Rocky Enterprise Linux 9.2 Manual Pages on command 'Type::Tiny::Manual::UsingWithClassTiny.3pm'

C:\>man Type::Tiny::Manual::UsingWithClassTiny.3pm

NAME

Type::Tiny::Manual::UsingWithClassTiny - use of Type::Tiny with Class::Tiny

MANUAL

Class::Tiny is an even-smaller-than-Moo class builder.

Let's translate the classic Horse class from Moo to Class::Tiny.

Moo:

```
 package Horse {
   use Moo;
   use Types::Standard qw( Str Num ArrayRef );
   use namespace::autoclean;

   has name       => ( is => 'ro', isa => Str, required => 1 );
   has gender     => ( is => 'ro', isa => Str );
   has age        => ( is => 'rw', isa => Num );
   has children   => (
     is       => 'ro',
```

```
    isa      => ArrayRef,
    default  => sub { return [] },
  );
}
```

Class::Tiny:

```
package Horse {
  use Class::Tiny qw( gender age ), {
    name     => sub { die "name is required"; },
    children => sub { return [] },
  };
  use Types::Standard qw( Str Num ArrayRef Dict Optional slurpy Any);
  use Type::Params qw( wrap_methods compile );
  use namespace::autoclean;

  # type checks
  wrap_methods(
    BUILD    => [Dict[
      name       => Str,
      gender     => Optional[Str],
      age        => Optional[Num],
      children   => Optional[ArrayRef],
      slurpy Any,
    ]],
    name     => [],
    gender   => [],
    age      => Optional[Num],
    children => [],
  );
}
```

What's going on here?

Well, Class::Tiny, after it has built a new object, will do this:

```
$self->BUILD($args);
```

(Technically, it calls "BUILD" not just for the current class, but for all parent classes too.) We can hook onto this in order to check type constraints for the constructor.

We use "wrap_methods" from Type::Params to wrap the original "BUILD" method (which doesn't exist, so "wrap_methods" will just assume an empty sub) with a type check for $args. The type check is just a Dict that checks the class's required and optional attributes and includes slurpy Any at the end to be flexible for subclasses adding new attributes.

Then we wrap the "name", "gender", and "children" methods with checks to make sure they're only being called as getters, and we wrap "age", allowing it to be called as a setter with a Num.

There are also a couple of CPAN modules that can help you out.

Class::Tiny::ConstrainedAccessor

Class::Tiny::ConstrainedAccessor creates a "BUILD" and accessors that enforce Type::Tiny constraints. Attribute types are passed to Class::Tiny::ConstrainedAccessor; attribute defaults are passed to Class::Tiny.

```
package Horse {
  use Types::Standard qw( Str Num ArrayRef );
  use Class::Tiny::ConstrainedAccessor {
    name     => Str,
    gender   => Str,
    age      => Num,
    children => ArrayRef,
```

```
  };
  use Class::Tiny qw( gender age ), {
    name     => sub { die "name is required"; },
    children => sub { return [] },
  };
}
```

Class::Tiny::Antlers

Class::Tiny::Antlers provides Moose-like syntax for Class::Tiny, including support for "isa". You do not also need to use Class::Tiny itself.

```
package Horse {
  use Class::Tiny::Antlers qw(has);
  use Types::Standard qw( Str Num ArrayRef );
  use namespace::autoclean;

  has name       => (
    is        => 'ro',
    isa       => Str,
    default   => sub { die "name is required" },
  );
  has gender     => ( is => 'ro',    isa => Str );
  has age        => ( is => 'rw',    isa => Num );
  has children   => (
    is        => 'ro',
    isa       => ArrayRef,
    default   => sub { return [] },
  );
}
```

NEXT STEPS

Here's your next step:

? Type::Tiny::Manual::UsingWithOther

Using Type::Tiny with Class::InsideOut, Params::Check, and Object::Accessor.

AUTHOR

Toby Inkster <tobyink@cpan.org>.

COPYRIGHT AND LICENCE

DISCLAIMER OF WARRANTIES