./ - lemplate

- a.lua
- lib/
- src/
- t/

a.lua - lemplate

Functions defined

- _M.process
- context_meta.include
- context_meta.plugin
- context_meta.process
- filter
- if type(v)
- if type(value)
- stash_get
- stash_set
- template_map['tmpQQS_k.tt2']
- tt2_not
- tt2_true

Source code

```
--[[
   This Lua code was generated by Lemplate, the Lua
   Template Toolkit. Any changes made to this file will be lost the next
   time the templates are compiled.

   Copyright 2016 - Yichun Zhang (agentzh) - All rights reserved.

   Copyright 2006-2014 - Ingy döt Net - All rights reserved.
]]

local gsub = ngx.re.gsub
local concat = table.concat
local type = type
local math_floor = math.floor
local table_maxn = table.maxn

local _M = {
    version = '0.01'
}

local template_map = {}

local function tt2_true(v)
    return v and v ~= 0 and v ~= "" and v ~= '0'
end

local function tt2_not(v)
    return not v or v == 0 or v == "" or v == '0'
end

local context_meta = {}

function context_meta.plugin(context, name, args)
    if name == "iterator" then
        local list = args[1]
        local count = table_maxn(list)
        return { list = list, count = 1, max = count - 1, index = 0, size = count, first = true, last = false,
prev = "" }
```

```lua
    else
        return error("unknown iterator: " .. name)
    end
end

function context_meta.process(context, file)
    local f = template_map[file]
    if not f then
        return error("file error - " .. file .. ": not found")
    end
    return f(context)
end

function context_meta.include(context, file)
    local f = template_map[file]
    if not f then
        return error("file error - " .. file .. ": not found")
    end
    return f(context)
end

context_meta = { __index = context_meta }

local function stash_get(stash, k)
    local v
    if type(k) == "table" then
        v = stash
        for i = 1, #k, 2 do
            local key = k[i]
            local typ = k[i + 1]
            if type(typ) == "table" then
                local value = v[key]
                if type(value) == "function" then
                    return value()
                end
                if value then
                    return value
                end
                if key == "size" then
                    if type(v) == "table" then
                        return #v
                    else
                        return 1
                    end
                else
                    return error("virtual method " .. key .. " not supported")
                end
            end
            if type(key) == "number" and key == math_floor(key) and key >= 0 then
                key = key + 1
            end
            if type(v) ~= "table" then
                return nil
            end
            v = v[key]
        end
    else
        v = stash[k]
    end
    if type(v) == "function" then
        return v()
    end
    return v
end

local function stash_set(stash, k, v, default)
    if default then
        local old = stash[k]
        if old == nil then
            stash[k] = v
        end
    else
        stash[k] = v
    end
end

```

```
function _M.process(file, params)
    local stash = params
    local context = {
        stash = stash,
        filter = function (bits, name, params)
            local s = concat(bits)
            if name == "html" then
                s = gsub(s, "&", '&amp;', "jo")
                s = gsub(s, "<", '&lt;', "jo");
                s = gsub(s, ">", '&gt;', "jo");
                s = gsub(s, '"', '&quot;', "jo"); -- " end quote for emacs
                return s
            end
        end
    }
    context = setmetatable(context, context_meta)
    local f = template_map[file]
    if not f then
        return error("file error - " .. file .. ": not found")
    end
    return f(context)
end
-- tmpQQS_k.tt2
template_map['tmpQQS_k.tt2'] = function (context)
    if not context then
        return error("Lemplate function called without context\n")
    end
    local stash = context.stash
    local output = {}
    local i = 0

i = i + 1 output[i] = 'Hello, '
-- line 1 "tmpQQS_k.tt2"
i = i + 1 output[i] = stash_get(stash, 'world')
i = i + 1 output[i] = '!\n'

    return output
end

return _M
```

One Level Up   Top Level

lib/ - lemplate

- Lemplate/
- Lemplate.pm

lib/Lemplate/ - lemplate

- Directive.pm
- Parser.pm
- Runtime/
- Runtime.pm

lib/Lemplate/Directive.pm - lemplate

Data types defined

- _attempt_range_expand_val
- args
- assign
- block
- break
- call
- capture
- clear
- default
- filenames
- filter
- foreach
- get
- ident
- if
- include
- javascript
- macro
- multi_wrapper
- new
- next
- no_javascript
- process
- quoted
- raw
- return
- set
- stop
- switch
- template

- text
- textblock
- throw
- use
- while
- wrapper

Source code

```perl
package Lemplate::Directive;
use strict;
use warnings;

# VERSION

our $OUTPUT = 'i = i + 1 output[i] =';
our $WHILE_MAX = 1000;

 # parser state variable
 # only true when inside JAVASCRIPT blocks
 our $INJAVASCRIPT = 0;

 sub new {
     my $class = shift;

     return bless {}, $class
 }

 sub template {
     my ($class, $block) = @_;

     return "function() return '' end" unless $block =~ /\S/;

     return <<"...";
 function (context)
     if not context then
         return error("Lemplate function called without context\\n")
     end
     local stash = context.stash
     local output = {}
     local i = 0

 $block

     return output
 end
 ...
 }

 # Try to do 1 .. 10 expansions
 sub _attempt_range_expand_val ($) {
     my $val = shift;
     return $val unless
         my ( $from, $to ) = $val =~ m/\s*\[\s*(\S+)\s*\.\.\s*(\S+)\s*\]/;

     die "Range expansion is current supported for positive/negative integer values only (e.g. [ 1 .. 10
])\nCannot expand: $val" unless $from =~ m/^-?\d+$/ && $to =~ m/^-?\d+$/;

     return join '', '[', join( ',', $from .. $to ), ']';
 }

 #------------------------------------------------------------------------
 # textblock($text)
 #------------------------------------------------------------------------

 sub textblock {
     my ($class, $text) = @_;
     return $text if $INJAVASCRIPT;
```

```perl
    return "$OUTPUT " . $class->text($text);
}

#------------------------------------------------------------------------
# text($text)
#------------------------------------------------------------------------

sub text {
    my ($class, $text) = @_;
    for ($text) {
        s/(['\\])/\\$1/g;
        s/\n/\\n/g;
        s/\r/\\r/g;
    }
    return "'" . $text . "'";
}

#------------------------------------------------------------------------
# ident(\@ident)                                             foo.bar(baz)
#------------------------------------------------------------------------

sub ident {
    my ($class, $ident) = @_;
    return "''" unless @$ident;
    my $ns;

    # does the first element of the identifier have a NAMESPACE
    # handler defined?
    if (ref $class && @$ident > 2 && ($ns = $class->{ NAMESPACE })) {
        my $key = $ident->[0];
        $key =~ s/^'(.+)'$/$1/s;
        if ($ns = $ns->{ $key }) {
            return $ns->ident($ident);
        }
    }

    if (scalar @$ident <= 2 && ! $ident->[1]) {
        $ident = $ident->[0];
    }
    else {
        $ident = '{' . join(', ', @$ident) . '}';
     }
     return "stash_get(stash, $ident)";
 }


 #------------------------------------------------------------------------
 # assign(\@ident, $value, $default)                              foo = bar
 #------------------------------------------------------------------------

 sub assign {
     my ($class, $var, $val, $default) = @_;

     if (ref $var) {
         if (scalar @$var == 2 && ! $var->[1]) {
             $var = $var->[0];
         }
         else {
             $var = '{' . join(', ', @$var) . '}';
         }
     }
     $val =  _attempt_range_expand_val $val;
     $val .= ', 1' if $default;
     return "stash_set(stash, $var, $val)";
 }


 #------------------------------------------------------------------------
 # args(\@args)                                        foo, bar, baz = qux
 #------------------------------------------------------------------------

 sub args {
     my ($class, $args) = @_;
     my $hash = shift @$args;
     push(@$args, '{ ' . join(', ', @$hash) . ' }')
         if @$hash;

     return '{}' unless @$args;
     return '{ ' . join(', ', @$args) . ' }';
 }


 #------------------------------------------------------------------------
 # filenames(\@names)
 #------------------------------------------------------------------------

 sub filenames {
     my ($class, $names) = @_;
     if (@$names > 1) {
         $names = '[ ' . join(', ', @$names) . ' ]';
     }
     else {
         $names = shift @$names;
     }
     return $names;
 }


 #------------------------------------------------------------------------
 # get($expr)                                                    [% foo %]
 #------------------------------------------------------------------------

 sub get {
     my ($class, $expr) = @_;
     return "$OUTPUT $expr";
 }

 sub block {
     my ($class, $block) = @_;
     return join "\n", map {
         s/^#(?=line \d+)/-- /gm;
         $_;
     } @{ $block || [] };
 }

 #------------------------------------------------------------------------
 # call($expr)                                              [% CALL bar %]
 #------------------------------------------------------------------------

 sub call {
     my ($class, $expr) = @_;
     $expr .= ';';
     return $expr;
 }


 #------------------------------------------------------------------------
 # set(\@setlist)                               [% foo = bar, baz = qux %]
 #------------------------------------------------------------------------

 sub set {
     my ($class, $setlist) = @_;
     my $output;
     while (my ($var, $val) = splice(@$setlist, 0, 2)) {
         $output .= $class->assign($var, $val) . ";\n";
     }
     chomp $output;
     return $output;
 }


 #------------------------------------------------------------------------
 # default(\@setlist)                   [% DEFAULT foo = bar, baz = qux %]
 #------------------------------------------------------------------------

 sub default {
     my ($class, $setlist) = @_;
     my $output;
     while (my ($var, $val) = splice(@$setlist, 0, 2)) {
         $output .= &assign($class, $var, $val, 1) . ";\n";
     }
     chomp $output;
     return $output;
 }


 #------------------------------------------------------------------------
 # include(\@nameargs)                    [% INCLUDE template foo = bar %]
 #         # => [ [ $file, ... ], \@args ]
 #------------------------------------------------------------------------

 sub include {
     my ($class, $nameargs) = @_;
     my ($file, $args) = @$nameargs;
     my $hash = shift @$args;
     $file = $class->filenames($file);
     (my $raw_file = $file) =~ s/^'|'$//g;
     $Lemplate::ExtraTemplates{$raw_file} = 1;
     my $file2 = "'$Lemplate::TemplateName/$raw_file'";
     my $str_args = (@$hash ? ', { ' . join(', ', @$hash) . ' }' : '');
     return "$OUTPUT context.include(context, template_map['$Lemplate::TemplateName/$raw_file'] and $file2 or
 $file$str_args)";
 }


 #------------------------------------------------------------------------
 # process(\@nameargs)                    [% PROCESS template foo = bar %]
 #         # => [ [ $file, ... ], \@args ]
 #------------------------------------------------------------------------

 sub process {
     my ($class, $nameargs) = @_;
     my ($file, $args) = @$nameargs;
     my $hash = shift @$args;
     $file = $class->filenames($file);
     (my $raw_file = $file) =~ s/^'|'$//g;
     $Lemplate::ExtraTemplates{$raw_file} = 1;
     $file .= @$hash ? ', { ' . join(', ', @$hash) . ' }' : '';
     return "$OUTPUT context.process(context, $file)";
 }


 #------------------------------------------------------------------------
 # if($expr, $block, $else)                              [% IF foo < bar %]
 #                                                          ...
 #                                                       [% ELSE %]
 #                                                          ...
 #                                                       [% END %]
 #------------------------------------------------------------------------

 sub if {
     my ($class, $expr, $block, $else) = @_;
     my @else = $else ? @$else : ();
     $else = pop @else;

     my $output = "if tt2_true($expr) then\n$block\n";

     foreach my $elsif (@else) {
         ($expr, $block) = @$elsif;
         $output .= "elseif tt2_true($expr) then\n$block\n";
     }
     if (defined $else) {
         $output .= "else\n$else\nend\n";
     } else {
         $output .= "end\n";
     }

     return $output;
 }

 #------------------------------------------------------------------------
 # foreach($target, $list, $args, $block)    [% FOREACH x = [ foo bar ] %]
 #                                                ...
 #                                           [% END %]
 #------------------------------------------------------------------------

 sub foreach {
     my ($class, $target, $list, $args, $block) = @_;
     $args  = shift @$args;
     $args  = @$args ? ', { ' . join(', ', @$args) . ' }' : '';

     my ($loop_save, $loop_set, $loop_restore, $setiter);
     if ($target) {
         $loop_save =
             'local oldloop = ' . $class->ident(["'loop'"]);
         $loop_set = "stash['$target'] = value";
         $loop_restore = "stash_set(stash, 'loop', oldloop)";
     }
     else {
         die "XXX - Not supported yet";
         $loop_save = 'stash = context.localise()';
         $loop_set =
             "stash.get(['import', [value]]) if typeof(value) == 'object'";
         $loop_restore = 'stash = context.delocalise()';
     }

     $list = _attempt_range_expand_val $list;

     return <<EOF;

 -- FOREACH
 do
     local list = $list
     local iterator
     if list.list then
         iterator = list
         list = list.list
     end
     $loop_save
     local count
     if not iterator then
         count = table_maxn(list)
         iterator = { count = 1, max = count - 1, index = 0, size = count, first = true, last = false, prev = 
 "" }
     else
         count = iterator.size
     end
     stash.loop = iterator
     for idx, value in ipairs(list) do
         if idx == count then
             iterator.last = true
         end
         iterator.index = idx - 1
         iterator.count = idx
         iterator.next = list[idx + 1]
         $loop_set
 $block
         iterator.first = false
         iterator.prev = value
     end
     $loop_restore
 end
 EOF
 }


 #------------------------------------------------------------------------
 # next()                                                        [% NEXT %]
 #
 # Next iteration of a FOREACH loop (experimental)
 #------------------------------------------------------------------------

 sub next {
   return <<EOF;
   return error("NEXT not implemented yet")
 EOF
 }

 #------------------------------------------------------------------------
 # wrapper(\@nameargs, $block)            [% WRAPPER template foo = bar %]
 #          # => [ [$file,...], \@args ]
 #------------------------------------------------------------------------
 sub wrapper {
     my ($class, $nameargs, $block) = @_;
     my ($file, $args) = @$nameargs;
     my $hash = shift @$args;

     s/ => /: / for @$hash;
     return $class->multi_wrapper($file, $hash, $block)
         if @$file > 1;
     $file = shift @$file;
     push(@$hash, "'content': output");
     $file .= @$hash ? ', { ' . join(', ', @$hash) . ' }' : '';

     return <<EOF;

 // WRAPPER
 $OUTPUT (function() {
     var output = '';
 $block;
     return context.include($file);
 })();
 EOF
 }

 sub multi_wrapper {
     my ($class, $file, $hash, $block) = @_;

     push(@$hash, "'content': output");
     $hash = @$hash ? ', { ' . join(', ', @$hash) . ' }' : '';

     $file = join(', ', reverse @$file);
 #    print STDERR "multi wrapper: $file\n";

     return <<EOF;

 // WRAPPER
 $OUTPUT (function() {
     var output = '';
 $block;
     var files = new Array($file);
     for (var i = 0; i < files.length; i++) {
         output = context.include(files[i]$hash);
     }
     return output;
 })();
 EOF
 }


 #------------------------------------------------------------------------
 # while($expr, $block)                                 [% WHILE x < 10 %]
 #                                                         ...
 #                                                      [% END %]
 #------------------------------------------------------------------------

 sub while {
     my ($class, $expr, $block) = @_;

     return <<EOF;

 -- WHILE
 do
     local failsafe = $WHILE_MAX;
     while $expr do
         failsafe = failsafe - 1
         if failsafe <= 0 then
             break
         end
 $block
     end
     if not failsafe then
         return error("WHILE loop terminated (> $WHILE_MAX iterations)\\n")
     end
 end
 EOF
 }

 #------------------------------------------------------------------------
 # javascript($script)                                   [% JAVASCRIPT %]
 #                                                           ...
 #                                                       [% END %]
 #------------------------------------------------------------------------
 sub javascript {
     my ( $class, $javascript ) = @_;
     return $javascript;
 }

 sub no_javascript {
     my ( $class ) = @_;
     die "EVAL_JAVASCRIPT has not been enabled, cannot process [% JAVASCRIPT %] blocks";
 }

 #------------------------------------------------------------------------
 # switch($expr, \@case)                                    [% SWITCH %]
 #                                                          [% CASE foo %]
 #                                                             ...
 #                                                          [% END %]
 #------------------------------------------------------------------------

 sub switch {
     my ($class, $expr, $case) = @_;
     my @case = @$case;
     my ($match, $block, $default);
     my $caseblock = '';

     $default = pop @case;

     foreach $case (@case) {
         $match = $case->[0];
         $block = $case->[1];
 #        $block = pad($block, 1) if $PRETTY;
         $caseblock .= <<EOF;
 case $match:
 $block
 break;

 EOF
     }

     if (defined $default) {
         $caseblock .= <<EOF;
 default:
 $default
 break;
 EOF
     }
 #    $caseblock = pad($caseblock, 2) if $PRETTY;

 return <<EOF;

     switch($expr) {
 $caseblock
     }

 EOF
 }


 #------------------------------------------------------------------------
 # throw(\@nameargs)                           [% THROW foo "bar error" %]
 #       # => [ [$type], \@args ]
 #------------------------------------------------------------------------

 sub throw {
     my ($class, $nameargs) = @_;
     my ($type, $args) = @$nameargs;
     my $hash = shift(@$args);
     my $info = shift(@$args);
     $type = shift @$type;

     return qq{return error({$type, $info})};
 }


 #------------------------------------------------------------------------
```

```perl
# clear()                                                     [% CLEAR %]
#
# NOTE: this is redundant, being hard-coded (for now) into Parser.yp
#------------------------------------------------------------------------

sub clear {
    return "output = {}";
}


#------------------------------------------------------------------------
# break()                                                     [% BREAK %]
#
# NOTE: this is redundant, being hard-coded (for now) into Parser.yp
#------------------------------------------------------------------------

sub break {
    return 'break';
}

#------------------------------------------------------------------------
# return()                                                   [% RETURN %]
#------------------------------------------------------------------------

sub return {
    return "return output"
}


#------------------------------------------------------------------------
# stop()                                                       [% STOP %]
#------------------------------------------------------------------------

sub stop {
    return "return error('Lemplate.STOP\\n' .. concat(output))";
}


#------------------------------------------------------------------------
# use(\@lnameargs)                         [% USE alias = plugin(args) %]
#     # => [ [$file, ...], \@args, $alias ]
#------------------------------------------------------------------------

sub use {
    my ($class, $lnameargs) = @_;
    my ($file, $args, $alias) = @$lnameargs;
    $file = shift @$file;       # same production rule as INCLUDE
    $alias ||= $file;
    $args = &args($class, $args);
    $file .= ", $args" if $args;
    return "-- USE\n"
         . "stash_set(stash, $alias, context.plugin(context, $file))";
}


#------------------------------------------------------------------------
# raw(\@lnameargs)                         [% RAW alias = plugin(args) %]
#     # => [ [$file, ...], \@args, $alias ]
#------------------------------------------------------------------------

sub raw {
    my ($class, $lnameargs) = @_;
    my ($file, $args, $alias) = @$lnameargs;
    $file = shift @$file;       # same production rule as INCLUDE
    $alias ||= $file;
    $args = &args($class, $args);
#    $file .= ", $args" if $args;
    $file =~ s/'|"//g;
    return "// RAW\n"
         . "stash_set(stash, $alias, $file)";
}


#------------------------------------------------------------------------
# stubs()                                                      [% STOP %]
#------------------------------------------------------------------------
```

```

sub filter {
    my ($class, $lnameargs, $block) = @_;
    my ($name, $args, $alias) = @$lnameargs;
    $name = shift @$name;
    $args = &args($class, $args);
    $args = $args ? "$args, $alias" : ", null, $alias"
        if $alias;
    $name .= ", $args" if $args;
    return <<EOF;

-- FILTER
local value
do
    local output = {}
    local i = 0

$block

    value = context.filter(output, $name)
end
$OUTPUT value
EOF
}

sub quoted {
    my $class = shift;
    if ( @_ && ref($_[0]) ) {
        return join( " .. ", @{$_[0]} );
    }
    return "return error('QUOTED called with unknown arguments in Lemplate')";
}

#------------------------------------------------------------------------
# macro($name, $block, \@args)
#------------------------------------------------------------------------

sub macro {
    my ($class, $ident, $block, $args) = @_;

    if ($args) {
        $args = join(';', map { "args['$_'] = fargs.shift()" } @$args);

        return <<EOF;

//MACRO
stash.set('$ident', function () {
    var output = '';
    var args = {};
    var fargs = Array.prototype.slice.call(arguments);
    $args;
    args.arguments = Array.prototype.slice.call(arguments);

    var params = fargs.shift() || {};

    for (var key in params) {
        args[key] = params[key];
    }

    context.stash.clone(args);
    try {
$block
    }
    catch(e) {
        var error = context.set_error(e, output);
        throw(error);
    }

    context.stash.declone();
    return output;
});

EOF

    }
    else {
        return <<EOF;

//MACRO

stash.set('$ident', function () {
    var output = '';
    var args = {};

    var fargs = Array.prototype.slice.call(arguments);
    args.arguments = Array.prototype.slice.call(arguments);

    if (typeof arguments[0] == 'object') args = arguments[0];

    context.stash.clone(args);
    try {
$block
    }
    catch(e) {
        var error = context.set_error(e, output);
        throw(error);
    }

    context.stash.declone();
    return output;});

EOF
    }
}

sub capture {
    my ($class, $name, $block) = @_;

    if (ref $name) {
        if (scalar @$name == 2 && ! $name->[1]) {
            $name = $name->[0];
        }
        else {
            $name = '[' . join(', ', @$name) . ']';
        }
    }

    return <<EOF;

// CAPTURE
(function() {
   var output = '';
   $block
   stash.set($name, output);
})();
EOF

}

BEGIN {
    return;  # Comment out this line to get callback traces
    no strict 'refs';
    my $pkg = __PACKAGE__ . '::';
    my $stash = \ %$pkg;
    use strict 'refs';
    for my $name (keys %$stash) {
        my $glob = $stash->{$name};
        if (*$glob{CODE}) {
            my $code = *$glob{CODE};
            no warnings 'redefine';
            $stash->{$name} = sub {
                warn "Calling $name(@_)\n";
                &$code(@_);
            };
        }
    }
}


1;

__END__
```

```

=encoding UTF-8

=head1 NAME

Lemplate::Directive - Lemplate Code Generating Backend

=head1 SYNOPSIS

    use Lemplate::Directive;

=head1 DESCRIPTION

Lemplate::Directive is the analog to Template::Directive, which is the
module that produces that actual code that templates turn into. The
Lemplate version obviously produces Lua code rather than Perl.
Other than that the two modules are almost exactly the same.

=head1 BUGS

Unfortunately, some of the code generation seems to happen before
Lemplate::Directive gets control. So it currently has heuristical code
to rejigger Perl code snippets into Lua. This processing needs to
happen upstream once I get more clarity on how Template::Toolkit works.

=head1 AUTHOR

Ingy döt Net <ingy@cpan.org>

=head1 COPYRIGHT

Copyright (c) 2016. Yichun Zhang (agentzh). All rights reserved.

Copyright (c) 2006-2014. Ingy döt Net. All rights reserved.

This program is free software; you can redistribute it and/or modify it
under the same terms as Perl itself.

See L<http://www.perl.com/perl/misc/Artistic.html>

=cut
```

One Level Up   Top Level

lib/Lemplate/Parser.pm - lemplate

Data types defined

- new

Source code

```
package Lemplate::Parser;
use strict;
use warnings;
use base 'Template::Parser';

# VERSION

use Lemplate::Grammar;
use Lemplate::Directive;

sub new {
    my $class = shift;
    my $parser = $class->SUPER::new(
        GRAMMAR => Lemplate::Grammar->new(),
        FACTORY => 'Lemplate::Directive',
        @_,
    );

    # flags passed from Lemplate object
    my %args = @_;

    # eval-javascript is default "on"
    $parser->{EVAL_JAVASCRIPT} = exists $args{EVAL_JAVASCRIPT}
      ? $args{EVAL_JAVASCRIPT} : 1;

    # tie the parser state-variable to the global Directive var
    $parser->{INJAVASCRIPT} = \$Lemplate::Directive::INJAVASCRIPT;

    return $parser;
}

1;

__END__

=encoding UTF-8

=head1 NAME

Lemplate::Parser - Lemplate Parser Subclass

=head1 SYNOPSIS

    use Lemplate::Parser;

=head1 DESCRIPTION

Lemplate::Parser is a simple subclass of Template::Parser. Not much
to see here.

=head1 AUTHOR

Ingy döt Net <ingy@cpan.org>

=head1 COPYRIGHT

Copyright (c) 2006-2014. Ingy döt Net. All rights reserved.

This program is free software; you can redistribute it and/or modify it
under the same terms as Perl itself.

See L<http://www.perl.com/perl/misc/Artistic.html>
```

```

=cut
```

One Level Up Top Level

lib/Lemplate/Runtime/ - lemplate

- Compact.pm

lib/Lemplate/Runtime/Compact.pm - lemplate

Data types defined

- ajax_jquery
- ajax_xhr
- ajax_yui
- json2
- json_json2
- json_json2_internal
- json_yui
- kernel
- main
- xhr_gregory
- xhr_ilinsky
- xxx

Source code

```
package Lemplate::Runtime::Compact;
use strict;
use warnings;

# VERSION

sub main { return &kernel }
sub kernel {
    <<'...';
 ...
 }

 sub ajax_jquery {
     <<'...';
 ...
 }

 sub ajax_xhr {
     <<'...';
 ...
 }

 sub ajax_yui {
     <<'...';
 ...
 }

 sub json_json2 {
     <<'...';
 ...
 }

 sub json_json2_internal {
     <<'...';
 ;(function(){

 var JSON;



 }());
 ...
 }

 sub json_yui {
     <<'...';
 ...
 }

 sub json2 {
     <<'...';
 ...
 }

 sub xhr_gregory {
     <<'...';
 ...
 }

 sub xhr_ilinsky {
     <<'...';
 ...
 }

 sub xxx {
     <<'...';
 ...
 }

 1;

 __END__

 =encoding UTF-8

 =head1 NAME

 Lemplate::Runtime - Perl Module containing the Lemplate JavaScript Runtime

 =head1 SYNOPSIS

     use Lemplate::Runtime;
     print Lemplate::Runtime->main;

 =head1 DESCRIPTION

 This module is auto-generated and used internally by Lemplate. It
 contains subroutines that simply return various parts of the Lemplate
 JavaScript Runtime code.

 =head1 METHODS

 head2 kernel

 head2 ajax_jquery

 head2 ajax_xhr

 head2 ajax_yui

 head2 json_json2

 head2 json_yui

 head2 json2

 head2 xhr_gregory

 head2 xhr_ilinsky

 head2 xxx

 =head1 COPYRIGHT

```

```
Copyright (c) 2014. Ingy döt Net.

This program is free software; you can redistribute it and/or modify it
under the same terms as Perl itself.

See L<http://www.perl.com/perl/misc/Artistic.html>

=cut
```

src/lib/Lemplate/Runtime/Compact.pm - lemplate

Data types defined

- ajax_jquery
- ajax_xhr
- ajax_yui
- json2
- json_json2
- json_json2_internal
- json_yui
- kernel
- main
- xhr_gregory
- xhr_ilinsky
- xxx

Source code

```
package Lemplate::Runtime::Compact;
use strict;
use warnings;

sub main { return &kernel }
sub kernel {
    <<'...';
...
}

 sub ajax_jquery {
     <<'...';
 ...
 }

 sub ajax_xhr {
     <<'...';
 ...
 }

 sub ajax_yui {
     <<'...';
 ...
 }

 sub json_json2 {
     <<'...';
 ...
 }

 sub json_json2_internal {
     <<'...';
 ;(function(){

 var JSON;



 }());
 ...
 }

 sub json_yui {
     <<'...';
 ...
 }

 sub json2 {
     <<'...';
 ...
 }

 sub xhr_gregory {
     <<'...';
 ...
 }

 sub xhr_ilinsky {
     <<'...';
 ...
 }

 sub xxx {
     <<'...';
 ...
 }

 1;

 __END__

 =encoding UTF-8

 =head1 NAME

 Lemplate::Runtime - Perl Module containing the Lemplate JavaScript Runtime

 =head1 SYNOPSIS

     use Lemplate::Runtime;
     print Lemplate::Runtime->main;

 =head1 DESCRIPTION

 This module is auto-generated and used internally by Lemplate. It
 contains subroutines that simply return various parts of the Lemplate
 JavaScript Runtime code.

 =head1 METHODS

 head2 kernel

 head2 ajax_jquery

 head2 ajax_xhr

 head2 ajax_yui

 head2 json_json2

  head2 json_yui

  head2 json2

  head2 xhr_gregory

  head2 xhr_ilinsky

  head2 xxx

  =head1 COPYRIGHT

  Copyright (c) 2014. Ingy döt Net.

```

```
This program is free software; you can redistribute it and/or modify it
under the same terms as Perl itself.

See L<http://www.perl.com/perl/misc/Artistic.html>

=cut
```

src/lib/Lemplate/Runtime/ - lemplate

- Compact.pm

src/lib/Lemplate/ - lemplate

- Runtime/
- Runtime.pm

src/lib/ - lemplate

- Lemplate/

src/ - lemplate

- Makefile
- bin/
- lib/
- parser/

src/Makefile - lemplate

```
.PHONY: fetch runtime

.DELETE_ON_ERROR: $(RUNTIME_MODULE) $(RUNTIME_COMPACT_MODULE)

JEMPLATE_SCRIPT=../bin/lemplate
JEMPLATE_STANDALONE_SCRIPT=../lemplate
GRAMMAR_MODULE=../lib/Lemplate/Grammar.pm
RUNTIME_MODULE=../lib/Lemplate/Runtime.pm
RUNTIME_COMPACT_MODULE=../lib/Lemplate/Runtime/Compact.pm
JEMPLATE_MODULES=$(GRAMMAR_MODULE) $(RUNTIME_MODULE) $(RUNTIME_COMPACT_MODULE)

all: $(JEMPLATE_STANDALONE_SCRIPT)

$(JEMPLATE_STANDALONE_SCRIPT): $(JEMPLATE_MODULES) _force
	./bin/make-standalone-script $(JEMPLATE_SCRIPT) > $@
	chmod +x $@

$(GRAMMAR_MODULE): parser _force
	(cd parser; ./yc)
	mv parser/Grammar.pm $@
	rm parser/Parser.output

$(RUNTIME_MODULE): lib/Lemplate/Runtime.pm _force
	bin/tpage $< > $@

$(RUNTIME_COMPACT_MODULE): lib/Lemplate/Runtime/Compact.pm _force
	bin/tpage $< > $@

_force:
```

src/bin/ - lemplate

- make-standalone-script

src/bin/make-standalone-script - lemplate

Data types defined

- disable_libs
- get_module
- guts

Source code

```
#!/usr/bin/env perl

use strict;
use warnings;
use FindBin qw($Bin);
use lib "$Bin/../../lib", "$Bin/../lib";
use Template;
use IO::All;

 {
     my $script = io(shift)->all;
     $script =~ s{^#!/usr/bin/perl$}{#!/usr/bin/env perl}m;

     $script =~ /(.*\n#BOOTSTRAP-BEGIN\n).*\n(#BOOTSTRAP-END\n.*)/s
         or die;

     print $1 . guts() . $2;
 }

 sub guts {
     my $output = '';
     for (qw(
         Number::Compare
         Text::Glob
         File::Find::Rule
         Template::Constants
         Template::Base
         Template::Config
         Template::Document
         Template::Exception
         Template::Service
         Template::Provider
         Template
         Template::Grammar
         Template::Directive
         Template::Parser
         Lemplate::Directive
         Lemplate::Grammar
         Lemplate::Parser
         Lemplate::Runtime
         Lemplate::Runtime::Compact
         Lemplate
     )) {
         $output .= get_module($_);
     }
     return disable_libs() . $output;
 }

 sub disable_libs {
     return <<'...';
 # This is the standalone Lemplate compiler.
 #
 # All you need is this program and the program called `perl`. You don't need
 # to install any Perl modules.
 #
 # If you downloaded this program from the internet, don't forget to put it in
 # your path and make sure it is executable. Like this:
 #
 #     mv lemplate /usr/local/bin/
 #     chmod +x /usr/local/bin/lemplate
 #
 # Try this command to make sure it works:
 #
 #     lemplate --help

 use Config;
 BEGIN {
     @INC = (
         $Config::Config{archlib},
         $Config::Config{privlib},
     );
 }
 use strict;
 use warnings;

 ...
 }

 sub get_module {
     my $module = shift;
     eval "require $module; 1" or die "$module not found";
     $module =~ s{::}{/}g;
     $module .= '.pm';
     my $content = io($INC{$module})->all;
     # Get rid of DATA section
     $content =~ s/^__(END|DATA)__.*//sm;
     # Remove POD
     $content =~ s/^=\w+.*?(\n=cut\n|\z)//msg;
     # Remove comments
     $content =~ s/^#.*\n//gm;

     # Return the concatenation of prerequisite modules
     return
         "#\n# Inline include of $module\n#\n" .
         "BEGIN { \$INC{'$module'} = 'dummy/$module'; }\n" .
         "BEGIN {\n" .
         "#line 0 \"$module\"\n" .
         $content .
         "\n}\n" .
         "";
 }
```

src/parser/ - lemplate

- Grammar.pm.skel
- Parser.yp
- yc

src/parser/Grammar.pm.skel - lemplate

```
#============================================================= -*-Perl-*-
#
# Lemplate::Grammar
#
# DESCRIPTION
#   Grammar file for the Template Toolkit language containing token
#   definitions and parser state/rules tables generated by Parse::Yapp.
#
# AUTHOR
#   Ingy döt Net   <ingy@cpan.org>
#
# ORIGINAL AUTHOR
#   Andy Wardley   <abw@kfs.org>
#
# COPYRIGHT
#   Copyright (C) 2006-2008 Ingy döt Net.
#   Copyright (C) 1996-2000 Andy Wardley.
#   Copyright (C) 1998-2000 Canon Research Centre Europe Ltd.
#
#   This module is free software; you can redistribute it and/or
#   modify it under the same terms as Perl itself.
#
#------------------------------------------------------------------------
#
# NOTE: this module is constructed from the parser/Grammar.pm.skel
# file by running the parser/yc script.  You only need to do this if
# you have modified the grammar in the parser/Parser.yp file and need
# to-recompile it.  See the README in the 'parser' directory for more
# information (sub-directory of the Lemplate distribution).
#
#========================================================================

package Lemplate::Grammar;

require 5.004;

use strict;
use vars qw( $VERSION );

$VERSION  = sprintf("%d.%02d", q$Revision: 2.10 $ =~ /(\d+)\.(\d+)/);

my (@RESERVED, %CMPOP, $LEXTABLE, $RULES, $STATES);
my ($factory, $rawstart);


#========================================================================
# Reserved words, comparison and binary operators
#========================================================================

@RESERVED = qw(
   GET CALL SET DEFAULT INSERT INCLUDE PROCESS WRAPPER BLOCK END
   USE RAW PLUGIN FILTER MACRO JAVASCRIPT TO STEP AND OR NOT DIV MOD
   IF UNLESS ELSE ELSIF FOR NEXT WHILE SWITCH CASE META IN
   TRY THROW CATCH FINAL LAST RETURN STOP CLEAR VIEW DEBUG
    );

# for historical reasons, != and == are converted to ne and eq to perform
# stringwise comparison (mainly because it doesn't generate "non-numerical
# comparison" warnings which != and == can) but the others (e.g. < > <= >=)
# are not converted to their stringwise equivalents.  I added 'gt' et al,
# briefly for v2.04d and then took them out again in 2.04e.

%CMPOP = qw(
    != ~=
    == ==
    <  <
    >  >
    >= >=
    <= <=
);

```

```

#========================================================================
# Lexer Token Table
#========================================================================

# lookup table used by lexer is initialised with special-cases
$LEXTABLE = {
    'FOREACH' => 'FOR',
    'BREAK'   => 'LAST',
    '&&'      => 'AND',
    '||'      => 'OR',
    '!'       => 'NOT',
    '|'         => 'FILTER',
    '.'       => 'DOT',
    '_'       => 'CAT',
    '..'      => 'TO',
#    ':'       => 'MACRO',
    '='       => 'ASSIGN',
    '=>'      => 'ASSIGN',
#    '->'      => 'ARROW',
    ','       => 'COMMA',
    '\\'      => 'REF',
    'and'     => 'AND',      # explicitly specified so that qw( and or
    'or'      => 'OR',      # not ) can always be used in lower case,
    'not'     => 'NOT',      # regardless of ANYCASE flag
    'mod'     => 'MOD',
    'div'     => 'DIV',
};

# localise the temporary variables needed to complete lexer table
{
#    my @tokens = qw< ( ) [ ] { } ${ $ / ; : ? >;
    my @tokens = qw< ( ) [ ] { } ${ $ + / ; : ? >;
    my @cmpop  = keys %CMPOP;
#    my @binop  = qw( + - * % );              # '/' above, in @tokens
    my @binop  = qw( - * % );              # '+' and '/' above, in @tokens

    # fill lexer table, slice by slice, with reserved words and operators
    @$LEXTABLE{ @RESERVED, @cmpop, @binop, @tokens }
   = ( @RESERVED, ('CMPOP') x @cmpop, ('BINOP') x @binop, @tokens );
}


#========================================================================
# CLASS METHODS
#========================================================================

sub new {
    my $class = shift;
    bless {
   LEXTABLE => $LEXTABLE,
   STATES   => $STATES,
   RULES    => $RULES,
    }, $class;
}

# update method to set package-scoped $factory lexical
sub install_factory {
    my ($self, $new_factory) = @_;
    $factory = $new_factory;
}


#========================================================================
# States
#========================================================================

$STATES = <<$states>>;


#========================================================================
# Rules
#========================================================================

$RULES = <<$rules>>;

```

```
1;
```

One Level Up Top Level

src/parser/Parser.yp - template

```
#============================================================= -*-Perl-*-
#
# Parser.yp
#
# DESCRIPTION
#   Definition of the parser grammar for the Template Toolkit language.
#
# AUTHOR
#   Ingy döt Net   <ingy@cpan.org>
#
# ORIGINAL AUTHOR
#   Andy Wardley   <abw@kfs.org>
#
# HISTORY
#   Totally re-written for version 2, based on Doug Steinwand's
#   implementation which compiles templates to Perl code.  The generated
#   code is _considerably_ faster, more portable and easier to process.
#
# WARNINGS
#   Expect 1 reduce/reduce conflict.  This can safely be ignored.
#   Now also expect 1 shift/reduce conflict, created by adding a rule
#   to 'args' to allow assignments of the form 'foo.bar = baz'.  It
#   should be possible to fix the problem by rewriting some rules, but
#   I'm loathed to hack it up too much right now.  Maybe later.
#
# COPYRIGHT
#   Copyright (C) 2006,2008 Ingy döt Net.
#   Copyright (C) 1996-2004 Andy Wardley.
#   Copyright (C) 1998-2004 Canon Research Centre Europe Ltd.
#
#   This module is free software; you can redistribute it and/or
#   modify it under the same terms as Perl itself.
#
#------------------------------------------------------------------------
#
# NOTE: this module is constructed from the parser/Grammar.pm.skel
# file by running the parser/yc script.  You only need to do this if
# you have modified the grammar in the parser/Parser.yp file and need
# to-recompile it.  See the README in the 'parser' directory for more
# information (sub-directory of the Template distribution).
#
#------------------------------------------------------------------------
#
# $Id: Parser.yp,v 2.20 2004/01/13 15:32:22 abw Exp $
#
#========================================================================

%right ASSIGN
%right '?' ':'
%left COMMA
%left AND OR
%left NOT
%left CAT
%left DOT
%left CMPOP
%left BINOP
%left '+'
%left '/'
%left DIV
%left MOD
%left TO
%%

#--------------------------------------------------------------------------
# START AND TOP-LEVEL RULES
#--------------------------------------------------------------------------

template:   block          { $factory->template($_[1])           }
;

block:       chunks          { $factory->block($_[1])              }
```

```
   |   /* NULL */          { $factory->block()                   }
;

chunks:       chunks chunk       { push(@{$_[1]}, $_[2])
               if defined $_[2]; $_[1]           }
    |   chunk          { defined $_[1] ? [ $_[1] ] : [ ]     }
;

chunk:  TEXT             { $factory->textblock($_[1])          }
    |   statement ';'    { return '' unless $_[1];
                           $_[0]->location() . $_[1];
                         }
;

statement:  directive
    |   defblock
   |   anonblock
   |   capture
   |   macro
   |   use
   |   raw
   |   view
   |   rawperl
   |   expr          { $factory->get($_[1])                 }
   |   META metadata        { $_[0]->add_metadata($_[2]);         }
   |   /* empty statement */
;

directive:  setlist          { $factory->set($_[1])                }
   |   atomdir
   |   condition
   |   switch
   |   loop
   |   try
        |   javascript
   |   perl
;


#-----------------------------------------------------------------------------
# DIRECTIVE RULES
#-----------------------------------------------------------------------------

atomexpr:   expr          { $factory->get($_[1])                }
   |   atomdir
;

atomdir:    GET expr          { $factory->get($_[2])                }
   |   CALL expr          { $factory->call($_[2])               }
   |   SET setlist           { $factory->set($_[2])                }
   |   DEFAULT setlist         { $factory->default($_[2])            }
   |   INSERT nameargs         { $factory->insert($_[2])             }
   |   INCLUDE nameargs       { $factory->include($_[2])            }
   |   PROCESS nameargs       { $factory->process($_[2])            }
   |   THROW nameargs          { $factory->throw($_[2])              }
   |   RETURN          { $factory->return()                  }
   |   STOP          { $factory->stop()                    }
   |   CLEAR                   { $factory->clear()                   }
   |   LAST                    { $factory->break()                   }
   |   NEXT          { $factory->next()                    }
   |   DEBUG nameargs          { if ($_[2]->[0]->[0] =~ /^'(on|off)'$/) {
                      $_[0]->{ DEBUG_DIRS } = ($1 eq 'on');
                 $factory->debug($_[2]);
                  }
                  else {
                 $_[0]->{ DEBUG_DIRS } ? $factory->debug($_[2]) : '';
                  }
                }
        |   wrapper
   |   filter
;

condition:  IF expr ';'
         block else END       { $factory->if(@_[2, 4, 5])           }
   |   atomexpr IF expr       { $factory->if(@_[3, 1])              }
   |   UNLESS expr ';'
```

```
          block else END       { $factory->if("tt2_not($_[2])", @_[4, 5])  }
    |   atomexpr UNLESS expr    { $factory->if("tt2_not($_[3])", $_[1])     }
;

else:       ELSIF expr ';'
          block else       { unshift(@{$_[5]}, [ @_[2, 4] ]);
                    $_[5];                              }
    |   ELSE ';' block       { [ $_[3] ]                            }
    |   /* NULL */              { [ undef ]                            }
;

switch:       SWITCH expr ';'
          block case END       { $factory->switch(@_[2, 5])          }
;

case:       CASE term ';' block
          case          { unshift(@{$_[5]}, [ @_[2, 4] ]);
                    $_[5];                              }
    |   CASE DEFAULT ';' block  { [ $_[4] ]                            }
    |   CASE ';' block       { [ $_[3] ]                            }
    |   /* NULL */          { [ undef ]                            }
;

loop:       FOR loopvar ';'          { $_[0]->{ INFOR }++                  }
       block END       { $_[0]->{ INFOR }--;
                    $factory->foreach(@{$_[2]}, $_[5])  }
#loop:       FOR loopvar ';'
#      block END       { $factory->foreach(@{$_[2]}, $_[4])  }
    |   atomexpr FOR loopvar    { $factory->foreach(@{$_[3]}, $_[1])  }
    |   WHILE expr ';'          { $_[0]->{ INWHILE }++                 }
          block END          { $_[0]->{ INWHILE }--;
                                        $factory->while(@_[2, 5])           }
    |   atomexpr WHILE expr       { $factory->while(@_[3, 1])           }
;

loopvar:    IDENT ASSIGN term args  { [ @_[1, 3, 4] ]                      }
        |   IDENT IN term args      { [ @_[1, 3, 4] ]                      }
    |   term args               { [ 0, @_[1, 2] ]                      }
;

wrapper:    WRAPPER nameargs ';'
          block END          { $factory->wrapper(@_[2, 4])         }
    |   atomexpr
          WRAPPER nameargs       { $factory->wrapper(@_[3, 1])         }
;

try:       TRY ';'
          block final END       { $factory->try(@_[3, 4])             }
;

final:       CATCH filename ';'
          block final       { unshift(@{$_[5]}, [ @_[2,4] ]);
                    $_[5];                              }
    |   CATCH DEFAULT ';'
          block final       { unshift(@{$_[5]}, [ undef, $_[4] ]);
                    $_[5];                              }
    |   CATCH ';'
          block final       { unshift(@{$_[4]}, [ undef, $_[3] ]);
                    $_[4];                              }
    |    FINAL ';' block       { [ $_[3] ]                            }
    |   /* NULL */          { [ 0 ] } # no final
;

use:       USE lnameargs       { $factory->use($_[2])                }
;

raw:       RAW lnameargs       { $factory->raw($_[2])                }
;

view:       VIEW nameargs ';'       { $_[0]->push_defblock();        }
          block END             { $factory->view(@_[2,5],
                         $_[0]->pop_defblock) }
;

javascript: JAVASCRIPT ';'                 { ${$_[0]->{ INJAVASCRIPT }}++;             }
          block END          { ${$_[0]->{ INJAVASCRIPT }}--;
```

```
                  $_[0]->{ EVAL_JAVASCRIPT }
                  ? $factory->javascript($_[4])
                  : $factory->no_javascript();              }
;

filter:       FILTER lnameargs ';'
         block END          { $factory->filter(@_[2,4])           }
   |   atomexpr FILTER
         lnameargs          { $factory->filter(@_[3,1])           }
;

defblock: defblockname
          blockargs ';'
          template END          { my $name = join('/', @{ $_[0]->{ DEFBLOCKS } });
                  pop(@{ $_[0]->{ DEFBLOCKS } });
                  $_[0]->define_block($name, $_[4]);
                  undef
                }
;

defblockname: BLOCK blockname       { push(@{ $_[0]->{ DEFBLOCKS } }, $_[2]);
                  $_[2];
                }
;

blockname:  filename
        |   LITERAL          { $_[1] =~ s/^'(.*)'$/$1/; $_[1]      }
;

blockargs:  metadata
   |   /* NULL */
;

anonblock:  BLOCK blockargs ';' block END
                { local $" = ', ';
                  print STDERR "experimental block args: [@{ $_[2] }]\n"
                 if $_[2];
                  $factory->anon_block($_[4])         }
;

capture:    ident ASSIGN mdir       { $factory->capture(@_[1, 3])         }
;

macro:      MACRO IDENT '(' margs ')'
      mdir                { $factory->macro(@_[2, 6, 4])        }
   |   MACRO IDENT mdir        { $factory->macro(@_[2, 3])           }
;

mdir:       directive
   |   BLOCK ';' block END       { $_[3]                               }
;

margs:       margs IDENT          { push(@{$_[1]}, $_[2]); $_[1]        }
   |   margs COMMA             { $_[1]                               }
   |   IDENT                   { [ $_[1] ]                           }
;

metadata:   metadata meta       { push(@{$_[1]}, @{$_[2]}); $_[1]     }
   |   metadata COMMA
   |   meta
;

meta:       IDENT ASSIGN LITERAL       { for ($_[3]) { s/^'//; s/'$//;
                         s/\\'/'/g  };
                [ @_[1,3] ] }
   |   IDENT ASSIGN '"' TEXT '"'  { [ @_[1,4] ] }
   |   IDENT ASSIGN NUMBER        { [ @_[1,3] ] }
;


#------------------------------------------------------------------------
# FUNDAMENTAL ELEMENT RULES
#------------------------------------------------------------------------

term:       lterm
   |   sterm
```

```
;

lterm:       '[' list  ']'       { "{ $_[2] }"                        }
    |   '[' range ']'           { "{ $_[2] }"                        }
    |   '['       ']'           { "{ }"                              }
    |   '{' hash  '}'       { "{ $_[2]  }"                       }
;

sterm:       ident          { $factory->ident($_[1])             }
    |   REF ident               { $factory->identref($_[2])          }
    |   '"' quoted '"'       { $factory->quoted($_[2])            }
    |   LITERAL
    |   NUMBER
;

list:       list term           { "$_[1], $_[2]"                     }
    |   list COMMA
    |   term
;

range:       sterm TO sterm       { $_[1] . '..' . $_[3]               }
;


hash:       params
    |   /* NULL */          { "" }
;

params:       params param       { "$_[1], $_[2]"                     }
    |   params COMMA
    |   param
;

param:      LITERAL ASSIGN expr     { "[$_[1]] = $_[3]"                    }
         |   item ASSIGN expr       { "[$_[1]] = $_[3]"                    }
;

ident:      ident DOT node       { push(@{$_[1]}, @{$_[3]}); $_[1]     }
    |   ident DOT NUMBER       { push(@{$_[1]},
                  map {($_, 0)} split(/\./, $_[3]));
                  $_[1];                     }
    |   node
;

node:       item          { [ $_[1], 0 ]                       }
    |   item '(' args ')'       { [ $_[1], $factory->args($_[3]) ]    }
;

item:       IDENT          { "'$_[1]'"                          }
    |   '${' sterm '}'       { $_[2]                              }
    |   '$' IDENT          { $_[0]->{ V1DOLLAR }
                   ? "'$_[2]'"
                   : $factory->ident(["'$_[2]'", 0])  }
;

expr:       expr BINOP expr       { "$_[1] $_[2] $_[3]"                 }
    |   expr '/' expr       { "$_[1] $_[2] $_[3]"                 }
    |   expr '+' expr       { "$_[1] $_[2] $_[3]"                 }
    |   expr DIV expr       { "math_floor($_[1] / $_[3])"         }
    |   expr MOD expr       { "$_[1] % $_[3]"                     }
    |   expr CMPOP expr       { "$_[1] $CMPOP{ $_[2] } $_[3]"       }
    |   expr CAT expr       { "$_[1] .. $_[3]"                    }
    |   expr AND expr       { "tt2_true($_[1]) and tt2_true($_[3])"                  }
    |   expr OR expr       { "tt2_true($_[1]) or tt2_true($_[3])"                  }
    |   NOT expr          { "tt2_not($_[2])"                       }
    |   expr '?' expr ':' expr  { "tt2_true($_[1]) and $_[3] or $_[5]"          }
    |   '(' assign ')'       { $factory->assign(@{$_[2]})          }
    |   '(' expr ')'       { "($_[2])"                          }
    |   term
;

setlist:    setlist assign       { push(@{$_[1]}, @{$_[2]}); $_[1]     }
    |   setlist COMMA
    |   assign
;

```

```

assign:       ident ASSIGN expr       { [ $_[1], $_[3] ]                    }
    |   LITERAL ASSIGN expr       { [ @_[1,3] ]                         }
;

# The 'args' production constructs a list of named and positional
# parameters.  Named parameters are stored in a list in element 0
# of the args list.  Remaining elements contain positional parameters

args:       args expr          { push(@{$_[1]}, $_[2]); $_[1]        }
    |   args param             { push(@{$_[1]->[0]}, $_[2]); $_[1]   }
    |   args ident ASSIGN expr  { push(@{$_[1]->[0]}, "'', " .
                  $factory->assign(@_[2,4])); $_[1]  }
    |   args COMMA          { $_[1]                               }
    |   /* init */          { [ [ ] ]                             }
;


# These are special case parameters used by INCLUDE, PROCESS, etc., which
# interpret barewords as quoted strings rather than variable identifiers;
# a leading '$' is used to explicitly specify a variable.  It permits '/',
# '.' and '::' characters, allowing it to be used to specify filenames, etc.
# without requiring quoting.

lnameargs:  lvalue ASSIGN nameargs  { push(@{$_[3]}, $_[1]); $_[3]        }
    |   nameargs
;

lvalue:       item
    |   '"' quoted '"'       { $factory->quoted($_[2])             }
    |   LITERAL
;

nameargs:   '$' ident args       { [ [$factory->ident($_[2])], $_[3] ]   }
    |   names args          { [ @_[1,2] ] }
    |   names '(' args ')'       { [ @_[1,3] ] }
;

names:       names '+' name       { push(@{$_[1]}, $_[3]); $_[1] }
    |   name          { [ $_[1] ]                    }
;

name:       '"' quoted '"'       { $factory->quoted($_[2])  }
    |   filename                 { "'$_[1]'" }
    |    LITERAL
;

#nameargs:   literal args       { [ @_[1,2] ] }
#   |   literal '(' args ')'    { [ @_[1,3] ] }
#   |   '$' ident
#;

#namesargs:  names args          { [ @_[1,2] ] }
#;

filename:   filename DOT filepart   { "$_[1].$_[3]" }
    |   filepart
;

filepart: FILENAME | IDENT | NUMBER
;


# The 'quoted' production builds a list of 'quotable' items that might
# appear in a quoted string, namely text and identifiers.  The lexer
# adds an explicit ';' after each directive it finds to help the
# parser identify directive/text boundaries; we're not interested in
# them here so we can simply accept and ignore by returning undef

quoted:       quoted quotable       { push(@{$_[1]}, $_[2])
                       if defined $_[2]; $_[1]         }
    |   /* NULL */          { [ ]                                 }
;

quotable:   ident          { $factory->ident($_[1])              }
    |   TEXT          { $factory->text($_[1])              }
    |   ';'                     { undef                                 }
;


%%



```

src/parser/yc - lemplate

```
#!/bin/sh
#========================================================================
#
# yc - yapp compile
#
# This calls 'yapp', distributed with the Parse::Yapp module, to
# compile the parser grammar and construct the ../Template/Grammar.pm
# file.  The grammar is defined in ./Parser.yp.  The skeleton file
# Grammar.pm.skel is used as a template for creating the grammar file.
# An output file 'Parser.output' is generated containing a summary of
# the rule and state tables.
#
# You only need to run this script if you have changed the grammar and
# wish to recompile it.
#
# Andy Wardley <abw@kfs.org>
#
#========================================================================

: ${GRAMMAR:="Parser.yp"}
# : ${OUTPUT:="../lib/Lemplate/Grammar.pm"}
: ${OUTPUT:="Grammar.pm"}
: ${TEMPLATE:="Grammar.pm.skel"}

echo "Compiling parser grammar (${GRAMMAR} -> ${OUTPUT})"

yapp -v -s -o ${OUTPUT} -t ${TEMPLATE} ${GRAMMAR}

```

src/lib/Lemplate/Runtime.pm - lemplate

Data types defined

- ajax_jquery
- ajax_xhr
- ajax_yui
- json2
- json_json2
- json_json2_internal
- json_yui
- kernel
- main
- xhr_gregory
- xhr_ilinsky
- xxx

Source code

```
package Lemplate::Runtime;
use strict;
use warnings;

sub main { return &kernel }
sub kernel {
    <<'...';
...
}

 sub ajax_jquery {
     <<'...';
 ...
 }

 sub ajax_xhr {
     <<'...';
 ...
 }

 sub ajax_yui {
     <<'...';
 ...
 }

 sub json_json2 {
     <<'...';
 ...
 }

 sub json_json2_internal {
     <<'...';
 ;(function(){

 var JSON;

 }());
 ...
 }

 sub json_yui {
     <<'...';
 ...
 }

 sub json2 {
     <<'...';
 ...
 }

 sub xhr_gregory {
     <<'...';
 ...
 }

 sub xhr_ilinsky {
     <<'...';
 ...
 }

 sub xxx {
     <<'...';
 ...
 }

 1;

 __END__

 =encoding UTF-8

 =head1 NAME

 Lemplate::Runtime - Perl Module containing the Lemplate JavaScript Runtime

 =head1 SYNOPSIS

     use Lemplate::Runtime;
     print Lemplate::Runtime->main;

 =head1 DESCRIPTION

 This module is auto-generated and used internally by Lemplate. It
 contains subroutines that simply return various parts of the Lemplate
 JavaScript Runtime code.

 =head1 METHODS

 head2 kernel

 head2 ajax_jquery

 head2 ajax_xhr

 head2 ajax_yui

 head2 json_json2

 head2 json_yui

  head2 json2

  head2 xhr_gregory

  head2 xhr_ilinsky

  head2 xxx

  =head1 COPYRIGHT

  Copyright (c) 2014. Ingy döt Net.

  This program is free software; you can redistribute it and/or modify it
  under the same terms as Perl itself.
```

```

See L<http://www.perl.com/perl/misc/Artistic.html>

=cut
```

One Level Up Top Level

lib/Lemplate/Runtime.pm - lemplate

Data types defined

- ajax_jquery
- ajax_xhr
- ajax_yui
- json2
- json_json2
- json_json2_internal
- json_yui
- kernel
- main
- xhr_gregory
- xhr_ilinsky
- xxx

Source code

```
package Lemplate::Runtime;
use strict;
use warnings;

# VERSION

sub main { return &kernel }
sub kernel {
    <<'...';
 ...
 }

 sub ajax_jquery {
     <<'...';
 ...
 }

 sub ajax_xhr {
     <<'...';
 ...
 }

 sub ajax_yui {
     <<'...';
 ...
 }

 sub json_json2 {
     <<'...';
 ...
 }

 sub json_json2_internal {
     <<'...';
 ;(function(){

 var JSON;

 }());
 ...
 }

 sub json_yui {
     <<'...';
 ...
 }

 sub json2 {
     <<'...';
 ...
 }

 sub xhr_gregory {
     <<'...';
 ...
 }

 sub xhr_ilinsky {
     <<'...';
 ...
 }

 sub xxx {
     <<'...';
 ...
 }

 1;

 __END__

 =encoding UTF-8

 =head1 NAME

 Lemplate::Runtime - Perl Module containing the Lemplate Lua Runtime

 =head1 SYNOPSIS

     use Lemplate::Runtime;
     print Lemplate::Runtime->main;

 =head1 DESCRIPTION

 This module is auto-generated and used internally by Lemplate. It
 contains subroutines that simply return various parts of the Lemplate
 Lua Runtime code.

 =head1 METHODS

 head2 kernel

 head2 ajax_jquery

 head2 ajax_xhr

 head2 ajax_yui

 head2 json_json2

  head2 json_yui

  head2 json2

  head2 xhr_gregory

  head2 xhr_ilinsky

  head2 xxx

  =head1 COPYRIGHT

  Copyright (c) 2014. Ingy döt Net.

```

```
This program is free software; you can redistribute it and/or modify it
under the same terms as Perl itself.

See L<http://www.perl.com/perl/misc/Artistic.html>

=cut
```

lib/Lemplate.pm - lemplate

Data types defined

Source code

```
# ToDo:
# - Use TT:Simple in Makefiles

# ABSTRACT: compiles Perl TT2 templates to standalone Lua modules for OpenResty

package Lemplate;
use strict;
use warnings;
use Template 2.14;
use Getopt::Long;

# VERSION

use Lemplate::Parser;

#------------------------------------------------------------------------------

our %ExtraTemplates;
our %ProcessedTemplates;
our $TemplateName;

sub usage {
    <<'...';
Usage:

    lemplate --runtime [runtime-opt]

    lemplate --compile [compile-opt] <template-list>

    lemplate --runtime [runtime-opt] --compile [compile-opt] <template-list>

    lemplate --list <template-list>

```

```
Where "--runtime" and "runtime-opt" can include:

    --runtime           Equivalent to --ajax=ilinsky --json=json2
    --runtime=standard

    --runtime=lite      Same as --ajax=none --json=none
    --runtime=jquery    Same as --ajax=jquery --json=none
    --runtime=yui       Same as --ajax=yui --json=yui
    --runtime=legacy    Same as --ajax=gregory --json=json2

    --json              By itself, equivalent to --json=json2
    --json=json2        Include http://www.json.org/json2.js for parsing/stringifying
    --json=yui          Use YUI: YAHOO.lang.JSON (requires external YUI)
    --json=none         Doesn't provide any JSON functionality except a warning

    --ajax              By itself, equivalent to --ajax=xhr
    --ajax=jquery       Use jQuery for Ajax get and post (requires external jQuery)
    --ajax=yui          Use YUI: yui/connection/connection.js (requires external YUI)
    --ajax=xhr          Use XMLHttpRequest (will automatically use --xhr=ilinsky if --xhr is not set)
    --ajax=none         Doesn't provide any Ajax functionality except a warning

    --xhr               By itself, equivalent to --xhr=ilinsky
    --xhr=ilinsky       Include http://code.google.com/p/xmlhttprequest/
    --xhr=gregory       Include http://www.scss.com.au/family/andrew/webdesign/xmlhttprequest/

    --xxx               Include XXX and JJJ helper functions

    --compact           Use the YUICompressor compacted version of the runtime

Where "compile-opt" can include:

    --include_path=DIR  Add directory to INCLUDE_PATH

    --start-tag
    --end-tag
    --pre-chomp
    --post-chomp
    --trim
    --any-case
    --eval
    --noeval
    -s, --source
    --exclude

For more information use:
    perldoc lemplate
...
}

sub main {
    my $class = shift;

    my @argv = @_;

    my ($template_options, $lemplate_options) = get_options(@argv);
    my ($runtime, $compile, $list) = @$lemplate_options{qw/runtime compile list/};

    if ($runtime) {
        print runtime_source_code(@$lemplate_options{qw/runtime ajax json xhr xxx compact/});
        return unless $compile;
    }

    my $templates = make_file_list($lemplate_options->{exclude}, @argv);
    print_usage_and_exit() unless @$templates;

    if ($list) {
         foreach (@$templates) {
             print STDOUT $_->{short} . "\n";
         }
         return;
     }

     if ($compile) {
         my $lemplate = Lemplate->new(%$template_options);
         print STDOUT $lemplate->_preamble;
         for (my $i = 0; $i < @$templates; $i++) {
            my $template = $templates->[$i];
            #warn "processing $template->{short}";
            my $content = slurp($template->{full});
            if ($content) {
                %ExtraTemplates = ();
                print STDOUT $lemplate->compile_template_content(
                    $content,
                    $template->{short}
                );
                my @new_files;
                for my $new_template (keys %ExtraTemplates) {
                    if (!$ProcessedTemplates{$new_template}) {
                        if (!-f $new_template) {
                            $new_template = "t/data/" . $new_template;
                        }
                        #warn $new_template;
                        if (-f $new_template) {
                            #warn "adding new template $new_template";
                            push @new_files, $new_template;
                        }
                    }
                }
                push @$templates, @{ make_file_list({}, @new_files) };
            }
        }
        print STDOUT "return _M\n";
        return;
    }

    print_usage_and_exit();
}

sub get_options {
    local @ARGV = @_;

    my $runtime;
    my $compile = 0;
    my $list = 0;

    my $start_tag = exists $ENV{LEMPLATE_START_TAG}
        ? $ENV{LEMPLATE_START_TAG}
        : undef;
    my $end_tag = exists $ENV{LEMPLATE_END_TAG}
        ? $ENV{LEMPLATE_END_TAG}
        : undef;
    my $pre_chomp = exists $ENV{LEMPLATE_PRE_CHOMP}
        ? $ENV{LEMPLATE_PRE_CHOMP}
        : undef;
    my $post_chomp = exists $ENV{LEMPLATE_POST_CHOMP}
        ? $ENV{LEMPLATE_POST_CHOMP}
        : undef;
    my $trim = exists $ENV{LEMPLATE_TRIM}
        ? $ENV{LEMPLATE_TRIM}
        : undef;
    my $anycase = exists $ENV{LEMPLATE_ANYCASE}
        ? $ENV{LEMPLATE_ANYCASE}
        : undef;
    my $eval_javascript = exists $ENV{LEMPLATE_EVAL_JAVASCRIPT}
        ? $ENV{LEMPLATE_EVAL_JAVASCRIPT}
        : 1;

    my $source  = 0;
    my $exclude = 0;
    my ($ajax, $json, $xxx, $xhr, $compact, $minify);

    my $help = 0;
    my @include_paths;

    GetOptions(
        "compile|c"     => \$compile,
        "list|l"        => \$list,
        "runtime|r:s"   => \$runtime,

        "start-tag=s"   => \$start_tag,
        "end-tag=s"     => \$end_tag,
        "trim=s"        => \$trim,
        "pre-chomp"     => \$pre_chomp,
        "post-chomp"    => \$post_chomp,
        "any-case"      => \$anycase,
        "eval!"         => \$eval_javascript,

        "source|s"      => \$source,
        "exclude=s"     => \$exclude,

        "ajax:s"        => \$ajax,
        "json:s"        => \$json,
        "xxx"           => \$xxx,
        "xhr:s"         => \$xhr,

        "include_path"  => \@include_paths,
        "compact"       => \$compact,
        "minify:s"      => \$minify,

        "help|?"        => \$help,
    ) or print_usage_and_exit();

    if ($help) {
        print_usage_and_exit();
    }

    ($runtime, $ajax, $json, $xxx, $xhr, $minify) = map { defined $_ && ! length $_ ? 1 : $_ } ($runtime,
$ajax, $json, $xxx, $xhr, $minify);
    $runtime = "standard" if $runtime && $runtime eq 1;

    print_usage_and_exit("Don't understand '--runtime $runtime'") if defined $runtime && ! grep { $runtime =~
m/$_/ } qw/standard lite jquery yui legacy/;
    print_usage_and_exit("Can't specify --list with a --runtime and/or the --compile option") if $list &&
($runtime || $compile);
    print_usage_and_exit() unless $list || $runtime || $compile;

    my $command =
        $runtime ? 'runtime' :
        $compile ? 'compile' :
        $list ? 'list' :
        print_usage_and_exit();

    my $options = {};
    $options->{START_TAG} = $start_tag if defined $start_tag;
    $options->{END_TAG} = $end_tag if defined $end_tag;
    $options->{PRE_CHOMP} = $pre_chomp if defined $pre_chomp;
    $options->{POST_CHOMP} = $post_chomp if defined $post_chomp;
    $options->{TRIM} = $trim if defined $trim;
    $options->{ANYCASE} = $anycase if defined $anycase;
    $options->{EVAL_JAVASCRIPT} = $eval_javascript if defined $eval_javascript;
    $options->{INCLUDE_PATH} = \@include_paths;

    return (
        $options,
        { compile => $compile, runtime => $runtime, list => $list,
            source => $source,
            exclude => $exclude,
            ajax => $ajax, json => $json, xxx => $xxx, xhr => $xhr,
            compact => $compact, minify => $minify },
    );
}


sub slurp {
    my $filepath = shift;
    open(F, '<', $filepath) or die "Can't open '$filepath' for input:\n$!";
    my $contents = do {local $/; <F>};
    close(F);
    return $contents;
}

sub recurse_dir {
    require File::Find::Rule;

    my $dir = shift;
    my @files;
    foreach ( File::Find::Rule->file->in( $dir ) ) {
        if ( m{/\.[^\.]+} ) {} # Skip ".hidden" files or directories
```

```perl
        else {
            push @files, $_;
        }
    }
    return @files;
}

sub make_file_list {
    my ($exclude, @args) = @_;

    my @list;

    foreach my $arg (@args) {
        unless (-e $arg) { next; } # file exists
        unless (-s $arg or -d $arg) { next; } # file size > 0 or directory (for Win platform)
        if ($exclude and $arg =~ m/$exclude/) { next; } # file matches exclude regex

        if (-d $arg) {
            foreach my $full ( recurse_dir($arg) ) {
                $full =~ /$arg(\/|)(.*)/;
                my $short = $2;
                push(@list, {full=>$full, short=>$short} );
            }
        }
        else {
            my $full = $arg;
            my $short = $full;
            $short =~ s/.*[\/\\]//;
            push(@list, {full=>$arg, short=>$short} );
        }
    }

    return [ sort { $a->{short} cmp $b->{short} } @list ];
}

sub print_usage_and_exit {
    print STDOUT join "\n", "", @_, "Aborting!", "\n" if @_;
    print STDOUT usage();
    exit;
}

sub runtime_source_code {
    require Lemplate::Runtime;
    require Lemplate::Runtime::Compact;

    unshift @_, "standard" unless @_;

    my ($runtime, $ajax, $json, $xhr, $xxx, $compact) = map { defined $_ ? lc $_ : "" } @_[0 .. 5];

    my $Lemplate_Runtime = $compact ? "Lemplate::Runtime::Compact" : "Lemplate::Runtime";

    if ($runtime eq "standard") {
        $ajax ||= "xhr";
        $json ||= "json2";
        $xhr ||= "ilinsky";
    }
    elsif ($runtime eq "jquery") {
        $ajax ||= "jquery";
    }
    elsif ($runtime eq "yui") {
        $ajax ||= "yui";
        $json ||= "yui";
    }
    elsif ($runtime eq "legacy") {
        $ajax ||= "xhr";
        $json ||= "json2";
        $xhr ||= "gregory";
        $xxx = 1;
    }
    elsif ($runtime eq "lite") {
    }

    $ajax = "xhr" if $ajax eq 1;
    $xhr ||= 1 if $ajax eq "xhr";
    $json = "json2" if $json eq 1;
    $xhr = "ilinsky" if $xhr eq 1;

    my @runtime;

    push @runtime, $Lemplate_Runtime->kernel if $runtime;

    push @runtime, $Lemplate_Runtime->json2 if $json =~ m/^json2?$/i;

    push @runtime, $Lemplate_Runtime->ajax_xhr if $ajax eq "xhr";
    push @runtime, $Lemplate_Runtime->ajax_jquery if $ajax eq "jquery";
    push @runtime, $Lemplate_Runtime->ajax_yui if $ajax eq "yui";

    push @runtime, $Lemplate_Runtime->json_json2 if $json =~ m/^json2?$/i;
    push @runtime, $Lemplate_Runtime->json_json2_internal if $json =~ m/^json2?[_-]?internal$/i;
    push @runtime, $Lemplate_Runtime->json_yui if $json eq "yui";

    push @runtime, $Lemplate_Runtime->xhr_ilinsky if $xhr eq "ilinsky";
    push @runtime, $Lemplate_Runtime->xhr_gregory if $xhr eq "gregory";

    push @runtime, $Lemplate_Runtime->xxx if $xxx;

    return join ";", @runtime;
}

#------------------------------------------------------------------------------

sub new {
    my $class = shift;
    return bless { @_ }, $class;
}

sub compile_module {
    my ($self, $module_path, $template_file_paths) = @_;
    my $result = $self->compile_template_files(@$template_file_paths)
      or return;
    open MODULE, "> $module_path"
        or die "Can't open '$module_path' for output:\n$!";
    print MODULE $result;
    close MODULE;
    return 1;
}

sub compile_module_cached {
    my ($self, $module_path, $template_file_paths) = @_;
    my $m = -M $module_path;
    return 0 unless grep { -M($_) < $m } @$template_file_paths;
    return $self->compile_module($module_path, $template_file_paths);
}

sub compile_template_files {
    my $self = shift;
    my $output = $self->_preamble;
    for my $filepath (@_) {
        my $filename = $filepath;
        $filename =~ s/.*[\/\\]//;
        open FILE, $filepath
          or die "Can't open '$filepath' for input:\n$!";
        my $template_input = do {local $/; <FILE>};
        close FILE;
        $output .=
            $self->compile_template_content($template_input, $filename);
    }
    return $output;
}

sub compile_template_content {
    die "Invalid arguments in call to Lemplate->compile_template_content"
      unless @_ == 3;
    my ($self, $template_content, $template_name) = @_;
    $TemplateName = $template_name;
    my $parser = Lemplate::Parser->new( ref($self) ? %$self : () );
    my $parse_tree = $parser->parse(
        $template_content, {name => $template_name}
    ) or die $parser->error;
    my $output =
        "-- $template_name\n" .
        "template_map['$template_name'] = " .
        $parse_tree->{BLOCK} .
        "\n";
    for my $function_name (sort keys %{$parse_tree->{DEFBLOCKS}}) {
        my $name = "$template_name/$function_name";
        next if $ProcessedTemplates{$name};
        #warn "seen $name";
        $ProcessedTemplates{$name} = 1;
        $output .=
            "template_map['$name'] = " .
            $parse_tree->{DEFBLOCKS}{$function_name} .
            "\n";
    }
    return $output;
}

sub _preamble {
    return <<'...';
--[[
   This Lua code was generated by Lemplate, the Lua
   Template Toolkit. Any changes made to this file will be lost the next
   time the templates are compiled.

   Copyright 2016 - Yichun Zhang (agentzh) - All rights reserved.

   Copyright 2006-2014 - Ingy döt Net - All rights reserved.
]]

local gsub = ngx.re.gsub
local concat = table.concat
local type = type
local math_floor = math.floor
local table_maxn = table.maxn

local _M = {
    version = '0.02'
}

local template_map = {}

local function tt2_true(v)
    return v and v ~= 0 and v ~= "" and v ~= '0'
end

local function tt2_not(v)
    return not v or v == 0 or v == "" or v == '0'
end

local context_meta = {}

function context_meta.plugin(context, name, args)
    if name == "iterator" then
        local list = args[1]
        local count = table_maxn(list)
        return { list = list, count = 1, max = count - 1, index = 0, size = count, first = true, last =
false, prev = "" }
    else
        return error("unknown iterator: " .. name)
    end
end

function context_meta.process(context, file)
    local f = template_map[file]
    if not f then
        return error("file error - " .. file .. ": not found")
    end
    return f(context)
end

function context_meta.include(context, file)
    local f = template_map[file]
    if not f then
        return error("file error - " .. file .. ": not found")
    end
    return f(context)
end

```

```lua
context_meta = { __index = context_meta }

local function stash_get(stash, k)
    local v
    if type(k) == "table" then
        v = stash
        for i = 1, #k, 2 do
            local key = k[i]
            local typ = k[i + 1]
            if type(typ) == "table" then
                local value = v[key]
                if type(value) == "function" then
                    return value()
                end
                if value then
                    return value
                end
                if key == "size" then
                    if type(v) == "table" then
                        return #v
                    else
                        return 1
                    end
                else
                    return error("virtual method " .. key .. " not supported")
                end
            end
            if type(key) == "number" and key == math_floor(key) and key >= 0 then
                key = key + 1
            end
            if type(v) ~= "table" then
                return nil
            end
            v = v[key]
        end
    else
        v = stash[k]
    end
    if type(v) == "function" then
        return v()
    end
    return v
end

local function stash_set(stash, k, v, default)
    if default then
        local old = stash[k]
        if old == nil then
            stash[k] = v
        end
    else
        stash[k] = v
    end
end

function _M.process(file, params)
    local stash = params
    local context = {
        stash = stash,
        filter = function (bits, name, params)
            local s = concat(bits)
            if name == "html" then
                s = gsub(s, "&", '&amp;', "jo")
                s = gsub(s, "<", '&lt;', "jo");
                s = gsub(s, ">", '&gt;', "jo");
                s = gsub(s, '"', '&quot;', "jo"); -- " end quote for emacs
                return s
            end
        end
    }
    context = setmetatable(context, context_meta)
    local f = template_map[file]
    if not f then
        return error("file error - " .. file .. ": not found")
    end
    return f(context)
```

```
end
...
}

1;

__END__

=encoding utf8

=head1 Name

Lemplate - OpenResty/Lua template framework implementing Perl's TT2 templating language

=head1 Status

This is still under early development. Check back often.

=head1 Synopsis

    local templates = require "myapp.templates"
    ngx.print(tempaltes.process("homepage.tt2", { var1 = 32, var2 = "foo" }))

From the command-line:

    lemplate --compile path/to/lemplate/directory/ > myapp/templates.lua

=head1 Description

Lemplate is a templating framework for OpenResty/Lua that is built over
Perl's Template Toolkit (TT2).

Lemplate parses TT2 templates using the TT2 Perl framework, but with a
twist. Instead of compiling the templates into Perl code, it compiles
them into Lua that can run on OpenResty.

Lemplate then provides a Lua runtime module for processing
the template code. Presto, we have full featured Lua
templating language!

Combined with OpenResty, Lemplate provides a really simple
and powerful way to do web stuff.

=head1 HowTo

Lemplate comes with a command line tool call C<lemplate> that you use to
precompile your templates into a Lua module file. For example if you have
a template directory called C<templates> that contains:

    $ ls templates/
    body.tt2
    footer.tt2
    header.tt2

You might run this command:

    $ lemplate --compile template/* > myapp/templates.lua

This will compile all the templates into one Lua module file which can be loaded in your
main OpenResty/Lua application as the module C<myapp.templates>.

Now all you need to do is load the Lua module file in your OpenResty app:

    local templates = require "myapp.templates"

and do the HTML page rendering:

    local results = templates.process("some-page.tt2",
                                      { var1 = val1, var2 = val2, ...})

Now you have Lemplate support for these templates in your OpenResty application.

=head1 Public API

The Lemplate Lua runtime module has the following API method:

```

```
=over

=item process(template-name, data)

The C<template-name> is a string like C<'body.tt2'> that is the name of
the top level template that you wish to process.

The optional C<data> specifies the data object to be used by the
templates. It can be an object, a function or a url. If it is an object,
it is used directly. If it is a function, the function is called and the
returned object is used.

=back

=head1 Current Support

The goal of Lemplate is to support all of the Template Toolkit features
that can possibly be supported.

Lemplate now supports almost all the TT directives, including:

    * Plain text
    * [% [GET] variable %]
    * [% CALL variable %]
    * [% [SET] variable = value %]
    * [% DEFAULT variable = value ... %]
    * [% INCLUDE [arguments] %]
    * [% PROCESS [arguments] %]
    * [% BLOCK name %]
    * [% FILTER filter %] text... [% END %]
    * [% WRAPPER template [variable = value ...] %]
    * [% IF condition %]
    * [% ELSIF condition %]
    * [% ELSE %]
    * [% SWITCH variable %]
    * [% CASE [{value|DEFAULT}] %]
    * [% FOR x = y %]
    * [% WHILE expression %]
    * [% RETURN %]
    * [% THROW type message %]
    * [% STOP %]
    * [% NEXT %]
    * [% LAST %]
    * [% CLEAR %]
    * [%# this is a comment %]
    * [% MACRO name(param1, param2) BLOCK %] ... [% END %]

ALL of the string virtual functions are supported.

ALL of the array virtual functions are supported:

ALL of the hash virtual functions are supported:

MANY of the standard filters are implemented.

The remaining features will be added very soon. See the DESIGN document
in the distro for a list of all features and their progress.

=head1 Community

=head2 English Mailing List

The L<openresty-en|https://groups.google.com/group/openresty-en> mailing list is for English speakers.

=head2 Chinese Mailing List

The L<openresty|https://groups.google.com/group/openresty> mailing list is for Chinese speakers.

=head1 Code Repository

The bleeding edge code is available via Git at
git://github.com/openresty/lemplate.git

=head1 Bugs and Patches

Please submit bug reports, wishlists, or patches by
```

```
=over

=item 1.

creating a ticket on the L<GitHub Issue Tracker|https://github.com/openresty/lua-nginx-module/issues>,

=item 2.

or posting to the L</Community>.

=back

=head1 CREDIT

This project is based on Ingy dot Net's excellent L<Jemplate> project.

=head1 AUTHOR

Yichun Zhang (agentzh), E<lt>agentzh@gmail.comE<gt>, CloudFlare Inc.

=head1 Copyright

Copyright (C) 2016 Yichun Zhang (agentzh).  All Rights Reserved.

Copyright (C) 1996-2014 Andy Wardley.  All Rights Reserved.

Copyright (c) 2006-2014. Ingy döt Net. All rights reserved.

Copyright (C) 1998-2000 Canon Research Centre Europe Ltd

This module is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

=head1 See Also

=over

=item *

Perl TT2 Reference Manual: http://www.template-toolkit.org/docs/manual/index.html

=item *

Jemplate for compiling TT2 templates to client-side JavaScript: http://www.jemplate.net/

=back
```

t/ - lemplate

- TestLemplate.pm
- binop.t
- block.t
- blocks.t
- iterator.t
- pod.t
- sanity.t

t/TestLemplate.pm - lemplate

Data types defined

- run_test
- run_tests

Source code

```
package t::TestLemplate;

use lib 'inc';
use Test::Base -Base;
use File::Temp qw( tempfile );
use File::Copy qw( copy );
use IPC::Run3 qw( run3 );
use Lemplate;

 our @EXPORT = qw( run_tests );

 sub run_tests {
     for my $block (blocks()) {
         run_test($block);
     }
 }

 sub run_test ($) {
     my $block = shift;
     #print $json_xs->pretty->encode(\@new_rows);
     #my $res = #print $json_xs->pretty->encode($res);
     my $name = $block->name;

     my $tt2 = $block->tt2;
     if (!defined $tt2) {
         die "No --- tt2 specified for test $name\n";
     }

     my ($out_fh, $tt2file) = tempfile("tmpXXXXX", SUFFIX => '.tt2', UNLINK => 1);
     print $out_fh $tt2;
     close $out_fh;

     my @cmd = ($^X, "./bin/lemplate", "--compile", $tt2file);

     my ($comp_out, $comp_err);

     run3(\@cmd, undef, \$comp_out, \$comp_err);

     #warn "res:$res\nerr:$comp_err\n";

     if (defined $block->comp_err) {
         if (ref $block->comp_err) {
             like $comp_err, $block->comp_err, "$name - comp_err expected";
         } else {
             is $comp_err, $block->comp_err, "$name - comp_err expected";
         }

     } elsif ($?) {
         if (defined $block->fatal) {
             pass("failed as expected");

         } else {
             fail("failed to compile TT2 source for test $name: $comp_err\n");
             return;
         }

     } else {
         if ($comp_err) {
             if (!defined $block->comp_err) {
                 warn "$comp_err\n";
```

```

            } else {
                is $comp_err, $block->comp_err, "$name - err ok";
            }
        }
    }

    my $expected_lua = $block->lua;
    if (defined $expected_lua) {
        if (ref $expected_lua) {
            like $comp_out, $expected_lua, "$name - lua expected";
        } else {
            is $comp_out, $expected_lua, "$name - lua expected";
        }
    }

    my $luafile;
    ($out_fh, $luafile) = tempfile("tmpXXXXX", SUFFIX => '.lua', UNLINK => 1);
    print $out_fh $comp_out;
    close $out_fh;

    copy($luafile, "a.lua") or die $!;

    (my $luamod = $luafile) =~ s/\.lua$//;

    my $define = $block->define // '';
    my $init = $block->init // '';

    @cmd = ("resty", "-e", qq{$init ngx.print(require("$luamod").process("$tt2file", {$define}))});
    #warn "cmd: @cmd";

    my ($run_out, $run_err);

    run3(\@cmd, undef, \$run_out, \$run_err);

    if (defined $block->lua_err) {
        $run_err =~ s/^\S+\.lua:\d+:\s*//;
        if (ref $block->lua_err) {
            like $run_err, $block->lua_err, "$name - run_err expected";
         } else {
            is $run_err, $block->lua_err, "$name - run_err expected";
         }

     } elsif ($?) {
         if (defined $block->fatal) {
             pass("failed as expected");

         } else {
             fail("failed to run Lua code for test $name: $run_err\n");
             return;
         }

     } else {
         if ($run_err) {
             if (!defined $block->lua_err) {
                 warn "$run_err\n";

             } else {
                 is $run_err, $block->lua_err, "$name - err ok";
             }
         }
     }

     my $expected_out = $block->out;
     if (defined $expected_out) {
         if (defined $run_out) {
             $run_out =~ s/^\n+//gs;
             $run_out =~ s/\n\n+$/\n/gs;
         }
         if (ref $expected_out) {
             like $run_out, $expected_out, "$name - out expected";
         } else {
             is $run_out, $expected_out, "$name - out expected";
         }
     }
}
```

```

1;
```

One Level Up Top Level

t/binop.t - lemplate

```
# vim:set ft= ts=4 sw=4 et fdm=marker:

use t::TestLemplate;

plan tests => 1 * blocks();

$ENV{LEMPLATE_POST_CHOMP} = 1;

run_tests;

 __DATA__

 === TEST 1:
 --- tt2
 maybe
 [% IF yes %]
 yes
 [% END %]

 --- define: yes = 1
 --- out
 maybe
 yes



 === TEST 2:
 --- tt2
 [% IF yes %]
 yes
 [% ELSE %]
 no
 [% END %]

 --- define: yes = 1
 --- out
 yes



 === TEST 3:
 --- tt2
 [% IF yes %]
 yes
 [% ELSE %]
 no
 [% END %]

 --- define: yes = 1
 --- out
 yes



 === TEST 4:
 --- tt2
 [% IF yes and true %]
 yes
 [% ELSE %]
 no
 [% END %]

 --- define: yes = 1, ['true'] = 'this is true'
 --- out
 yes



 === TEST 5:
 --- tt2
 [% IF yes && true %]
```

```
yes
[% ELSE %]
no
[% END %]

--- define: yes = 1, ['true'] = 'this is true'
--- out
yes



=== TEST 6:
--- tt2
[% IF yes && sad || happy %]
yes
[% ELSE %]
no
[% END %]

--- define: yes = 1, sad = '', happy = 'yes'
--- out
yes



=== TEST 7:
--- tt2
[% IF yes AND ten && true and twenty && 30 %]
yes
[% ELSE %]
no
[% END %]

--- define: yes = 1, ten = 10, ['true'] = 'this is true', twenty = 20
--- out
yes



=== TEST 8:
--- tt2
[% IF ! yes %]
no
[% ELSE %]
yes
[% END %]

--- define: yes = 1
--- out
yes



=== TEST 9:
--- tt2
[% UNLESS yes %]
no
[% ELSE %]
yes
[% END %]

--- define: yes = 1
--- out
yes



=== TEST 10:
--- tt2
[% "yes" UNLESS no %]

--- define: yes = 1, no = 0
--- out chomp
yes


```

```

=== TEST 11:
--- tt2
[% IF ! yes %]
no
[% ELSE %]
yes
[% END %]

--- define: yes = 1, no = 0
--- out
yes



=== TEST 12:
--- tt2
[% IF yes || no %]
yes
[% ELSE %]
no
[% END %]

--- define: yes = 1, no = 0
--- out
yes



=== TEST 13:
--- tt2
[% IF yes || no || true || false %]
yes
[% ELSE %]
no
[% END %]

--- define: yes = 1, no = 0, ['true'] = 'this is true', ['false'] = '0'
--- out
yes



=== TEST 14:
--- tt2
[% IF yes or no %]
yes
[% ELSE %]
no
[% END %]

--- define: yes = 1, no = 0
--- out
yes



=== TEST 15:
--- tt2
[% IF not false and not sad %]
yes
[% ELSE %]
no
[% END %]

--- define: ['false'] = '0', sad = ''
--- out
yes



=== TEST 16:
--- tt2
[% IF ten == 10 %]
yes
[% ELSE %]
```

```
no
[% END %]

--- define: ten = 10
--- out
yes



=== TEST 17:
--- tt2
[% IF ten == twenty %]
I canna break the laws of mathematics, Captain.
[% ELSIF ten > twenty %]
Your numerical system is inverted.  Please reboot your Universe.
[% ELSIF twenty < ten %]
Your inverted system is numerical.  Please universe your reboot.
[% ELSE %]
Normality is restored.  Anything you can't cope with is your own problem.
[% END %]

--- define: ten = 10, twenty = 20
--- out
Normality is restored.  Anything you can't cope with is your own problem.



=== TEST 18:
--- tt2
[% IF ten >= twenty or false %]
no
[% ELSIF twenty <= ten  %]
nope
[% END %]
nothing

--- define: ten = 10, twenty = 20, ['false'] = '0'
--- out
nothing



=== TEST 19:
--- tt2
[% IF ten >= twenty or false %]
no
[% ELSIF twenty <= ten  %]
nope
[% END %]
nothing

--- define: ten = 10, twenty = 20, ['false'] = '0'
--- out
nothing



=== TEST 20:
--- tt2
[% IF ten > twenty %]
no
[% ELSIF ten < twenty  %]
yep
[% END %]

--- define: ten = 10, twenty = 20, ['false'] = '0'
--- out
yep



=== TEST 21:
--- tt2
[% IF ten != 10 %]
no
[% ELSIF ten == 10  %]
```

```
yep
[% END %]

--- define: ten = 10
--- out
yep



=== TEST 22:
--- tt2
[% IF alpha AND omega %]
alpha and omega are true
[% ELSE %]
alpha and/or omega are not true
[% END %]
count: [% count %]

--- init
local counter = 0
--- define
alpha = function () counter = counter + 1 return counter end,
omega = function () counter = counter + 10 return 0 end,
count = function () return counter end,
reset = function () return counter == 0 end
--- out chomp
alpha and/or omega are not true
count: 11



=== TEST 23:
--- tt2
[% IF omega AND alpha %]
omega and alpha are true
[% ELSE %]
omega and/or alpha are not true
[% END %]
count: [% count %]

--- init: local counter = 11
--- define
['true'] = 'this is true',
alpha = function () counter = counter + 1 return counter end,
omega = function () counter = counter + 10 return 0 end,
count = function () return counter end,
reset = function () return counter == 0 end

--- out chomp
omega and/or alpha are not true
count: 21



=== TEST 24:
--- tt2
[% IF alpha OR omega %]
alpha and/or omega are true
[% ELSE %]
neither alpha nor omega are true
[% END %]
count: [% count %]

--- init: local counter = 21
--- define
['true'] = 'this is true',
alpha = function () counter = counter + 1 return counter end,
omega = function () counter = counter + 10 return 0 end,
count = function () return counter end,
reset = function () return counter == 0 end

--- out chomp
alpha and/or omega are true
count: 22
```

```
=== TEST 25:
--- tt2
[% IF omega OR alpha %]
alpha and/or omega are true
[% ELSE %]
neither alpha nor omega are true
[% END %]
count: [% count %]

--- init: local counter = 22
--- define
alpha = function () counter = counter + 1 return counter end,
omega = function () counter = counter + 10 return 0 end,
count = function () return counter end,
--- out chomp
alpha and/or omega are true
count: 33



=== TEST 26:
--- tt2
[% small = 5
   mid   = 7
   big   = 10
   both  = small + big
   less  = big - mid
   half  = big / small
   left  = big % mid
   mult  = big * small
%]
both: [% both +%]
less: [% less +%]
half: [% half +%]
left: [% left +%]
mult: [% mult +%]
maxi: [% mult + 2 * 2 +%]
mega: [% mult * 2 + 2 * 3 %]

--- out chomp
both: 15
less: 3
half: 2
left: 3
mult: 50
maxi: 54
mega: 106

```

One Level Up   Top Level

t/block.t - lemplate

```
# vim:set ft= ts=4 sw=4 et fdm=marker:

use t::TestLemplate;

plan tests => 1 * blocks();

$ENV{LEMPLATE_POST_CHOMP} = 1;

run_tests;

 __DATA__

 === TEST 1: line 1
 --- tt2
 [% BLOCK block1 %]
 This is the original block1
 [% END %]
 [% INCLUDE block1 %]
 [% INCLUDE blockdef %]
 [% INCLUDE block1 %]

 --- out
 This is the original block1
 start of blockdef
 end of blockdef
 This is the original block1
 --- LAST



 === TEST 2: line 60
 --- tt2
 [% BLOCK block1 %]
 This is the original block1
 [% END %]
 [% INCLUDE block1 %]
 [% PROCESS blockdef %]
 [% INCLUDE block1 %]

 --- out
 This is the original block1
 start of blockdef
 end of blockdef
 This is block 1, defined in blockdef, a is alpha



 === TEST 3: line 74
 --- tt2
 [% INCLUDE block_a +%]
 [% INCLUDE block_b %]

 --- out
 this is block a
 this is block b



 === TEST 4: line 81
 --- tt2
 [% INCLUDE header
    title = 'A New Beginning'
 +%]
 A long time ago in a galaxy far, far away...
 [% PROCESS footer %]

 --- out
 <html><head><title>A New Beginning</title></head><body>
 A long time ago in a galaxy far, far away...
 </body></html>

```

```


=== TEST 5: line 93
--- tt2
[% BLOCK foo:bar %]
blah
[% END %]
[% PROCESS foo:bar %]

--- out
blah



=== TEST 6: line 101
--- tt2
[% BLOCK 'hello html' -%]
Hello World!
[% END -%]
[% PROCESS 'hello html' %]

--- out
Hello World!

```

One Level Up Top Level

t/blocks.t - lemplate

```
# vim:set ft= ts=4 sw=4 et fdm=marker:

use t::TestLemplate;

plan tests => 1 * blocks();

$ENV{LEMPLATE_POST_CHOMP} = 1;

run_tests;

 __DATA__

 === TEST 1: line 1
 --- tt2
 [% INCLUDE blockdef/block1 %]

 --- lua_err
 file error - blockdef/block1: not found
 --- LAST



 === TEST 2: line 61
 --- tt2
 [% INCLUDE blockdef/block1 %]

 --- out
 This is block 1, defined in blockdef, a is alpha



 === TEST 3: line 68
 --- tt2
 [% INCLUDE blockdef/block1 a='amazing' %]

 --- out
 This is block 1, defined in blockdef, a is amazing



 === TEST 4: line 74
 --- tt2
 [% TRY; INCLUDE blockdef/none; CATCH; error; END %]

 --- out
 file error - blockdef/none: not found



 === TEST 5: line 79
 --- tt2
 [% INCLUDE "$dir/blockdef/block1" a='abstract' %]

 --- out
 This is block 1, defined in blockdef, a is abstract

```

t/iterator.t - lemplate

```
# vim:set ft= ts=4 sw=4 et fdm=marker:

use t::TestLemplate;

plan tests => 1 * blocks();

$ENV{LEMPLATE_POST_CHOMP} = 1;

run_tests;

 __DATA__

 === TEST 1: line 1
 --- tt2
 [% items = [ 'foo' 'bar' 'baz' 'qux' ] %]
 [% FOREACH i = items %]
    * [% i +%]
 [% END %]

 --- out
    * foo
    * bar
    * baz
    * qux



 === TEST 2: line 99
 --- tt2
 [% items = [ 'foo' 'bar' 'baz' 'qux' ] %]
 [% FOREACH i = items %]
    #[% loop.index %]/[% loop.max %] [% i +%]
 [% END %]

 --- out
    #0/3 foo
    #1/3 bar
    #2/3 baz
    #3/3 qux



 === TEST 3: line 110
 --- tt2
 [% items = [ 'foo' 'bar' 'baz' 'qux' ] %]
 [% FOREACH i = items %]
    #[% loop.count %]/[% loop.size %] [% i +%]
 [% END %]

 --- out
    #1/4 foo
    #2/4 bar
    #3/4 baz
    #4/4 qux



 === TEST 4: line 121
 --- SKIP
 --- tt2
 [% items = [ 'foo' 'bar' 'baz' 'qux' ] %]
 [% FOREACH i = items %]
    #[% loop.number %]/[% loop.size %] [% i +%]
 [% END %]

 --- out
    #1/4 foo
    #2/4 bar
    #3/4 baz
    #4/4 qux

```

```
=== TEST 5: line 134
--- tt2
[% USE iterator(data) %]
[% FOREACH i = iterator %]
[% IF iterator.first %]
List of items:
[% END %]
   * [% i +%]
[% IF iterator.last %]
End of list
[% END %]
[% END %]

--- define
data = {'foo', 'bar', 'baz', 'qux', 'wiz', 'woz', 'waz'}
--- out
List of items:
   * foo
   * bar
   * baz
   * qux
   * wiz
   * woz
   * waz
End of list



 === TEST 6: line 157
 --- tt2
 [% FOREACH i = [ 'foo' 'bar' 'baz' 'qux' ] %]
 [% "$loop.prev<-" IF loop.prev -%][[% i -%]][% "->$loop.next" IF loop.next +%]
 [% END %]

 --- out
 [foo]->bar
 foo<-[bar]->baz
 bar<-[baz]->qux
 baz<-[qux]

```

One Level Up Top Level

t/pod.t - lemplate

```
use Test::More;

eval "use Test::Pod";
plan skip_all => "Test::Pod required for testing POD" if $@;
all_pod_files_ok();
```

t/sanity.t - lemplate

```
# vim:set ft= ts=4 sw=4 et fdm=marker:

use t::TestLemplate;

plan tests => 1 * blocks();

run_tests;

__DATA__

 === TEST 1: simple varaible interpolation
 --- tt2
 Hello, [% world %]!
 --- define: world = "Lua"
 --- out
 Hello, Lua!
```