Exploring Perl 6 Through Its Modules

Jonathan Worthington
@jnthnwrthngtn | 6guts.wordpress.com
Originally from England
Since then, I've lived in...

Spain

Slovakia
...and now I'm in Sweden!
I hack on Perl 6
Snippets ➔ Modules

Last time I was at YAPC::Asia, I gave a talk “Solved in Perl 6”

Lots of small snippets of code showing how to solve a range of problems in Perl 6

Perl 6 has been growing up. Thanks to the module ecosystem, we can look at how Perl 6 is put to use in larger, more practical examples
Multiple Dispatch

Write multiple subroutines or methods that have the same name, but take a different number or different types of parameters.

```perl
multi sub double(Int $x) { $x * 2 }
multi sub double(Str $x) { $x x 2 }

say double(21);       # 42
say double('can');    # cancan
```
JSON::Tiny: to-json

Turns simple Perl data structures into JSON – powered by multiple dispatch

```perl
# Real numbers simply stringify
multi to-json(Real:D $d) {
    ~$d
}
# Booleans become true or false literals
multi to-json(Bool:D $d) {
    $d ?? 'true' !! 'false';
}
```
JSON::Tiny: to-json

Turns simple Perl data structures into JSON – powered by multiple dispatch

# Strings need various bits of escaping
multi to-json(Str:D $d) {
    """ ~ $d.trans(
        [""", "\", "f", "n", "r", "t"] =>
        [""", "\\\\", "f", "n", "r", "t"]
    ).subst(/<-[\c32..\c126]>/,
    { ord(~$_).fmt(\u%04x) }, :g
    ) ~ """
}
JSON::Tiny: to-json

Turns simple Perl data structures into JSON – powered by multiple dispatch

# For anything that can be positionally
# indexed, emit a JSON array
multi to-json(Positional:D $d) {  
    '[ ' ~  
        $d.map(&to-json).join(', ', ') ~  
    ' ]';
}
JSON::Tiny: to-json

Turns simple Perl data structures into JSON – powered by multiple dispatch

# Any undefined values become a null
multi to-json(Any:U $) { 'null' }

# Error on unrecognized types
multi to-json(Any:D $s) {
  die "Can't serialize an object of type " ~ $s.^name
}
Grammars

Regexes have always been a key part of Perl

Perl 6 revises regex syntax, and takes them to the next level by adding support for grammars

The step up from regexes to grammars in small, but a grammar can stay clean and maintainable when scaled up to parse something complex
# Grammars are a kind of package, so we start
# with a package-like declaration.
grammar JSON::Tiny::Grammar;

# The TOP rule is the default entry rule when
# a grammar is used to parse something. We use
# rule to get automatic whitespace handling.
rule TOP { ^ [ <object> | <array> ] $ }
A grammar for parsing JSON

# Parsing of JSON objects ({ "foo": 42, … })
rule object { '{' ~ '}' } <pairlist>
rule pairlist { <pair> * % \, }
rule pair { <string> ':' <value> }

# Parsing of JSON arrays ([ 1, 2, 3, … ])
rule array { '[' ~ ']' } <arraylist>
rule arraylist { <value>* % [ \, ] }
JSON::Tiny: Grammar

A grammar for parsing JSON

# A proto-regex is a bit like an alternation, # but easily and cleanly extensible.
proto | token | value | { * }
--- | --- | --- | ---
token | value: sym<true> | { <sym> }
token | value: sym<false> | { <sym> }
token | value: sym>null> | { <sym> }
token | value: sym<object> | { <object> }
token | value: sym<array> | { <array> }
token | value: sym<string> | { <string> }
A grammar for parsing JSON

# String parsing – mostly just char classes.

```plaintext
token string {
   " ~ " ( <str> | \ \ <str_escape> )* 
}
token str {
   <-["\\\t\\n]+ 
}
token str_escape {
   ["\\/bfnrt] | u <xdigit>**4 
}
```
JSON::Tiny: Actions

Action methods are invoked for each grammar rule, and build a Perl 6 data structure

```perl
method value:sym<number>($/) {
    make +$/ . Str
}
method value:sym<string>($/) {
    make $<string> . ast
}
method value:sym<true>($/) {
    make Bool::True
}
```

Get passed the match object for the rule in $/>. This is just a few – there is about one per rule.
A simple sub drives the overall JSON to Perl 6 data structure process

# Create actions object, then pass it to the parse method on the grammar.
sub from-json($text) is export {
    my $a = JSON::Tiny::Actions.new();
    my $o = JSON::Tiny::Grammar.parse($text,
        :actions($a));
    return $o.ast;
}
Traits

A way to attach extra information and/or behavior to declarations (for example, of classes, subroutines, attributes...)

```perl
sub some-lvalue-sub() is rw {
    ...
}
```

The `is rw` is a trait attached to the subroutine declaration

Modules can provide extra traits too!
NativeCall

Provides an is native trait for routines

This indicates they are really implemented in native code, which should be loaded and called

The Perl 6 signature is introspected and used to work out how to pass the arguments

Write native bindings...without writing C!
NativeCall: Win32 API

Here’s an example of calling a Windows API using the NativeCall library

use NativeCall;

sub MessageBoxA(int32, Str, Str, int32) returns int32
is native('user32')
{
  *
}

MessageBoxA(0, "We can haz NCI?", "Hi!", 64);
NativeCall: DBlish

A simple database interface for Perl 6 that feels somewhat like Perl 5's DBI, but with an API that feels more natural in Perl 6

Supports SQLite, mysql and Pg

Drivers are built using the NativeCall library, meaning that they are written in pure Perl 6
DBlish: Pg driver example

```perl
sub PQexecPrepared(
    OpaquePointer $conn,
    Str $statement_name,
    Int $n_params,
    CArray[Str] $param_values,
    CArray[int] $param_length,
    CArray[int] $param_formats,
    Int $resultFormat)
returns OpaquePointer
is native('libpq')
{
    ...
}
```
NativeCall

Also supports...

Structures
Callbacks

More bindings are in progress, including an SDL one that already has enough to support implementing a Game::BubbleBreaker.
Meta-programming

The Perl 6 object system is based around a **MOP** (Meta-Object Protocol)

Can **customize the way objects work**, for example, by overriding method dispatch

Can even **add entire new features** that are not in core Perl 6, such as aspect orientation
Grammar::Tracer

Grammars are really just like classes

The various regexes, tokens and rules are just like methods in the class

Each call to a sub-rule is a method dispatch

Idea: use the MOP to hook method dispatch and trace which rules are being called
Grammar::Tracer

The aim is to output a tree diagram as the grammar calls down to sub-rules
Change the meaning of grammar

```perl
# Inherit from the default grammar package.
my class TracedGrammarHOW is Metamodell::GrammarHOW {
    ...
}

# Export our subclass as the default one for the
# "grammar" package declarator.
my module EXPORTHOW {
    EXPORTHOW::<grammar> = TracedGrammarHOW;
}
```
method find_method($obj, $name) {
    my $meth := callsame;
    $name eq any(<parse MATCH pos from>)
        ?? $meth
        !! -> $c, |args {
            # Output rule name here...
            my $result := $meth($obj, |args);
            # Output result here...
            $result
        }
}
say ('| ' x $indent) ~ BOLD() ~ $name ~ RESET();

(indent++)
my $result := $meth($obj, |args);
(indent--);

my $match := $result.MATCH;
say ('| ' x $indent) ~ '* ' ~ ($result.MATCH
?? colored('MATCH', 'white on_green') ~
summary($match)
!! colored('FAIL', 'white on_red'));
Grammar::Tracer

[Live Demo]
Rakudo

The Rakudo Perl 6 compiler is written largely in NQP (Not Quite Perl 6), a small Perl 6 subset.

The **CORE setting**, which provides many of the built-ins, is written in Perl 6.

This makes it relatively easy to hack on and extend the compiler.
Rakudo Debugger

A small core written in NQP

All the user facing stuff is built in Perl 6! 😊

Was built without having to extend the core of the compiler itself

Supports single stepping, breakpoints, evaluation, changing variables, etc.
Rakudo Debugger

[Live Demo]
Panda

Panda is a simple module installation tool for Perl 6 modules, written in Perl 6

panda install NativeCall
Where to learn more

To learn more about the modules discussed today – and many more – check out modules.perl6.org

The Rakudo debugger and Panda are included in the Rakudo Star releases; for more see rakudo.org
Where next?

We’re beyond the age of snippets

These days, it’s already very possible to build small tools and write modules in Perl 6

In the coming years, in addition to a growing module ecosystem, I hope to see larger applications developed in Perl 6
Thank you!

Questions?

Twitter
@jnthswrthngtn

Blog
http://6guts.wordpress.com