livingmarkup

Feb 05, 2021

Project Information

LivingMarkup is an PHP implementation of a LHTML parser.
It is a powerful and flexible way to build dynamic web pages.

```
<?php
use LivingMarkup\Factory\ProcessorFactory;

$processor = ProcessorFactory::getInstance();

$processor->addElement([
'xpath' => '//partial',
'class_name' => 'Partial\{name}'
]);

$processor->addRoutine([
'method' => 'onRender',
'execute' => 'RETURN_CALL'
]);

$processor->parseBuffer();
?>
<html lang="en">
<partial name="Alert" type="success">
    This is a success alert.
</partial>
</html>
```

CHAPTER 1

Installation

Get an instance of LivingMarkup up and running in less than 5 minutes.

LivingMarkup is available on Packagist.

Install with Composer:

```
composer require ouxsoft/livingmarkup
```

That's it. You're done! Please take the rest of the time to read our docs.

CHAPTER 2

Contribute

- Issue Tracker: https://github.com/ouxsoft/LivingMarkup/issues
- Source Code: https://github.com/ouxsoft/LivingMarkup

CHAPTER 3

Navigation

3.1 Routines

Routines are methods that are automatically called by the processor during run time.

During a routine call all instantiated elements featuring the routine's method are called.

3.1.1 Prefix

It is recommended to establish a naming convention for routines that distinguish them from other methods.

Often packages choose to prefix Routine methods with the word `before`, `on`, or `after` followed by an explanation of the stage of execute.

For example:

Parameter	Comments
`beforeLoad`	Execute before object data load
`onLoad`	Execute during object data load
`afterLoad`	Execute after object data loaded
`beforeRender`	Execute before object render
`onRender`	Execute during object render
`afterRender`	Execute after object rendered

3.2 Elements

Elements are the working bees of LivingMarkup.

Elements are objects that process DOMElements.

Much of how an DOMElement is processed is deteremined by their class.

During the object's construction, the element receives arguments that were found in both the DOMElement's attributes and child `arg` DOMElements from `Engine`.

3.2.1 Element Development

New elements are easy to make. Simply create a class that extends the abstract class `\LivingMarkup\Element\AbstractElement`.

Make sure to add it to the `Configuration` to active the element.

Example

The basic syntax of a element class is shown below.

```
<?php

namespace LivingMarkup\Elements;

class HelloWorld extends \LivingMarkup\Element
{
    public function onRender()
    {
        return 'Hello, World';
    }
}
```

3.3 Configuration

The `Configuration` class is responsible for the instructions that explain to the `Builder` how to build the LHTML `Document`. These instructions can be set by modifying the config file that is loaded.

3.3.1 Config File

Configurations can be loaded from a file or created using an object. Below is an example of a config file.

Example

```
{
  "version": 3,
  "elements": [
    {
      "xpath": "//bitwise",
      "class_name": "LivingMarkup\\Test\\Bitwise"
    }
  ],
  "routines": [
    {
      "method": "beforeLoad",
      "description": "Execute before object data is loaded"
    },
```

(continues on next page)

(continued from previous page)

```
    {
      "method": "onLoad",
      "description": "Execute when object data is loading"
    },
    {
      "method": "afterLoad",
      "description": "Execute after object data is loaded"
    },
    {
      "method": "beforeRender",
      "description": "Execute before object is rendered"
    },
    {
      "method": "onRender",
      "description": "Execute while object is rendering",
      "execute": "RETURN_CALL"
    },
    {
      "method": "afterRender",
      "description": "Execute after object is rendered"
    }
  ]
}
```

Autoloading

If a config file has been specified during construction, it will be loaded.
If a config file has not been specified the `Configuration` class tries to load a `config.json` file if present.

If the `config.json` is not present, the `Configuration` will try

to load the | packaged config `config.dist.json` file.

Parameters

Parameter	Comments
`version`	Indicates the file structure to the `Configuration` for stability purposes.
`elements`	An array containing the types of elements to load at runtime. Each type contains contain an array with a `name`, a `class_name`, and a `xpath` expression.
`element:*:name`	Defines what the elements is named.
`element:*:xpath`	Specifies exactly how find DOMElements to initialize as elements. Xpath expressions are a powerful syntax for searching within a the `Document` for DOMElements.
`element:*:class_name`	Specifies which class to instantiate the DOMElement as. The `class_name` provided must refer to a class that extends the abstract `Element` class. The class name may feature a `{name}` variable which is automatically populated by the DOMElement's name attribute during runtime.
`routines`	An array containing automated method calls that will be made to all Elements during runtime. The order of items in this array determines the order of execution.
`routines:*:name`	The exact name of the method being executed.
`routines:*:order`	An explanation of what the method is doing that indicates its order.
`routines:*:action`	Determines whether the method should be ran differently. Currently, the following commands are supported * RETURN_CALL - The output of the method will replace the DOMElement in the DOMDocument. Is optional
`markup:`	String containing the actual LHTML that will be parsed by the `Builder`. This field is typically omitted from the config file and is instead appended to `Configuration` during runtime, often by the `Autoloader`.

3.4 Security

3.4.1 Escaping HTML and XSS

It is the responsibility of the library client to escape HTML to avoid XSS. This library itself will not alter its input.

3.4.2 Disable Entity Loader

You may want to choose to disable external entities.

```
libxml_disable_entity_loader(true);
```

For more information, see PHP Security Injection Attacks