Iteration & Loops in Puppet 4

Repeating yourself using the Future Parser

Stefan Möding

Puppet Camp Amsterdam
February 5th, 2016

Loops in Puppet?

- Puppet describes a state to enforce
- A resource can’t be declared more than once
- So why would we need loops?

Use case: create resources for each array/hash item

- Avoid copy/paste manifests

Purpose

Iterate an array or hash and do something for each element

Loops in Puppet 3

- Essentially no loop statements

Workarounds

- Array of strings as resource title
- `create_resources` with a parameter hash
- Recursion using a defined type

Evolution of Puppet

Puppet 3 – Example: array as resource title

```
# Array of filenames
$files = [ '/dir/file1', '/dir/file2', ... ]

# Declare multiple resources by using an array
file { $files:
  ensure => absent,
}
```

Limitations

- Resource parameters affect all resources
- Defined type required for more challenging tasks

Evolution of Puppet

Puppet 3 – Example: create_resources

`create_resources`

- Indirect way to create resources
- Uses type name and hash of type title & parameters

```
# Create mail aliases
create_resources('mailalias', {
    'postmaster' => { recipient => 'root' },
    'admins'     => { recipient => 'fred,barney' },
  }
)

# Can we generate the parameter hash from this hash?
{ 'postmaster' => 'root', 'admins' => 'fred,barney' }
```

Evolution of Puppet

Puppet 3 – Example: create_resources

```
# Our mail aliases
$aliases = { 'postmaster' => 'root', 'admins' => 'fred,barney' }

# Convert key-values into strings
$aliases_1 = join_keys_to_values($aliases, ':')

# Build JSON key-value pairs
$aliases_2 = regsubst($aliases_1, '^(.*):(.*)$', '"\1": {"recipient": "\2"}')

# Concatenate into one string
$aliases_3 = join($aliases_2, ', ')

# Finalize JSON representation
$aliases_4 = "{ ${aliases_3} }"

# Convert into a Puppet hash
$aliases_5 = parsejson($aliases_4)

create_resources('mailalias', $aliases_5)
```

Limitations

- Purpose of code is not obvious at a first glance
- Complex resources require complex regular expressions

Evolution of Puppet

Puppet 3 – Example: recursion

```
define recurse($array) {
  if !empty($array) {
    $car = $array[0]              # first item
    $cdr = delete_at($array, 0) # rest of array

    # Recursive declaration for remaining resources
    recurse { "recurse-${car}": array => $cdr }

    # Do something useful here using $car
  }
}
```

Limitations

- Needs a way to guarantee unique resource names
- Error: Somehow looped more than 1000 times . . .

Loops in Puppet 4

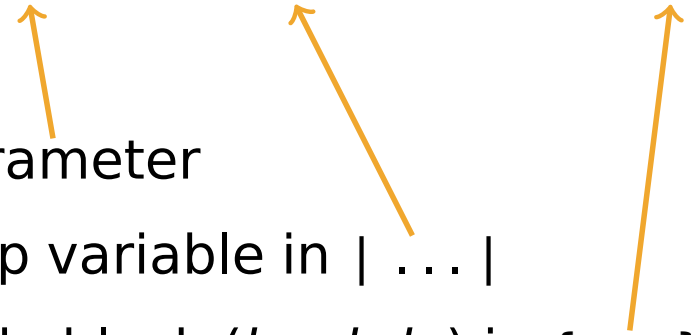
- `each`
- `map`
- `filter`
- `reduce`
- `slice`

Also available in Puppet 3.x using `parser = future`

Loops in Puppet 4

Syntax

```
each($array) |$item| { notice($item) }
```



- parameter
- loop variable in `| ... |`
- code block (*lambda*) in `{ ... }`

type of parameter	unit of work
array	single element
hash	key-value pair
string	character

Alternative object-oriented syntax

```
$array.each |$item| { notice($item) }
```

Loops in Puppet 4

each

each

Generic loop function

- Execute block of code for each item
- Return original data structure unchanged

Loops in Puppet 4

each – Example: Declare resource for each array value

```
# Array of filenames
$files = [ '/dir/file1', '/dir/file2', ... ]

# Declare multiple resources by using a loop
each($files) |$file| {
  file { $file:
    ensure => absent,
  }
}
```

Loops in Puppet 4

each – Example: More program logic in the code block

```
# Tidy using retention time based on the directory name
$directories = [ '/tmp', '/var/tmp', '/opt/jboss/log', ]

each($directories) |$dir| {
  $age = $dir ? {
    /tmp/   => '1d',
    /log/   => '4w',
    default => '1w',
  }

  tidy { $dir:
    age    => $age,
    backup => false,
  }
}
```

Loops in Puppet 4

map

map

Transform one data structure into another

- Execute block of code for each item
- Use return value of each cycle to build new data structure
- Return new data structure

Loops in Puppet 4

map – Example: Iterate string character by character

```
# Scramble mail addresses
$mail = 'fred@example.com'

$s = map($mail) |$char| {
  # Use selector expression to substitute characters
  $char ? {
    '@'     => ' at ',
    '.'     => ' dot ',
    default => $char,
  }
}


$scrambled_mail = join($s)


   'fred at example dot com'
```

Loops in Puppet 4

map – Example: Generate array of hashes from an array

```
# Aliases for apache::vhost
$aliases = [
  { path => '/var/www/css',     alias => '/css',     },
  { path => '/var/www/gfx',     alias => '/gfx',     },
  { path => '/var/www/js',      alias => '/js',      },
  { path => '/usr/lib/cgi-bin', alias => '/cgi-bin', },
]
```

Loops in Puppet 4

map – Example: Generate array of hashes from an array

```
# Generate Apache aliases from array of path names
$dirs = [
  '/var/www/css',
  '/var/www/gfx',
  '/var/www/js',
  '/usr/lib/cgi-bin',
]

$aliases = map($dirs) |$d| {
  # Extract last component of path including '/'
  $alias = regsubst($d, '^.*(/[^/]+/?)$', '\1')

  # Return a hash
  { path => $d, alias => $alias, }
}
```

Loops in Puppet 4

`filter`

`filter`

Conditionally remove items from arrays and hashes

- Execute block of code for each item
- Return `true` if item belongs to the result, `false` otherwise
- Collect wanted items into new data structure
- Return new data structure

Loops in Puppet 4

`filter` – Example: Filter structured facts

```
networking => {
  fqdn => "caprice.example.net",
  interfaces => {
    eth0 => {
      ip => "64.15.112.20",
      ...
    },
    eth1 => {
      ip => "192.168.17.38",
      ...
    },
    ...
  },
  ...
}
```

Loops in Puppet 4

`filter` – Example: Filter structured facts

```
# Get hash of interfaces from the networking fact
$ifaces = $::networking['interfaces']

# Network facts for our internal interfaces only
$net = filter($ifaces) |$ifname, $attr| {
  $attr['ip'] =~ /^192\.168/
}
```

Loops in Puppet 4

`filter` – Example: Determine Java JDK to install

```
# Java packages we want to manage
$java = [
  { packages     => [ 'ibm-java7-jdk:s390', ],
    java_version => '1.7',
    java_bitness => '32bit',
    architecture => [ 's390', ],
    os_rel       => [ 'Debian-7', ],
  },
  { packages     => [ 'openjdk-7-jdk', ],
    java_version => '1.7',
    java_bitness => '64bit',
    architecture => [ 's390x', 'amd64' ],
    os_rel       => [ 'Debian-7', 'Ubuntu-14.04', ],
  },
  ...
]
```

Loops in Puppet 4

`filter` – Example: Determine Java JDK to install

```
# Combined os-release fact: 'Ubuntu-14.04', 'Debian-7'
$os_rel = "${::os['name']}-${::os['release']['major']}"

# Filter inappropriate options
$candidates = filter($java) |$hash| {
  ($java_version   == $hash['java_version']) and
  ($java_bitness   == $hash['java_bitness']) and
  ($::architecture in $hash['architecture']) and
  ($os_rel         in $hash['os_rel'])
}

if empty($candidates) {
  fail("No appropriate Java JDK available")
}

# Priority: first candidate wins
ensure_packages($candidates[0]['packages'])
```

Loops in Puppet 4

reduce

reduce

Incremental processing or calculation

- Execute block of code for each item
- Update a memorized result calculated so far
- Pass memorized result into next cycle
- Return memorized result from last cycle

Loops in Puppet 4

reduce – Example: character filter

```
# Reverse a sentence
$sentence = 'Strawberry Fields Forever'

$reverse = reduce($sentence, '') |$memo, $c| {
  "${c}${memo}"
}
```

```
'reveroF sdleiF yrrebwartS'
```

Loops in Puppet 4

reduce – Example: Better validation rules and error messages

Excerpt from `/etc/gai.conf`

```
#
# Configuration for getaddrinfo(3).
#
# precedence  <mask>   <value>
#
precedence  ::/96            20
precedence  2002::/16        30
precedence  ::/0             40
precedence  ::1/128          50
precedence  ::ffff:0:0/96    100
```

Implement a validation rule to prevent duplicated values

Loops in Puppet 4

reduce – Example: Better validation rules and error messages

```
# Puppet 3 version
$labels = {
  '::/96'          => 20,
  '2002::/16'      => 30,
  '::/0'           => 40,
  '::1/128'        => 50,
  '::ffff:0:0/96' => 100,
}

# Get hash values as array
$priorities = values($labels)

# Check non-unique values
if size(unique($priorities)) < size($priorities) {
  fail('Precedence values are not unique')
}
```

Loops in Puppet 4

reduce – Example: Better validation rules and error messages

```
# Puppet 4 version
$labels = {
  '::/96'          => 20,
  ...
  '::ffff:0:0/96' => 100,
}

reduce($labels, []) |$memo, $item| {
  # Fail if the current precedence has been seen before
  if member($memo, $item[1]) {
    fail("${item[0]} has non-unique value ${item[1]}")
  }

  # Add precedence to list of processed items
  concat($memo, $item[1])
}
```

Loops in Puppet 4

slice

slice

Divide an array into smaller pieces and process each part

- Execute block of code for groups of *n* items

Loops in Puppet 4

`slice` – Example: Iterate array in pairs

```
# Consecutive numbers are on opposite sides of a dice
$dice = [ '1', '6', '2', '5', '3', '4', ]

slice($dice, 2) |$num1, $num2| {
  notice("${num1} is opposite of ${num2}")
}
```

```
Notice: Scope(Class[main]): 1 is opposite of 6
Notice: Scope(Class[main]): 2 is opposite of 5
Notice: Scope(Class[main]): 3 is opposite of 4
```

Unfortunately no *real life*™ example . . .

Wrap up

Puppet 4 adds five powerful iteration functions

Use cases: process arrays/hashes/strings to

- individually manage each item
- transform the data structure
- select items based on conditions
- create validations with better error messages

Arrays and hashes can be

- class/type parameters
- structured facts

Contact

stm@kill-9.net

@UnixMagus

http://www.moeding.net

https://github.com/smoeding

https://forge.puppetlabs.com/stm

Icons by Dan Leech, http://simpleicons.org