The Drupal 8 Theming Guide

by Sander Tirez (@sqndr)

Table of Contents

Drupal 8 主题开发指南（中文版）

Drupal 8 相对于 Drupal 7 发生了巨大的变化。Drupal 8 的主题层基于新的模板引擎 Twig。

Drupal 8 主题开发指南的目标是提供完整的 Drupal 8 带来的前端变化概要，让你能够开始创建自定义 Drupal 8 主题。

本指南 fork 自 https://www.drupal.org/u/sqndr 的 https://github.com/sqndr/d8-theming-guide

中文版

中文资料匮乏，决定翻译此指南为中文

参与贡献

项目地址：https://github.com/ranqiangjun/The-Drupal-8-Theming-Guide

1. Fork 本 Repository
    - 如何与本 Repository 保持同步 Syncing a fork
2. 领取任务，在这个 Issue (https://github.com/ranqiangjun/The-Drupal-8-Theming-Guide/issues/1) 下发表评论，领取对应的章节
3. 翻译 --> 完成翻译
4. 发送 pull request 到本 Repository

> 建议用词

在线阅读

https://ranqiangjun.gitbooks.io/the-drupal-8-theming-guide/content/

下载电子版

访问 https://www.gitbook.com/book/ranqiangjun/the-drupal-8-theming-guide

或直接点击以下链接下载

- pdf
- epub
- mobi

Foreword

A lot of time has gone into writing and fine tuning the content of this book. The book has been written for two kinds of people:

- People who are familiar with Drupal 7 theming and want to learn more about how theming has changed in Drupal 8,
- People who are new to Drupal and want to learn how theming in Drupal 8 works.

The book is rather technical and contains lots of code examples.

Note

I am not responsible for any mistakes or faults that are currently in this book.

About the author

Hi, my name is Sander and I'm a web developer from Belgium. Early 2014, I started an intership at company in Ghent. The company was mainly developing websites with Drupal. During this internship, I started developing website with the software and I got really excited about it. "Come for the code, stay for the community," is what I read on the Drupal.org website. Later that year, at a local sprint, I had the chance to start contributing and join that community myself. It all got very real when my very first patch for Drupal 8 core got committed. I came for the code and I stayed for the community… Later that year, I started traveling around Europe to attend various Drupal events. During those travels, I met so many new interesting and talented people. These amazing people have inspired me during the writing of this book.

About this book

Somewhere after the summer of 2014, before DrupalCon Amsterdam (2014), I decided to start writing a theming guide. At first, it was a very small markdown document. It was a (Github) gist with an overview of front-end changes in Drupal 8. Since I felt that there was so much to talk about, I decided to split up the document into chapters. Before I knew, some sort of book was born. What you're reading right now is the result of hours of work and free time. If you enjoy reading this book, please let me know. The appreciation from several people has been my motivation to continue working on this book.

Get in touch

If you have any questions about the book or it's content, if you want to buy me a cup of coffee or if you've found a mistake, feel free to contact me.

- sqndr | d.o
- @sqndr | twitter

Licence

Drupal 8

Drupal 8 从 2011 年 3 月 10 日开始，经过 4 年的开发之后，发布了第一个稳定版本。在新的基主题 Classy 提交到核心之后，Drupal 8 的 Beta 1 版本于 **2014** 年 **10** 月 **1** 日面世。新的主题层降低了主题开发的复杂性。在 **2015** 年 **10** 月 **7** 日，第一个候选版本发布。最后，在 2015 年 11 月 19 日，发布了正式版。

延伸阅读:

- Huge architectural changes

Drupal 核心

`/core` 是 Drupal 的核心所在。Drupal 根目录中的 `/modules` 以及 `/themes` 用于包含自开发和第三方模块。核心主题包含在 `/core/themes` 目录中。

HTML5

Drupal 8 的标记使用 HTML 5 实现。在核心中使用了 `header` 、 `nav` 以及 `article` 等新标记。

延伸阅读：

- The Drupal HTML5 Group

可用性

加入了 WAI-ARIA Roles 角色。这是一系列的角色、状态以及事物，可用于提供丰富的语义，提高可访问性以及交互能力。WAI-ARIA 在 xhtml 1.0 中是无效的，但在 HTML5 中是有效的，能够应用在 Drupal 8 的标记之中。利用 HTML 元素的角色属性，作者可以为页面组件提供更多的目的信息。

响应式

Drupal 8 是一个开箱可用的移动友好、响应式设计的系统。

延伸阅读：

- Drupal is now out-of-the-box responsive and mobile ready.

浏览器支持

Drupal 8 在管理界面中用 SVG 代替 PNG，用以提供分辨率无关的图标。对于 **IE 8** 以及更早版本，以及 **Android** 浏览器 **2.3** 以及更早版本，或类似的不支持 SVG 的浏览器，不提供相应的兼容支持。

因为不再支持这些过时的浏览器，Drupal 核心中的 CSS 得以更进一步。不再需要添加 `odd` 、 `even` 、 `first` 以及 `last` 这样的类；CSS3 伪类选择器可以完成这些任务。

延伸阅读：

- Drupal 8 does not support browsers that do not support SVG（ Drupal 8 不支持不兼容 SVG 的浏览器）
- Most first/last/odd/even classes removed in favor of CSS3 pseudo selectors（ 按照 CSS3 的伪类选择器风格，绝大多数 first/last/odd/even 类被移除）

延伸阅读

> 在熟悉 Durpal 7 的读者眼中会有更多不同，对主题作者来说，所有重要的变更记录都可以在这里找到。

编码规范

下面是 **CSS**、**javascript** 以及新加入的 **Twig** 模板引擎的编码规范

- CSS 编码规范.
- Javascript 编码规范.
- Twig 编码规范.

CSS

CSS 编码规范来源于 **SMACSS** 以及 **BEM** 这两种知名的方法论。

SMACSS

SMACSS（读作 "smacks"）标准是可扩展的、模块化的 **CSS** 架构。Drupal 8 使用 **SMACSS** 对 CSS 进行概念上的组织。

Base

> **Base** 通常包含 css reset 或者 normalize，以及 HTML 元素样式。Drupal 8 在 *system* 模块中包含了 **normalize.css**。

布局（Layout）

> **Layout** 规则定义了通用模块在各个页面上的典型呈现方式。**Grid** 系统中的 CSS 也在这一类别中。

在 Drupal 核心主题中，所有的布局样式类（全局页面布局）都以 `.layout-` 为类名的前缀。举例如下：

```
.layout-content
.layout-container
.layout-sidebar-left
.layout-sidebar-right
```

注：这些 `.layout-` 类绝不应该包含 colors、borders 等内容，而只推荐包括 width、loats、margins 以及 paddings 这类内容，仅满足全局的页面 Grid 布局需求即可。

组件（Components）

SMACSS 官方称之为 **module**，然而 *modules* 在 Drupal 中使用已久，所以为防止冲突，这里将其重命名为 组件（**Components**）。

> 组件 是独立的可复用的 UI 元素。组件规则应该有弹性、可复用。例如对话框、轮播、微件等。

状态（State）

> 状态以参数化的形式覆盖所有其他的样式。一般用来描述某种动作或者触发器。最好的例子是 `is-active` 状态，用来标识是否激活。这些类一般用 `is-` 作为前缀。

在 Drupal 核心中，状态通常和组件一起定义。

主题（Theme）

为了防止和 Drupal 的 **Themes**（主题）相混淆，也会使用 **skin**（皮肤）的叫法。

> 主题 定义了颜色和图像，为前面的任何元素提供视觉支持

例如下面的代码：

```
// component.css

.component {
    border: 1px solid;
}

// theme.css

.component {
    border-color: blue;
}
```

在 SMACSS 可以进一步了解相关知识。

Seven 主题 中的 SMACCS

主题 “Seven” 用了这些分类来拆分 CSS 规则。Seven 主题包含了一个叫 `css` 的目录，在这个目录中有四个子目录：

- `core/themes/seven/css/base`
- `core/themes/seven/css/components`
- `core/themes/seven/css/layout`
- `core/themes/seven/css/theme`

BEM

BEM (Block Element Modifier 块元素修饰符) 是一种命名原则。基于 **BEM** 方法论 的 Drupal 8 CSS 类命名更加清晰易读。

例如：

```
.block__element--modifier
```

- **Block**（块） 是一个独立实体，表现了页面中的一个片段的界面。
- **Element**（元素） 是 Block 的一部分，表达 Block 的功能和语义。其有效性仅限于所属 Block 的范围。Block 不一定包含 Element。
- **Modifier**（修饰符） 是 Block 或 Elements 的标志（属性），他对属性以及状态进行了定义。他可以是布尔值（例如：visible: true 或者 false），也可以是键值对（size: large, medium, small）- 有点像 HTML 的属性，不过不完全相同。如果一个项目有多个属性，可以包含多个修饰符。

来源： Smashing Magazine

`buttons.theme.css (Seven)`：

- `.button {}`，块。
- `.button--primary {}`，修饰符。

`node.css (Seven)`：

- `.node__submitted {}`，元素。

延伸阅读

- SMACSS 主页
- Split Seven's style.css into SMACSS categories（把 Seven 的 style.css 拆分为 SMACSS 分类），**Seven** 主题按照新的编码规范重构的过程。
- Organize Your Styles - An Introduction to SMACSS（组织你的样式 -- SMACSS 简介），Acquia 的博客。
- BEM and SMACSS: Advice From Developers Who've Been There（BEM 和 SMACSS：先行者的忠告），New Relic 的博客。
- BEM methodology for small projects（面向小项目的 BEM 方法）

Drupal core themes

Drupal 8 的四个核心主题位于 `core/themes` 文件夹内。这些主题是 Drupal 8 移动提案的一部分，因此他们都是响应式的。所有的核心主题（不是基主题）都能充当管理主题进行使用。因为 Drupal 8 是多语言的，所以核心主题也都支持双向文字。

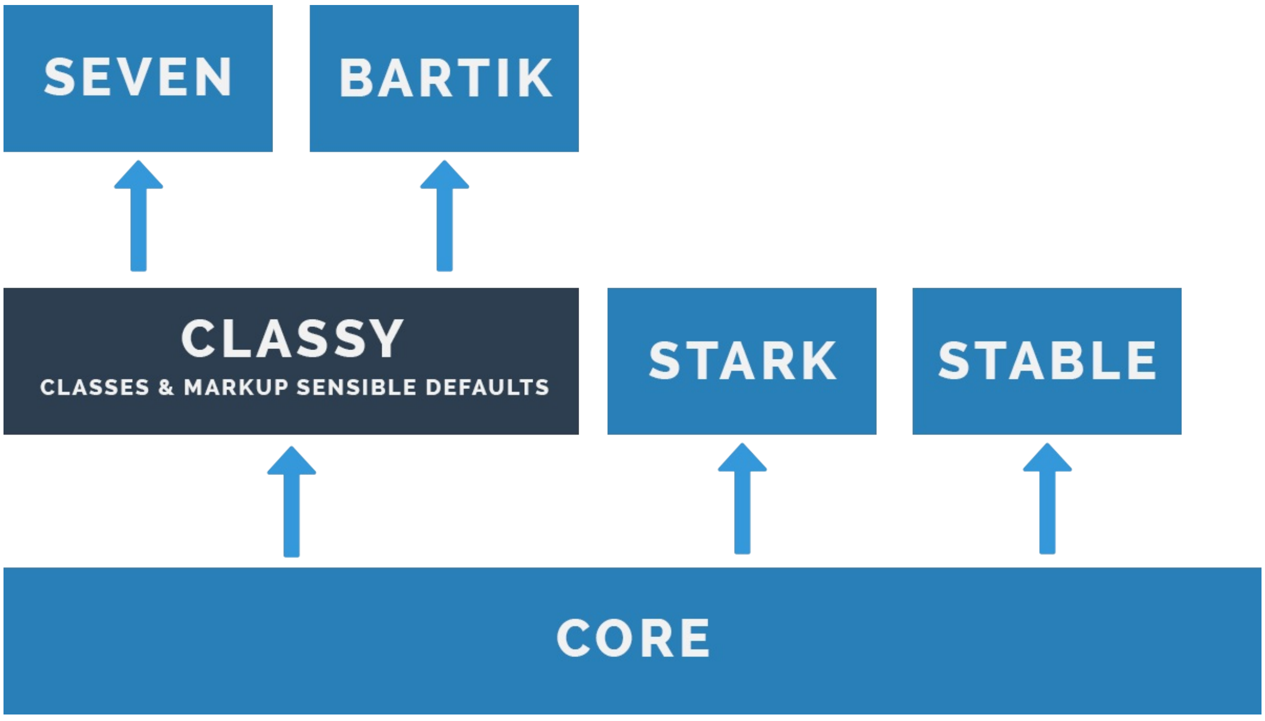
> 注： Stark 主题中有一个用于响应式布局的样式表。后来这些样式被清除掉了。 在 **Stark** 一节中可以了解这部分内容

- **bartik**: 移动优先的响应式主题，有弹性，可定制颜色，带有很多的区块。
- **seven**: *Drupal 8* 的缺省管理主题，具有干净的线条，简单的块，使用 *sans-serif* 字库，便于管理工作的进行。
- **stark**: 一个几乎没有样式的主题，用来演示 *Drupal* 缺省的 *HTML* 和 *CSS*

上述三个 Drupal 7 主题你可能还是有印象的，不过还有另外两个

- **classy**: [meta] Results of Drupalcon Austin's Consensus Banana: 在 2014 年的奥斯汀 DrupalCon 中，认为需要在核心中加入新的主题。Classy 是 Seven 以及 Bartik 的基主题。
- **stable**: Stable 主题作为 Drupal 8 核心标记、CSS 以及 JS 的向后兼容的布局而存在。如果一个主题的 `.info.yml` 文件中没有注明基主题，则会使用该主题作为基主题。

下面是一个概括，说明 Drupal 8 主题之间的关系。



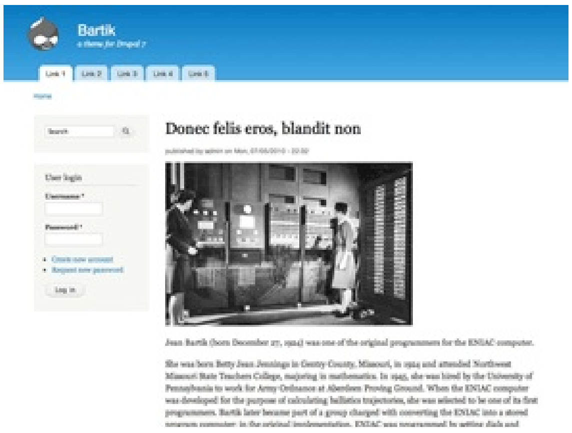
Bartik

Bartik 是一个随 Drupal 7 诞生的简洁主题。这个主题在 Drupal 8 升级为响应式，并提供了更多新特性。Bartik 是 Drupal 8 下的缺省的面向用户的主题。

`base theme: classy`

Bartik 的进步

在 Drupal 8 的开发过程中，Bartik 主题几乎没有任何进展。所以在 Drupal 8 的 Beta 期间，由 Emma Karayiannis 担任新的 Bartik 管理人，从这之后，Bartik 才开始跟上进度。



Donec felis eros, blandit non

published by admin on Mon, 07/05/2010 - 22:32

Jean Bartik (born December 27, 1924) was one of the original programmers for the ENIAC computer.

She was born Betty Jean Jennings in Gentry County, Missouri, in 1924 and attended Northwest Missouri State Teachers College, majoring in mathematics. In 1945, she was hired by the University of Pennsylvania to work for Army Ordnance at Aberdeen Proving Ground. When the ENIAC computer was developed for the purpose of calculating ballistics trajectories, she was selected to be one of its first programmers. Bartik later became part of a group charged with converting the ENIAC into a stored program computer; in the original implementation, ENIAC was programmed by setting dials and

---

延伸阅读

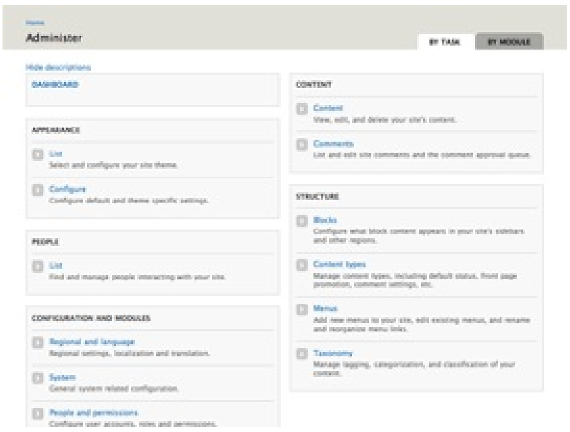
- 官方网站的 Bartik 文档
- [META] Overhaul Bartik's code

Seven

Seven 在 Drupal 7 中诞生，用做缺省的管理主题，他的设计目的是增强 Drupal 7 的用户体验。Seven 在 Drupal 8 中仍然是缺省的管理主题。使用 Seven 样式指南来进一步改善体验和外观。Seven 主题当前由 Lewis Nyman 管理。

```
base theme: classy
```

> 有些用户可能会奇怪，为什么这个主题不改名叫 `eight` （D8 中重新命名）？ 回答是 不


Home
Administer
BY TASK
BY MODULE
Hide descriptions

DASHBOARD

APPEARANCE
List
Select and configure your site theme.
Configure
Configure default and theme specific settings.

PEOPLE
List
Find and manage people interacting with your site.

CONFIGURATION AND MODULES
Regional and language
Regional settings, localization and translation.
System
General system related configuration.
People and permissions
Configure user accounts, roles and permissions.

CONTENT
Content
View, edit, and delete your site's content.
Comments
List and edit site comments and the comment approval queue.

STRUCTURE
Blocks
Configure what block content appears in your site's sidebars and other regions.
Content types
Manage content types, including default status, front page promotion, comment settings, etc.
Menus
Add new menus to your site, edit existing menus, and rename and reorganize menu links.
Taxonomy
Manage tagging, categorization, and classification of your content.

采访 Lewis Nyman

Seven 主题的管理者的工作内容是什么？

有完善的文档来说明核心组件管理者的工作内容。我的首要职责是概览 Seven Queue 中的所有 Issue，很少有人会做这个事情，但是只有这样，在有人报告 BUG 的时候或者发起讨论的时候，才能判断这些 Issue 是否已经存在。

我还要负责确认 Issue 已经被正确的组织，并且被有序的处理。另外，我丰富的经验确保我有能力评审 Seven 所有的补丁，检查他们的质量，确保他们的一致性。

Drupal 8 中的 Seven 主题有什么新特性？

Seven 的设计追随了 Drupal 8 的脚步。Ry5n 、Yoroy 和 Bojhan 编制了一份样式指南，增强了这一主题的外观表现，同时也为很多核心模块的用户体验的提升提供了良好的基础。

有些例子还会持续使用不同的按钮类型、下拉按钮、新的内容编辑布局，以及行内的 Form 错误提示。

Seven 主题以后会变成怎样？对未来的 Drupal 8.1 或者 Drupal 9 已经有计划了么？

Seven 正在缓慢的向组件化设计靠拢。根据样式指南和我们的 CSS 新标准，Seven 中的很多 CSS 正在使用更加抽象、可复用的 CSS 组件。我想，Seven 会在 8.x 的组件基础上发展，以适应核心和第三方不断增长的需求。

在 Drupal 9 中，正在讨论的 组件化设计不仅仅涵盖了 CSS，而是整个 Drupal 主题系统。

如果 Drupal 9 的计划能够实现，Seven 会是这个系统的第一个测试用例。

现在的 CSS 如此强大，我们还讨论了关于向 Seven 主题增加更多的视觉效果的问题。有人开玩笑说为界面增加音效，天知道这是不是玩笑。

我们刚在讨论的内容中有一部分是关于 Drupal 的样式指南，这方面的进展如何呢？

我们遇到了一些跟 原有 Photoshop 样式指南 有关的维护问题，要在设计实现过程中跟进是个比较痛苦的事情。

我们现在已经有了一个计划，使用 KSS 来实现 Seven 的 CSS 组件，这样我们可以为核心代码生成和 api.drupal.org 以及 Drupal 核心 Javascript 文档类似格式的样式指南。因为 KSS 的语法和现有 Drupal 的注释标准颇有不同，造成了一些障碍。在 Drupal 8 发布 RC 版本之后，HTML 和 CSS 代码会被冻结，在这之前还有很多事情要做，所以这部分内容只能推迟完成。

有些人觉得这名字很古怪。你觉得呢？如果让你来命名，你会起个什么名字？

近年来，我想我已经不太在意这个名字了。我认为名字的意义是我们赋予的。大家都知道，Seven 是随 Drupal 发行的管理主题。改了名可能会更加麻烦。

如果我们现在创建了这个主题，并给他起个名字，我可能会像 Bartik 一样，用名人来命名。也许是忍者神龟中的一个。

为 Drupal 创建管理主题的最大挑战是什么？

现在最大的问题是，大量的模块都需要不同的用户界面。有的模块会为了自己的需要来调整现有界面，有些甚至会从头做起。一个管理主题想要无损的支持所有的需求是个难题。

我的梦想是，在模块中完全没有管理界面的 CSS。切近一些的目标是尽量减少这种 CSS。我希望持续发展 Seven 主题，使之成为一个真正有用的界面框架，持续的影响模块开发者，利用已有的组件来编写一致的用户体验。

延伸阅读

- 官方网站的 Seven 文档
- Lewis Nyman

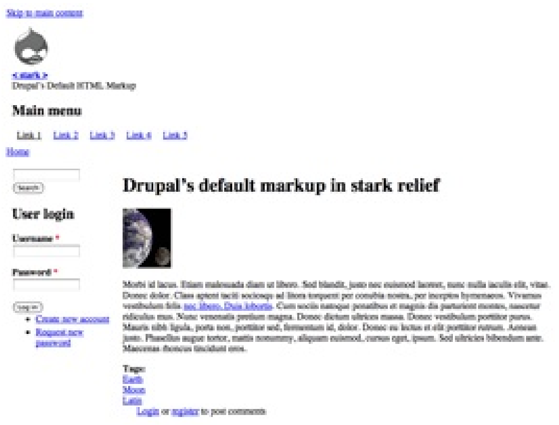
Stark

Stark 这一核心主题演示了 Drupal 的缺省标记。他不包含任何的模板或 CSS 文件。也就是说他使用的是 Drupal （核心）模块的模板和 CSS 文件。

> 为了避免混淆 Drupal 和 第三方模块加入到页面里的 CSS，Stark 主题除了 “页头、侧栏、内容以及页脚” 布局之外，没有加入其他样式。这是受 Drupal 8 移动提案的影响，Start 在 Drupal 8 中使用了响应式布局，这一响应式布局的 CSS 规则可以在 `css/layout.css` 中找到。
>
> 在 Drupal 7 中，Stark 包含了布局和页面区块所使用的 CSS。起初 Stark 布局也是响应式的，但是后来 CSS 被移除了（从 Stark 中删除所有视觉(元素)）。因此，Stark 现在只输出 HTML 以及模块中的 CSS 了。

Stark 对 开发人员 来说很有用，用来判断模块相关的 CSS 和 JavaScript 是否影响到了复杂的主题。对于 设计人员 来说，他可以在学习 Drupal 标记的时候避免额外的干扰。

> 总而言之，Stark 输出的是 Drupal 最基础的 CSS 和 HTML。


Skip to main content

< stark >
Drupal's Default HTML Markup

Main menu

Link 1 Link 2 Link 3 Link 4 Link 5
Home

Search

User login

Username *

Password *

Log in
- Create new account
- Request new password

Drupal's default markup in stark relief

Morbi id lacus. Etiam malesuada diam ut libero. Sed blandit, justo nec euismod laoreet, nunc nulla iaculis elit, vitae. Donec dolor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus vestibulum felis nec libero. Duis lobortis. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nunc venenatis pretium magna. Donec dictum ultrices massa. Donec vestibulum porttitor purus. Mauris nibh ligula, porta non, porttitor sed, fermentum id, dolor. Donec eu lectus et elit porttitor rutrum. Aenean justo. Phasellus augue tortor, mattis nonummy, aliquam euismod, cursus eget, ipsum. Sed ultricies bibendum ante. Maecenas rhoncus tincidunt eros.

Tags:
Earth
Moon
Latin
Login or register to post comments

延伸阅读

Classy

```
<div class="do-not-add-classes-in-drupal-core"></div>
```

2014 年奥斯汀 DrupalCon 上提出对新的核心基主题的需求。这一需求后来成为 **Consensus Banana** 的一部分。

共识之蕉（Consensus Banana）

> 香蕉 从哪里来？
>
> 奥斯汀 DrupalCon 之前两年的一个 BadCamp 上，John Albin（JohnAlbin）在海盗节前夕手持一把塑料剑，这把剑称为共识之剑。Moten (mortendk) 在会上经常手持一根香蕉指向他人提问 “所以我们就 x 问题达成共识了？”，这就是共识之蕉的由来。说的就是一个可用于指点的工具。

Scott Reeves

Meta issue

下面是一个对 Meta issue 的概述。

> 创建一个名为 "classy" 的核心基主题，包含目前所有核心模板的文件，。然后修改所有的核心/模块模板文件，把其中的类做最小化处理（仅包含有用部分）。然后确保 Seven 和 Bartik 能够正常工作。

技术更新

技术的角度来说：移植核心类到 Classy 基主题。这一工作分成两个步骤：第一阶段，把 preprocess 函数中把类移动到核心模板文件中；第二阶段，把包含所有类的核心模板文件转到 Classy 中，这样就保证了核心模板中不再有这些内容。接下来就可以再认为所有核心模板文件都不再使用 class 了（核心中不再使用 `class="whatever"` 了）。

主题开发者不再需要一个类似 mothership 这样的基主题来清理多余的 div 了。调查表明，不是所有的主题开发者都需要这些一致的标记，他们不需要那么多的 `<div>` 。感谢 Classy，不再需要在反抗核心上继续浪费精力了。一些主题开发者在开发自己的主题的时候，需要利用 Classy 获得有用的缺省类；而另一些开发者需要能够从头开始，不需要进行重载就获得完全的控制，拜托 Drupal 7 时代的阴影。

Classy，新的基主题

在阿姆斯特丹 DrupalCon 上，**classy.info.yml** 由 Dries 提交到了 Drupal 8 核心中。变更记录

这意味着 Classy 是新的核心 `基主题` ，Bartik 和 Seven 都以该主题为基础派生而来。

classy.info.yml 加入核心，并成为 Bartik 和 Seven 的基主题

向后兼容性

Classy 遵从 Drupal 8 的规则，提供可靠的向后兼容性。

延伸阅读

- Modules Unraveled, episode 119: The Classy Base Theme for Drupal 8 with Scott Reeves and David Hernandez.
- The official change record: Added a new base theme to core called Classy

stable

缺省的基主题。

Stable was introduced in Drupal 8 as a backwards compatibility layer for Drupal 8's core markup, CSS and JS. When no `base theme` is provided by a theme, stable will automatically be used.

Stable 在 Drupal 8 中是用来作为核心标记、CSS 以及 JS 的向下兼容层而存在的。当主题没有指定基主题时，就会自动使用 Stable 主题。

> Drupal 8 核心通过这一主题提供了一个最小化的标记集。对前端开发者来说，可以在这个基础上加入更加丰富的内容，而无需去移除 Drupal 8 自带的缺省类。对 Bootstrap 或者 Foundation 这样的前端框架尤其有益。

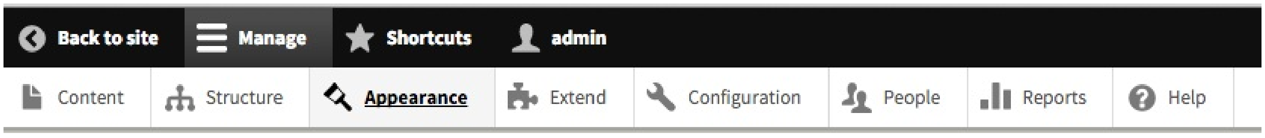
把基主题设为 `false` ，就能够禁用这种向后的兼容

```
base theme: false
```

- 官方的变更记录：缺省加入 Stable 基主题，保证向后兼容性

Theme administration

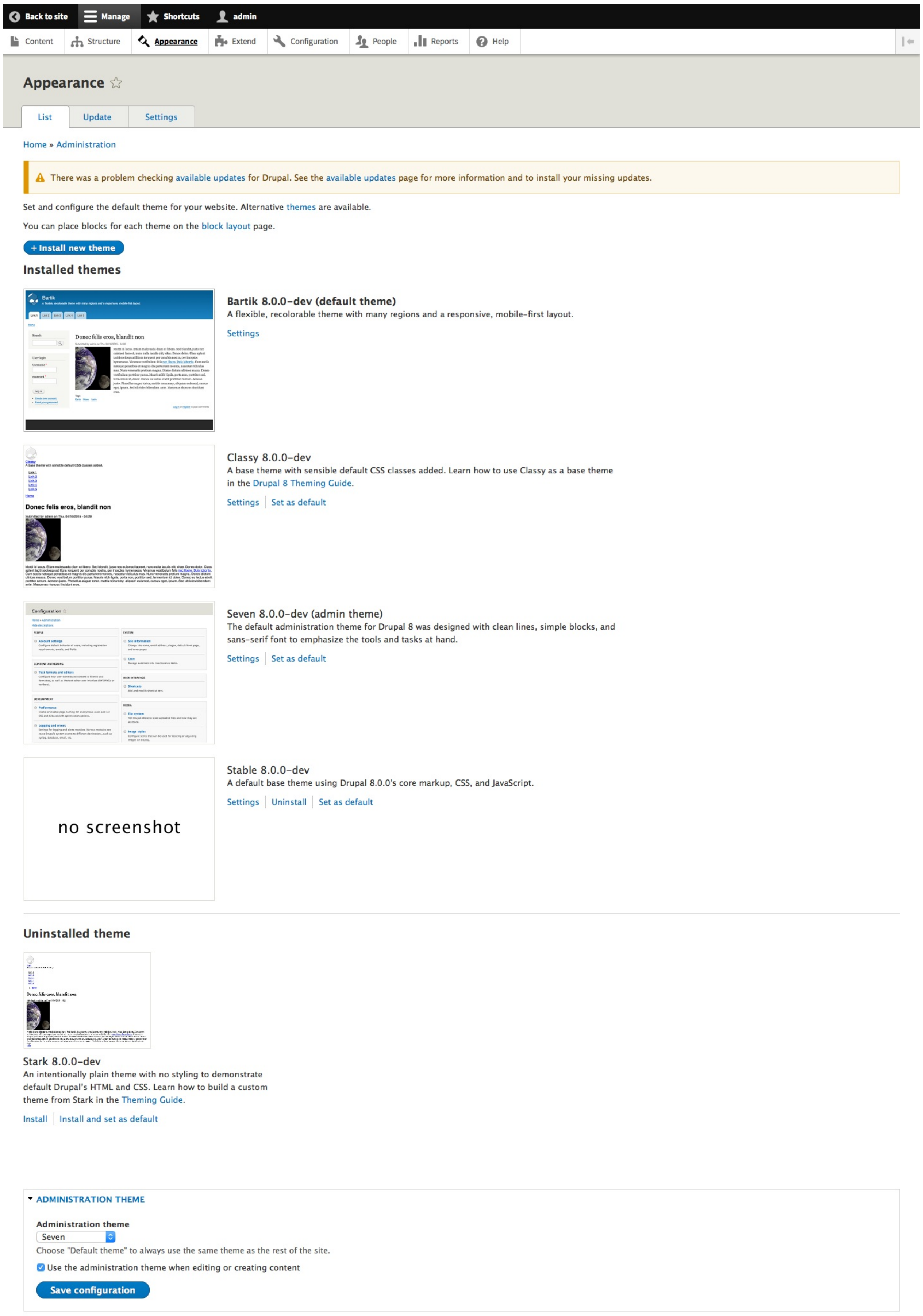
All - or most of the - theme related settings, are located in the admin's **Appearance** section.



From this page, it's possible to enable and disable themes. Site builders can even install themes using a URL or by uploading a theme archive (*tar tgz gz bz2 zip*). From this page, a theme can be set as default user-facing theme using the **Set as default** button which appears right from the screenshot (this button won't appear for the theme that is currently set as the default theme).

A theme that is installed won't - by default - appear in the front-end, unless it's set as the **default theme**.

At the bottom of the page, the administration theme can be switched.



Appearance ☆

Home » Administration

⚠ There was a problem checking available updates for Drupal. See the available updates page for more information and to install your missing updates.

Set and configure the default theme for your website. Alternative themes are available.

You can place blocks for each theme on the block layout page.

+ Install new theme

Installed themes

Bartik 8.0.0-dev (default theme)
A flexible, recolorable theme with many regions and a responsive, mobile-first layout.
Settings

Classy 8.0.0-dev
A base theme with sensible default CSS classes added. Learn how to use Classy as a base theme in the Drupal 8 Theming Guide.
Settings | Set as default

Seven 8.0.0-dev (admin theme)
The default administration theme for Drupal 8 was designed with clean lines, simple blocks, and sans-serif font to emphasize the tools and tasks at hand.
Settings | Set as default

no screenshot

Stable 8.0.0-dev
A default base theme using Drupal 8.0.0's core markup, CSS, and JavaScript.
Settings | Uninstall | Set as default

Uninstalled theme

Stark 8.0.0-dev
An intentionally plain theme with no styling to demonstrate default Drupal's HTML and CSS. Learn how to build a custom theme from Stark in the Theming Guide.
Install | Install and set as default

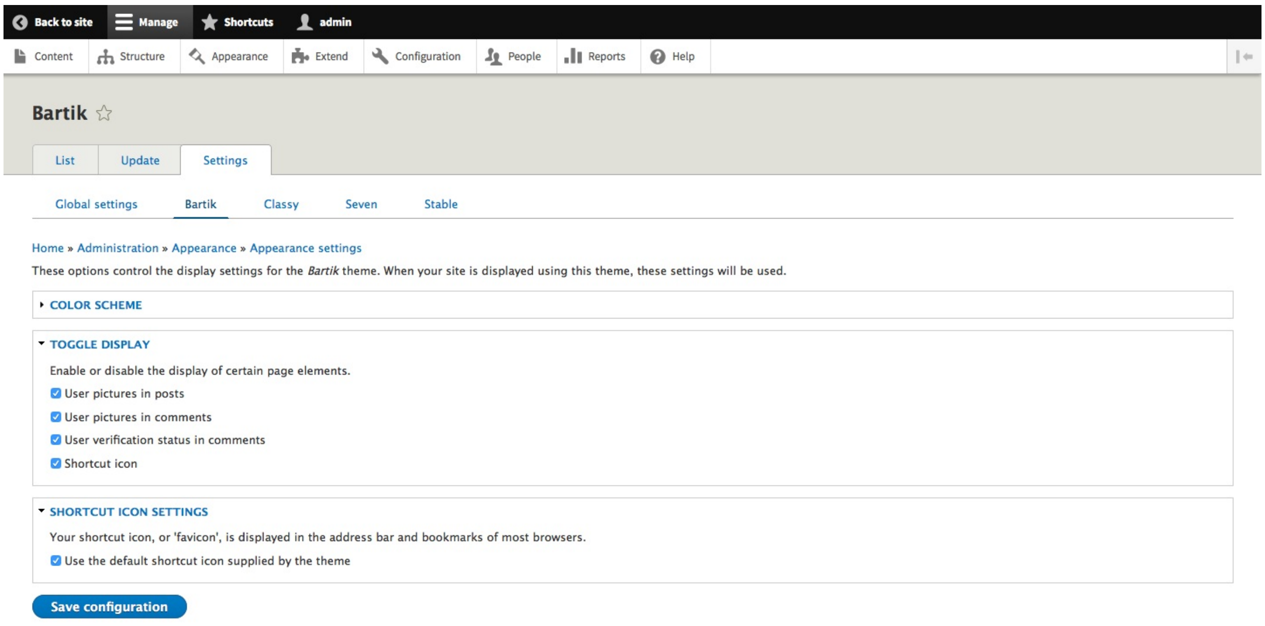
▾ ADMINISTRATION THEME

Administration theme
Seven
Choose "Default theme" to always use the same theme as the rest of the site.
✓ Use the administration theme when editing or creating content

Save configuration

Theme settings

Below is an example of Bartik's theme settings page.



Home » Administration » Appearance » Appearance settings

These options control the display settings for the *Bartik* theme. When your site is displayed using this theme, these settings will be used.

▸ COLOR SCHEME

▾ TOGGLE DISPLAY

Enable or disable the display of certain page elements.

- [✓] User pictures in posts
- [✓] User pictures in comments
- [✓] User verification status in comments
- [✓] Shortcut icon

▾ SHORTCUT ICON SETTINGS

Your shortcut icon, or 'favicon', is displayed in the address bar and bookmarks of most browsers.

- [✓] Use the default shortcut icon supplied by the theme

Installing a new theme

Drupal scans the `themes` directory and looks `*.info.yml` to find new themes. Make sure to install themes in the correct directories.

```
/themes
```

> For themes that should be available to all sites.

```
/sites/{site}/themes
```

> For themes that should be available on a specific site (on a multisite installation).

@todo: Change location of this part:

- The default settings and config are changed to be fast and safe production values. In a default Drupal 8 installation, CSS and JS aggregation is turned on.

The new theme layer

Drupal 8 comes with a completely new theme layer. The old theme layer was a big frustration for front-end developers and was far too complex. The most significant change of the new layer is **Twig**. Before diving into this new theming layer, let's take a closer look at the *old* one and sum up some disadvantages.

Maintainers

The Drupal 8 theme system is maintained by the following people:

- Alex Bronstein 'effulgentsia'
- Scott Reeves 'Cottser'
- Fabian Franz 'Fabianx'
- Joël Pittet 'joelpittet'
- Lauri Eskola 'lauriii'

Read more

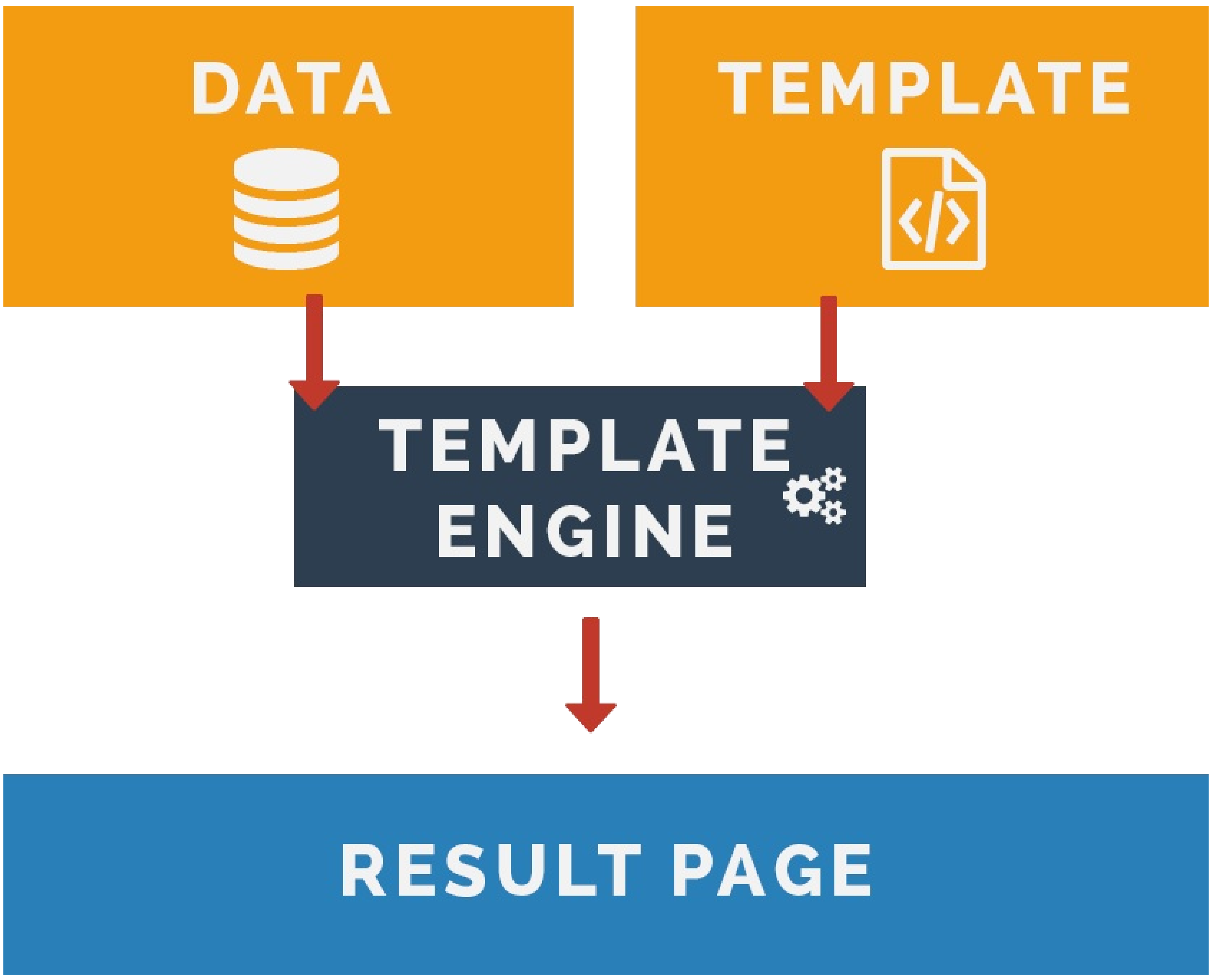
- Blog post about the future of Drupal, a blog post by Larry Garfield.
- Drupal 8's new theming layer, a video interview with Joël Pittet and Scott Reeves.
- The "themer" role is a Drupalism, a blog post by …
- Rethinking Drupal’s Theme/Render Layer, a blog post by by John Albin.
- (meta) Convert all core *.tpl.php templates to Twig as singular patch

Theme engines

Inside `core/themes` there is a sixth folder (besides `bartik` , `seven` , `stark` , `classy` and `stable`), called `engines` . This folder contains the theme engines. The default template engine In Drupal 8 is **Twig**. The default template engine from Drupal 7 however, PHPTemplate, does still live in there. It's not recommended to continue using the PHPTemplate engine. Bear in mind that Twig has a lot more to offer and is much more secure than PHPTemplate.

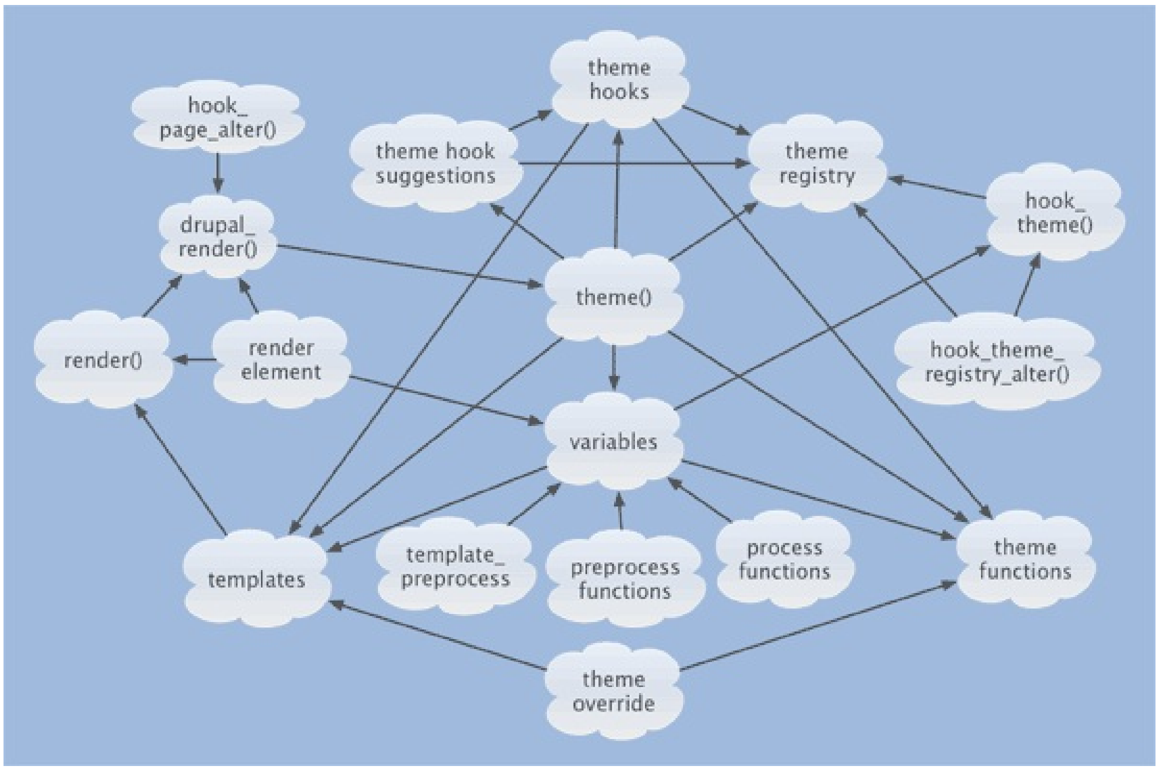
What is a theme engine?

> A theme engine (also called template engine, template processor or template parser) is a software component that **combines data with templates** from themes and shows the result - the final HTML - to the user.



The old (Drupal 7) theme layer

The Drupal 7 theming layer had lots of disadvantages:



> *A simplified wiring guide to the Theme Layer, by John Albin*

- Drupal-specific template conventions
- With **PHPTemplate**, anything was possible; even dropping your entire database from a template file. This made PHPTemplate - and the whole theming layer - very insecure.

  `<?php db_query("DROP TABLE {node}"); ?>`

- There are too many different ways to display content in the template. Below are three examples. For people unfamiliar with PHP, it's hard to find out how to display content in a template file. From `node.tpl.php` :
  - `<?php print $node->nid; ?> // object`
  - `<?php print $attributes; ?> // string`
  - `<?php print render($content); ?> // array`

> In the first example, a property from an object is printed. In the second example, a string is printed. In the last example, the $content variable is a render array. The render() function converts this (render) array to HTML markup. Since it returns HTML, it should be used along with print in templates.

- There were too many template files.

- Besides template files, Drupal 7 also used `theme()` function. There were even more `theme()` functions than template files. These PHP functions were used to create a string containing html markup based on (more) some complex logic. Syntax highlighting is almost impossible with these functions.
- The theme layer was a complex system, almost *impossible* to understand. This wasn't just for people new to Drupal. Even people working on core daily got confused and frustrated about the system. This makes Drupal 7 hard to learn and is one of the reasons the **learning curve** for new Drupal developers was so high.

PHPTemplate

PHPTemplate was the default template engine for Drupal 7. It was written by Adrian Rossouw.

The engine uses individual template files (`*.tpl.php` extension, such as `example.tpl.php`), to theme Drupal's `theme_` functions, such as `theme_pager()`

An extract from **node.tpl.php** in Drupal 7:

```
<div id="node-<?php print $node->nid; ?>" class="<?php print $classes; ?> clearfix"<?php

  <?php print $user_picture; ?>

  <?php print render($title_prefix); ?>
  <?php if (!$page): ?>
    <h2<?php print $title_attributes; ?>><a href="<?php print $node_url; ?>"><?php print
  <?php endif; ?>
  <?php print render($title_suffix); ?>
  …
```

Removing PHP Template

After a discussion about the advantages, disadvantages and features of various templating languages, the decision was made to include the Twig templating engine from the Symfony framework. Drupal could have created its own, new, PHP based “token" system, but that way Drupal would still be on its own. By introducing Twig, Drupal is moving away from the Drupal island it's been on for many years. Twig is, just like Symfony, maintained by Sensio Labs. The fact that Drupal adopts Symfony components doesn't necessarily have anything to do with the adoption of **Twig** as the new templating language. Twig was chosen because it was the best choice after comparing various templating languages.

> "… We don't have Twig because we have Symfony. It's more that, we have Twig because it's **AWESOME**"

- Scott Reeves, @Cottser

Twig makes the Drupal theme layer faster and more secure. It's impossible to run PHP scripts, make database calls or access the file system. Autoescaping is also enabled by default (more detail in the Twig chapter), a major improvement concerning XSS (Cross-site scripting).

> In order to use the raw data (not escaped), you have to use the Twig `|raw` filter. In the Twig chapter, more information can be found.

All of the PHPTemplate files (`*.tpl.php`) were converted to Twig template files (`*.html.twig`).

Twig

Twig is a modern PHP-based compiled templating language. It is part of the Symfony 2 framework and is the default template engine in Drupal 8 (Read the official change record).

All of the `theme_*` functions and PHPTemplate based `*.tpl.php` files have been completely removed and replaced with Twig template files. These template files have a new Twig extension: `*.html.twig` .

When your web page renders, the Twig template engine takes the template file and converts it into a 'compiled' PHP template which it stores in a protected directory in `sites/default/files/php_storage/*` The compilation is done once. Template files are cached for reuse and are recompiled on clearing the Twig cache.

Coding standards

Twig is a new core programming language and all of the core template files must follow the coding standards. There are both official coding standards from the Twig template engine and coding standards for the usage of Twig in Drupal.

- Generic Twig coding standards: http://twig.sensiolabs.org/doc/coding_standards.html
- Twig in Drupal coding standards: http://drupal.org/node/1823416

Advantages

Compared to PHPTemplate, Twig offers some major advantages. Especially when it comes to security.

- Twig is more secure, due to the fact that only a number of tags can be used. In the previous PHPTemplate, it was possible for a template file to execute the following code:

  ```
  <?php
    db_query('DROP TABLE {users}');
  ?>
  ```

  This should of course not be the case. In case you're wondering what it does: it removes the entire `users` table from your database. Not good, right?

- There is now a clear separation between the *logic* and the *view*. This means: no more PHP code inside your template files.

- The syntax is very easy to understand, making the code more readable as well. Also,

many IDE's have syntax highlighting for `*.twig` files.
- Template files are reusable, thanks to Twig includes.
- Debugging is much more easy. First of all, Twig outputs a helpful message with the filename and the line number whenever there is a syntax problem within a template. Secondly, you can turn on a Twig debug function. More on that later.
- Twig is very well documented. Go ahead and start reading the official documentation here.
- It's not only used in Drupal core, so it's not a Drupaly-thing.

Disadvantage

- Although the syntax is very easy to read and understand, it's a new syntax you have to learn and get used to before getting started. Once you know the syntax however, you get the advantage of knowing a template engine that can be used outside of Drupal as well. This is due to the fact that Twig is part of the Symfony framework.

Read more

- The official Twig documentation from Sensio Labs.

From Drupal 7 to Drupal 8

Converting `theme()` functions to Twig templates

- Convert core theme functions to Twig templates

All of the `theme()` functions are deprecated in Drupal 8. They are all converted into twig template files.

> **ALL GONE:** All of the `theme()` functions have been converted to Twig templates.

webchick Credit commented 11 days ago #59

Status: ~~Reviewed & tested by the community~~ » Fixed

DIE LAST THEME FUNCTION DIE!!

Committed and pushed to 8.0.x. YEAH! :D

report as spam

Removing the template process layer

- Remove the process layer (hook_process and hook_process_HOOK)
- The template process layer has been removed

The process layer was created to flatten complicated data structures - such as objects or arrays - into strings. In practice, render arrays themselves allow for late rendering via theme function or template file (and can be manipulated in other preprocess functions), and objects can implement `__toString` methods for printing. There is no longer a need for this whole extra layer of processing before rendering.

Since the process layer got removed, the only layer between the data and the template is the *preprocess* layer.

The preprocess layer

The preprocess layer still exists, but it's used for a different purpose. In Drupal 8, the preprocess layer should not be used to add css classes. This should be done in the template files.

- CSS classes being moved from preprocess to Twig templates.

Render arrays

In Drupal 8, everything is a render array. This allows for late rendering, and allows data alterations to happen right before the template is being rendered.

Theme Hooks And Theme Hook Suggestions

hook_theme().

@todo: more info (see sandwich)

Theme Hook Suggestions

This hook allows any module or theme to provide alternative theme function or template name suggestions and reorder or remove suggestions provided by `hook_theme_suggestions_HOOK()` or by earlier invocations of this hook.

These theme suggestions will show up in the Twig debug output (when enabled). This allows to create a more complex logic to decide which template file should be used to render the data.

> Don't get confused with the double `hook`. The first hook stands for the name of the **theme or module**, the second stands for the **template**.

Drupal 7:

```
<?php
/**
* Implements hook_preprocess_HOOK() for node templates.
*/
function MYTHEME_preprocess_node(&$variables) {
  $variables['theme_hook_suggestions'][] = 'node__' . 'first';
  $variables['theme_hook_suggestions'][] = 'node__' . 'second';
}
```

Drupal 8:

```
<?php
/**
* Implements hook_theme_suggestions_HOOK_alter() for node templates.
*/
function MYTHEME_theme_suggestions_node_alter(array &$suggestions, array $variables) {
  $suggestions[] = 'node__' . 'first';
  $suggestions[] = 'node__' . 'second';
}
```

To add a template file suggestion for the `page` hook, you would implement `hook_theme_suggestions_page_alter()` .

hook_preprocess for suggestions

Each theme hook suggestion will automatically get it's own preprocess function. For the example above, the following two preprocess functions will be available:

```
/* for $suggestions[] = 'node__' . 'first'; */
function hook_preprocess_node__first(&$variables);

/* for $suggestions[] = 'node__' . 'second'; */
function hook_preprocess_node__second(&$variables);
```

Twig

在前面的章节中提到了新的 Drupal 模板引擎 Twig。在本章中会简单的介绍一下如何使用这一模板引擎。

Twig 的第一个版本发布于 2009 年 10 月 12 日。语法参考了 Jinja 和 Django 模板。目前 Twig 由 **Fabien Potencier** 维护。

Twig 基础

启用 Twig 的自动转义

> 字符串转义的意思是减少字符串中的歧义字符。

（Drupal 在）缺省情况下启用 Twig 的自动转义。这意味着 Twig 模板输出的字符串（ `{{ }}` 之间的内容 ）都是被转义处理过的。

输出一个变量

用 `{{ 变量 }}` 的方式输出模板中的一个变量，例如 `{{ foo }}`

句点 （ `.` ）用于访问对象的属性：

```
{{ foo.bar }}
```

> （上面的语句，）Twig 会自动进行如下尝试：

```
// 数组键。
$foo['bar'];
// 对象属性。
$foo->bar;
// 对象方法。
$foo->bar();
// 对象属性的 Getter。
$foo->getBar();
// 尝试 is + 方法名。
$foo->isBar();
// isset 和 get 方法
`$foo->__isset('bar');`
`$foo->__get('bar');`
```

PHP

```
$awesome_array = array(
  'a_key' => 'a_value',
  'another_key' => array(
    'foo' => 'bar',
  ),
);
```

Twig

```
{{ awesome_array.a_key }} # 返回 'a_value'
{{ awesome_array.another_key.foo }} # 返回 'bar'
```

Twig 过滤器（filter）

可以在输出变量之前用过滤器对变量进行处理，使用方式是： `{{ variable|filter }}` 。

Twig 自有过滤器

下面是一些 Twig 引擎自带的过滤器的例子。

- `|length` ，返回字典或序列类型数据的数量，或者字符串的长度。
- `|lower` ，把值转换为小写。
- `|upper` ，把值转换为大写。
- …

缺省 Twig 过滤器的完整列表。

Drupal 过滤器

Drupal 提供了额外的过滤器。

翻译过滤器

- `t` ：这个过滤器会使用 Drupal `t()` 函数处理变量，该函数会返回一个翻译结果，例如：

```
{% raw %}
<a href="{{ url('<front>') }}" title="{{ 'Home'|t }}" rel="home" class="site-logo"></a>
{% endraw %}
```

- `passthrough`
- `placeholder`

为了安全的对 `{% trans %}` 标记中检测到的变量进行转义，缺省情况下会使用 `@` 前缀。要传递变量（!）或者使用占位符（%），可以使用 `passthrough` 或 `placeholder` 过滤器。

- **passthrough** gets no sanitization or formatting and should only be used for text that has already been prepared for HTML display.
- **placeholder** 会转义为 HTML 并使用 `drupal_placeholder()` 格式化，并显示为 `<em>emp</em>` 的形式。

- `raw` 会显示原始数据（也就是不做转义处理）。

替换 Twig 的自有过滤器

- `drupal_escape` 是 Twig 的转义过滤器的替代品。参见 `TwigExtension::escapeFilter`

实现安全的字符串连接

- `safe_join` 用于安全的连接多个字符串。参见 `twig_drupal_join_filter` 。

数组过滤器

- `without` 创建一个可渲染数组的拷贝，并根据过滤器的参数移除指定的子元素。这样这个拷贝就可以在没有这些元素的情况下被输出；而原有的可渲染数组还可以在模板中继续被输出（带有全部子元素）。

CSS 和 ID 过滤器

- `clean_class` 会把字符串转换为一个有效的 HTML 类名。使用 Html::getClass
- `clean_id` 把字符串转换为有效的 HTML ID。使用 Html::getId

翻译

Twig 的 i18n 扩展允许把模板的一部分标记为可翻译内容。我们以下面的代码为例：

```
{% trans %} Hello world {% endtrans %}
```

可以在可翻译字符串中间利用 `{{ variable }}` 语法嵌入变量,

```
{% trans %} Hello {{ name }} {% endtrans %}
```

要在可翻译块中使用过滤器对变量进行处理，可以首先把过滤器结果赋值给一个变量。

```
{% set name = name|capitalize %}

{% trans %}
  Hello {{ name }}!
{% endtrans %}
```

可翻译字符串可以有复数操作，只要实现一个 `{% plural ... %}` 开关就完成了：

```
{% trans %}
  Hello star.
{% plural count %}
  Hello {{ count }} stars.
{% endtrans %}
```

注释

```
{# 用这一对标记来包含注释 #}
```

条件判断 （此处原文为 **Function**，疑似错误）

`if` ：

```
{% if site_slogan %}
  <div class="site-slogan">{{ site_slogan }}</div>
{% endif %}
```

循环

一个 `for` 循环。输出结果为 `0, 1, 2, 3` 。

> `range` 函数返回一个包含顺序递增的整数列表。

```
{% for i in range(0, 3) %}
  {{ i }},
{% endfor %}
```

来自 `field--node--title.html.twig` 模板的另一个例子:

```
{% for item in items %}
  {{ item.content }}
{% endfor %}
```

在 For 循环中，可以使用一些特殊变量。

变量	描述
items.index	当前的循环次数（从 1 开始）
items.index0	当前的循环次数（从 0 开始）
items.revindex	剩余的循环次数（从 1 开始）
items.revindex0	剩余的循环次数（从 0 开始）
items.first	是否初次执行
items.last	是否最后一次执行
items.length	序列中的元素数量
items.parent	上级的上下文

如果循环列表为空，可以利用一个 `else` 语句块来代替输出：

```
<ul>
{% for user in users %}
  <li>{{ user.username|e }}</li>
{% else %}
  <li><em>查询结果为空</em></li>
{% endfor %}
</ul>
```

创建一个 Twig 变量

Sometimes it might be useful to define variables in a template file. `{% set foo="bar" %}` declares a variable `foo` and assigns the value `bar` to it. Later in the template, the variable can be printed out using `{{ foo }}` .

有时候需要在模板中定义变量。 `{% set foo="bar" %}` 声明了一个变量 `foo` ，并赋值为 `bar` 。后面的模板中可以使用 `{{ foo }}` 来输出。

example.twig.php

```
{% set foo="bar" %}

# 其他代码

Hi, here's my variable: {{ foo }}
```

除了上面的简单变量之外，变量还可以是数组：

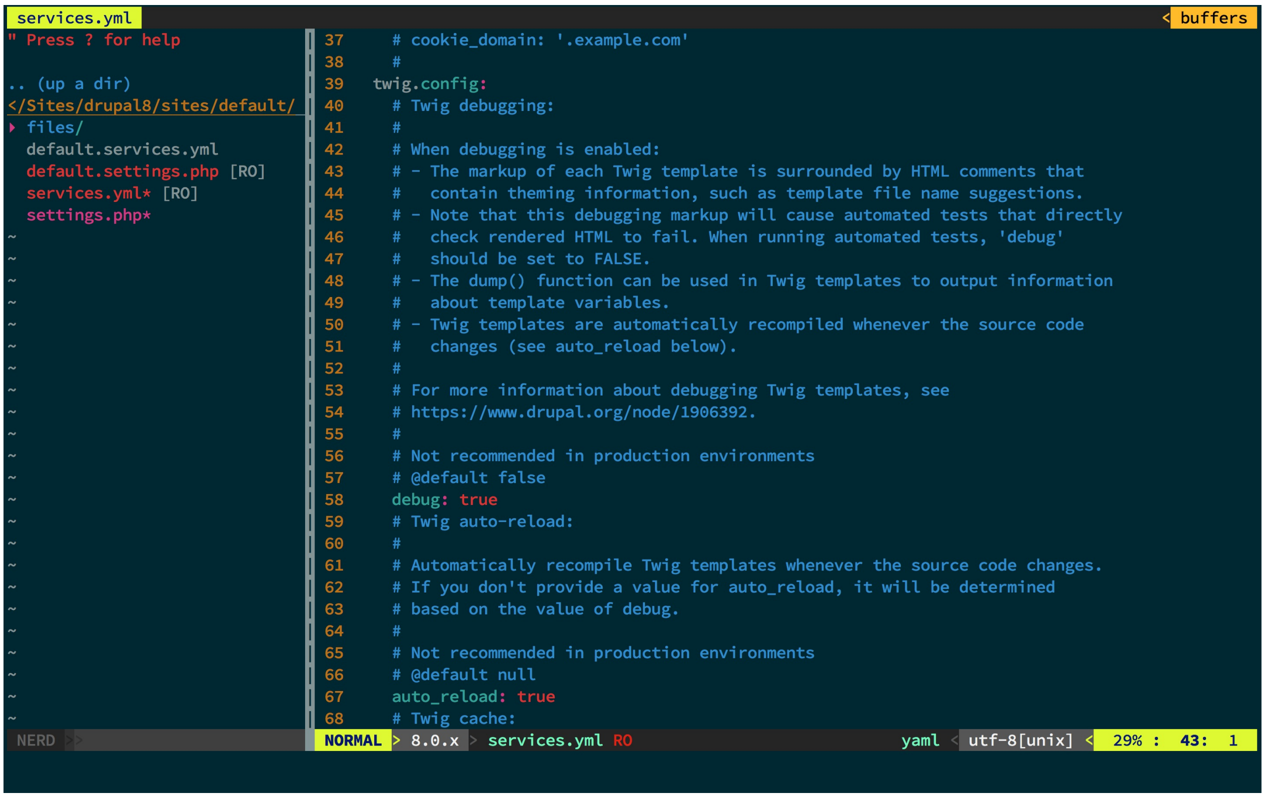
another_example.twig.php

```
{%
  set foo_array = [
    'foo',
    'bar',
  ]
%}
```

Twig 调试

A new feature in Drupal core is the **Twig debug** tool. It allows developers to trace from which template files certain markup comes. To enable Twig Debugging, set the `debug` parameter in the `twig.config` to `true` .

Drupal 核心的一个新特性就是 **Twig** 调试工具。这一工具让开发者可以通过跟踪，来得知某些标记来自于哪一个模板文件。要启用 Twig 调试，可以在 `twig.config` 文件中把 `debug` 参数设置为 `true`



```
services.yml                                                              buffers
" Press ? for help
.. (up a dir)
</Sites/drupal8/sites/default/
▸ files/
  default.services.yml
  default.settings.php [RO]
  services.yml* [RO]
  settings.php*
NERD
```

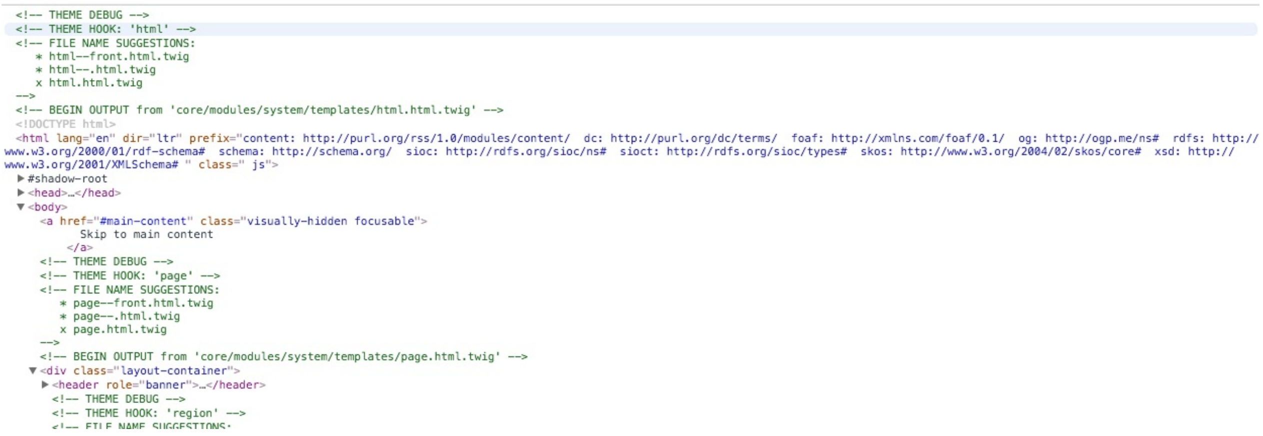
```
    # cookie_domain: '.example.com'
    #
  twig.config:
    # Twig debugging:
    #
    # When debugging is enabled:
    # - The markup of each Twig template is surrounded by HTML comments that
    #   contain theming information, such as template file name suggestions.
    # - Note that this debugging markup will cause automated tests that directly
    #   check rendered HTML to fail. When running automated tests, 'debug'
    #   should be set to FALSE.
    # - The dump() function can be used in Twig templates to output information
    #   about template variables.
    # - Twig templates are automatically recompiled whenever the source code
    #   changes (see auto_reload below).
    #
    # For more information about debugging Twig templates, see
    # https://www.drupal.org/node/1906392.
    #
    # Not recommended in production environments
    # @default false
    debug: true
    # Twig auto-reload:
    #
    # Automatically recompile Twig templates whenever the source code changes.
    # If you don't provide a value for auto_reload, it will be determined
    # based on the value of debug.
    #
    # Not recommended in production environments
    # @default null
    auto_reload: true
    # Twig cache:
NORMAL > 8.0.x > services.yml RO                    yaml < utf-8[unix] < 29% :  43:  1
```

sites/default/services.yml:

```
parameters:
  twig.config:
    debug: true # originally false
```

当打开 Web 控制台时，会看到加入了 HTML 注释：

```
<!-- FILE NAME SUGGESTIONS:
   * html--front.html.twig
   * html--.html.twig
   x html.html.twig
-->
<!-- BEGIN OUTPUT from 'core/modules/system/templates/html.html.twig' -->
```



```
<!-- THEME DEBUG -->
<!-- THEME HOOK: 'html' -->
<!-- FILE NAME SUGGESTIONS:
   * html--front.html.twig
   * html--.html.twig
   x html.html.twig
-->
<!-- BEGIN OUTPUT from 'core/modules/system/templates/html.html.twig' -->
<!DOCTYPE html>
<html lang="en" dir="ltr" prefix="content: http://purl.org/rss/1.0/modules/content/  dc: http://purl.org/dc/terms/  foaf: http://xmlns.com/foaf/0.1/  og: http://ogp.me/ns#  rdfs: http://
www.w3.org/2000/01/rdf-schema#  schema: http://schema.org/  sioc: http://rdfs.org/sioc/ns#  sioct: http://rdfs.org/sioc/types#  skos: http://www.w3.org/2004/02/skos/core#  xsd: http://
www.w3.org/2001/XMLSchema# " class=" js">
 ▶#shadow-root
 ▶<head>…</head>
 ▼<body>
    <a href="#main-content" class="visually-hidden focusable">
          Skip to main content
        </a>
    <!-- THEME DEBUG -->
    <!-- THEME HOOK: 'page' -->
    <!-- FILE NAME SUGGESTIONS:
       * page--front.html.twig
       * page--.html.twig
       x page.html.twig
    -->
    <!-- BEGIN OUTPUT from 'core/modules/system/templates/page.html.twig' -->
  ▼<div class="layout-container">
    ▶<header role="banner">…</header>
      <!-- THEME DEBUG -->
      <!-- THEME HOOK: 'region' -->
      <!-- FILE NAME SUGGESTIONS:
```

> **Drupal 7** 中也有类似功能。可以通过在 `settings.php` 设置 `$conf['theme_debug'] = TRUE;` 来激活此功能。

模板建议

The files with **(*)** are theme hook suggestions, and can be used to override the template file in more specific and less generic use cases. The file marked with **(x)** is template file that is currently being used to print the markup. The comments end with the file that is currently being using to print the markup.

带有（*）的文件是主题建议，在特定条件下可以用来覆盖模板文件。 带有（**x**）的文件是当前正在使用并输出被跟踪标记的模板。注释中最后一个文件是用于指出最后输出的模板。

```
<!-- FILE NAME SUGGESTIONS:
   * html--front.html.twig
   * html--.html.twig
   x html.html.twig
-->
<!-- BEGIN OUTPUT from 'core/modules/system/templates/html.html.twig' -->
```

调试变量

要输出（模板文件中）所有的可用变量，可以使用 `dump()` 函数。要获取一个特定变量的内容，只要把变量传递给这个函数即可。

> 注：当使用 `dump()` 的时候，可能会遭遇白屏。这个命令会递归遍历和输出所有变量，造成大量内存消耗。

```
# 只输出可用键
{{ dump(\_context|keys) }}
```

输出全部可用变量

```
{{ dump() }}
```

> 注：同下面的 `dump(_context)` 等效。

输出 `$foo` 变量：

```
{{ dump(foo) }}
```

输出 `$foo` 和 `$bar` 变量：

```
{{ dump(foo, bar) }}
```

在所有的 Twig 模板中都有两个全局变量：

- `_context` , which contains all variables that are passed to the template file. When using `dump()` without passing in a variable, the `_context` will be printed.
- `_context` ，包含了传递给这一模板文件的所有变量。当不带有参数使用 `dump()` 的时候，就会输出 `_context` 。
- `_charset` 代表当前的字符集。

Devel 和 Kint

`dump()` 在调试单个变量的时候很好用，但是在输出所有变量的时候就比较辛苦了。这就是使用 **Devel** module 的时机了。这个第三方模块包含很多的子模块，其中一个就是 **Kint**，它提供了一个友好的界面来展现调试数据。

```
# 下载 Devel 模块
drush dl devel
# 启用 Devel 以及 Kint 模块
drush en devel kint
```

激活之后，可以使用 `kint()` 来替换所有的 `dump()` ，以获得更好的调试体验。

延伸阅读

- 在 Twig 模板 中加入 Kint 支持
- 在 Twig 模板中发现和查看变量
- 调试编译后的 Twig 模板

模板文件 (Twig)

下一节将会详细介绍如何修改和覆盖这些模板。还将讲述如何获得加入到布局中的全部类的控制权。

/templates 文件夹

因为 `theme_` 函数已经不复存在，几乎所有的核心主题（以及模块）都包含了新的名为 `templates` 的目录。在这个目录中保存了 Twig 的模板文件。

Libraries

Libraries are the most imported part for Drupal 8 themes. They are used to add any kind of assets (css or javascript) to the page.

Global styling

Stylesheets that should be included on all pages are called globing styling. It's considered a good practice to use `{theme}/globing-styling` as a key for this sort of stylesheets.

Global scripts

Scripts that should be included on all pages are called globing scripts. It's considered a good practice to use `{theme}/globing-scrips` as a key for this sort of javascript.

> **NOTE:** Don't just include all your css or javascript files on every page!

Adding a javascript library

Here is an example of how to add a custom javascript file to theme.

Start by creating a `*.libraries.yml` file, e.g. `awesome.libraries.yml` (`{theme-or-module-name}` .libraries.yml) and save it into the root of the theme.

```
base:
  version: 1.x
  js:
    js/awesome.js: {}
  dependencies:
    - core/drupal
    - core/jquery
    - core/jquery.once
```

> We need `core/drupal` in order to take advantage of the `Drupal.behaviors`. To include the Drupal core version of jQuery, we add `core/jquery`. The final dependency is `core/jquery.once`, a jQuery plugin allowing to only apply a function once to an element.

Open up the `*.info.yml` file of the theme and add the following lines to include the *library* into the theme.

```
libraries:
  - awesome/base
```

This includes the custom javascript and also makes sure that all dependencies are loaded. For example: Drupal will make sure that jQuery is loaded before importing `awesome.js` .

The history of libraries, part 1

During the early development of Drupal 8, assets such as javascript and stylesheets had their own, different way of getting imported into themes. Importing a custom javascript file was done using the `scripts` property inside the `.info.yml` -file of the theme. To import stylesheets, the `stylesheets` property was used. This was more or less the way it was done in Drupal 7. Here is an example from a Drupal 7 `theme.info` file.

```
...

# Adding stylesheets
stylesheets[all][] = css/styles.css
stylesheets[print][] = css/print.css

# Adding javascript files
scripts[] = js/script.js

...
```

An alternative path was chosen. Javascript maintainer **_nod** created an issue on using AMD for JS architecture. The patch for the issue became so big that a separate issue was created. This issue handled the splitting up of the dependencies. This patch, Explicitly declare all JS dependencies, don't use drupal_add_js, where all javascript files are declared as libraries and added the relevant dependencies to the scripts, got committed about two weeks after the creation. The original issue [**AMD architecture**] however was posponed to a later Drupal release (Drupal 9.x).

{}.libraries.yml

During the early stages of Drupal 8, this was done using `hook_library_info`. Since this was one of the last remaining hooks in Drupal 8, it got replaced by a `*.libraries.yml` file). This means modules, themes and profiles can define their own `*.libraries.yml` file. The file should be in the root directory of the theme (or module).

```
# example.libraries.yml

base:
  version: 1.x
  js:
    js/scripts.js: {}
  css:
    component:
      css/components/component.css: {}
  dependencies:
    - core/drupal
    - core/jquery
```

Inside the `.info.yml` file from a module or theme, the library can be included using the `libraries` property.

```
# example.info.yml

libraries:
  - example/base
```

The *example* theme now has a global dependency on the `base` library (from the *example* theme/module); `example/base` . Therefor it's included on **every page** (where this theme is loaded). The `base` library itself contains a single javascript file (`scripts.js`), a css file (categoried as a component) and has dependencies on the `drupal` and `jquery` library from core (read more on this in the **Javascript** chapter).

The history of libraries, part 2

Due to this change, the `scripts` tag from the `.info.yml` , was removed and replaced with the `libraries` property. This means scripts can only be included using the `libraries` property and a `libraries.yml` file. Stylesheets however (in themes) could still included using the old `stylesheets` property:

_nod:

> remove scripts[] from info files and replace it with library[] …
>
> stylesheets[] can be kept as is, it's totally fair to add a single css file without dependencies. It's highly unlikely for scripts, they will need to depend on jquery, drupal.js 99% of the time.

It became clear this new way of adding assets had some major advantages. Wim Leers created a new issue stating Themes should use libraries, not individual stylesheets. Due to the advantages pointed out in this issue, the `stylesheets` property from the `.info` file got removed. Thanks to this, there is now a unified way for adding assets libraries.

The advantages of libraries and dependencies

Copied from the issue summary of 2377397:

1. dependencies, hence a step towards getting rid of weights
2. one uniformly applied system for modules and themes alike
3. themes are also forced to think about categorisation in their CSS
4. libraries can contain CSS and JS as logically associated units, and can declare dependencies on other libraries
5. themes are encouraged to create a library per component/topic/logical unit and hence are encouraged to not load all their CSS on all pages anymore, but to attach it when appropriate in a preprocess hook and/or only on the relevant pages via hook_page_attachments_alter()

Attach libraries directly from a template file

As mentioned, the libraries inside the `*.info.yml` file are include globally, on every page. To conditionally add a library, a `preprocess` function (`#attached`), just like in Drupal 7, had to be used. In order to improve the front-end developer experience, a simple Twig function has been added to easify this. `{{ attach_library('theme/library') }}` makes it possible to add libraries directy from template files. A huge win.

> The only template files that doesn't support this is **html.html.twig**. More information can be found here.

Core libraries

A quick overview with some of the Drupal core libraries.

> To get a **complete overview** of all the core libraries, take a look inside `core/core.libraries.yml`.

- `core/drupal`

We need `core/drupal` in order to take advantage of the `Drupal.behaviors`.

- `core/jquery`

To include the Drupal core version of jQuery, we add `core/jquery`.

- `core/jquery.once`

A jQuery plugin allowing to only apply a function once to an element.

> There are plenty more libraries in core.

Javascript

[Introduction]

Read more

File-closure

As part of the Drupal 8 Javascript Coding Standards, all of the javascript code **must** be declared inside a closure wrapping the whole file. This closure **must** be in strict mode.

```
(function () {
  'use strict';
  // Custom javascript
})();
```

ESHint

> As of Drupal 8, we use ESLint to make sure our JavaScript code is consistent and free from syntax error and leaking variables and that it can be properly minified.

More on ESLint

jQuery

Drupal 8 doesn't load any additional scripts by default. This means that jQuery is not included on every page any more. This is mostly due to performance reasons: you don't need jQuery on every page, so Drupal should only load it when needed. In order to use jQuery, you have to declare jQuery (or any other core library) as a dependency for your script.

> Since Drupal 8 does not support IE8 - and below - and because Javacript has evolved, you might not need jQuery. If however you do want to use jQuery, make sure to look up the best practices for using jQuery.

jQuery and Drupal

Drupal uses jQuery in a no-conflict mode. In jQuery, `$` is an alias for `jQuery`. Other javascript libraries often use `$` as a function or variable name. When using jQuery in a no-conflict mode, this `$` variable is released so other libraries can use this `$` variable.

Read more:

- **Drupal 7**: Added methods to avoid loading jQuery and related JavaScript libraries on all pages when they are not needed
- jQuery.noConflict(), from the jQuery API.

Multilingual in Javascript

The Drupal core javascript library (`core/drupal`) ships with two functions to support multilingual in Javascript. `Drupal.t()` and `Drupal.formatPlural()` are two functions available, that work just like their PHP equivalents:

Drupal.t()

> **PHP**: ?

```
var close = Drupal.t('Close');
var nodesOfType = Drupal.t('Showing nodes of @type', {@type: nodeType});
```

Drupal.formatPlural()

> **PHP**: \Drupal\Core\StringTranslation\TranslationInterface::formatPlural()

```
var nodesCount = Drupal.formatPlural(count, '1 node', '@count nodes');
var nodesCountOfType = Drupal.formatPlural(count, '1 node of @type', '@count nodes of @ty
```

Theme function (`Drupal.theme`)

`Drupal.theme` is the client-side counterpart of the server-side `theme()` -function. All of the server-side `theme()` function has been removed in Drupal 8. The client-side function however remains part of the Javascript API. All markup (defined in Javascript) should go through these theme functions.

The beauty of this system is that it allows the reuse of theme functions as templates and the override of these functions in themes. A module can define a theme function; a theme that wants to change the markup can override the function to define its own template.

The `Drupal.theme` function has been simplified. Previously it was possible to declare functions in `Drupal.theme.prototype` , where workaround code made `Drupal.theme()` behave as intended. Instead these Javascript theme functions are now using `Drupal.theme` directly.

Drupal 7:

```
Drupal.theme.prototype.example = function () {
  var markup = '<p>Hello world!</p>';
  return markup;
};

Drupal.theme.prototype.anotherExample = function (title, name) {
  var markup = '<p>Hello ' + title + ' ' + name '!</p>';
  return markup;
};

var example = Drupal.theme('example'); // Hello world!
var another_example = Drupal.theme('anotherExample', 'Mr.', 'Dries'); // Hello Mr. Dries!
```

Drupal 8:

```
Drupal.theme.example = function () {
  var markup = '<p>Hello world!</p>';
  return markup;
};

Drupal.theme.anotherExample = function (title, name) {
  var markup = '<p>Hello ' + title + ' ' + name '!</p>';
  return markup;
};

var example = Drupal.theme('example'); // Hello world!
var another_example = Drupal.theme('anotherExample', 'Mr.', 'Dries');
```

Declaring several theme functions

Declaring several theme functions can be done using the `$.extend()` pattern. This is a considered a better approach for declaring serveral functions.

```
$.extend(Drupal.theme, {
  example: function () { return 'Hello world!'; },
  anotherExample: function (title, name) { return '<p>Hello ' + title + ' ' + name '!</p>
});
```

Drupal behaviors (`Drupal.behaviors`)

Drupal behaviors (`Drupal.behaviors`) are still part of javascript in core. These behaviors will be executed on every request, including AJAX requests.

> Don't forget to add a dependency to `core/drupal` in order to use Behaviors.

Below is an example of a Drupal Javascript Behavior.

```
(function ($) {
  'use strict';

  Drupal.behaviors.awesome = {
    attach: function(context, settings) {
      $('main', context).once('awesome').append('<p>Hello world</p>');
    }
  };

}(jQuery));
```

Let's have a quick look at what this does.

- **namespace**: A Drupal behavior has to have a unique namespace. In this example, the namespace is `awesome` (`Drupal.behaviors.awesome`).
- **attach**: Contains the actual function that should be executed.
- **context**: The `context` variable is the part of the page for which this applies. On a page load, this will be the entire document. On a AJAX request however, the `context` variable will only contain all the newly loaded elements.
- **settings**: The `settings` variable is used to pass information from the PHP code to the javascript (`core/drupalSettings`).
- **once**: Using the `.once('awesome')` will make sure the code only runs once. Otherwise, the code will be executed on every AJAX request. It adds a `processed` - class to the `main` tag (`<main role="main" class="awesome-processed">`) in order to accomplish this (`core/jquery.once`).

```
(function ($) {
  'use strict';

  Drupal.behaviors.awesome = {
    attach: function(context, settings) {
    },
    detach: function (context, settings, trigger) {
    }
  };

}(jQuery));
```

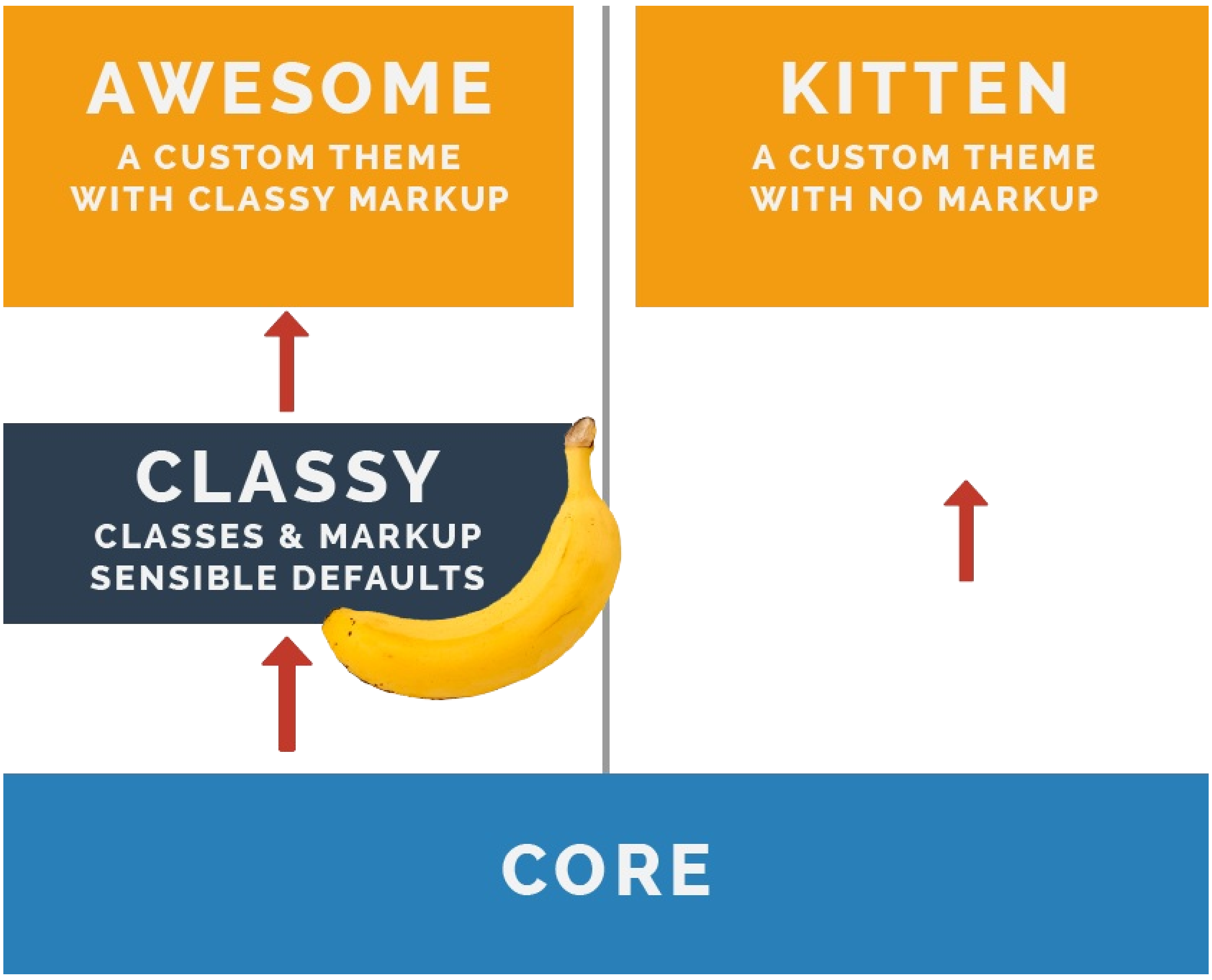
- **detach**: Alongside `attach` lives `detach`, which can be used to detach registered behaviors from a page element. Besides from a `context` and `settings`, it also expects a `trigger`. The `trigger` is a string containing the causing of the behavior to be detached. Possible causings are:
    - *unload*: (default) The context element is being removed from the DOM.
    - *move*: The element is about to be moved within the DOM (for example, during a tabledrag row swap). After the move is completed, Drupal.attachBehaviors() is called, so that the behavior can undo whatever it did in response to the move. Many behaviors won't need to do anything simply in response to the element being moved, but because IFRAME elements reload their "src" when being moved within the DOM, behaviors bound to IFRAME elements (like WYSIWYG editors) may need to take some action.
    - *serialize*: When an Ajax form is submitted, this is called with the form as the context. This provides every behavior within the form an opportunity to ensure that the field elements have correct content in them before the form is serialized. The canonical use-case is so that WYSIWYG editors can update the hidden textarea to which they are bound.

Creating a custom theme

When creating a custom theme, there are three possibilities:

1. Use **Classy** as a base theme. This will result in some default classes and markup in your template files (since the template files from **Classy** will be used). You can however still override these template files in the custom theme.
2. Using **Stable** as a base theme. The minimum amount of default markup and classes will be added to the template files. This way, you have full control of all the markup and classes. **This is the default option!**
3. **Not using a base theme**, by setting `base theme: false` .

In the following example, the relationship between two custom themes and Classy is demonstrated. The first theme, *Awesome*, uses **Classy** as a base theme. The second theme, *Kitten*, doesn't use **Classy** as a base theme.



The /themes directory

All right. Now is the time to get really excited. We're about to create a Drupal 8 theme! The question that raises is: where should this theme be located, where should I put the files?

> If you're familiar with Drupal 7 theming, the first place for you to look at would be `drupal/sites/all/themes/{custom/}` (the place where all your custom themes lived in Drupal 7).

In Drupal 8, this location has changed. Custom and contrib themes now live in `drupal/themes`. Also notice that (custom and contrib) modules now live inside `drupal/modules` (instead of drupal/sites/all/modules/).

> Did you notice {*custom*}? It's always a good practice to separate the contrib themes (the one's you've been downloading from d.o) and the ones you've written yourself. This can simply be done by creating two folders inside the `themes` directory:
>
> `contrib`: for contrib themes
> `custom`: for custom themes

Create your custom theme directory

Let's create an `example` directory inside `themes/custom`, resulting in `themes/custom/example`. Inside this directory, all the code for our custom theme will live.

Creating an info file

A simple .info.yml file

In Drupal 8, the `.info` files from themes, modules and profiles are replaced by `.info.yml` files (read the change record). These files are parsed using the Symphony YAML Component.

```
name: Awesome Theme
```

Fairly simple. This is the name of the theme. It's the name that also appears on the *Appearance* page, where you can activate your theme.

```
description: 'An awesome D8 theme.'
```

A theme description. This description is also displayed on the *Appearance* page.

```
package: Custom
```

The package in which your theme lives. For the core themes, this package is of course `core`.

```
type: theme
```

Since the `info.yml` files are used for besides themes also used for modules and profiles, this line lets Drupal know if it's a theme, module or a profile.

```
version: 1.0
```

The version of the theme.

```
core: 8.x
```

The version of Drupal core the theme requires.

Using a base theme

```
base theme: classy
```

The line above gives you the power to extend from a base theme. This line can be found in both `bartik.info.yml` and `seven.info.yml`, since these two core themes extend **Classy**.

*.info.yml

To wrap things up, this is our `.info.yml` file so far:

```
name: Awesome Theme
description: 'An awesome D8 theme.'
base theme: classy
package: Custom
type: theme
version: 1.0
core: 8.x
```

Save this as `{theme_name}.info.yml` (`awesome.info.yml`) inside the created custom theme directory (eg. `themes/custom/example/example.info.yml`). Now navigate to `admin/appearance` and see your theme displayed. You can now even enable the theme. Hooray!

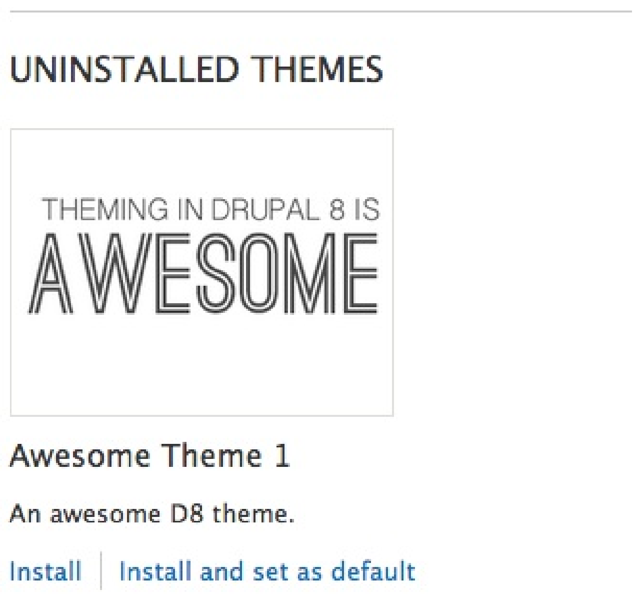
Adding a screenshot

When navigation to the appearance page, you might notice a *no screenshot* image for our theme. This is because no screenshot has been found (duh!). A screenshot is automatically detected and displayed if the filename is `screenshot.png`. If we add this file, we would see the screenshot on the appearance page.

Otherwise you can manually add a screenshot with the following line:

```
screenshot: otherfilename.png
```

If everything went well, you should be able to see the screenshot:



The screenshot of cause doesn't have to be a `*.png`. It can be an `*.svg` or even an animated `*.gif`

Conclusion: the filename for a screenshot does not have to be `screenshot.png`, as long as it is defined in the *{theme}.info.yml* file.

Adding stylesheets

In Drupal 8, both stylesheets and scripts should be added using libraries. Read more on how to create a library with css and javascript files in the **libraries** chapter.

Removing stylesheets

Alternatively, a stylesheet can also be removed.

```
# Remove a CSS file:
stylesheets-remove:
  - core/assets/vendor/jquery.ui/themes/base/dialog.css
```

To remove a stylesheet, the full path to the css file should be used (instead of just a file name). This due to the fact that multiple css files can have the same name.

In cases where a Drupal core asset is being removed the full file path is needed. In cases where the file is part of a library that belongs to a module or theme, a token can be used.

> When using a token, it needs to be quoted because `@` is a reserved indicator in YAML.

```
# Removing a stylesheet from Bartik using the @ token.
stylesheets-remove:
  - '@bartik/css/style.css'
```

Regions

Regions can be defined using the `regions` tag. Here is an example where 3 regions are defined: a *header*, a *content* and a *footer* region:

```
# Regions
regions:
  header: 'Header'
  content: 'Content'
  footer: 'Footer'
```

A region that is required in all theme is the `content` region.

Regions hidden

The `regions_hidden` can be applied to any previous defined *regions*. Regions with this attribute will not show up on the block administration page. This means they can't have blocks assigned to them by ordinary mechanisms. Drupal uses this feature to protect the special 'page_top' and 'page_bottom' regions from adventurous themers. This can be used by module writers and theme writers to protect a given region from having unexpected content inserted into the output. The `seven.info.yml` contains this tag, in order to *hide* the 'Sidebar First' region.

```
regions_hidden:
  - sidebar_first
```

Breakpoints and responsive images

This chapter takes a closer look inside both the `breakpoint` and `responsive_image` modules.

Breakpoints

The *breakpoint* module keeps track of the height, width and resolution breakpoints where a responsive design needs to change in order to respond to different devices being used to view the site. The breakpoint module standardises the use of breakpoints, and enables modules and themes to expose or use each others' breakpoints. The core module does not have a user interface. The breakpoints are stored inside a `{module-or-theme}.breakpoints.yml` file.

The change record can be found here: Breakpoint added to Drupal 8

Breakpoints configuration

Both themes and modules can define breakpoints by creating a configuration file called `{name}.breakpoints.yml` where `{name}` is the name of your theme or module.

To get a good example, let's take a look at `bartik.breakpoints.yml` :

```
bartik.mobile:
  label: mobile
  mediaQuery: '(min-width: 0px)'
  weight: 0
  multipliers:
    - 1x
bartik.narrow:
  label: narrow
  mediaQuery: 'all and (min-width: 560px) and (max-width: 850px)'
  weight: 1
  multipliers:
    - 1x
bartik.wide:
  label: wide
  mediaQuery: 'all and (min-width: 851px)'
  weight: 2
  multipliers:
    - 1x
```

Each breakpoint has it's own identifier. Bartik has 3 unique breakpoints: *mobile*, *narrow* and *wide*. Each breakpoint is defined as `{module or theme name}.{label}` , with `bartik.mobile` as an example. The name of the module or theme is used to make sure the identifier is unique. This way, modules can include breakpoints from each other.

label

Human readable label for breakpoint. It should be unique within a module or theme.

mediaQuery

The media query for the breakpoint.

weight

Weight used for ordering breakpoints.

multipliers

Breakpoint multipliers. Multipliers are a measure of the viewport's device resolution, defined as the ratio between the physical pixel size of the active device and the device-independent pixel size. The breakpoint module defines multipliers of 1, 1.5 and 2. When defining breakpoints, modules and themes can define which multipliers apply to each breakpoint.

Responsive images

The `responsive_image` module inside core …