NAME

Type::Tiny::Manual::UsingWithMouse – how to use Type::Tiny and Type::Library with Mouse

SYNOPSIS

```
{
   package Person;

   use Mouse;
   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);
   }
}
```

STATUS

Mouse support in Type::Tiny was somewhat of an afterthought. It should work, but is not anywhere near as well-tested as Moo or Moose support.

DESCRIPTION

Type::Tiny is tested with Mouse 1.00 and above.

Type::Tiny type constraints have an API almost identical to that of Mouse::Meta::TypeConstraint. As a result, you can use a Type::Tiny object pretty much anywhere you'd use a Mouse::Meta::TypeConstraint and you are unlikely to notice the difference. (And Mouse is unlikely to notice the difference too!)

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

Mouse's built-in type constraints are implemented using XS and are stupidly fast. For many type constraints, if Type::Tiny notices Mouse is loaded early enough, Type::Tiny will borrow Mouse's XS subs.

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

Interactions with MouseX-Types

Type::Tiny and MouseX::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 MouseX::Types::Common::Numeric, then the following should "just work":

```
isa => ArrayRef[ PositiveInt ]

isa => PositiveInt | ArrayRef
```

SEE ALSO

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

AUTHOR

Toby Inkster <tobyink@cpan.org>.

COPYRIGHT AND LICENCE

DISCLAIMER OF WARRANTIES