Template Inheritance

Pug supports template inheritance. Template inheritance works via the `block` and `extends` keywords.

In a template, a `block` is simply a “block” of Pug that a child template may replace. This process is recursive.

Pug blocks can provide default content, if appropriate. Providing default content is purely optional, though. The example below defines `block scripts`, `block content`, and `block foot`.

```
//- layout.pug
html
  head
    title My Site - #{title}
    block scripts
      script(src='/jquery.js')
  body
    block content
    block foot
      #footer
        p some footer content
```

To extend this layout, create a new file and use the `extends` directive with a path to the parent template. (If no file extension is given, `.pug` is automatically appended to the file name.) Then, define one or more blocks to override the parent block content.

Below, notice that the `foot` block is *not* redefined, so it will use the parent’s default and output “some footer content”.

```
//- page-a.pug
extends layout.pug

block scripts
  script(src='/jquery.js')
  script(src='/pets.js')

block content
  h1= title
  - var pets = ['cat', 'dog']
  each petName in pets
    include pet.pug
```

```
//- pet.pug
p= petName
```

It's also possible to override a block to provide additional blocks, as shown in the following example. As it shows, `content` now exposes a `sidebar` and `primary` block for overriding. (Alternatively, the child template could override `content` altogether.)

```
//- sub-layout.pug
extends layout.pug

block content
  .sidebar
    block sidebar
      p nothing
  .primary
    block primary
      p nothing
```

```
//- page-b.pug
extends sub-layout.pug

block content
  .sidebar
    block sidebar
      p nothing
  .primary
    block primary
      p nothing
```

Block `append` / `prepend` ¶

Pug allows you to `replace` (default), `prepend`, or `append` blocks.

Suppose you have default scripts in a `head` block that you wish to use on *every* page. You might do this:

```
//- layout.pug
html
  head
    block head
      script(src='/vendor/jquery.js')
      script(src='/vendor/caustic.js')
  body
    block content
```

Now, consider a page of your JavaScript game. You want some game related scripts as well as these defaults. You can simply `append` the block:

```
//- page.pug
extends layout.pug

block append head
  script(src='/vendor/three.js')
  script(src='/game.js')
```

When using `block append` or `block prepend`, the word “`block`” is optional:

```
//- page.pug
extends layout

append head
  script(src='/vendor/three.js')
  script(src='/game.js')
```

Common mistakes ¶

Pug’s template inheritance is a powerful feature that allows you to split complex page template structures into smaller, simpler files. However, if you chain many, many templates together, you can make things a lot more complicated for yourself.

Note that **only named blocks and mixin definitions** can appear at the top (unindented) level of a child template. This is important! Parent templates define a page’s overall structure, and child templates can only `append`, `prepend`, or replace specific blocks of markup and logic. If a child template tried to add content outside of a block, Pug would have no way of knowing where to put it in the final page.

This includes unbuffered code, which can also contain markup. If you need to define variables for use in a child template, you can do so a few different ways:

- Add the variables to the Pug options object, or define them in unbuffered code in a parent template. The child template will inherit these variables.

- Define the variables *in a block* in the child template. Extending templates must have at least one block, or it would be empty — just define your variables there.

For the same reason, Pug's buffered comments cannot appear at the top level of an extending template: they produce HTML comments which would have nowhere to go in the resulting HTML. (Unbuffered Pug comments, however, can still go anywhere.)