NAME

MouseX::NativeTraits::ArrayRef − Helper trait for ArrayRef attributes

SYNOPSIS

```
package Stuff;
use Mouse;

has 'options' => (
   traits     => ['Array'],
   is         => 'ro',
   isa        => 'ArrayRef[Str]',
   default    => sub { [] },
   handles    => {
       all_options    => 'elements',
       add_option     => 'push',
       map_options    => 'map',
       filter_options => 'grep',
       find_option    => 'first',
       get_option     => 'get',
       join_options   => 'join',
       count_options  => 'count',
       has_options    => 'count',
       has_no_options => 'is_empty',
       sorted_options => 'sort',
   },
);
```

DESCRIPTION

This module provides an Array attribute which provides a number of array operations.

PROVIDED METHODS

These methods are implemented in MouseX::NativeTraits::MethodProvider::ArrayRef.

count

Returns the number of elements in the array.

```
$stuff = Stuff->new;
$stuff->options(["foo", "bar", "baz", "boo"]);

my $count = $stuff->count_options;
print "$count\n"; # prints 4
```

is_empty

Returns a boolean value that is true when the array has no elements.

```
$stuff->has_no_options ? die "No options!\n" : print "Good boy.\n";
```

elements

Returns all of the elements of the array.

```
my @option = $stuff->all_options;
print "@options\n"; # prints "foo bar baz boo"
```

get($index)

Returns an element of the array by its index. You can also use negative index numbers, just as with Perl's core array handling.

```
my $option = $stuff->get_option(1);
print "$option\n"; # prints "bar"
```

pop

push($value1, $value2, value3 ...)

shift

unshift($value1, `$value2`, value3 ...)
splice($offset, `$length`, `@values`)
These methods are all equivalent to the Perl core functions of the same name.

first(sub { ... })
This method returns the first item matching item in the array, just like List::Util's `first` function. The matching is done with a subroutine reference you pass to this method. The reference will be called against each element in the array until one matches or all elements have been checked.

```
my $found = $stuff->find_option( sub { /^b/ } );
print "$found\n"; # prints "bar"
```

any(sub { ... })
This method returns true if any item in the array meets the criterion given through the subroutine, otherwise returns false. It sets `$_` for each item in the array.

grep(sub { ... })
This method returns every element matching a given criteria, just like Perl's core `grep` function. This method requires a subroutine which implements the matching logic.

```
my @found = $stuff->filter_options( sub { /^b/ } );
print "@found\n"; # prints "bar baz boo"
```

map(sub { ... })
This method transforms every element in the array and returns a new array, just like Perl's core `map` function. This method requires a subroutine which implements the transformation.

```
my @mod_options = $stuff->map_options( sub { $_ . "-tag" } );
print "@mod_options\n"; # prints "foo-tag bar-tag baz-tag boo-tag"
```

apply(sub { ... })
This method also transform every element in the array and returns a new array, just like List::MoreUtils's `apply` function.his is similar to `map`, but does not modify the element of the array.

reduce(sub { ... })
This method condenses an array into a single value, by passing a function the value so far and the next value in the array, just like List::Util's `reduce` function. The reducing is done with a subroutine reference you pass to this method.

```
my $found = $stuff->reduce_options( sub { $_[0] . $_[1] } );
print "$found\n"; # prints "foobarbazboo"
```

sort(\&compare)
Returns the array in sorted order.

You can provide an optional subroutine reference to sort with (as you can with Perl's core `sort` function). However, instead of using `$a` and `$b`, you will need to use `$_[0]` and `$_[1]` instead.

```
# ascending ASCIIbetical
my @sorted = $stuff->sort_options();

# Descending alphabetical order
my @sorted_options = $stuff->sort_options( sub { lc $_[1] cmp lc $_[0] } );
print "@sorted_options\n"; # prints "foo boo baz bar"
```

sort_in_place(\&compare)
Sorts the array *in place*, modifying the value of the attribute.

You can provide an optional subroutine reference to sort with (as you can with Perl's core `sort` function). However, instead of using `$a` and `$b`, you will need to use `$_[0]` and `$_[1]` instead.

sort_by(\&by, \&compare)
Returns the array in sorted order, applying *\&by* function to each item.

This is equivalent to `sort(sub{ by($_[0]) cmp by($_[1]) })`, but implemented effectively.

Currently (as of Moose 0.98) this is a Mouse specific method.

sort_in_place_by(\&by, \&compare)
Sorts the array, applying *\&by* function to each item, modifying the value of the attribute.

This is equivalent to `sort_in_place(sub{ by($_[0]) cmp by($_[1]) })`, but implemented effectively.

Currently (as of Moose 0.98) this is a Mouse specific method.

shuffle
Returns the array, with indices in random order, like `shuffle` from List::Util.

uniq
Returns the array, with all duplicate elements removed, like `uniq` from List::MoreUtils.

join($str)
Joins every element of the array using the separator given as argument, just like Perl's core `join` function.

```
my $joined = $stuff->join_options( ':' );
print "$joined\n"; # prints "foo:bar:baz:boo"
```

set($index, `$value`)
Given an index and a value, sets the specified array element's value.

delete($index)
Removes the element at the given index from the array.

insert($index, `$value`)
Inserts a new element into the array at the given index.

clear
Empties the entire array, like `@array = ()`.

accessor
This method provides a get/set accessor for the array, based on array indexes. If passed one argument, it returns the value at the specified index. If passed two arguments, it sets the value of the specified index.

for_each(sub{ ... })
This method calls the given subroutine with each element of the array, like Perl's core `foreach` statement.

Currently (as of Moose 0.98) this is a Mouse specific method.

for_each_pair(sub{ ... })
This method calls the given subroutine with each two element of the array, as if the array is a list of pairs.

Currently (as of Moose 0.98) this is a Mouse specific method.

METHODS

meta
method_provider_class
helper_type

SEE ALSO

MouseX::NativeTraits