NAME

Mouse::Role − The Mouse Role

VERSION

This document describes Mouse version 0.97

SYNOPSIS

```
package Comparable;
use Mouse::Role; # the package is now a Mouse role

# Declare methods that are required by this role
requires qw(compare);

# Define methods this role provides
sub equals {
    my($self, $other) = @_;
    return $self->compare($other) == 0;
}

# and later
package MyObject;
use Mouse;
with qw(Comparable); # Now MyObject can equals()

sub compare {
    # ...
}

my $foo = MyObject->new();
my $bar = MyObject->new();
$obj->equals($bar); # yes, it is comparable
```

DESCRIPTION

This module declares the caller class to be a Mouse role.

The concept of roles is documented in Moose::Manual::Roles. This document serves as API documentation.

EXPORTED FUNCTIONS

Mouse::Role supports all of the functions that Mouse exports, but differs slightly in how some items are handled (see ''CAVEATS'' below for details).

Mouse::Role also offers two role-specific keywords:

`requires(@method_names)`

Roles can require that certain methods are implemented by any class which `does` the role.

Note that attribute accessors also count as methods for the purposes of satisfying the requirements of a role.

`excludes(@role_names)`

This is exported but not implemented in Mouse.

IMPORT AND UNIMPORT

import

Importing Mouse::Role will give you sugar. `-traits` are also supported.

unimport

Please unimport (`no Mouse::Role`) so that if someone calls one of the keywords (such as ''has'') it will break loudly instead breaking subtly.

CAVEATS

Role support has only a few caveats:

- Roles cannot use the `extends` keyword; it will throw an exception for now. The same is true of the `augment` and `inner` keywords (not sure those really make sense for roles). All other Mouse

keywords will be *deferred* so that they can be applied to the consuming class.

- Role composition does its best to **not** be order-sensitive when it comes to conflict resolution and requirements detection. However, it is order-sensitive when it comes to method modifiers. All before/around/after modifiers are included whenever a role is composed into a class, and then applied in the order in which the roles are used. This also means that there is no conflict for before/around/after modifiers.

  In most cases, this will be a non-issue; however, it is something to keep in mind when using method modifiers in a role. You should never assume any ordering.

SEE ALSO

Mouse

Moose::Role

Moose::Manual::Roles

Moose::Spec::Role