NAME

Type::Tiny::Manual::UsingWithMoose – how to use Type::Tiny and Type::Library with Moose

SYNOPSIS

```
{
   package Person;

   use Moose;
   use Types::Standard qw( Str Int );

   has name => (
      is      => "ro",
      isa     => Str,
   );

   my $PositiveInt = Int
      -> where( sub { $_ > 0 } )
      -> plus_coercions( Int, sub { abs $_ } );

   has age => (
      is      => "ro",
      isa     => $PositiveInt,
      coerce  => 1,
      writer  => "_set_age",
   );

   sub get_older {
      my $self = shift;
      my ($years) = @_;
      $PositiveInt->assert_valid($years);
      $self->_set_age($self->age + $years);
   }
}
```

DESCRIPTION

Type::Tiny is tested with Moose 2.0007 and above.

Type::Tiny type constraints have an API almost identical to that of Moose::Meta::TypeConstraint. It is also able to build a Moose::Meta::TypeConstraint constraint from a Type::Tiny constraint, and will do so automatically when needed. When Moose.pm is loaded, Type::Tiny will use Perl's `AUTOLOAD` feature to proxy method calls through to the Moose::Meta::TypeConstraint object. In short, you can use a Type::Tiny object pretty much anywhere you'd use a Moose::Meta::TypeConstraint and you are unlikely to notice the difference.

Per-Attribute Coercions

Type::Tiny offers convenience methods to alter the list of coercions associated with a type constraint. Let's imagine we wish to allow our `name` attribute to be coerced from an arrayref of strings.

```
has name => (
   is      => "ro",
   isa     => Str->plus_coercions(
      ArrayRef[Str], sub { join " ", @{$_} },
   ),
   coerce  => 1,
);
```

This coercion will apply to the `name` attribute only; other attributes using the `Str` type constraint will be unaffected.

See the documentation for `plus_coercions`, `minus_coercions` and `no_coercions` in Type::Tiny.

Optimization

The usual advice for optimizing type constraints applies: use type constraints which can be inlined whenever possible.

Defining coercions as strings rather than coderefs won't give you as much of a boost with Moose as it does with Moo, because Moose doesn't inline coercion code. However, it should still improve performance somewhat because it allows Type::Coercion to do some internal inlining.

See also Type::Tiny::Manual::Optimization.

Interactions with MooseX-Types

Type::Tiny and MooseX::Types type constraints should "play nice". If, for example, `ArrayRef` is taken from Types::Standard (i.e. a Type::Tiny−based type library), and `PositiveInt` is taken from MooseX::Types::Common::Numeric, then the following should "just work":

```
isa => ArrayRef[ PositiveInt ]

isa => PositiveInt | ArrayRef
```

SEE ALSO

For examples using Type::Tiny with Moose see the SYNOPSIS sections of Type::Tiny and Type::Library, and the Moose integration tests <https://github.com/tobyink/p5-type-tiny/tree/master/t/30-integration/Moose>, and MooseX-Types integration tests <https://github.com/tobyink/p5-type-tiny/tree/master/t/30-integration/MooseX-Types> in the test suite.

AUTHOR

Toby Inkster <tobyink@cpan.org>.

COPYRIGHT AND LICENCE

DISCLAIMER OF WARRANTIES