EBook Gratis

APRENDIZAJE

jade

Free unaffiliated eBook created from
Stack Overflow contributors.

#jade

Tabla de contenido

Acerca de

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

It is an unofficial and free jade 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 jade.

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

Capítulo 1: Empezando con jade

Observaciones

Pug (anteriormente conocido como jade) es un motor de plantillas de alto rendimiento fuertemente influenciado por Haml e implementado con JavaScript para Node.js y navegadores.

1. Produce HTML
2. Soporta código dinámico
3. Apoya la reutilización (SECO)

Página de inicio

Repositorio en GitHub

Versiones

versión	fecha de lanzamiento
0.0.2	2010-07-03
1.0.0	2013-12-22
1.11.0	2015-06-12

Examples

Instalación o configuración

Antes de iniciarte a codificar con Pug, debes tener algunos requisitos previos.

Necesitará instalar:

- NodeJS con NPM
- ExpressJS (opcional)

Después de instalar NodeJS, puede verificar en su terminal la instalación correcta haciendo:

```
$ node -v
```

Si tiene éxito, imprimirá el número de la versión de Node.

Para instalar Pug en su proyecto, la forma preferida y fácil es a través de NPM (Node Package Manager). Si está familiarizado con eso, simplemente ejecute esta línea de código en su Terminal:

```
$ npm install pug
```

Si quieres instalarlo globalmente, puedes escribir:

```
$ npm install pug-cli -g
```

y correr con

```
$ pug --help
```

Sintaxis

Pug (el nombre antiguo es Jade) es una sintaxis limpia y sensible al espacio en blanco para escribir HTML. Aquí hay un ejemplo simple:

```
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.
```

Produce la siguiente salida como HTML

```
<!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>
```

Aquí están las reglas para hacer Pug al código HTML:

1. Al sangrar el texto, se construirá el árbol HTML. La sangría se puede utilizar con espacios o tabulaciones. Esto no se pudo mezclar!
2. Las etiquetas HTML se escriben sin `<` y `>` . Los atributos son lugares entre paréntesis.
3. El comentario podría hacerse con `//` o `<!-- -->` . Los comentarios con `//-` no son visibles en el HTML renderizado.
4. Con `#{ }` generará un modelo ofrecido: `#{header} #{user.username}` .
5. El `#` (hashtag) sin llaves será un elemento `div` creado con el texto como ID. El ejemplo `#myID` se representará como `<div id="myID"></div>` .
6. Con un `.` *(punto)* será un `div` generado con un atributo de clase. Ejemplo: `.myClass` se representará como `<div class="myClass"></div>`
7. Con `=` (signo de igualdad seguido de un espacio), se recuperará una variable. Ejemplo: `h1= title`
8. A `!=` (No igual a) recuperó una variable sin escapar.
9. A `-` (guión) le permite escribir JavaScript. Ejemplo: `- console.log("foo");`
10. El enlace a un archivo externo puede ser el siguiente: `script(src="/js/chat.js")`
11. El script en línea podría usar este `script.` .
12. Una directiva para añadir el diseño básico: se `extends ../layout` .
13. En `layout.pug` sucede la inserción mediante el uso de `block content`
14. El uso de parciales se puede realizar de dos maneras:
    1. por parcial `!= partial(template file name/options)` .
    2. Por incluir: `include ../includes/footer`
15. Lo inverso de incluir es `extend` . Esto permite desde una página "partes de `block` html" enviar a una página de diseño, por ejemplo: `extend layout`
16. La concatenación ocurre con el carácter `+` (más) o `#` (hashtag). El plus se utiliza en el código JavaScript. El hashtag en HTML y muestra el contenido: `p El nombre es: # {miNombre}

Usando pug con Node.js

```
var pug = require('pug');

// compile
var fn = pug.compile('string of pug', options);
var html = fn(locals);

// render
var html = pug.render('string of pug', merge(options, locals));

// renderFile
var html = pug.renderFile('filename.pug', merge(options, locals));
```

Opciones

- *nombre de archivo* Se utiliza en excepciones y se requiere cuando se utilizan incluye
- *compileDebug* Cuando es falso, no se compila la instrumentación de depuración
- *bonito* Agregue espacios en blanco de sangría bonita a la salida (falso por defecto)

Lea Empezando con jade en línea: https://riptutorial.com/es/jade/topic/1709/empezando-con-jade

Capítulo 2: arpa js

Introducción

Harp es un servidor web estático con preprocesamiento incorporado. Harp puede compilar su proyecto en activos estáticos, HTML, CSS y JavaScript, sin necesidad de configuración. También puede usar Harp como una biblioteca de nodos para compilar o ejecutar como un servidor.

Harp incluye los preprocesadores comunes y útiles de forma predeterminada. Sirve Jade (Pug), Markdown, EJS, CoffeeScript, LESS, Sass y Stylus.

Examples

Cómo configurar arpa

Arpa no requiere ninguna configuración para empezar. Instala Harp en tu terminal usando el comando: `npm install -g harp` .

```
$ sudo npm install -g harp
$ harp init myproject
$ harp server myproject
```

Lea arpa js en línea: https://riptutorial.com/es/jade/topic/9863/arpa-js

Capítulo 3: Atributos

Examples

Atributos normales

Etiqueta HTML:

Los atributos de las etiquetas son similares a html, sin embargo, sus valores son solo JavaScript normal.

```
a(href='google.com') Google
a(class='button', href='google.com') Google
```

Resultado:

```
<a href="google.com">Google</a><a href="google.com" class="button">Google</a>
```

Variable:

Todas las expresiones normales de JavaScript funcionan bien también:

```
- var authenticated = true
body(class=authenticated ? 'authed' : 'anon')
```

Resultado:

```
<body class="authed"></body>
```

Muchos atributos

Si tiene muchos atributos, también puede distribuirlos en varias líneas:

```
input(
  type='checkbox'
  name='agreement'
  checked
)
```

Resultado:

```
<input type="checkbox" name="agreement" checked="checked"/>
```

Atributos Sin Escapar

De forma predeterminada, todos los atributos se escapan (reemplazando los caracteres especiales con secuencias de escape) para evitar ataques como los scripts entre sitios. Si necesita usar caracteres especiales, puede usar `!=` lugar de `=` .

Código:

```
div(escaped="<code>")
div(unescaped!="<code>")
```

Resultado:

```
<div escaped="&lt;code&gt;"></div>
<div unescaped="<code>"></div>
```

Atributos booleanos

Los atributos booleanos son reflejados por Jade y aceptan bools, también conocidos como `true` o `false` . Cuando no se especifica ningún valor se asume verdadero.

Código:

```
input(type='checkbox', checked)
input(type='checkbox', checked=true)
input(type='checkbox', checked=false)
input(type='checkbox', checked=true.toString())
```

Resultado:

```
<input type="checkbox" checked="checked"/>
<input type="checkbox" checked="checked"/>
<input type="checkbox"/>
<input type="checkbox" checked="true"/>
```

Si el doctype es `html` jade sabe que no debe reflejar el atributo y utiliza el estilo conciso (comprendido por todos los navegadores).

Código:

```
doctype html
input(type='checkbox', checked)
input(type='checkbox', checked=true)
input(type='checkbox', checked=false)
input(type='checkbox', checked=true && 'checked')
```

Resultado:

```
<!DOCTYPE html>
<input type="checkbox" checked>
<input type="checkbox" checked>
<input type="checkbox">
<input type="checkbox" checked="checked">
```

Atributos de estilo

El atributo de `style` puede ser una cadena (como cualquier atributo normal) pero también puede ser un objeto, lo cual es útil cuando JavaScript genera partes del estilo.

Código:

```
a(style={color: 'red', background: 'green'})
```

Resultado:

```
<a style="color:red;background:green"></a>
```

Atributos de clase

El atributo de `class` puede ser una cadena (como cualquier atributo normal) pero también puede ser una matriz de nombres de clase, que es útil cuando se genera desde JavaScript.

Código:

```
- var classes = ['foo', 'bar', 'baz']
a(class=classes)
//- the class attribute may also be repeated to merge arrays
a.bing(class=classes class=['bing'])
```

Resultado:

```
<a class="foo bar baz"></a><a class="bing foo bar baz bing"></a>
```

También puede ser un objeto que asigna nombres de clase a valores verdaderos o falsos, lo cual es útil para aplicar clases condicionales

Código:

```
- var currentUrl = '/about'
a(class={active: currentUrl === '/'} href='/') Home
a(class={active: currentUrl === '/about'} href='/about'
```

Resultado:

```
<a href="/">Home</a><a href="/about" class="active">About</a>
```

Clase literal

Las clases se pueden definir usando una sintaxis de `.classname` :

Código:

```
a.button
```

Resultado:

```
<a class="button"></a>
```

Dado que los div son una elección de etiqueta tan común, es la opción predeterminada si omite el nombre de la etiqueta:

Código:

```
.content
```

Resultado:

```
<div class="content"></div>
```

ID Literal

Los ID se pueden definir utilizando una sintaxis `#idname` :

Código:

```
a#main-link
```

Resultado:

```
<a id="main-link"></a>
```

Dado que los div son una elección de etiqueta tan común, es la opción predeterminada si omite el nombre de la etiqueta:

Código:

```
#content
```

Resultado:

```
<div id="content"></div>
```

y atributos

Se pronuncia "y atributos", la sintaxis de `&attributes` se puede utilizar para explotar un objeto en los atributos de un elemento.

Código:

```
div#foo(data-bar="foo")&attributes({'data-foo': 'bar'})
```

Resultado:

```
<div id="foo" data-bar="foo" data-foo="bar"></div>
```

El objeto no tiene que ser un objeto literal. También puede ser simplemente una variable que tenga un objeto como valor (consulte también Atributos de mezcla)

Código:

```
- var attributes = {'data-foo': 'bar'};
div#foo(data-bar="foo")&attributes(attributes)
```

Resultado:

```
<div id="foo" data-bar="foo" data-foo="bar"></div>
```

Lea Atributos en línea: https://riptutorial.com/es/jade/topic/6641/atributos

Capítulo 4: Caso

Examples

Caso

```
- var friends = 10
case friends
  when 0
    p you have no friends
  when 1
    p you have a friend
  default
    p you have #{friends} friends
```

El resultado es:

```
<p>you have 10 friends</p>
```

Lea Caso en línea: https://riptutorial.com/es/jade/topic/4012/caso

Creditos

S. No	Capítulos	Contributors
1	Empezando con jade	Community, David Dias, H. Pauwelyn, Huy Nguyen, Naeem Shaikh
2	arpa js	Alice Tribuleva, Mohit Bhardwaj
3	Atributos	Huy Nguyen
4	Caso	Huy Nguyen