## NAME

Role::Tiny – Roles: a nouvelle cuisine portion size slice of Moose

## SYNOPSIS

```
 package Some::Role;

 use Role::Tiny;

 sub foo { ... }

 sub bar { ... }

 around baz => sub { ... };

 1;
```

elsewhere

```
 package Some::Class;

 use Role::Tiny::With;

 # bar gets imported, but not foo
 with 'Some::Role';

 sub foo { ... }

 # baz is wrapped in the around modifier by Class::Method::Modifiers
 sub baz { ... }

 1;
```

If you wanted attributes as well, look at Moo::Role.

## DESCRIPTION

`Role::Tiny` is a minimalist role composition tool.

## ROLE COMPOSITION

Role composition can be thought of as much more clever and meaningful multiple inheritance. The basics of this implementation of roles is:

- If a method is already defined on a class, that method will not be composed in from the role. A method inherited by a class gets overridden by the role's method of the same name, though.
- If a method that the role "requires" to be implemented is not implemented, role application will fail loudly.

Unlike Class::C3, where the **last** class inherited from "wins," role composition is the other way around, where the class wins. If multiple roles are applied in a single call (single with statement), then if any of their provided methods clash, an exception is raised unless the class provides a method since this conflict indicates a potential problem.

## IMPORTED SUBROUTINES

### requires

```
 requires qw(foo bar);
```

Declares a list of methods that must be defined to compose role.

### with

```
 with 'Some::Role1';

 with 'Some::Role1', 'Some::Role2';
```

Composes another role into the current role (or class via Role::Tiny::With).

If you have conflicts and want to resolve them in favour of Some::Role1 you can instead write:

```
with 'Some::Role1';
with 'Some::Role2';
```

If you have conflicts and want to resolve different conflicts in favour of different roles, please refactor your codebase.

### before

```
before foo => sub { ... };
```

See "before method(s) => sub { ... }" in Class::Method::Modifiers for full documentation.

Note that since you are not required to use method modifiers, Class::Method::Modifiers is lazily loaded and we do not declare it as a dependency. If your Role::Tiny role uses modifiers you must depend on both Class::Method::Modifiers and Role::Tiny.

### around

```
around foo => sub { ... };
```

See "around method(s) => sub { ... }" in Class::Method::Modifiers for full documentation.

Note that since you are not required to use method modifiers, Class::Method::Modifiers is lazily loaded and we do not declare it as a dependency. If your Role::Tiny role uses modifiers you must depend on both Class::Method::Modifiers and Role::Tiny.

### after

```
after foo => sub { ... };
```

See "after method(s) => sub { ... }" in Class::Method::Modifiers for full documentation.

Note that since you are not required to use method modifiers, Class::Method::Modifiers is lazily loaded and we do not declare it as a dependency. If your Role::Tiny role uses modifiers you must depend on both Class::Method::Modifiers and Role::Tiny.

### Strict and Warnings

In addition to importing subroutines, using `Role::Tiny` applies strict and warnings to the caller.

## SUBROUTINES

### does_role

```
if (Role::Tiny::does_role($foo, 'Some::Role')) {
  ...
}
```

Returns true if class has been composed with role.

This subroutine is also installed as −>does on any class a Role::Tiny is composed into unless that class already has an −>does method, so

```
 if ($foo->does('Some::Role')) {
   ...
 }
```

will work for classes but to test a role, one must use ::does_role directly.

Additionally, Role::Tiny will override the standard Perl `DOES` method for your class. However, if `any` class in your class' inheritance hierarchy provides `DOES`, then Role::Tiny will not override it.

## METHODS

### make_role

```
Role::Tiny->make_role('Some::Role');
```

Makes a package into a role, but does not export any subs into it.

**apply_roles_to_package**

```
Role::Tiny->apply_roles_to_package(
  'Some::Package', 'Some::Role', 'Some::Other::Role'
);
```

Composes role with package. See also Role::Tiny::With.

**apply_roles_to_object**

```
Role::Tiny->apply_roles_to_object($foo, qw(Some::Role1 Some::Role2));
```

Composes roles in order into object directly. Object is reblessed into the resulting class. Note that the object's methods get overridden by the role's ones with the same names.

**create_class_with_roles**

```
Role::Tiny->create_class_with_roles('Some::Base', qw(Some::Role1 Some::Role2));
```

Creates a new class based on base, with the roles composed into it in order. New class is returned.

**is_role**

```
Role::Tiny->is_role('Some::Role1')
```

Returns true if the given package is a role.

## CAVEATS

- On perl 5.8.8 and earlier, applying a role to an object won't apply any overloads from the role to other copies of the object.
- On perl 5.16 and earlier, applying a role to a class won't apply any overloads from the role to any existing instances of the class.

## SEE ALSO

Role::Tiny is the attribute-less subset of Moo::Role; Moo::Role is a meta-protocol-less subset of the king of role systems, Moose::Role.

Ovid's Role::Basic provides roles with a similar scope, but without method modifiers, and having some extra usage restrictions.

## AUTHOR

mst – Matt S. Trout (cpan:MSTROUT) <mst@shadowcat.co.uk>

## CONTRIBUTORS

dg – David Leadbeater (cpan:DGL) <dgl@dgl.cx>

frew – Arthur Axel "fREW" Schmidt (cpan:FREW) <frioux@gmail.com>

hobbs – Andrew Rodland (cpan:ARODLAND) <arodland@cpan.org>

jnap – John Napiorkowski (cpan:JJNAPIORK) <jjn1056@yahoo.com>

ribasushi – Peter Rabbitson (cpan:RIBASUSHI) <ribasushi@cpan.org>

chip – Chip Salzenberg (cpan:CHIPS) <chip@pobox.com>

ajgb – Alex J. G. Burzyski (cpan:AJGB) <ajgb@cpan.org>

doy – Jesse Luehrs (cpan:DOY) <doy at tozt dot net>

perigrin – Chris Prather (cpan:PERIGRIN) <chris@prather.org>

Mithaldu – Christian Walde (cpan:MITHALDU) <walde.christian@googlemail.com>

ilmari – Dagfinn Ilmari Mannsåker (cpan:ILMARI) <ilmari@ilmari.org>

tobyink – Toby Inkster (cpan:TOBYINK) <tobyink@cpan.org>

haarg – Graham Knop (cpan:HAARG) <haarg@haarg.org>

## COPYRIGHT

Copyright (c) 2010–2012 the Role::Tiny "AUTHOR" and "CONTRIBUTORS" as listed above.

## LICENSE

This library is free software and may be distributed under the same terms as perl itself.