
WORK
TRUE
千锋教育
QIAN FENG EDUCATION
做真实的自己，用良心做教育
PHP+原型
H5教学部

目录

做真实的自己，用良心做教育

千锋教育
QIAN FENG EDUCATION

PHP语法介绍

PHP简介

PHP是目前最流行的服务器的脚本语言

PHP(全称：PHP：Hypertext Preprocessor，即"PHP：超文本预处理器")

是一种通用开源脚本语言

文件扩展名为.php

PHP开发环境:

1, PhpStorm: 编写php代码(也可以使用其他开发工具)

2, wampServer: 集成开发环境

3, MySQL: 数据库开发环境

PHP语法介绍

PHP变量

变量以 $ 符号开始，后面跟着变量的名称

```
$a = 6;
```

输出

```
$b = 6;
echo $b;
echo "<b>Hello world</b>";
```

PHP语法介绍

PHP数据类型

PHP和JS一样, 是弱类型:

Integer（整型）, Float（浮点型）,

Boolean（布尔型）,

String（字符串）,

Array（数组）,

Object（对象）,

NULL（空值）。

var_dump(): 函数返回变量的数据类型和值

```
$x = 5;
var_dump($x);
```

PHP语法介绍

整型

```
$a = 5;
var_dump($a);
```

浮点型

```
$a = 1.1;
var_dump($a);
```

布尔型

```
$a = true;
var_dump($a);
```

PHP语法介绍

空值

```
$x=null;
var_dump($x);
```

字符串

```
$str = "Hello world!";
echo $str;
```

字符串连接

```
$str1 = "hello";
$str2 = " world";
$str3 = $str1.$str2;
echo  $str3;
```

字符串长度

```
echo  strlen("hello world")
```

PHP语法介绍

数组

```
$arr = array("张三", "李四", "王五");
 var_dump($arr);
```

数组元素

```
echo $arr[0]." love ".$arr[1]." hate ".$arr[2];
```

数组长度

```
echo count($arr);
```

遍历数组

```
for($i=0; $i<count($arr); $i++){
    echo $arr[$i];
    echo "<br>";
}
```

PHP语法介绍

PHP运算符: 和JS运算符类似

PHP条件判断(if/switch): 和JS类似

print_r : 可以打印出复杂类型变量的值(如数组,对象)

```
$arr = array("Zhangsan", "Lisi", "Wangwu");
sort($arr);  //升序排序
print_r($arr);
```

PHP语法介绍

PHP运算符: 和JS运算符类似

PHP条件判断(if/switch): 和JS类似

print_r : 可以打印出复杂类型变量的值(如数组,对象)

```
$arr = array("Zhangsan", "Lisi", "Wangwu");
sort($arr);  //升序排序
print_r($arr);
```

PHP循环: while, do-while, for, foreach-as
其中foreach-as 可以用于遍历关联数组

PHP语法介绍

PHP函数: 和JS类似

1, 内建函数(即系统函数)

2, 自定义函数

```
function func($name) {
    echo $name;
}
func("张三");
```

PHP编写服务端接口

表单GET提交

html代码

```
<form action="demo2_GET.php" method="get">
        名字: <input type="text" name="username">
        年龄: <input type="text" name="age">
        <input type="submit" value="提交">
</form>
```

php代码

```
<h2>GET</h2>
姓名: <?php echo $_GET["username"]; ?> <br>
年龄: <?php echo $_GET["age"] ?>
```

PHP编写服务端接口

表单POST提交

html代码

```
<form action="demo3_POST.php" method="post">
    名字: <input type="text" name="username">
    年龄: <input type="text" name="age">
    <input type="submit" value="提交">
</form>
```

php代码

```
<h2>POST</h2>
姓名: <?php echo $_POST["username"]; ?> <br>
年龄: <?php echo $_POST["age"] ?>

<h2>$_REQUEST : 可以收集GET和POST发送的表单数据</h2>
姓名: <?php echo $_REQUEST["username"]; ?> <br>
年龄: <?php echo $_REQUEST["age"] ?>
```

PHP编写服务端接口

Ajax请求

js代码：使用ajax进行POST请求

php代码:

判断客户端采用GET还是POST请求

```
if($_SERVER["REQUEST_METHOD"] == "GET"){
    $name = $_GET["username"];
}
else if($_SERVER["REQUEST_METHOD"] == "POST"){
    $name = $_POST["username"];
}
```

PHP编写服务端接口

Ajax请求

php代码:

```
$username = $_POST["username"]; //从客户端获取的用户名
$password = $_POST["password"]; //从客户端获取的密码
class Res{
        public $status;
        public $msg;
        public $userid;
}
if($username=="zhangsan"&& $password=="123456"){
        $res = new Res();
        $res->status = 1;
        $res->msg = "success";
        $res->userid = "1001";
        echo json_encode($res);  //json编码(json序列化)
}
```

PHP编写服务端接口

PHP后台设置支持跨域

支持跨域访问

```
header('Access-Control-Allow-Origin: *');
```

原型

1, 原型是什么

原型, 英文名prototype是函数中一个自带的属性, 我们创建的每个函数都有一个prototype(原型)属性, 这个属性是一个对象.

2, 原型的作用

原型的作用是: 可以让同一个构造函数创建的所有对象共享属性和方法. 也就是说, 你可以不在构造函数中定义对象的属性和方法, 而是可以直接将这些信息添加到原型对象中。

原型

例如:

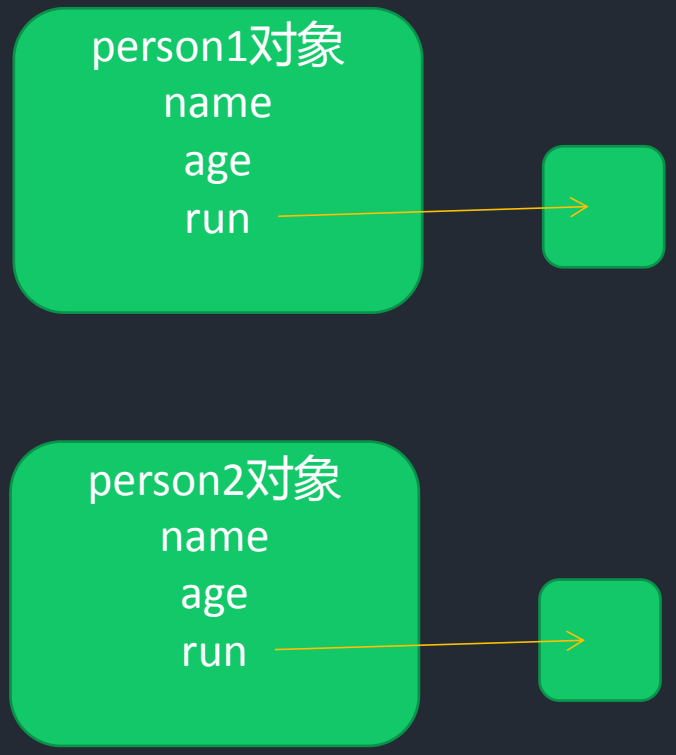
```
function Person() {}    //声明一个构造函数
Person.prototype.name =  “zhang”     //在原型里添加属性
Person.prototype.age = 100;
Person.prototype.show = function () {     //在原型里添加方法
    return this.name + this.age;
};
var person = new Person();
```

比较一下原型内的方法地址是否一致

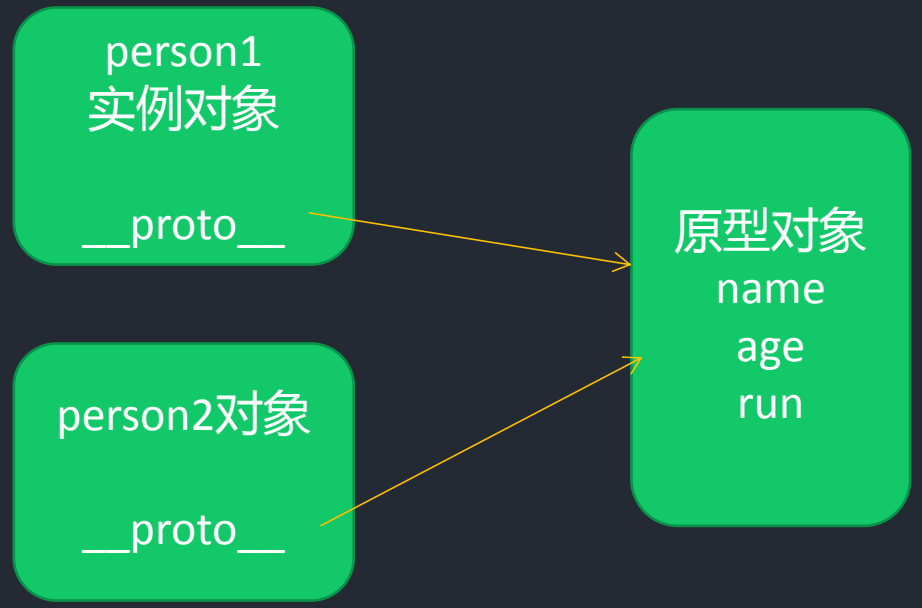
```
var person1 = new Person();
var person2 = new Person();
console.log(person1.show == person2.show);  //true，方法的引用地址一致
```

原型

构造函数中写属性和方法



原型中写属性和方法



原型

3, 原型的属性

constructor: 是原型的属性, 指向原型对象所属的构造函数;

```
console.log(Person.prototype.constructor === Box);
```

__proto__: 是对象的属性，指向构造函数的原型。

```
console.log(box1.__proto__);   //[object Object]
console.log(box1.__proto__ === Box.prototype);
```

isPrototypeOf(): 判断一个对象是否指向了该构造函数的原型对象

```
console.log(Box.prototype.isPrototypeOf(box));  //true, 实例对象都会指向
```

原型

原型模式的执行流程：

1.先查找实例对象里的属性或方法，如果有，立刻返回；

2.如果实例对象里没有，则去它的原型对象里找，如果有就返回；

例如:

```
function Person() {}
Person.prototype.name = "张三";

var person = new Person();
person.name = "李四";
console.log(person.name); //李四, 找到了实例对象的值
console.log(person.__proto__.name); //张三
```

原型

hasOwnProperty(): 判断实例对象中是否存在该属性

```
console.log(person.hasOwnProperty( ‘name’ ));  //实例对象里有则为true,否则为false
```

in操作符: 判断属性是否存在于该实例对象或者该对象的原型中

```
console.log( ‘name’  in person);     //true，存在实例中或原型中
```

如果要判断某属性是否存在原型中, 则可以根据in操作符和hasOwnProperty()来判断

```
function isInPrototype(obj, name){
    return (name in obj) && (!obj.hasOwnProperty(name));
}
var box = new Box();
console.log(isInPrototype(person1, "name")); //true，如果原型有,返回true
```

原型

4, 原型使用字面量的写法

```
function Person() {};
Person.prototype = {
    name: '张三',
    show: function () {
        return this.name;
    }
};
var person = new Person();
console.log(person.constructor == Person); //字面量方式，返回false，否则true
console.log(person.constructor == Object);  //字面量方式，返回true，否则false
```

如果想让字面量方式的constructor指向实例对象，那么可以这么做：

```
Person.prototype = {
    constructor: Person  //直接强制指向即可
};
```

原型

5, 内置对象的原型

```
console.log(Array.prototype.sort);           //sort 就是 Array 类型的原型方法
console.log(String.prototype.substring);  //substring 就是 String 类型的原型方法
```

使用原型可以给已有构造函数添加方法:

例如:

给String类型添加一个方法

```
String.prototype.addstring = function () {
    return this +  ‘，被添加了！’ ;  //this 代表调用的字符串
};
console.log( ‘张三’.addstring());    //使用这个方法
```

原型

1, 仅使用原型的缺点

单独使用原型来给对象添加属性和方法, 是有缺点的, 具体有以下两点:

1, 它省略了构造函数传参初始化这一过程, 带来的缺点就是初始化的值都是一致的

2, 原型对象共享的属性或者方法是公用的, 在一个对象修改后,会影响其他对象对该属性或方法的使用.

原型

例如:

```
function Person() {};
Person.prototype = {
    constructor : Person,
    name : '张三',
    age : 100,
    family : ['父亲', '母亲', '妹妹'],   //添加了一个数组属性
    run : function () {
        return this.name + this.age + this.family;
    }
};
var person1 = new Person();
person.family.push(“哥哥”);  //通过box1给原型中的family数组添加了一个元素'哥哥'
console.log(person1.run());
var person2 = new Person();
console.log(person2.run());   //共享带来的麻烦，也有'哥哥'了
```

原型

2, 构造函数+原型模式

使用构造函数添加私有属性和方法,

使用原型添加共享的属性和方法

优点:

1, 实例对象都有自己的独有属性

2, 同时共享了原型中的方法,最大限度的节省了内存

3, 支持向构造函数传递参数

原型

例如:

```
function Person(name, age) {
    this.name = name;
    this.age = age;
    this.family = ["爸爸", "妈妈"];
}
Person.prototype = {
    constructor: Person,
    show: function() {
        console.log("姓名:" + this.name + ", 年龄:" + this.age);
    }
}
```

- 上面创建不同的对象只能共享show方法;
- 这种混合模式很好的解决了传参和引用类型共享的问题.

原型

练习

1, 采用原型实现拖拽功能

2, 采用构造函数+原型实现拖拽功能

作业

1, 使用构造函数+原型模式实现萤火虫案例

2, 使用构造函数+原型模式实现烟花效果

THANK YOU

千锋教育
QIAN FENG EDUCATION

做真实的自己，用良心做教育