eBook Gratuit

APPRENEZ pug

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#pug

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: pug

It is an unofficial and free pug ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official pug.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec Carlin

Remarques

Pug est un moteur de gabarit riche en fonctionnalités, robuste et élégant. Il a été influencé par Haml et implémenté avec JavaScript pour Node.js et les navigateurs. Les implémentations existent pour Laravel, PHP Scala, Ruby, Python et Java.

Il comporte:

- Intégration express
- Conditionnels
- Des filtres
- Comprend
- Héritage
- Interpolation
- Itération
- Mixins

Pug était auparavant connu sous le nom de *Jade,* mais a été renommé en raison d'une marque déposée.

Cette section de remarque devrait également mentionner tous les grands sujets dans Carlin et établir un lien avec les sujets connexes. La documentation de pug étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Versions

Version	Date de sortie
2.0.0-beta11	2017-02-04
2.0.0-beta10	2017-01-29
2.0.0-beta9	2017-01-25
2.0.0-beta1	2016-06-03
1.11.0	2015-06-11

Examples

Installation

Pour installer le système de rendu de modèle Pug, procédez comme suit:

1. Avoir l' environnement Node.js installé sur votre machine
2. Exécutez `npm install pug --save` pour installer le module `pug` dans votre projet en cours.

Vous pouvez maintenant utiliser `pug` dans votre projet via le mécanisme de `require` standard:

```
const pug = require("pug");
```

Si vous utilisez Express dans votre application, vous n'avez pas besoin de `require("pug")` . Toutefois, vous devez définir la propriété du `view engine` de votre application Express sur `pug` .

```
app.set("view engine", "pug");
```

De plus, vous devez définir le répertoire d'affichage de votre application pour qu'Express sache où rechercher vos fichiers Pug (pour la compilation).

```
app.set("views", "path/to/views");
```

Dans votre itinéraire Express, vous pouvez ensuite rendre vos fichiers Pug en appelant la fonction `res.render` avec le chemin du fichier (en commençant par le répertoire défini par l' `app.set("views")`).

```
app.get("/", function (req, res, next) {
    // Your route code
    var locals = {
        title: "Home",
    };
    res.render("index", locals);
});
```

Dans ce qui précède, l' `index` pointe vers un fichier situé à `views/index.pug` , et les `locals` représentent un objet de variables exposées à votre fichier. Comme expliqué dans les sections suivantes, Pug peut accéder aux variables qui lui sont transmises et effectuer diverses actions (conditions, interpolation, itération, etc.).

Bonjour Monde Exemple

Tout d'abord, créons un modèle à rendre!

```
p Hello World, #{name}!
```

Enregistrez-le dans un fichier se terminant par l'extension `.pug` (vous pouvez l'appeler comme vous voulez, mais nous utiliserons `view.pug` dans le code suivant pour le compiler).

Il ne reste plus qu'à compiler ce modèle! Créez un fichier de script JS (nous appellerons ours `main.js`) et ajoutez le contenu suivant:

```
// Import the pug module
const pug = require('pug');
```

```
// Compile the template (with the data not yet inserted)
const templateCompiler = pug.compileFile('view.pug');

// Insert your data into the template file
console.log(templateCompiler({ name: 'John' }));
```

Lorsque vous exécutez ce fichier avec `npm main.js` , vous devez obtenir le code HTML suivant dans votre console:

```
<p>Hello World, John!</p>
```

Félicitations, vous venez de créer et de compiler votre premier modèle! Sur des trucs plus avancés, tels que les conditionnels , les itérations et bien plus encore!

Lire Commencer avec Carlin en ligne: https://riptutorial.com/fr/pug/topic/8613/commencer-avec-carlin

Chapitre 2: Conditionnels

Introduction

Pug peut exécuter le code de manière conditionnelle en fonction des variables (transmises depuis votre serveur ou basées sur Pug lui-même).

Syntaxe

- si (déclaration)

```
// Pug code
```

- else if (statement)

```
// Pug code
```

- autre

```
// Pug code
```

- sauf si (déclaration)

```
// Pug code
```

Paramètres

Paramètre	Détails
si (déclaration)	Evalue l' `statement` pour voir si elle renvoie true ou false. Le code imbriqué en dessous `if` s'exécutera uniquement si `statement` renvoie true.
else if (statement)	Enchaîné à une instruction `if` ou `else if` existante; il ne s'exécute que si l'instruction précédente est évaluée à false. Le code imbriqué sous l'instruction `else if` s'exécutera uniquement si la valeur de l' `statement` true.
autre	Le code imbriqué sous l'instruction `else` s'exécutera uniquement si toutes les instructions précédentes renvoyaient false.
sauf si (déclaration)	La négation de `if (statement)` ; le code niché sous `if` ne fonctionnera que si la `statement` retourne false. C'est la même chose que `if (!statement)` .

Remarques

Documentation officielle PugJS sur les conditions

Examples

Déclaration If / Else en Carlin

Les conditions dans Pug peuvent évaluer les instructions d'une manière similaire à JavaScript. Vous pouvez évaluer les variables créées dans Pug ou celles transmises par votre route (`res.render` , `pug.renderFile` , etc.).

index.js

```
var authorized = true
res.render("index", {
    authorized: authorized
});
```

index.pug

```
- var showLogin = false;
if authorized && showLogin === true
    .welcome Welcome back to our website!
else
    .login
        a(href="/login") Login
```

sortie index.pug

```
<div class="login"><a href="/login">Login</a></div>
```

Déclaration If / Else dans Pug (avec un tiret)

Vous pouvez choisir d'ajouter un opérateur `if` ou `else` avec un tiret, mais ce n'est pas nécessaire. Vous devrez envelopper l'instruction entre parenthèses, cependant (si vous omettez un tiret, vous n'avez pas besoin de parenthèses).

```
- var showLogin = false;
- if (showLogin === true)
    .welcome Welcome back to our website!
- else
    .login
        a(href="/login") Login
```

sortie index.pug

```
<div class="login"><a href="/login">Login</a></div>
```

Sinon si déclaration

Vous pouvez enchaîner un nombre quelconque d'instructions `else if` à une instruction `if` existante pour évaluer une séquence d'instructions.

index.pug

```
- var page = 60;
if page => 52
    h1 Lots of numbers!
else if page > 26 && page < 52
    h1 A few numbers
else
    h1 Not a lot of numbers
```

sortie index.pug

```
<h1>Lots of numbers!</h1>
```

À moins d'opérateur

`unless` l'opération inverse de `if` dans Pug. C'est analogue à `if !(statement)` .

index.pug

```
- var likesCookies = true;
unless likesCookies === true
    h2 You don't like cookies :(
else
    h2 You like cookies!
```

sortie index.pug

```
<h1>You like cookies!</h1>
```

Note : `else unless` instructions ne fonctionnent pas à `unless` ; vous pouvez enchaîner une instruction `else if` à une instruction `unless` , mais à `else unless` cela ne fonctionne pas.

Lire Conditionnels en ligne: https://riptutorial.com/fr/pug/topic/9662/conditionnels

Chapitre 3: Génération de syntaxe et de balisage

Introduction

Un aperçu de la différence entre le code de publication et le balisage généré

Remarques

Pug permet d'écrire du code HTML de la manière la plus simple, en utilisant une syntaxe propre et sensible aux espaces.

Examples

De Pug au HTML

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) bar(1 + 5)
  body
    h1 Pug - node template engine
    #container.col
      if youAreUsingPug
        p You are amazing
      else
        p Get on it!
      p.
        Pug is a terse and simple templating language with a
        strong focus on performance and powerful features.
```

Devient:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Pug</title>
    <script type="text/javascript">
      if (foo) bar(1 + 5)
    </script>
  </head>
  <body>
    <h1>Pug - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>Pug is a terse and simple templating language with a strong focus on performance and
powerful features.</p>
    </div>
```

```
  </body>
</html>
```

Lire Génération de syntaxe et de balisage en ligne:
https://riptutorial.com/fr/pug/topic/9549/generation-de-syntaxe-et-de-balisage

Chapitre 4: Interpolation avec Carlin

Introduction

Il est important de pouvoir utiliser les variables côté serveur sur votre site Web. Pug vous permet d'interpoler les données générées par votre serveur en HTML, CSS et même en code JavaScript.

Syntaxe

- res.render (path, variables) // Recherche un fichier pug pour effectuer un rendu au chemin "path" et lui transmet des "variables"
- # {variable} // Interpole "variable" en ligne avec le code Jade environnant, après avoir évalué "variable"
- ! {variable} // Interpole "variable" en ligne avec le code Jade environnant, sans évaluer "variable".
- # [element] // Interpole "element" à l'intérieur d'un élément HTML Pug existant. La syntaxe des éléments HTML interpolés est identique à celle des éléments HTML normaux.

Paramètres

Paramètre	Détails
chemin	Utilisé dans `res.render` . C'est le chemin du fichier Pug que nous allons rendre. Le chemin provient de la racine du dossier défini sur votre application Express: `app.set("views", "templates/views")` . Par exemple, `res.render("index")` recherchera un fichier Pug dans `templates/views/index.pug` . Les sous-répertoires peuvent également être spécifiés. `res.render("admin/index")` recherche un fichier Pug dans `templates/views/admin/index.pug` .
les variables	Utilisé dans `res.render` . Un objet JavaScript de variables à rendre accessible au fichier Pug défini par `path` (ci-dessus). Dans le fichier Pug, les clés de l'objet JavaScript ci-dessus deviennent disponibles en tant que variables. Si `variables = {title: "Hello", color: "red"}` , nous pourrions utiliser la variable `title` et `color` . Les sous-propriétés des objets imbriqués sont également disponibles.
variable	Utilisé dans la syntaxe entre parenthèses `#{}` ou `!{}` . La valeur de `variable` sera sortie dans le contexte de son code Pug environnant. Si un symbole dièse est ajouté au crochet, la `variable` sera évaluée avant d'être sortie. Si un point d'exclamation est ajouté à l'accolade d'ouverture, la `variable` **ne sera pas** évaluée.
élément	Utilisé entre crochets sytax `#[]` . L'élément HTML (dans la syntaxe Pug, pas la syntaxe HTML normale) sera évalué et généré en ligne avec le code Pug environnant.

Remarques

Pour plus d'informations sur l'interpolation PugJS, voir la documentation officielle sur l'interpolation PugJS.

Examples

Interpolation des variables côté serveur

Il est possible de passer des variables de votre serveur dans Pug pour générer du contenu dynamique ou des scripts. Les modèles Pug peuvent accéder aux variables transmises à la fonction `res.render` dans Express (ou `pug.renderFile` si vous n'utilisez pas Express, les arguments sont identiques).

index.js

```
let colors = ["Red", "Green", "Blue"];
let langs  = ["HTML", "CSS", "JS"];
let title  = "My Cool Website";

let locals = {
    siteColors: colors,
    siteLangs:  langs,
    title:      title
};
res.render("index", locals);
```

Dans votre fichier `index.pug` , vous avez ensuite accès à la variable `locals` via ses clés. Les noms des variables de votre fichier Pug deviennent `siteColors` et `siteNames` .

Pour définir l'intégralité d'un élément HTML égal à une variable, utilisez l'opérateur égal à `=` pour le faire. Si votre variable doit être incorporée en ligne, utilisez la syntaxe de parenthèse `#{}` pour le faire.

index.pug

```
doctype html
html
    head
        title= title
    body
        p My favorite color is #{siteColors[0]}.
        p Here's a list of my favorite coding languages
        ul
            each language in siteLangs
                li= language
```

sortie index.pug

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <title>My Cool Website</title>
    </head>
    <body>
        <p>My favorite color is Red.</p>
        <p>Here's a list of my favorite coding languages</p>
        <ul>
            <li>HTML</li>
            <li>CSS</li>
            <li>JS</li>
        </ul>
    </body>
</html>
```

Interpolation des variables brutes en HTML

Le contenu interpolé avec la syntaxe de parenthèse sera évalué pour le code, dont la sortie est incluse dans votre sortie HTML.

> title suit le modèle de base pour évaluer un modèle local, mais le code entre `#{` et `}` est évalué, échappé et le résultat mis en mémoire tampon dans la sortie du modèle en cours de rendu. [La source]

Si vous devez inclure la syntaxe HTML brute, utilisez un point d'exclamation au lieu d'un symbole dièse (`!{}` lieu de `#{}`).

index.js

```
let tag = "<div>You can't escape me!</div>";
res.render("index", {
    myTag: tag
});
```

index.pug

```
doctype html
html
    head
    body
        !{myTag}
```

sortie index.pug

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <div>You can't escape me!</div>
    </body>
</html>
```

Interpolation de valeur dans le code JavaScript

L'interpolation des valeurs est utile si vous devez transmettre une variable côté serveur à JavaScript côté client (ou à d'autres langages qui le requièrent).

Dans le cas de variables, de nombres, de chaînes de caractères, etc., vous pouvez transmettre ces types de variables directement dans votre code JavaScript avec une syntaxe entre crochets et un point d'explication (le code entre parenthèses *n'est pas* évalué). Code JavaScript nécessitant quelque chose de votre serveur.

Dans l'exemple ci-dessous, nous devons envelopper le `username` entre guillemets pour que JavaScript puisse l'interpréter comme une chaîne. Pug affichera le contenu de la variable tel quel, nous devons donc le mettre entre guillemets pour qu'il devienne une chaîne JavaScript correcte. Ce n'est pas nécessaire pour le `number` , où JavaScript interprétera notre numéro comme nous le souhaitons (sous forme de nombre).

index.js

```
let number   = 24;
let username = "John";
res.render("index", {
    number:   number,
    username: username
});
```

index.pug

```
html
    head
    script.
        // Sets the username of the current user to be displayed site-wide
        function setUsername(username) {
            // ...
        }
        var number   = #{number};
        var username = "#{username}";
        setUsername(username);

    body
        p Welcome to my site!
```

sortie index.pug

```
<html>
    <head>
        <script>
            // Sets the username of the current user to be displayed site-wide
            function setUsername(username) {
                // ...
            }
            var number   = 24;
            var username = "John";
            setUsername(username);
        </script>
    </head>
    <body>
```

```
        <p>Welcome to my site!</p>
    </body>
</html>
```

Si vous devez interpoler la valeur d'un objet JavaScript (par exemple, toutes les informations relatives à un utilisateur), vous devez contraindre la sortie dans Pug pour qu'elle soit traitée comme un objet JavaScript. Il est également nécessaire de générer le contenu brut de la variable, au lieu de la forme évaluée. Si vous deviez faire sortie la variable échappé (`var user = #{JSON.stringify(user)}`), vous recevrez une version *échappée* de l'objet (où les guillemets et apostrophes sont convertis en `&quot;`), ce qui est ce que nous voulons que `JSON.stringify` fonctionne dessus.

index.js

```
var myUser = {
    name:    "Leeroy Jenkins",
    id:      1234567890,
    address: "123 Wilson Way, New York NY, 10165"
};

res.render('index', {
    user: myUser
});
```

index.pug

```
doctype html
html
    head
        script.
            window.onload = function () {
                function setUsername(username) {
                    return username;
                }

                var user = !{JSON.stringify(user)};
                document.getElementById("welcome-user").innerHTML = setUsername(user.name);
            };

    body
        div(id="welcome-user")
```

sortie index.pug

```
<!DOCTYPE html>
<html>
    <head>
        <script>
            window.onload = function() {
                function setUsername(username) {
                    return username;
                }

                var user = {
                    "name": "Leeroy Jenkins",
```

```
                    "id": 1234567890,
                    "address": "123 Wilson Way, New York NY, 10165"
                };
                document.getElementById("welcome-user").innerHTML = setUsername(user.name);
        };
        </script>
    </head>
    <body>
        <div id="welcome-user"></div>
    </body>
</html>
```

Le `innerHTML` de `#welcome-user` devient égal à `Leeroy Jenkins` . Le contenu de la variable `user` est imprimé directement dans la source HTML

Interpolation d'éléments HTML

Il peut être nécessaire d'imbriquer les balises HTML les unes dans les autres. L'interpolation d'éléments est effectuée dans une syntaxe similaire à l'interpolation de variables; les crochets au lieu des accolades sont utilisés ici. La syntaxe des éléments HTML interpolés est identique à celle des éléments HTML normaux.

index.pug

```
doctype html
html
    head
        title My Awesome Website
    body
        p The server room went #[b boom]!
        p The fire alarm, however, #[u failed to go off...]
        p Not even #[a(href="https://stackoverflow.com/") Stack Overflow] could save them now.
```

sortie index.pug

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Awesome Website</title>
    </head>
    <body>
        <p>The server room went <b>boom</b>!</p>
        <p>The fire alarm, however, <u>failed to go off...</u></p>
        <p>Not even <a href="https://stackoverflow.com/">Stack Overflow</a> could save them
now.</p>
    </body>
</html>
```

Lire Interpolation avec Carlin en ligne: https://riptutorial.com/fr/pug/topic/9565/interpolation-avec-carlin

Chapitre 5: Itération avec Carlin

Introduction

Comment itérer sur un simple objet JSON et économiser beaucoup de saisie

Remarques

Vous devez avoir Node.js et Pug installés

Examples

Chaque itération

Construisez un `app.js` avec un simple *magasin de données* :

```
app.get("/bookstore", function (req, res, next) {
    // Your route data
   var bookStore = [
        {
            title: "Templating with Pug",
            author: "Winston Smith",
            pages: 143,
            year: 2017
        },
        {
            title: "Node.js will help",
            author: "Guy Fake",
            pages: 879,
            year: 2015
        }
    ];
    res.render("index", {
        bookStore: bookStore
    });
});
```

Itérer sur le magasin de données en utilisant un fichier `index.pug` et une boucle:

```
each book in bookStore
  ul
    li= book.title
    li= book.author
    li= book.pages
    li= book.year
```

Le résultat sera:

```
<ul>
  <li>Templating with Pug</li>
```

```
  <li>Winston Smith</li>
  <li>143</li>
  <li>2017</li>
</ul>
<ul>
  <li>Node.js will help</li>
  <li>Guy Fake</li>
  <li>879</li>
  <li>2015</li>
</ul>
```

Référence

Lire Itération avec Carlin en ligne: https://riptutorial.com/fr/pug/topic/9545/iteration-avec-carlin

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec Carlin	Community, gandreadis, Shea Belsky, smonff
2	Conditionnels	Shea Belsky
3	Génération de syntaxe et de balisage	smonff
4	Interpolation avec Carlin	Shea Belsky
5	Itération avec Carlin	Shea Belsky, smonff