

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Tuesday, February 9, 2010

A quick look at XHP

Facebook released a new PHP extension today that supports inlining XML. This is a feature known as XML Literals in Visual Basic. Go read their description here:
<http://www.facebook.com/notes/facebook-engineering/xhp-a-new-way-to-write-php/294003943919>

It adds an extra parsing step which maps inlined XML elements to PHP classes. These classes are core.php and html.php which covers all the main HTML elements. The syntax of those class definitions is a bit odd. That oddness is explained in the How It Works document.

Essentially, it lets you turn:

```
renderBaseAttrs() . ' />';  
}  
}
```

which extends html-element which in turn extends primitive. You can go read all the code for those yourself.

Note that to build XHP you will need flex 2.5.35 which most distros won't have installed by default. Grab the flex tarball and ./configure && make install it. Then you are ready to go.

I pointed Siege at my rather underpowered AS1410 SU2300 with the above trivial form examples. The plain PHP one and the XHP version. Ran each one 5 times benchmarking for 30s each time. The plain PHP one averaged around 1300 requests/sec. Here is a representative sample:

```
acer:~> siege -c 3 -b -t30s http://xhp.localhost/1.php  
** SIEGE 2.68  
** Preparing 3 concurrent users for battle.  
The server is now under siege...  
Lifting the server siege... done.  
Transactions: 38239 hits  
Availability: 100.00 %  
Elapsed time: 29.60 secs  
Data transferred: 3.97 MB  
Response time: 0.00 secs  
Transaction rate: 1291.86 trans/sec  
Throughput: 0.13 MB/sec  
Concurrency: 2.93  
Successful transactions: 38239  
Failed transactions: 0  
Longest transaction: 0.05  
Shortest transaction: 0.00
```

And the XHP version:

```
Transactions: 868 hits  
Availability: 100.00 %  
Elapsed time: 29.28 secs  
Data transferred: 0.08 MB  
Response time: 0.10 secs  
Transaction rate: 29.64 trans/sec
```

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Throughput: 0.00 MB/sec
Concurrency: 2.99
Successful transactions: 868
Failed transactions: 0
Longest transaction: 0.21
Shortest transaction: 0.05

So, a drop from 1300 to around 30 requests per second and latency from less than 10ms to 100ms. Running XHP on plain PHP is definitely out of the question. But, knowing that Facebook uses APC heavily and looking through the code (see the MINIT function in ext.cpp) we can see that it should play nicely with APC. So, re-running our PHP version of the form, now with APC enabled, that goes from 1300 to around 1460 requests per second, and no measurable latency:

Transactions: 43773 hits
Availability: 100.00 %
Elapsed time: 29.88 secs
Data transferred: 4.55 MB
Response time: 0.00 secs
Transaction rate: 1464.96 trans/sec
Throughput: 0.15 MB/sec
Concurrency: 2.93
Successful transactions: 43773
Failed transactions: 0
Longest transaction: 0.07
Shortest transaction: 0.00

The XHP version of the form now with APC enabled:

Transactions: 9707 hits
Availability: 100.00 %
Elapsed time: 29.45 secs
Data transferred: 0.94 MB
Response time: 0.01 secs
Transaction rate: 329.61 trans/sec
Throughput: 0.03 MB/sec
Concurrency: 2.97
Successful transactions: 9707
Failed transactions: 0
Longest transaction: 0.21
Shortest transaction: 0.00

Much better. But it is still around a 75% performance drop from 1460 to 330 and a ~10ms latency penalty. And yes, I did have a default filter enabled for these tests, so there was basic XSS filtering in place for the naked `$_POST['name']` variable in the plain PHP version. Of course, the default filtering would likely fail if the user data was used in a different context. And this 75% is obviously going to depend on what else is going on during the request. If you are spending most of your time calculating a fractal or waiting on MySQL, you may not notice XHP very much at all.

The bulk of the time is spent in all the tag to class interaction. If the core.php and html.php code was all baked into the XHP extension, it would be a lot quicker, of course. So, when you combine XHP with HipHop PHP you can start to imagine that the performance penalty would be a lot less than 75% and it becomes a viable approach. Of course, this also means that if you are unable to run HipHop you probably want to think a bit and run some tests before adopting this. If you are already doing some sort of external templating, XHP could very well be a faster approach.

Update: Here are the callgraphs.

The first is the plain PHP+APC version without XHP. And the second is the PHP+APC+XHP version. In the first you see all sorts of bits and pieces all across the stack getting cpu time.

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

In the second graph we see the effect of needing to copy and instantiate those core and html classes on every request. They are cached in APC, of course, but because of PHP's perfect sandbox, they cannot persist.

So we went from spending around 1% of our time in the executor to over 80%.

This isn't an entirely fair comparison, of course, since the plain version has close to no PHP to execute while the XHP version has 93 userspace classes to deal with. I would guess that XHP could get quite a boost if at least the primitives in core.php could be baked into the extension. Ideally all 93 basic html classes would be in C++ in the extension itself, but that would be a bit of a tedious undertaking.

Posted by Rasmus at 21:22

It's too bad that Facebook didn't make a version that lets me keep my current LAMP setup, and create Linux shared object library extensions that I can load in. This is why the 'phc' project interests me and I must experiment more. It will let me code stuff in PHP, compile into C++ and then binary, and then load as an extension inside of PHP.

Anonymous on Feb 10 2010, 00:14

Volomike? We are talking about XHP here. It lets you keep your LAMP setup. It's a plain old extension that you can compile and load into your existing PHP. Your comment sounds like it is directed at HipHop.

Anonymous on Feb 10 2010, 00:19

Hey Rasmus, glad to see you checking out XHP! Just a couple things I wanted to point out--

The "xhp:html-singleton" class is a misnomer. It's not a singleton in the classical sense, it's a singleton in the context of HTML. That is, an element like `
` or `<input type="text" name="name" />` may not have any children, so they are HTML singletons. You can still instantiate as many of these as you want :).

Also, if you grab the latest release tag of XHP (as opposed to master) all generated files are included, so you don't need bison, flex, or re2c.

Regarding performance, this is one of the most common questions I receive about XHP. It's true that XHP is 2-3 times slower than raw string concatenation, but for most applications, this is not an issue. For instance, I determined that most of our pages at Facebook spent at most 1-2% of wall time doing basic string ops. While APC is definitely a must (as it is for most websites), I wouldn't say that HipHop is. I believe it was only recently that our Lite site even switched to HipHop, and Lite uses XHP exclusively with pretty reasonable gen times. In real world applications XHP doesn't contribute much to the bottom line, but it is a non-zero cost. It's just a matter of picking a place in between brevity and performance, and I think where XHP landed is a pretty good compromise.

Anonymous on Feb 10 2010, 00:22

thanks for sharing us your in depth thoughts on XHP

Anonymous on Feb 10 2010, 02:19

(Note, using square vs angle brackets because of some issues with the form)

Let's observe this example:

```
echo
[form method="post"]
  What is your name?[br /]
  [input type="text" name="name" /]
  [input type="submit" /]
[/form];
```

Behind the scenes what it does, is it quickly sets up an object tree, an object for each node at least, then renders it back to exactly the same string we put up there, thus not gaining much.

Sure, it validates it, but if they instead opted to stick to fully parse-time validation for XML syntax, and instead converted it to a string literal for runtime execution, it would be a nice zero-performance-penalty tool for avoiding typos and other mistakes in your generated HTML content.

Furthermore, most PHP IDEs will start throwing syntax errors at you when they encounter any of the XHP extensions of syntax, which doesn't happen with other existing validating template engines that use a more classical syntax, i.e.:

```
[? $bar = "Hi" ?]
[foo]{$bar}[/foo]
```

instead of:

```
[?php
$bar = "Hi";
echo [foo]{$bar}[/foo]
?].
```

So as a result we have incompatibility with existing toolsets, slow performance, which I find hard to justify, given a lot of this can be done at parse time with the right modifications, including context-sensitive filtering that Rasmus mentioned.

Anonymous on Feb 10 2010, 06:21

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Hi

How did you generate those callgraphs?

Thanks

Anonymous on Feb 10 2010, 11:48

The callgraphs come from Callgrind and exported from kcachegrind.

Anonymous on Feb 10 2010, 13:09

Thankyou.

Anonymous on Feb 10 2010, 13:55

Hello Rasmus,

Do you trust "Siege" more than "ab" or any other benchmarking tool for benchmarking your web apps?

Thank You!

Anonymous on Feb 10 2010, 15:32

I for one cannot see any single true gain that XHP may bring. It's just another way of making simple things complicated. A PHP that is aware of the string type being echoed will never beat plain and blind PHP. It's not that it doesn't introduce any notable performance boost, but moreover it gives you the tightest bottleneck one could think of. 4-5 times slower? That's a lot. Could be used for an intranet application though.

@Marcel Laverdet: Even though the html-singleton is not a singleton class, it still is a class. You do not make a class for a tag. And you do not add inheritance on top of that. But you can :) And "2-3 times slower than raw string concatenation" is quite an issue for most apps.

PHP is not meant to be semantic in the way that XML and HTML are.

Anonymous on Mar 18 2010, 03:04

Rasmus,

with determination, I finally got an answer to my question that I asked you 2days ago. The answer is "variable function" ;) Today I bought a php 5.3 book. Finally came across the right one.

//kwasseem

(7php.com)

Anonymous on Apr 26 2010, 13:14