

hey!

Path::Dispatcher
kicks ass!

Shawn M Moore

Wednesday, June 23, 2010

Presented 2010-06-23, Ohio State University, Columbus Ohio

simple defects

SD

Wednesday, June 23, 2010

First a bit of background.

Jesse Vincent and Chia-Liang Kao wrote this distributed bug-tracker named SD (<http://syncwith.us>). It's built on Prophet which is your parachute out of the cloud. It's cool stuff.

command line interface

CLI

Wednesday, June 23, 2010

We had a command-line interface for SD, but it wasn't particularly good. So Jesse asked me to improve SD's CLI. Since he gives me a fat stack of cash every week, it was hard to say no. I like writing text-based user interfaces anyway so I probably would have done it for free. I'm only saying that because it's Jesse's three-year wedding anniversary today so I know--hope he's not here.

just do it

JFDI

Jifty

Wednesday, June 23, 2010

I started hacking up an ad-hoc CLI parser but before I got too far, I realized what I actually wanted was the URI dispatcher that is built into Jifty (<http://jifty.org>), our web framework.

```
$ sd '/ticket/update/10  
?summary=crap&due=today'
```

Wednesday, June 23, 2010

Jifty::Dispatcher is great for URLs but it obviously wouldn't work so well for parsing command-line arguments.

```
$ sd ticket update 10  
  --summary=crap  
  --due=today
```

Wednesday, June 23, 2010

Instead we want our users to write something like this. It's way more natural for a command-line interface.

```
/ticket/update/10  
?summary=crap  
&due=today
```

```
ticket update 10  
--summary=crap  
--due=today
```

Wednesday, June 23, 2010

But these are similar enough right? In both of these paths, we have some positional arguments with some delimiter, and we have some named arguments. Clearly there's a niche for a module that can handle both styles of paths. So I wrote one.

Path::Dispatcher

Wednesday, June 23, 2010

I called it Path::Dispatcher. What Path::Dispatcher does is, using a set of rules you provide, takes a path, does a rain dance, then returns a dispatch object of the matching rules that you can execute.

```
under ticket => sub {  
  on ['update', qr/^\d+$/] => sub {  
    say "Updating $2";  
  };  
};
```

Wednesday, June 23, 2010

Let's jump right into that. This is what a Path::Dispatcher rule looks like.

```
under ticket => sub {  
    on ['update', qr/^\d+$/] => sub {  
        say "Updating $2";  
    };  
};
```

Wednesday, June 23, 2010

"under" rules let you match a prefix of the path. There can be many rules inside an "under" rule. It's just a way of structuring the dispatcher a little more. It also improves efficiency by matching the prefix only once.

```
under ticket => sub {  
    on ['update', qr/^\d+$/] => sub {  
        say "Updating $2";  
    };  
};
```

Wednesday, June 23, 2010

"on" rules are more standard. Most of your dispatcher rules are going to be "on" rules. This "on" rule says match the string "update", and then match the rest of the path against a regex which looks for an integer.

```
under ticket => sub {  
  on ['update', qr/^\d+$/] => sub {  
    say "Updating $2";  
  };  
};
```

Wednesday, June 23, 2010

And finally if these rules match we run the codeblock.

```
mo money => sub {  
  mo ['problems', qr/^\d+$/] => sub {  
    say "Updating $2";  
  };  
};
```

Wednesday, June 23, 2010

Path::Dispatcher provides the matched path segments in the dollar-number variables. So the integer ID the regex matched will show up in \$2, even though it doesn't use capture parentheses. The string "update" will show up in \$1 but that's not particularly useful since it's a constant. I'll talk more about this dollar-number variable funny business in the bonus slides.

```
$dispatcher->run("ticket update 10");  
    Updating 10
```

Wednesday, June 23, 2010

So we give the dispatcher the path "ticket update 10" and it will run that codeblock.

```
$ENV{PATH_DISPATCHER_TRACE}=99;  
$dispatcher->run("ticket update 10");
```

```
Path::Dispatcher::Rule::Tokens=HASH(0x100834768) [ticket] matched against (ticket update 10)  
with (update 10) left over.  
((D - rule test.pl:10) - rule test.pl:9) [update,(?-xism:^(d+$)] matched against (update 10).  
((D - rule test.pl:10) - rule test.pl:9) [update,(?-xism:^(d+$)] running codeblock with path  
(update 10): {  
    package D;  
    use warnings;  
    use strict 'refs';  
    warn "Updating $2";  
}.  
}
```

Updating 10

Wednesday, June 23, 2010

We also have various levels of tracing if you're into that voyeur kinda thing.

dispatch
execution

Wednesday, June 23, 2010

Path::Dispatcher is split into two phases: dispatch then execution. Both of those are synonyms for kill, by the way.

```
my $dispatch = $dispatcher->dispatch(...);

if ($dispatch->has_matches) {
    $dispatch->run;
}
else {
    warn "404";
}
```

Wednesday, June 23, 2010

We can stop before executing the dispatch if we want and inspect the dispatch object that Path::Dispatcher built up. This is useful if you want to, say, check to ensure there are actually matches, and if not, let the user know. You can also do weirder things like see if a particular rule matched. Path::Dispatcher keeps track of everything you would want to ask for and if it doesn't please tell me so I can fix it.

Alternation

Always

Chain

CodeRef

Dispatch

Empty

Enum

Eq

Intersection

Metadata

Regex

Sequence

Tokens

Under

Wednesday, June 23, 2010

Path::Dispatcher has builtin support for a lot of different rule classes. I feel like that's the most important thing that distinguishes Path::Dispatcher from most of the other modules in its niche which are content with just regex matching.

Most of these rule classes have parameters. For example, Alternation and Intersection take as a parameter a set of rules and just make sure one or all of them match the same path.

Alternation

Always

Chain

CodeRef

Dispatch

Empty

Enum

Eq

Intersection

Metadata

Regex

Sequence

Tokens

Under

Wednesday, June 23, 2010

We saw a bunch of rules already in the ticket update example.

Alternation

Always

Chain

CodeRef

Dispatch

Empty

Enum

Eq

Intersection

Metadata

Regex

Sequence

Tokens

Under

Wednesday, June 23, 2010

There are two rule classes I like a lot.

The Dispatch rule takes as a parameter another Path::Dispatcher object. This turns out to be massively useful for layering dispatchers as part of subclassing or implementing a plugin architecture. And all the structure is preserved for more introspection.

Alternation

Always

Chain

CodeRef

Dispatch

Empty

Enum

Eq

Intersection

Metadata

Regex

Sequence

Tokens

Under

Wednesday, June 23, 2010

Metadata is special because it's a rule for letting you match against a path's metadata. Paths are not just a plain string, they also have a hash of metadata. For example in a web context, your metadata could be the PSGI (<http://plackperl.org>) environment hash. In a CLI context, it could be either the %ENV hash or the named parameters, or both, depending on what you care about. And of course Metadata also takes as a parameter a rule object for matching the metadata you care about!

Alternation

Always

Chain

CodeRef

Dispatch

Empty

Enum

Eq

Intersection

Metadata

Regex

Sequence

Tokens

Under

Application-specific rules!

Wednesday, June 23, 2010

And of course since this is all OOP, you can write your own rules that have application-specific logic. More on that later.

Path::Dispatcher

Wednesday, June 23, 2010

So that's an overview of what Path::Dispatcher does. Whattdya think?

Hurdles

Wednesday, June 23, 2010

Now the fun part of the talk. How Path::Dispatcher's design has evolved due to hurdles thrown at it by the complexity of Prophet and SD.

Layering

SD::Dispatcher

+

Prophet::Dispatcher

Wednesday, June 23, 2010

First up is the problem of layering. Prophet has a lot of dispatcher rules. SD has a lot of dispatcher rules too. We want to be able to combine them sensibly without duplication.

Layering



Wednesday, June 23, 2010

Yuval Kogman (nothingmuch) was visiting Boston in 2008. He was designing KiokuDB (<http://www.iinteractive.com/kiokudb/>) at the time and wanted feedback. I don't think I helped him much with that, but I was able to pick his brain enough for a good solution to the layering problem.

Layering

Path::Dispatcher::Rule::Dispatch

Wednesday, June 23, 2010

So the Dispatch rule is mostly Yuval's fault. His idea was to let a rule return multiple matches. After that everything just kind of made sense and the problem was completely solved. Obvious-in-hindsight solutions like that are just great!

So a rule of class Dispatch has a dispatcher object that it runs the path against and returns all of the matches. It's kind of like calling a superclass.

Layering

```
redispach_to  
    'Prophet::Dispatcher';
```

Wednesday, June 23, 2010

SD's dispatcher has a line like this which is kind of like calling a superclass method. SD has regular dispatcher rules before and after it. But this lets SD reuse all of Prophet's CLI logic.

Layering

```
for $plugin ($app->plugins) {  
    redispatch_to  
        $plugin->dispatcher;  
}
```

Wednesday, June 23, 2010

This solution turns out to be useful as a generic hook for extensibility. We do something like this in `Jifty::Dispatcher` for Jifty plugins. Jason May's Dataninja IRC bot (<http://github.com/jasonmay/dataninja>) does exactly this to let command classes hook into message dispatch.

Metadata

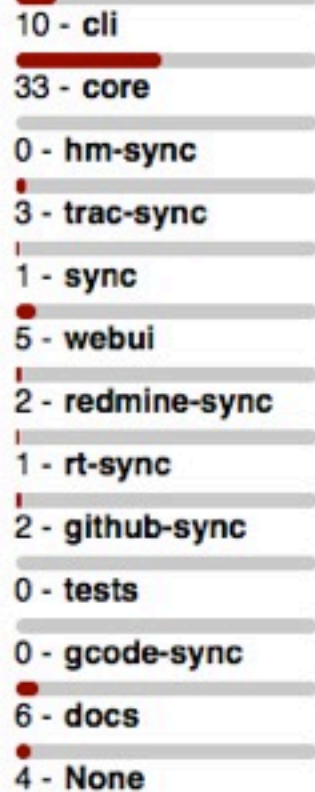
Path::Dispatcher::Rule::Metadata

Wednesday, June 23, 2010

The next hurdle was metadata. Or rather, the next hurdle was solved by the creation of the Metadata rule.

Project overview

Components



Statuses



Your active tickets for diana

id	Status	Milestone	Component	Owner	Reporter	Due	Created
445	new	diana	core		spang@bestpractical.com		2009-08-19 15:19:17
sd log spits out uninitialised errors, make them go away							
484	new	diana	core		spang@bestpractical.com		2009-08-21 18:33:33
make syncwith.us git repo public							
420	new	diana	redmine-sync		spang@bestpractical.com		2009-08-14 16:23:42
redmine tests assume redmine id = sd luid							
336	new	diana	core		jesse@bestpractical.com		2009-04-09 22:05:09
make it easier to see the full ticket summary from listings							
344	open	diana	webui		jesse@bestpractical.com	2009-06-19 04:46:11	2009-04-14 03:01:44
Can't unset a ticket due date from the web ui							
441	new	diana	core		spang@bestpractical.com		2009-08-18 17:23:47
sd/prophet testsuite should not leave crap all over /tmp							
424	new	diana	cli		spang@bestpractical.com		2009-08-15 13:20:03
server command cannot be stopped without killing sd shell							
497							
server does not allow edits from IPV6 loopback address							

Wednesday, June 23, 2010

SD has a nice web interface. It uses Path::Dispatcher for its own URI dispatching. The web interface is read-write. We like RESTy design, so our server cares about GET versus POST, etc.

Metadata

```
on "/GET/ticket/update" => sub { ... };  
on "/POST/ticket/update" => sub { ... };  
SD::Server::Dispatcher->run("/$method/$url");
```

Wednesday, June 23, 2010

We *could* have solved it with something like this but that is just frigging awful.

Metadata

```
under { method => 'POST' } => sub { ... };  
under { method => 'GET' }  => sub { ... };
```

```
my $path = Path::Dispatcher::Path->new(  
    path => $uri,  
    metadata => {  
        method => $request_method,  
    },  
);  
  
$dispatcher->run($path);
```

Wednesday, June 23, 2010

Instead, we can include a hash of metadata as part of the path. That way we can match on whatever extra data you want. So for matching a URI, we can match environment stuff.

Metadata

```
under { REQUEST_METHOD => 'POST' } => sub { ... };  
under { REQUEST_METHOD => 'GET' } => sub { ... };  
  
my $path = Path::Dispatcher::Path->new(  
    path => $uri,  
    metadata => \%ENV,  
);  
  
$dispatcher->run($path);
```

Wednesday, June 23, 2010

We could do something like this if we're using Plack. Whatever metadata makes sense for your application, just go for it.

Tab-completion

```
$ sd a<tab>
```

about

alias

aliases

attachment

Wednesday, June 23, 2010

Tab completion is friggin great. You want to be able to do that automatically for your Path-Dispatcher classes, right?

Tab-completion

```
$ sd he<tab>
```

```
$ sd help <tab>
```

about	authors	environment	log	settings
alias	clone	find	publish	summary-format
aliases	comment	help	pull	sync
attach	comments	history	push	ticket
attachment	config	init	search	ticket.summary-format
attachments	copying	intro	server	ticket_summary_format
author	env	list	setting	tickets

Wednesday, June 23, 2010

Tab completion needs to work at multiple levels of dispatcher rule.

Alternation

Always

Chain

CodeRef

Dispatch

Empty

Enum

Eq

Intersection

Metadata

Regex

Sequence

Tokens

Under

Wednesday, June 23, 2010

Each rule defines specific logic for tab completion.

Alternation

Always

Chain

CodeRef

Dispatch

Empty

Enum

Eq

Intersection

Metadata

Regex

Sequence

Tokens

Under

Wednesday, June 23, 2010

Tab completion doesn't make sense for a lot of rules. For example there's no way to tab complete on a coderef since the coderef can be doing **anything** it wants to match against the path. Also currently there's no tab completion logic for regular expressions though it's certainly possible for a very useful subset of regular expressions.

Alternation

Eq
Intersection

Dispatch

Sequence
Tokens
Under

Enum

Wednesday, June 23, 2010

There are also rules that don't actually do any matching themselves, they just manage other rules, dispatchers, etc.

Alternation

Eq
Intersection

Dispatch

Sequence
Tokens
Under

Enum

Wednesday, June 23, 2010

And finally these three red rule classes define actual real logic for tab completion. Turns out it's a pretty easy problem.

```
return grep {  
  
    my $partial = substr($value, 0,  
length($path));  
  
    $partial eq $path;  
  
} @{ $self->enum };
```

Wednesday, June 23, 2010

This is basically all the code you need for tab completion of "enum" rules. In practice it's a bit more complicated because enum rules can be case sensitive or not, but that's the gist of it. You just return a list of strings that represent the rest of the path.

(go look at the completion file, play with `_gencomp` etc)

App-specific rules

Handwritten cursive text, likely a signature or decorative flourish, consisting of two lines of stylized script.

Wednesday, June 23, 2010

App-specific rules are why Path-Dispatcher will rule them all.

Alternation

Always

Chain

CodeRef

Dispatch

Empty

Enum

Eq

Intersection

Metadata

Regex

Sequence

Tokens

Under

Wednesday, June 23, 2010

Since we have all these rule classes, it's going to `_have_` to be easy to use, and write, different rule classes.

`Prophet::CLI::Dispatcher::Rule::RecordId`

Wednesday, June 23, 2010

This is an example of an application-specific rule. Prophet is a database, so records have IDs. In the command-line interface you want to be able to create, read, update, and delete objects. So you're going to have to type in IDs. Since many commands look for IDs, we're going to want to factor out the record ID matching logic. Before `Path::Dispatcher` could really support custom rules, we just used a shared regular expression.

```
under ticket => sub {  
  on create => sub { ... };  
  on ['read', RecordId()] => sub {  
    ...  
  };  
  on ['update', RecordId()] => sub {  
    ...  
  };  
  on ['delete', RecordId()] => sub {  
    ...  
  };  
};
```

Wednesday, June 23, 2010

We can write something like this to match record IDs. The RecordID function returns an object of that Prophet-specific rule class.

```
sub RecordId {  
    return Prophet::CLI::Dispatcher::Rule::RecordId->new;  
}
```

Wednesday, June 23, 2010

That RecordId function is just sugar for creating a new object of this rule class, because that's a lot of typing.

```
on ['read', RecordId('ticket')] => sub {  
    ...  
};  
on ['update', RecordId('ticket')] => sub {  
    ...  
};  
on ['delete', RecordId('ticket')] => sub {  
    ...  
};
```

Wednesday, June 23, 2010

We can go one step further and limit the type of the record to tickets since we're under the "ticket" rule.

(go look at the rule in Prophet)

Learn More

<http://search.cpan.org/perldoc?Jifty::Dispatcher>

<http://bit.ly/d7RusL>

<http://github.com/miyagawa/plack-dispatching-samples>

<http://bit.ly/a6sa2e>

More Hurdles

Wednesday, June 23, 2010

There are some hurdles that Path-Dispatcher has not yet overcome but it will.

Sugar

```
under ticket => sub {  
    on create => sub { ... };  
  
    on ['update', qr/\d+/] => sub { ... };  
};  
  
on help => sub { ... };
```

Wednesday, June 23, 2010

Path-Dispatcher is a very Moosey module, so like Stevan was talking about last night in his keynote, it's designed to be a solid foundation first, and sugar is built on top of it. So it's bottom-up not top-down. This is important because I'm starting to hate the sugar Path-Dispatcher does have.

Sugar

```
under ticket => sub {  
  on create => sub { ... };  
  
  on ['update', qr/\d+/] => sub { ... };  
};  
  
on help => sub { ... };
```

Wednesday, June 23, 2010

Having to use "sub" here sucks because it's just used to figure out which rules are being stuck into the "under" and which are not. So the "create" and "update" rules are in the "under" rule, and the "help" is not. We use dynamic scope to figure out when the "under" stops.

Sugar

```
under ticket => sub {  
  on create => sub { ... };  
  
  on ['update', qr/\d+/] => sub { ... };  
};  
  
on help => sub { ... };
```

Wednesday, June 23, 2010

And this is using the legacy Tokens rule but it needs to be updated to use Sequence, Eq, and Regex. I think switching will break some users' code.

Sugar

`Path::Dispatcher::Declarative`

Wednesday, June 23, 2010

I'm halfway over this hurdle. This module, which gives us the sugar you just saw, used to be part of the core `Path::Dispatcher` distribution but now it's a separate dist. I want to encourage other sugar modules. I've been playing with `Devel::Declare` specifically for `Path::Dispatcher` sugar.

Extracting this into a new dist was actually a problem because it broke every `Path::Dispatcher`-using module on CPAN. In particular `Prophet` and `SD`. To fix it they just needed to upload new versions that explicitly depended upon `Path::Dispatcher::Declarative`. Lesson learned.



Wednesday, June 23, 2010

There's another problem, those dollar number variables.

\$100

```
on ['update', qr/\d+/] => sub {  
  warn "Updating $2";  
};
```

Wednesday, June 23, 2010

This \$2 is a core Path-Dispatcher feature. I feel like that's the wrong level for such a feature.

\$100

```
on ['update', qr/\d+/] => sub {  
  my $match = shift;  
  warn "Updating " . $match->capture(2);  
};
```

Wednesday, June 23, 2010

Instead the core feature should be something like this. That way it's still all just OOP and people can extend and change what's going on with these captures.

\$100

```
on ['update', qr/\d+/] => sub {  
  warn "Updating $2";  
};
```

Wednesday, June 23, 2010

The \$2 support can and should be provided by the sugar layer not the solid, core module.

(go look at Path::Dispatcher::Match)