NAME

Type::Registry – a glorified hashref for looking up type constraints

SYNOPSIS

```
package Foo::Bar;

use Type::Registry;

my $reg = "Type::Registry"->for_me;  # a registry for Foo::Bar

# Register all types from Types::Standard
$reg->add_types(-Standard);

# Register just one type from Types::XSD
$reg->add_types(-XSD => ["NonNegativeInteger"]);

# Register all types from MyApp::Types
$reg->add_types("MyApp::Types");

# Create a type alias
$reg->alias_type("NonNegativeInteger" => "Count");

# Look up a type constraint
my $type = $reg->lookup("ArrayRef[Count]");

$type->check([1, 2, 3.14159]);  # croaks
```

Alternatively:

```
package Foo::Bar;

use Type::Registry qw( t );

# Register all types from Types::Standard
t->add_types(-Standard);

# Register just one type from Types::XSD
t->add_types(-XSD => ["NonNegativeInteger"]);

# Register all types from MyApp::Types
t->add_types("MyApp::Types");

# Create a type alias
t->alias_type("NonNegativeInteger" => "Count");

# Look up a type constraint
my $type = t("ArrayRef[Count]");

$type->check([1, 2, 3.14159]);  # croaks
```

STATUS

This module is covered by the Type-Tiny stability policy.

DESCRIPTION

A type registry is basically just a hashref mapping type names to type constraint objects.

Constructors

`new`
: Create a new glorified hashref.

`for_class($class)`
: Create or return the existing glorified hashref associated with the given class.

  Note that any type constraint you have imported from Type::Library–based type libraries will be automatically available in your class' registry.

`for_me`
: Create or return the existing glorified hashref associated with the caller.

Methods

`add_types(@libraries)`
: The libraries list is treated as an "optlist" (a la Data::OptList).

  Strings are the names of type libraries; if the first character is a hyphen, it is expanded to the "Types::" prefix. If followed by an arrayref, this is the list of types to import from that library. Otherwise, imports all types from the library.

```
use Type::Registry qw(t);

t->add_types(-Standard);  # OR: t->add_types("Types::Standard");

t->add_types(
   -TypeTiny => ['HashLike'],
   -Standard => ['HashRef' => { -as => 'RealHash' }],
);
```

  MooseX::Types (and experimentally, MouseX::Types) libraries can also be added this way, but *cannot be followed by an arrayref of types to import*.

`add_type($type, $name)`
: The long-awaited singular form of `add_types`. Given a type constraint object, adds it to the registry with a given name. The name may be omitted, in which case `$type->name` is called, and Type::Registry will throw an error if `$type` is anonymous. If a name is explicitly given, Type::Registry cares not one wit whether the type constraint is anonymous.

  This method can even add MooseX::Types and MouseX::Types type constraints; indeed anything that can be handled by Types::TypeTiny's `to_TypeTiny` function. (Bear in mind that to_TypeTiny *always* results in an anonymous type constraint, so `$name` will be required.)

`alias_type($oldname, $newname)`
: Create an alias for an existing type.

`simple_lookup($name)`
: Look up a type in the registry by name.

  Returns undef if not found.

`foreign_lookup($name)`
: Like `simple_lookup`, but if the type name contains "::", will attempt to load it from a type library. (And will attempt to load that module.)

`lookup($name)`
: Look up by name, with a DSL.

```
t->lookup("Int|ArrayRef[Int]")
```

  The DSL can be summed up as:

```
X               type from this registry
My::Lib::X      type from a type library
~X              complementary type
X | Y           union
X & Y           intersection
X[...]          parameterized type
slurpy X        slurpy type
Foo::Bar::      class type
```

Croaks if not found.

`make_union(@constraints)`, `make_intersection(@constraints)`, `make_class_type($class)`, `make_role_type($role)`

Convenience methods for creating certain common type constraints.

`AUTOLOAD`

Overloaded to call `lookup`.

```
$registry->Str;  # like $registry->lookup("Str")
```

Functions

`t` This class can export a function `t` which acts like `"Type::Registry"->for_class($importing_class)`.

BUGS

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

SEE ALSO

Type::Library.

AUTHOR

Toby Inkster <tobyink@cpan.org>.

COPYRIGHT AND LICENCE

DISCLAIMER OF WARRANTIES