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

C:\>man Type::Tiny::ConstrainedObject.3pm

Type::Tiny::ConstrainedObjecUsermContributed Perl DocumeType::Tiny::ConstrainedObject(3pm)

NAME

Type::Tiny::ConstrainedObject - shared behavour for Type::Tiny::Class, etc

STATUS

This module is considered experiemental.

DESCRIPTION

Methods

The following methods exist for Type::Tiny::Class, Type::Tiny::Role, Type::Tiny::Duck, and any type constraints that inherit from "Object" or "Overload" in Types::Standard.

These methods will also work for Type::Tiny::Intersection if at least one of the types in the intersection provides these methods.

These methods will also work for Type::Tiny::Union if all of the types in the union provide these methods.

"stringifies_to($constraint)"

Generates a new child type constraint which checks the object's stringification against a constraint. For example:

```
my $type  = Type::Tiny::Class->new(class => 'URI');
my $child = $type->stringifies_to( StrMatch[qr/^http:/] );
$child->assert_valid( URI->new("http://example.com/") );
```

In the above example, $child is a type constraint that checks objects are blessed into (or inherit from) the URI class, and when stringified (e.g. though

overloading) the result matches the regular expression "qr/^http:/".

$constraint may be a type constraint, something that can be coerced to a type constraint (such as a coderef returning a boolean), a string of Perl code operating on $_, or a reference to a regular expression.

So the following would work:

```
my $child = $type->stringifies_to( sub { qr/^http:/ } );
my $child = $type->stringifies_to(       qr/^http:/   );
my $child = $type->stringifies_to(       'm/^http:/'  );
my $child = $type->where('"$_" =~ /^http:/');
```

"numifies_to($constraint)"

The same as "stringifies_to" but checks numification.

The following might be useful:

```
use Types::Standard qw(Int Overload);
my $IntLike = Int | Overload->numifies_to(Int)
```

"with_attribute_values($attr1 => $constraint1, ...)"

This is best explained with an example:

```
use Types::Standard qw(InstanceOf StrMatch);
use Types::Common::Numeric qw(IntRange);
my $person = InstanceOf['Local::Human'];
my $woman  = $person->with_attribute_values(
  gender   => StrMatch[ qr/^F/i  ],
  age      => IntRange[ 18 => () ],
);
$woman->assert_valid($alice);
```

This assertion will firstly check that $alice is a Local::Human, then check that "$alice->gender" starts with an "F", and lastly check that "$alice->age" is an integer at least 18.

Again, constraints can be type constraints, coderefs, strings of Perl code, or regular expressions.

Technically the "attributes" don't need to be Moo/Moose/Mouse attributes, but any methods which can be called with no parameters and return a scalar.

BUGS

Please report any bugs to <http://rt.cpan.org/Dist/Display.html?Queue=Type-Tiny>.

SEE ALSO

Type::Tiny::Manual.

Type::Tiny.

AUTHOR

Toby Inkster <tobyink@cpan.org>.

COPYRIGHT AND LICENCE

DISCLAIMER OF WARRANTIES