EBook Gratis

APRENDIZAJE pug

Free unaffiliated eBook created from
Stack Overflow contributors.

#pug

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con Pug

Observaciones

Pug es un motor de plantillas de alto rendimiento, robusto, elegante y rico en funciones. Fue influenciado por Haml e implementado con JavaScript para Node.js y navegadores. Existen implementaciones para Laravel, PHP Scala, Ruby, Python y Java.

Cuenta con:

- Integración expresa
- Condicionales
- Filtros
- Incluye
- Herencia
- Interpolación
- Iteración
- Mixins

Pug era conocido anteriormente con el nombre de *Jade,* pero se le cambió el nombre debido a un caso de marca registrada.

Esta sección de comentarios también debe mencionar cualquier tema grande dentro de pug, y vincular a los temas relacionados. Dado que la Documentación para pug es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Versiones

Versión	Fecha de lanzamiento
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

Instalación

Para instalar el sistema de representación de plantillas Pug, siga estos pasos:

1. Tener el entorno Node.js instalado en su máquina
2. Ejecute `npm install pug --save` para instalar el módulo `pug` en su proyecto actual.

Ahora puede usar `pug` en su proyecto a través del mecanismo estándar `require` :

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

Si está utilizando Express en su aplicación, no necesita `require("pug")` . Sin embargo, debe establecer la propiedad del `view engine` de su aplicación Express en `pug` .

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

Además, debe configurar el directorio de vista de su aplicación para que Express sepa dónde buscar sus archivos Pug (para compilación).

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

Dentro de su ruta Express, puede procesar sus archivos Pug llamando a la función `res.render` con la ruta del archivo (comenzando desde el directorio establecido por la `app.set("views")`).

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

En lo anterior, el `index` apunta a un archivo ubicado en `views/index.pug` , y los `locals` representan un objeto de variables que están expuestas a su archivo. Como se explicará en secciones posteriores, Pug puede acceder a las variables que se le pasan y realizar una variedad de acciones (condicionales, interpolación, iteración y más).

Hola mundo ejemplo

Primero, vamos a crear una plantilla para ser renderizada!

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

`.pug` en un archivo que termine con la extensión `.pug` (puede llamarlo como desee, pero usaremos `view.pug` en el siguiente código para compilarlo).

¡Todo lo que queda por hacer ahora es compilar esa plantilla! Cree un archivo de script JS (llamaremos a nuestro `main.js`), y agregue el siguiente contenido:

```
// 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' }));
```

Cuando ejecute este archivo con `npm main.js` , debería obtener el siguiente código HTML en su consola:

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

¡Felicidades, acabas de crear y compilar tu primera plantilla! ¡A cosas más avanzadas, como Condicionales , Iteración y mucho más!

Lea Empezando con Pug en línea: https://riptutorial.com/es/pug/topic/8613/empezando-con-pug

Capítulo 2: Condicionales

Introducción

Pug puede ejecutar condicionalmente código basado en variables (pasadas desde su servidor o basadas en Pug).

Sintaxis

- si (declaración)

```
// Pug code
```

- de lo contrario si (declaración)

```
// Pug code
```

- más

```
// Pug code
```

- a menos que (declaración)

```
// Pug code
```

Parámetros

Parámetro	Detalles
si (declaración)	Evalúa la `statement` para ver si devuelve verdadero o falso. El código anidado debajo `if` se ejecutará solo si la `statement` devuelve verdadero.
de lo contrario si (declaración)	Encadenado a una declaración existente `if` o `else if` ; solo se ejecuta si la sentencia anterior fue evaluada como falsa. El código anidado debajo de la sentencia `else if` solo se ejecutará si la `statement` evalúa como verdadera.
más	El código anidado debajo de la instrucción `else` solo se ejecutará si todas las declaraciones anteriores devolvieron falso.
a menos que (declaración)	La negación de `if (statement)` ; el código anidado debajo `if` se ejecutará solo si la `statement` devuelve falso. Es lo mismo que `if (!statement)` .

Observaciones

Documentación oficial de PugJS sobre condicionales.

Examples

Si / otra declaración en Pug

Los condicionales en Pug pueden evaluar declaraciones de una manera similar a JavaScript. Puede evaluar las variables creadas en Pug o las que le haya pasado su ruta (`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
```

salida de index.pug

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

Declaración If / Else en Pug (con un guión)

Puede elegir anteponer un operador `if` o `else` con un guión, pero no es necesario. Sin embargo, deberá ajustar la declaración entre paréntesis (si omite un guión, no necesita paréntesis).

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

salida de index.pug

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

Otra cosa si declaración

Puede encadenar cualquier número de sentencias `else if` a una sentencia `if` existente, para evaluar una secuencia de sentencias.

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
```

salida de index.pug

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

A menos que el operador

`unless` sea la operación inversa de `if` en Pug. Es análogo a `if !(statement)` .

index.pug

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

salida de index.pug

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

Nota : `else unless` declaraciones no funcionen a `unless` ; puede encadenar una sentencia `else if` en una sentencia a `unless` que `else unless` , pero a `else unless` que no funcione.

Lea Condicionales en línea: https://riptutorial.com/es/pug/topic/9662/condicionales

Capítulo 3: Generación de sintaxis y marcado.

Introducción

Una vista previa de la diferencia entre el código pug y el marcado generado

Observaciones

Pug hace posible escribir HTML de la forma más sencilla, utilizando una sintaxis limpia y sensible al espacio en blanco.

Examples

De Pug a 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.
```

Se convierte en

```
<!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>
```

Lea Generación de sintaxis y marcado. en línea: https://riptutorial.com/es/pug/topic/9549/generacion-de-sintaxis-y-marcado-

Capítulo 4: Interpolación con Pug

Introducción

Es importante poder utilizar las variables del lado del servidor en su sitio web. Pug le permite interpolar los datos generados por su servidor en HTML, CSS e incluso en el código JavaScript.

Sintaxis

- res.render (ruta, variables) // Busca un archivo pug para representarse en la ruta "ruta", y le pasa "variables"
- # {variable} // Interpola "variable" en línea con el código de Jade circundante, después de evaluar "variable"
- ! {variable} // Interpola "variable" en línea con el código de Jade circundante, sin evaluar "variable".
- # [elemento] // Interpola "elemento" dentro de un elemento HTML Pug existente. La sintaxis de los elementos HTML interpolados es idéntica a la de los elementos HTML normales.

Parámetros

Parámetro	Detalles
camino	Utilizado en `res.render` . Esta es la ruta del archivo Pug que vamos a renderizar. La ruta se toma de la raíz de la carpeta establecida en su aplicación Express: `app.set("views", "templates/views")` . Por ejemplo, `res.render("index")` buscará un archivo Pug en `templates/views/index.pug` . Los subdirectorios también se pueden especificar; `res.render("admin/index")` busca un archivo Pug en `templates/views/admin/index.pug` .
variables	Utilizado en `res.render` . Un objeto de JavaScript de variables que se hará accesible al archivo Pug definido por la `path` (arriba). Dentro del archivo Pug, las claves del objeto JavaScript anterior están disponibles como variables. Si `variables = {title: "Hello", color: "red"}` , podríamos usar el `title` y `color` variable de `color` . También están disponibles subpropiedades de objetos anidados.
variable	Se utiliza en la sintaxis de corchete `#{}` o `!{}` . El valor de la `variable` se emitirá en el contexto del código Pug que lo rodea. Si un símbolo de libra se añade al corchete de apertura, la `variable` se evaluará antes de emitirse. Si se añade un signo de exclamación a la llave de apertura, la `variable` **no** se evaluará.
elemento	Utilizado en el corchete de la sintaxis `#[]` . El elemento HTML (en la sintaxis de Pug, no en la sintaxis de HTML normal) se evaluará y emitirá en línea con el código Pug que lo rodea.

Observaciones

Para obtener más información sobre la interpolación de PugJS, consulte la documentación oficial de interpolación de PugJS.

Examples

Interpolación de variables del lado del servidor

Es posible pasar variables de su servidor a Pug para el contenido dinámico o la generación de scripts. Las plantillas de Pug pueden acceder a las variables pasadas a la función `res.render` en Express (o `pug.renderFile` si no está usando Express, los argumentos son idénticos).

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);
```

Dentro de su archivo `index.pug` , tendrá acceso a la variable `locals` través de sus claves. Los nombres de las variables en su archivo Pug se convierten en `siteColors` y `siteNames` .

Para establecer la totalidad de un elemento HTML igual a una variable, use el operador igual `=` para hacerlo. Si su variable necesita estar incrustada en línea, use la sintaxis de corchete `#{}` para hacerlo.

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
```

salida de 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>
```

Interpolación de variables sin procesar en HTML

El contenido interpolado con la sintaxis de corchete se evaluará para el código, cuya salida se incluye en su salida HTML.

> el título sigue el patrón básico para evaluar una plantilla local, pero el código entre `#{` y `}` se evalúa, se escapa y el resultado se almacena en la salida de la plantilla que se representa. [Fuente]

Si necesita incluir sintaxis HTML en bruto, use un signo de exclamación en lugar de un símbolo de libra (`!{}` lugar 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}
```

salida de index.pug

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

Interpolación de valor en el código JavaScript

La interpolación de valores es útil si necesita pasar una variable del lado del servidor a JavaScript

del lado del cliente (u otros idiomas que lo requieran).

En el caso de variables, números, cadenas y similares, puede pasar estos tipos de variables directamente a su JavaScript con la sintaxis de corchete más un punto de explicación (para que *no se* evalúe el código entre corchetes). Esto es útil para parametrizar Código JavaScript que requiere algo de su servidor.

En el siguiente ejemplo, tenemos que ajustar el `username` de `username` entre comillas para que JavaScript lo interprete como una cadena; Pug emitirá el contenido de la variable como está, por lo que debemos ponerla entre comillas para que sea una cadena de JavaScript adecuada. Esto no es necesario para el `number` , donde JavaScript interpretará nuestro número como lo intentamos (como un número).

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!
```

salida de 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 necesita interpolar el valor de un objeto de JavaScript (por ejemplo, toda la información sobre un usuario), debe establecer una cadena de caracteres en el Pug para que se trate como un objeto de JavaScript. También es necesario generar el contenido sin procesar de la variable, en lugar de la forma evaluada. Si `var user = #{JSON.stringify(user)}` que generar la variable escapada (`var user = #{JSON.stringify(user)}`), recibirá una versión *escapada* del objeto (donde las comillas y los apóstrofes se convierten a `&quot;`), que no es lo que queremos que `JSON.stringify` trabaje en ello.

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")
```

salida de 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>
```

El `innerHTML` de `#welcome-user` vuelve igual a `Leeroy Jenkins` . El contenido de la variable de `user` se imprime directamente en la fuente HTML

Interpolación de elementos HTML

Puede ser necesario anidar etiquetas HTML una dentro de la otra. La interpolación de elementos se realiza en una sintaxis similar a la interpolación de variables; aquí se utilizan corchetes en lugar de llaves. La sintaxis de los elementos HTML interpolados es idéntica a la implementación de los elementos HTML normales.

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.
```

salida de 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>
```

Lea Interpolación con Pug en línea: https://riptutorial.com/es/pug/topic/9565/interpolacion-con-pug

Capítulo 5: Iteración con Pug

Introducción

Cómo iterar sobre un objeto JSON simple y ahorrar mucha escritura

Observaciones

Necesitas tener Node.js y Pug instalados

Examples

Cada iteración

Construye un `app.js` con un simple *almacén de datos* :

```
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
    });
});
```

Iterar sobre el almacén de datos usando un archivo `index.pug` y un bucle de cada uno:

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

El resultado será:

```
<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>
```

Referencia

Lea Iteración con Pug en línea: https://riptutorial.com/es/pug/topic/9545/iteracion-con-pug

Creditos

S. No	Capítulos	Contributors
1	Empezando con Pug	Community, gandreadis, Shea Belsky, smonff
2	Condicionales	Shea Belsky
3	Generación de sintaxis y marcado.	smonff
4	Interpolación con Pug	Shea Belsky
5	Iteración con Pug	Shea Belsky, smonff