone = clone $test;
unset($test);
unset($clone);

//выведет
Конструктор
Конструктор копирования
Деструктор
Деструктор clone
```

PHP:инкапсуляция

В PHP 4 объекты обрабатываются также, как и простые типы данных, что приводит к копированию объекта оператором '='. В PHP 5 такого не происходит, так как каждый объект получает свой собственный числовой идентификатор (handle), а оператор '=' копирует не сам объект, а лишь его идентификатор.

```
<?php
     class MyClass {
         var $property;
     }
     $obj1 = new MyClass();
     $obj1->property = 1;
     $obj2 = $obj1;
     $obj2->property = 2;
     echo $obj1->property; // Выводит '1' в PHP 4 , а в PHP 5 – '2'
     echo $obj2->property; // Выводит '2'
?>
```

Такми образом в PHP 5 объекты копируются по ссылке.

PHP:инкапсуляция

Классы поддерживают статические поля и методы:

```
<?php
class Test {
    public static $var = 0;
    Function getVar{
        return Test::$var;
    }
}

$t1 = new Test();
Test::$var += 10;
$t2 = new Test();
echo “t1 = {$t1->getVar()}; t2 = {$t1->getVar()}”; //выведет : t1 =
                                                    // 10; t2 = 10
?>
```

Статические поля не принадлежать ни одному из объектов класса. Изменение такого поля где-либо приводит к изменению этого поля во всех остальных объектах данного класса. Кроме этого, становится возможным обратиться к такому полю вне контекста объекта. Однако, не допускается обращение к статическому полю через указатель $this.

PHP:инкапсуляция

Статические методы могут быть выполнены без создания экземпляра класса:

```
Class Counter {
    private static $count = 0;

    function __construct() {
        Counter::$count++;
    }

    function __destruct() {
        Counter::$count--;
    }

    static function getCount() {
        return Counter::$count;
    }
}

…// создание/удаление объектов Counter
echo Counter::getCount();
```

PHP:инкапсуляция

В PHP 5 появились специальные методы *__get()* и *__set()*:

```
<?php
class Product {
    private $properties; //будет использоваться как ассоциативный массив

    function __set($name, $value) {
        echo "Задаём свойство $name = $value";
        $this->properties[$name] = $value;
    }

    function __get($name) {
        echo “Читаем свойство $name”;
        return $this->properties[$name];
    }
}

$obj = new Product();
$obj->type = “book”; // выведет 'Задаём свойство type = book'
$productType = $obj->type;   // выведет 'Читаем свойство type'
echo $productType; // выведет 'book”;
?>
```

PHP:инкапсуляция

Методы *__toString()* и *__call()*:

//product.php

```
<?php
class Product {
    private $prop;
        function __set($name, $value) {
        $this->prop[$name] = $value;
    }
        function __get($name) {
                return $this->prop[$name];
    }
    function __toString() {
        $str =  "Product:\n";
        foreach ($this->prop as $key => $val){
            $str .= "$key  -  $val  \n";
        }
        return $str;
    }
    function __call($name, $params) {
        echo "Unknown method: $name\n";
    }
}
```

```
<?php
require “product.php”;

$obj = new Product();
$obj->type = “book”;
$obj->name = “PHP”;

echo $obj;
//выведет:
//  Product:
//  type - book
//  name – PHP

$obj->smth();
//выведет:
// Unknown method: smth
?>
```

PHP:наследование

В PHP есть наследование:

```
<?php
class Base{
    public $first;
    private $second;
    protected $third;
    function __construct($f, $s, $t){
        $this->first = $f;
        $this->second = $s;
        $this->third = $t;
    }
    function  showBase() {
        echo $this->first;
        echo $this->second;
        echo $this->third."\n";
    }
}
```

```
class Derived extends Base {
    function __construct() {
            parent::__construct(1,
2, 3);
    }
    function  showDerived() {
        echo $this->first;
        echo $this->second; //нет
                                //такой
        echo $this->third."\n";

    }
}

$der = new Derived();
$der->showDerived(); // 13
$der->showBase(); //123
?>
```

Если необходимо вызвать конструктор или деструктор базового класса, то необходимо это делать явно, через указатель parent.

PHP:наследование

```
<?php
class Base{
    function show() {
        echo “ Base ”;
    }
}

class Derived extends Base {
    function show() {
        echo “ Derived ”;
        parent::show();
    }

}

$der = new Derived();
$der->show() // выведет: Derived Base
```

По умолчанию вызывается метод наследника, если в наследнике он не найден, то ведется поиск в базовом классе и т. д.

PHP не поддерживает множественное наследование.

PHP:наследование

```
<?php
class Base{
    final function show() {
        echo “ Base ”;
    }
}

class Derived extends Base {
//Fatal error: cannot override final method Base::show()
    function show() {
        echo “ Derived ”;
        parent::show();
    }

}
?>
```

Метод, при определении которого используется ключевое слово final, не может быть переопределен в классах, производных от данного класса.

Кроме этого, если final используется при определении самого класса, то порождение от него других классов становится невозможным.

Определение полей как финальных не допустимо (Fatal Error)

PHP:полиморфизм

В PHP 5 введены абстрактные классы и методы. Абстрактные методы не имеют реализации. Класс, который содержит такие методы, должен быть обязательно объявлен как абстрактный.

```
<?php
abstract class Base {
    private $var = 0;
    abstract function showName();
    function showBase() {
        echo “Base”;
    }
}

class Derived extends Base {
    function showName() {
        echo “Derived”;
    }
}

$obj1 = new Base; // вызовет ошибку
$obj2 = new Derived;
```

Экземпляр абстрактного класса создать невозможно.

Наследник обязан реализовать все абстрактные методы базового класса, иначе он сам будет абстрактным.

Абстрактныи класс может содержать поля и не абстрактные методы.

PHP:полиморфизм

В PHP есть поддержка интерфейсов. Интерфейс – это класс у которого есть только абстрактные методы и нет полей.

```
<?php
interface A {
    function showA() ;
}

interface B {
    function showB() ;
}

abstract class Base {
    abstract showBase();
}
```

```
class Derived extends Base implements A, B {
    function showA() {
        echo “A”;
    }
    function showB() {
        echo “B”;
    }
    function showBase() {
        echo “Base”;
    }
}
```

Класс может реализовывать любое колличество интерфейсов, но наследовать разрешено только один класс.

PHP:полиморфизм

В PHP есть возможность уточнения типа объекта, передаваемого в функцию.

```
<?php
interface Translator {
    function translate($text);
}
interface Dictionary {
    function translate($word);
}
class ERTranslator implements Translator {
    function translate($text){
        echo “ERTranslator”;
        ...
    }
}
class ERDict implements Dictionary{
    function translate($word){
        echo “ERDict”;
        ...
    }
}
```

```
//можно передать только объекты
//производные от Translator
function translateText(Translator $t,
$text){
    ...
    $res = $t->translate($text);
    return $res;
}

//можно передать только объекты
//производные от Dictionary
function translateWord(Dictionary $d,
$word){
    ...
    $res = $d->translate($word);
    return $res;
}

$ert = new ERTranslator();
$erd = new ERDict();
echo translateText($ert, '....');
echo translateWord($erd, '....');
```

PHP:полиморфизм

Оператор '**==**' вернёт TRUE для объектов если они являются экземплярами одного и того же класса, причём их поля должны иметь равные значения.
Оператор проверки идентичности '**===**' вернёт TRUE для объектов если если они ссылаются на один и тот же экземпляр класса.

Специальное ключевое слово instanceof в PHP 5 позволяет определять является ли объект экземпляром определенного класса, или же экземпляром класса производного от какого-либо класса.

```
if ($derive instanceof Base) {
          echo "{$derive} – объект созданный на основе интерфейса Base";
}
```

PHP:Reflection

Классы Reflection позволяют получать информацию о объектах языка непосредственно во время выполнения скрипта. Например, можно получить информацию о некотором объекте, включая его методы, их параметры, в каком файле находится описание объекта и даже какой документационный комментарий находится перед ним.

Reflection классы сделанны для каждого структурного объекта языка:

- `ReflectionFunction`
- `ReflectionParameter`
- `ReflectionMethod`
- `ReflectionClass`
- `ReflectionProperty`
- `ReflectionExtension`

Простой пример:

```
$ php --rf strlen
$ php --rc Exception
$ php –rc Reflection
```

Можно посмотреть структуру без документации

```
Reflection {
    public static void export ( Reflector $reflector [, string $return=false ] )

    public static array getModifierNames ( int $modifiers )
}
```

PHP:Reflection

```php
<?php
class A{
    private $str;
    function __construct($s = null){
        $str = $s;
        print "$s";
    }
}
//вывод написан справа
Reflection::export(new ReflectionClass("A"));

print_r (Reflection::getModifierNames(3));

//выведет
//Array
//(
//    [0] => abstract
//    [1] => static
//)
?>
```

```
Class [ <user> class A ] {
  @@ /home/ruslan/unix/- 2-8
  - Constants [0] {
  }
  - Static properties [0] {
  }
  - Static methods [0] {
  }
  - Properties [1] {
    Property [ <default> private
$str ]
  }
  - Methods [1] {
    Method [ <user, ctor> public
method __construct ] {
      @@ /home/ruslan/unix/- 4 - 7

      - Parameters [1] {
        Parameter #0 [ <optional>
$s = NULL ]
      }
    }
  }
}
```

PHP:Reflection

Список некоторых рефлекторов:

- ReflectionClass – получает информацию о классе
- ReflectionMethod – о методе
- ReflectionFunction – о функции

```
<?php
class A{
    private $str;
    function __construct($s = null){
        $str = $s;
        print "$s";
    }
    function hello(){
        print "Hello";
    }
}
$ref = new ReflectionClass("A");
$inst = $ref->newInstance("constructor\n");
echo $ref;
$inst->hello();

$method = $ref->getMethod("hello");
$method->invoke(new A);
?>
```

У каждого рефлектора есть много полезных функций. Информацию можно найти по ссылке: http://ru.php.net/manual/en/book.reflection.php