Master PHP DateTime

> It's a common task to work with date and time as a web developer. If you are still using functions like **strtotime** and **date** to work with date and time, you are missing out.

PHP provides a dedicated class **DateTime** for dealing with date and time. Most of people have been ignoring it, despite it has been available since PHP 5.2.

A couple of reasons why you should use **DateTime** over **strtotime** and **date**:

- **DateTime** is able to process more date string formats comparing to **strtotime**.
- Working with object is just easier than arbitrary functions. E.g. Comparison of two date is direct with **DateTime**. Whereas with **strtotime**, we need to convert dates to timestamps and compare them.
- **DateTime**'s object-oriented interface encapsulates a lot of back-end logic and provides a clean interface for us to use.

Instantiation

Instantiating an **DateTime** object is no different from instantiating any other objects in PHP. We instantiate an **DateTime** object via its constructor.

When no parameter is supplied, an **DateTime** object with current timestamp and the default timezone will be created. The default timezone is normally configured in php.ini file.

For example, to create a "now" **DateTime** object:

```
$currentDateTime = new DateTime();
```

Optionally we can supply a string, which represents a valid date and time, as the first parameter of the **DateTime**'s constructor. The default timezone will still be used since it is not specified.

A few examples of creating **DateTime** objects with valid time string:

```
$yesterday = new DateTime('yesterday');

$twoDaysLater = new DateTime('+ 2 days');

$oneWeekEarly = new DateTime('- 1 week');
```

The second parameter of the **DateTime**'s constructor allows us to specify a timezone. This parameter accepts a **DateTimeZone** object only.

For example, to create a **DateTime** object with Singapore timezone:

```
$yesterdayInSingapore = new DateTime('yesterday', new
DateTimeZone('Singapore'));
```

Of cause, we can also create a **DateTime** object with traditional date string:

```
$dateTime = new DateTime('2015-01-01 12:30:12');
```

Format

We will always want a specific format depends on the system we are building.

Formatting a **DateTime** object to get a custom string representation is straightforward. It's done via **DateTime**'s **format()** method.

DateTime::format() accepts a single string parameter. This string should contain options listed at PHP official page.

For example, to get a yyyy-dd-mm format outputted by **DateTime**, we can do:

```
$now = new DateTime();

$ymdNow = $now->format('Y-m-d');
```

We can create any desired format, because the format parameter provides all the possible options.

A few more examples:

```
print_r($now->format('jS F Y'));

print_r($now->format('ga jS M Y'));

print_r($now->format('Y/m/d s:i:H'));

// Output:
6th January 2016
1am 6th Jan 2016
2016/01/06 11:39:01
```

Comparison

When working with date string, we need to convert them to timestamp using **strtotime** in order to compare their value.

DateTime objects work with comparison operators out of box. We can directly compare two **DateTime** objects just like two numeric values.

A few examples of comparing two **DateTime** objects:

```
$today = new DateTime('today');
$yesterday = new DateTime('yesterday');

var_dump($today > $yesterday);
var_dump($today < $yesterday);
var_dump($today == $yesterday);

// Output
bool(true)
bool(false)
bool(false)
```

Sometimes, a Boolean value of comparing two dates objects is not sufficient. We may want to know the exact difference between two dates.

DateTime::diff is a function for getting the difference between two **DateTime** objects. This function returns a **DateInterval** object, which we can use to retrieve any desired interval format through its function **DateInterval::format**.

For example, to get days difference between $today and $yesterday, we can do:

```
$interval = $today->diff($yesterday);
echo $interval->format('%d day ago')

// Output
1 day ago
```

There are a number of interval properties in **DateInterval** class, you can check them out at PHP official site.

Manipulation

Though in reality, we can never manipulate time, with **DateTime**, we can actually do that. This means **DateTime** objects are mutable.

Addition

Adding to an existing **DateTime** is done through function **DateTime::add**. **DateTime::add** accepts a single **DateInterval** object parameter.

Let's demonstrate how addition works in a simple example:

```
$today = new DateTime('today');

echo $today->format('Y-m-d') . PHP_EOL;

$today->add(new DateInterval('P2D'));

echo $today->format('Y-m-d') . PHP_EOL;

// Output
2016-01-06
2016-01-08
```

Most of the code above is easy to understand. The only tricky part may be creating a **DateInterval** object. **DateInterval**'s constructor accepts a string that contains a valid interval specification.

An intercept of explanation for interval specification from PHP official site:

> The format starts with the letter P, for "period." Each duration period is represented by an integer value followed by a period designator. If the duration contains time elements, that portion of the specification is preceded by the letter T.

In our code above, we have used 'P2D', which standards for two days, to create a **DateInterval** object.

Subtraction

Similar to **DateTime::add**, subtraction is done through function **DateTime::sub**. **DateTime::sub** function accepts a **DateInterval** object just like **DateTime::add** function.

To subtract two days from $today, we can do:

```
$today = new DateTime('today');

echo $today->format('Y-m-d') . PHP_EOL;

$today->modify('-2 days');

echo $today->format('Y-m-d') . PHP_EOL;

// Output
2016-01-06
2016-01-04
```

Modification

Modification of a **DateTime** object is also possible with **DateTime::modify** function.

Comparing to **DateTime::add** function and **DateTime::sub** function, **DateTime::modify** function takes a different type of parameter to do the job. It accepts a date/time string. There is a wide range of support for the date/time string, you can check them out in details at PHP official site.

To subtract two days from $today with **DateTime::modify** function, we can do:

```
$today = new DateTime('today');

echo $today->format('Y-m-d') . PHP_EOL;

$today->modify('-2 days');

echo $today->format('Y-m-d') . PHP_EOL;

// Output
2016-01-06
2016-01-04
```

End

You have learned how to create, format, compare and even manipulate **DateTime** object so far. With what you have learned, you should be able to replace functions such as **strtotime** and **date** with **DateTime** object confidently in your future projects.

Sooner or later, you will be using **DateTime** objects for doing the same tasks over and over again. That has been the case for most of developers.

Tasks such as calculating someone's age based on his birthday or testing if a date is tomorrow are pretty common. PHP developers decided to consolidate various tasks that involving **DateTime** into some generic packages. Hence a number of packages related to **DateTime** were developed over the years.

Some outstanding ones are recommended by the community. We will encourage you to check out two of them as shown below. They are notably good. Because they not only provide a lot of help functions for dealing with **DateTime**, but also they are well documented.

- Carbon: A simple PHP API extension for DateTime. (Documentation link)

- Chronos: It provides a zero-dependency collection of extensions to the DateTime object. (Documentation link)

Glossary

Links

Links to mentioned resources.

- Characters recognized when using **DateTime::format** function:

  http://php.net/manual/en/function.date.php#refsect1-function.date-parameters
- Date interval specification:

  http://php.net/manual/en/class.dateinterval.php#dateinterval.props
- Supported date/time string in PHP:

  http://php.net/manual/en/datetime.formats.php
- Carbon: A simple PHP API extension for DateTime:

  http://carbon.nesbot.com/
- Chronos: It provides a zero-dependency collection of extensions to the DateTime object.

  http://book.cakephp.org/3.0/en/chronos.html

https://startutorial.com/view/master-php-datetime

PHP DateTime

Summary: in this tutorial, you'll learn how to work with the date and time in an object-oriented way.

Introduction to the PHP DateTime class

PHP provides a set of date and time classes that allow you to work with the date and time in an object-oriented way.

To create a new date and time object, you use the `DateTime` class. For example:

```
<?php

$datetime = new DateTime();

var_dump($datetime);
Code language: PHP (php)
```

Output:

```
object(DateTime)#1 (3) {
    ["date"]=> string(26) "2021-07-15 06:30:40.294788"
    ["timezone_type"]=>  int(3)
    ["timezone"]=>  string(13) "Europe/Berlin"
}
Code language: PHP (php)
```

The `DateTime` object represents the current date and time in the timezone specified in the PHP configuration file (`php.ini`)

To set a new timezone, you create a new `DateTimeZone` object and pass it to the `setTimezone()` method of the `DateTime` object:

```
<?php

$datetime = new DateTime();

$timezone = new DateTimeZone('America/Los_Angeles');
$datetime->setTimezone($timezone);

var_dump($datetime);
Code language: PHP (php)
```

Output:

```
object(DateTime)#1 (3) {
    ["date"]=> string(26) "2021-07-14 21:33:27.986925"
    ["timezone_type"]=> int(3)
    ["timezone"]=> string(19) "America/Los_Angeles"
}
Code language: PHP (php)
```

In this example, we create a new `DateTimeZone` object and set it to `"America/Los_Angeles"`. To get valid timezones supported by PHP, check out the timezone list.

To format a `DateTime` object, you use the `format()` method. The format string parameters are the same as those you use for the `date()` function. For example:

```
<?php

$datetime = new DateTime();
echo $datetime->format('m/d/Y g:i A');
Code language: PHP (php)
```

Output:

```
07/15/2021 6:38 AM
Code language: PHP (php)
```

To set a specific date and time, you can pass a date & time string to the `DateTime()` constructor like this:

```
<?php

$datetime = new DateTime('12/31/2019 12:00 PM');
echo $datetime->format('m/d/Y g:i A');
Code language: PHP (php)
```

Or you can use the `setDate()` function to set a date:

```
<?php

$datetime = new DateTime();
$datetime->setDate(2020, 5, 1);
echo $datetime->format('m/d/Y g:i A');
Code language: PHP (php)
```

Output:

```
05/01/2020 6:42 AM
Code language: PHP (php)
```

The time is derived from the current time. To set the time, you use the `setTime()` function:

```
<?php

$datetime = new DateTime();
$datetime->setDate(2020, 5, 1);
$datetime->setTime(5, 30, 0);

echo $datetime->format('m/d/Y g:i A');
Code language: PHP (php)
```

Output:

```
05/01/2020 5:30 AM
Code language: PHP (php)
```

Since the `setDate()`, `setTime()`, and `setTimeZone()` method returns the `DateTime` object, you can chain them like this which is quite convenient.

```
<?php

$datetime = new DateTime();
echo $datetime->setDate(2020, 5, 1)
```

```
    ->setTime(5, 30)
    ->setTimezone(new DateTimeZone('America/New_York'))
    ->format('m/d/Y g:i A');
Code language: PHP (php)
```

Output:

```
04/30/2020 11:30 PM
Code language: PHP (php)
```

Creating a DateTime object from a string

When you pass the date string '06/08/2021′ to the `DateTime()` constructor or `setDate()` function, PHP interprets it as `m/d/Y`. For example:

```
<?php

$datetime = new DateTime('06/08/2021');
echo $datetime->format('F jS, Y');
Code language: PHP (php)
```

Output:

```
June 8th, 2021
Code language: PHP (php)
```

If you want to pass it as August 6th, 2021, you need to use the – or . instead of /. For example:

```
<?php

$datetime = new DateTime('06-08-2021');
echo $datetime->format('F jS, Y');
Code language: PHP (php)
```

Output:

```
August 6th, 2021
Code language: PHP (php)
```

However, if you want to parse the date string '06/08/2021′ as d/m/Y, you need to replace the / with – or . manually:

```
<?php

$ds = '06/08/2021';
$datetime = new DateTime(str_replace('/', '-', $ds));

echo $datetime->format('F jS, Y');
Code language: PHP (php)
```

Output:

```
August 6th, 2021
Code language: PHP (php)
```

A better way to do it is to use the `createFromFormat()` static method of the `DateTime` object:

```
<?php
```

```
$ds = '06/08/2021';
$datetime = DateTime::createFromFormat('d/m/Y', $ds);

echo $datetime->format('F jS, Y');
Code language: PHP (php)
```

In this example, we pass the date format as the first argument and the date string as the second argument.

Note that when you pass a date string without the time, the `DateTime()` constructor uses midnight time. However, the `createFromFormat()` method uses the current time.

Comparing two DateTime objects

PHP allows you to compare two DateTime objects using the comparison operators including >, >=, <, <=, ==, <=>. For example:

```
<?php

$datetime1 = new DateTime('01/01/2021 10:00 AM');
$datetime2 = new DateTime('01/01/2021 09:00 AM');

var_dump($datetime1 < $datetime2); // false
var_dump($datetime1 > $datetime2); // true
var_dump($datetime1 == $datetime2); // false
var_dump($datetime1 <=> $datetime2); // 1
Code language: PHP (php)
```

Calculating the differences between two DateTime objects

The `diff()` method of the `DateTime()` object returns the difference between two `DateTime()` objects as a `DateInterval` object. For example:

```
<?php

$dob = new DateTime('01/01/1990');
$to_date = new DateTime('07/15/2021');

$age = $to_date->diff($dob);
var_dump($age);
Code language: PHP (php)
```

Output:

```
object(DateInterval)#3 (16) {
    ["y"]=>    int(31)
    ["m"]=>    int(6)
    ["d"]=>    int(14)
    ["h"]=>    int(0)
    ["i"]=>    int(0)
    ["s"]=>    int(0)
    ["f"]=>    float(0)
    ["weekday"]=>    int(0)
    ["weekday_behavior"]=>    int(0)
    ["first_last_day_of"]=>    int(0)
    ["invert"]=>    int(1)
    ["days"]=>    int(11518)
```

```
    ["special_type"]=>    int(0)
    ["special_amount"]=>    int(0)
    ["have_weekday_relative"]=>    int(0)
    ["have_special_relative"]=>    int(0)
}
Code language: PHP (php)
```

The `DateInterval` represents the differences between two dates in the year, month, day, hour, etc. To format the difference, you use the `DateInterval`'s format. For example:

```
<?php

$dob = new DateTime('01/01/1990');
$to_date = new DateTime('07/15/2021');

echo $to_date->diff($dob)->format('%Y years, %m months, %d days');
Code language: PHP (php)
```

Output:

```
31 years, 6 months, 14 days
Code language: PHP (php)
```

Check out all the DateInterval format parameters.

Adding an interval to a DateTime object

To add an interval to date, you create a new `DateInterval` object and pass it to the `add()` method. The following example adds 1 year 2 months to the date `01/01/2021`:

```
<?php

$datetime = new DateTime('01/01/2021');
$datetime->add(new DateInterval('P1Y2M'));
echo $datetime->format('m/d/Y');
Code language: PHP (php)
```

Output:

```
03/01/2022
Code language: PHP (php)
```

To format a date interval, you use the date interval format strings.

To subtract an interval from a `DateTime` object, you create a negative interval and use the `add()` method.

Summary

- Use the `DateTime` class to work with the date and time.
- Use the `DateTimeZone` class to work with time zones.
- Use the comparison operators to compare two `DateTime` objects.
- Use the `diff()` method to calculate the difference between to `DateTime` objects.
- Use the `DateInterval` class to represent a date and time interval.

- Use the `add()` method to add an interval to or subtract an interval from a `DateTime` object.

https://www.phptutorial.net/php-oop/php-datetime/