Index of /~jharvard/S75-Sections/7-09_Objects_XPath_Regex

Name	Last modified	Size	Description
Parent Directory		-	
06_section_overview.php	08-Jul-2012 17:40	580	
07_objects.php	09-Jul-2012 17:02	402	
08_xpath.php	09-Jul-2012 19:16	1.2K	
app06/	08-Jul-2012 10:19	-	
menu.xml	09-Jul-2012 19:16	292	

Apache/2.2.22 (Fedora) Server at 192.168.119.128 Port 80

Previous Next

MVC Context from Last Section

In last section, we talked about `Model`, `View`, `Controller`, and introduced a simple MVC blogging application.

I'm going to continue using the application in examples today, so I will start with `app06`, continuing with last week's example. The diretory structure can be found by using the `tree` command in linux:

```
.
|-- 06_section_overview.php
`-- app06
    |-- controller
    |   `-- controller.php
    |-- model
    |   |-- data.xml
    |   |-- model.php
    |   `-- rss2sample.xml
    `-- view
        |-- content.php
        |-- header-content.php
        |-- pagination.php
        |-- post.php
        |-- top-bar.php
        `-- view.php
```

In general, the flow of control follows the normal flow of control for MVC applications, from Controller to Model back to Controller to View:

1. 06_section_overview.php

    1. controller/controller.php

    1. model/model.php
    2. model/data.xml
    3. model/model.php

1. controller/controller.php

1. view/view.php
2. view/header-content.php
3. view/view.php
4. view/top-bar.php
5. view/content.php
6. view/pagination.php
7. view/post.php
8. view/content.php
9. view/view.php

1. controller/controller.php

2. 06_section_overview.php

P. Myer Nore

Sat, 07 Jul 2012 14:24:21 GMT

Objects in PHP

- basic oop in php5

```
SimpleXMLElement Object
(
    [comment] => SimpleXMLElement Object
        (
        )

    [pizzas] => SimpleXMLElement Object
        (
            [item] => Array
                (
                    [0] => SimpleXMLElement Object
                        (
                            [name] => Tomato & Cheese
                            [large] => 9.75
                        )

                    [1] => SimpleXMLElement Object
                        (
                            [name] => Broccoli
                            [large] => 1085
                            [small] => 685
                        )

                )

        )

)
```

```
Array
(
    [0] => SimpleXMLElement Object
        (
            [name] => Broccoli
            [large] => 1085
            [small] => 685
        )
```

```
)
1
```

```
1085
```

```
$21.70
```

XPath and SimpleXMLElement

- SimpleXMLElement Documentation
- My notes on XPath in README.md
- example 1
- example 2

Previous Next

New Features of App06

In `app04`, we had what's called a `fat controller`; a lot of data model-specific information was computed in the controller. If you'll go back and look at last section's `app04/controller/controller.php`, you'll see how bloated the controller is.

Having a "fat controller" is usually an indication that your controller is doing things that your model should be doing. `In app06, we've moved this computation to the model`, using what may be a new function to you: `call_user_func( $function_name, $parameters)`.

As a result, the controller is responsible for requesting certain special variables names of the model - this is the model's API, or Application Programming Interface. At the top of the controller, we request any variables we need. Then we include the model, and the variables are filled in, ready to pass to the view.

P. Myer Nore

Sat, 07 Jul 2012 12:50:21 GMT

Index of /~jharvard/S75-Sections/7-09_Objects_XPath_Regex/app06/model

Name	Last modified	Size	Description
Parent Directory		-	
data.xml	09-Jul-2012 16:56	4.8K	
model.php	09-Jul-2012 19:16	3.5K	
rss2sample.xml	08-Jul-2012 10:19	2.5K	

Apache/2.2.22 (Fedora) Server at 192.168.119.128 Port 80

Index of /~jharvard/S75-Sections/7-09_Objects_XPath_Regex/app06/view

Name	Last modified	Size	Description
Parent Directory		-	
content.php	09-Jul-2012 12:30	317	
header-content.php	08-Jul-2012 10:19	378	
pagination.php	09-Jul-2012 08:40	799	
post.php	09-Jul-2012 12:19	235	
top-bar.php	09-Jul-2012 04:26	513	
view.php	08-Jul-2012 10:19	197	

Apache/2.2.22 (Fedora) Server at 192.168.119.128 Port 80

Previous Next

MVC Context from Last Section

In last section, we talked about `Model`, `View`, `Controller`, and introduced a simple MVC blogging application.

I'm going to continue using the application in examples today, so I will start with `app06`, continuing with last week's example. The diretory structure can be found by using the `tree` command in linux:

```
.
|-- 06_section_overview.php
`-- app06
    |-- controller
    |   `-- controller.php
    |-- model
    |   |-- data.xml
    |   |-- model.php
    |   `-- rss2sample.xml
    `-- view
        |-- content.php
        |-- header-content.php
        |-- pagination.php
        |-- post.php
        |-- top-bar.php
        `-- view.php
```

In general, the flow of control follows the normal flow of control for MVC applications, from Controller to Model back to Controller to View:

1. 06_section_overview.php

    1. controller/controller.php

    1. model/model.php
    2. model/data.xml
    3. model/model.php

1. controller/controller.php

1. view/view.php
2. view/header-content.php
3. view/view.php
4. view/top-bar.php
5. view/content.php
6. view/pagination.php
7. view/post.php
8. view/content.php
9. view/view.php

1. controller/controller.php

2. 06_section_overview.php

P. Myer Nore

Sat, 07 Jul 2012 14:24:21 GMT

Previous Next

Previous Section: Intro to MVC

Intro to MVC

Web Developers call dynamic websites `web applications` because from the coder's perspective, `dynamic websites are applications that run on a web server`.

Modern applications follow `design patterns`. One very common design pattern is Model, View, Controller, or MVC.

To follow MVC when writing an application, you *separate your code* according to the function of the code:

- `Model` code stores the application data and *models* your application's domain. For example, if you were writing a blogging application, the Model code would determine what data needs to be stored about a Blog Post, and take responsibility for storing and retrieving Blog Post data.
- `View` code produces anything *viewable* by the site visitor. For example, in a blogging application, the View code would turn a Blog Post's data into HTML and CSS, which would then be rendered in the site viewer's web browser and determine how the site appears to the visitor.
- `Controller` code receives all of site visitor's clicks and other input, and uses them to help the user *control* your application. For example, in a blogging site, if a user clicks on 'next post', the controller will receive that click, fetch the appropriate Model from the model code, send and send it to the View to turn it into the HTML that the site viewer's browser will render for the Site Viewer's viewing pleasure.

P. Myer Nore

Sat, 30 Jun 2012 12:39:21 GMT