EBook Gratis

APRENDIZAJE haxe

Free unaffiliated eBook created from
Stack Overflow contributors.

#haxe

Tabla de contenido

Acerca de

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

It is an unofficial and free haxe 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 haxe.

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 haxe

Observaciones

Haxe es un conjunto de herramientas de código abierto que es capaz de compilar a muchos lenguajes y plataformas de destino diferentes.

Consiste en:

- El lenguaje de programación Haxe : un lenguaje de programación moderno, de alto nivel y tipificado estrictamente.
- La biblioteca estándar de Haxe : una colección de API de propósito general, sistema y específicas de destino
- el compilador Haxe : un compilador cruzado rápido y optimizado con soporte de metadatos, eliminación de código muerto (DCE), modo de finalización, integración de recursos, información de tipo de tiempo de ejecución (RTTI), analizador estático, macros y más

Haxe se ha utilizado para crear juegos, aplicaciones web, móviles, de escritorio y de línea de comandos, así como API multiplataforma.

A partir de Haxe 3.3.0-rc.1, Haxe puede compilar en fuentes / códigos de bytes de los siguientes lenguajes: ActionScript 3, C #, C ++, Flash, HL, Lua, Java, JavaScript, Neko, PHP y Python.

Haxe tiene un administrador de paquetes, Haxelib , que se incluye con Haxe. También tiene un formato de archivo de compilación personalizado, `.hxml` , que ofrece una forma más sencilla de pasar los argumentos pasados al compilador Haxe.

Referencias

- Documentación Haxe

Examples

Instalación

Haxe está disponible en Windows, Linux y OS X. Se distribuye en dos formas:

- como **instalador** , proporciona una dependencia opcional de Neko VM y configura variables de entorno `haxe` y `haxelib` ;
- como **binarios** , proporcionando solo el compilador Haxe y el administrador de paquetes.

Windows

El instalador y los binarios están disponibles en el sitio web de Haxe .

Linux

Los binarios (32 bits y 64 bits) están disponibles en el sitio web de Haxe .

La Fundación Haxe también participa oficialmente en el mantenimiento de los paquetes Haxe y Neko para las distribuciones populares de Linux . Se recomienda usar esos paquetes si están disponibles.

Ubuntu

Se recomienda utilizar el PPA de Haxe, que proporciona las últimas versiones de Haxe y Neko para todas las versiones de Ubuntu compatibles actualmente. El PPA también se puede utilizar para distribuciones basadas en Ubuntu.

```
sudo add-apt-repository ppa:haxe/releases -y
sudo apt-get update
sudo apt-get install haxe -y
mkdir ~/haxelib && haxelib setup ~/haxelib
```

Tenga en cuenta que Neko se instala como una dependencia de Haxe.

Debian

Para instalar las versiones estables actualmente disponibles, ejecute los siguientes comandos:

```
sudo apt-get install haxe -y
mkdir ~/haxelib && haxelib setup ~/haxelib
```

Tenga en cuenta que Neko se instalará como una dependencia de Haxe.

Para instalar nuevas versiones desde el canal inestable, haga lo siguiente:

1. En `/etc/apt/sources.list` , agregue

   ```
   deb http://httpredir.debian.org/debian unstable main contrib non-free
   ```

2. En `/etc/apt/preferences.d/` , cree un nuevo archivo llamado `unstable` con el siguiente contenido:

   ```
   Package: *
   Pin: release a=unstable
   Pin-Priority: 100

   Package: haxe neko libneko*
   Pin: release a=unstable
   Pin-Priority: 999
   ```

3. Extraiga los archivos de índice del paquete de la fuente recién agregada:

   ```
   sudo apt-get update
   ```

4. Instalar Haxe (y Neko):

```
sudo apt-get install haxe -y
```

Fedora

La Fundación Haxe mantiene los paquetes de RPM Haxe y Neko en el repositorio de Fedora. Los paquetes están actualizados la mayor parte del tiempo. Sin embargo, cuando se lance una nueva versión de Haxe, tomará unos días, hasta 2 semanas, enviar un paquete actualizado a las versiones estables de Fedora. Las actividades de actualización se pueden rastrear en el Sistema de actualización de Bodhi Fedora .

Para instalar las versiones actualmente disponibles de Haxe y Neko, ejecute los siguientes comandos:

```
sudo dnf install haxe -y
mkdir ~/haxelib && haxelib setup ~/haxelib
```

Tenga en cuenta que Neko se instala como una dependencia de Haxe.

openSuse

La Fundación Haxe mantiene los paquetes de RPM Haxe y Neko en el repositorio de fábrica de openSUSE. Los paquetes están actualizados la mayor parte del tiempo. Sin embargo, cuando se lance una nueva versión de Haxe, demorará algunos días, hasta 2 semanas, en ser aceptado por openSUSE: Factory.

Para instalar las versiones actualmente disponibles de Haxe y Neko, ejecute los siguientes comandos:

```
sudo zypper install haxe
mkdir ~/haxelib && haxelib setup ~/haxelib
```

Tenga en cuenta que Neko se instala como una dependencia de Haxe.

Para obtener la última versión de Haxe que no esté disponible para openSUSE: Factory o una versión de openSUSE, use el proyecto devel: languages: haxe en el servicio de creación de openSUSE. Visite la página del paquete Haxe , haga clic en "Descargar paquete" en la esquina superior derecha y siga las instrucciones. Nuevamente, Neko también se instalará como una dependencia de Haxe.

Arco de linux

Hay paquetes Haxe y Neko en el repositorio de la comunidad de Arch Linux. La Fundación Haxe continuará ayudando a mantener los paquetes actualizados. Sin embargo, cuando se lance una nueva versión de Haxe, tomará tiempo actualizar el paquete, dependiendo de la disponibilidad del mantenedor del paquete.

Para las versiones actualmente disponibles de Haxe y Neko, consulte las siguientes páginas:

- Haxe en Arch Linux
- Neko en Arch Linux

Para instalar las versiones actualmente disponibles de Haxe y Neko, ejecute los siguientes comandos:

```
sudo pacman -S haxe
mkdir ~/haxelib && haxelib setup ~/haxelib
```

Tenga en cuenta que Neko se instala como una dependencia de Haxe.

OS X

El instalador y los binarios están disponibles en el sitio web de Haxe .

También es posible instalar la versión estable actual de Haxe a través del gestor de paquetes Brew .

```
brew install haxe
```

Referencias

- "Descargas", sitio web de Haxe
- "Paquetes de software de Linux", sitio web de Haxe

Hola Mundo

Requerimientos

1. Se debe instalar una versión del kit de herramientas Haxe.
2. Haxe debe estar presente en su ruta del sistema
3. La línea de comando debe ser accesible

Código

Navegue a un directorio de proyecto deseado y cree un archivo fuente `Test.hx` con el siguiente contenido:

```
class Test {
    static function main() {
        trace("Hello world");
    }
}
```

Los archivos fuente de Haxe se llaman **módulos** . Un módulo *debe* definir un tipo (`abstract` , `class` , `enum` , `interface` o `typedef`) con el mismo identificador que el nombre del módulo; en este

caso, la clase `Test` . Una vez que se cumple ese requisito, un módulo puede definir un número arbitrario de diferentes tipos.

Los programas Haxe requieren un **punto de entrada** , como se indica en la función `main` estática. La clase que implementa el punto de entrada es la **clase de inicio** o la clase principal. De nuevo, en este caso, la clase principal es la clase `Test` .

La función `trace()` es una función de registro de propósito general expuesta al espacio de nombres global por conveniencia. Da salida al identificador de salida estándar del idioma de destino (por ejemplo, consola del navegador para JavaScript, línea de comandos para C ++). Consulte la documentación de la API para obtener más información.

Ejecución

Navegue a la carpeta del proyecto desde su línea de comando. Pruebe a ver si Haxe está configurado en su entorno llamando al:

```
haxe --help
```

El intérprete de Haxe se puede usar para probar el código que no se basa en ninguna API de idioma de destino específico. Utilice el intérprete llamando al

```
haxe -main Test --interp
```

Recuerde , el módulo de `Test` contiene la clase de inicio de `Test` , por lo que se pasa la `-main Test` al compilador.

Las fuentes de Haxe pueden compilar (*transpilar*) a fuentes / *códigos de bytes* de varios idiomas diferentes. La siguiente tabla muestra el idioma de destino, el indicador del compilador, el tipo de argumento y el resultado de la compilación. Úsalo llamando

```
haxe -main Test [flag] [argument] .
```

Idioma	Bandera	Argumento	Resultado
ActionScript 3	-as3	Directorio	Fuente
DO#	-cs	Directorio	Fuente + bytecode opcional (.exe)
C ++	-cpp	Directorio	Fuente + binario opcional (nativo)
Destello	-swf	Expediente	Bytecode (.swf)
HL	-hl	Expediente	Fuente
Lua	-lua	Expediente	Fuente
Java	-Java	Directorio	Fuente + bytecode opcional (.jar)
JavaScript	-js	Expediente	Fuente

Idioma	Bandera	Argumento	Resultado
Neko	-neko	Expediente	Bytecode (.n)
PHP	-php	Directorio	Fuente
Pitón	-pitón	Expediente	Fuente
HashLink	-hl	Expediente	Bytecode (.hl)

Tenga en cuenta que los argumentos de ruta aquí son relativos a la ruta `haxe` se llamó. Las salidas de bytecode / binary opcionales se pueden excluir agregando los indicadores `-D no-compilation` , para evitar un paso de compilación adicional que implique llamar al compilador del idioma de destino.

Referencias

- Documentación API para `haxe.Log`
- Entrada "Hola mundo" en el libro de cocina de código Haxe

Lea Empezando con haxe en línea: https://riptutorial.com/es/haxe/topic/2593/empezando-con-haxe

Capítulo 2: Bucles

Sintaxis

- para (*identificador de variable* en *colección iterativa*) { *expresión* }
- while (*condición*) { *expresión* }
- do { *expresión* } while (*condición*);
- descanso;
- continuar;

Examples

por

Los bucles For se repiten sobre una **colección iterativa** . Una colección iterativa es cualquier clase que se unifica estructuralmente con los tipos `Iterator<T>` o `Iterable<T>` de la biblioteca estándar de Haxe.

Un for-loop que registra números en el rango de 0 a 10 (exclusivo) se puede escribir de la siguiente manera:

```
for (i in 0...10) {
    trace(i);
}
```

El identificador de la variable `i` contiene el valor individual de los elementos en la colección iterativa. Este comportamiento es similar a para cada uno en otros idiomas.

Por lo tanto, un bucle for que registra elementos en una matriz se puede escribir de la siguiente manera:

```
for (char in ['a', 'b', 'c', 'd']) {
    trace(char);
}
```

Pruebe el ejemplo en try.haxe.org .

Referencias

- "Para", manual de Haxe.
- "Iteradores", manual Haxe

Mientras

While-loops ejecuta una expresión de cuerpo siempre que la condición de bucle se evalúe como `true`

.

Un bucle while que registra números en el rango 9 a 0 (inclusive) se puede escribir de la siguiente manera:

```
var i = 10;
while (i-- > 0) {
    trace(i);
}
```

Pruebe el ejemplo en try.haxe.org .

Referencias

- "While", manual de Haxe.

Do-while

Do-while-loops ejecuta una expresión del cuerpo al menos una vez, y luego continúa ejecutándola mientras la condición del loop se evalúe como `true` .

Un bucle do-while que registra números en el rango de 10 a 0 (inclusive) se puede escribir de la siguiente manera:

```
var i = 10;
do {
    trace(i);
} while (i-- > 0);
```

Pruebe el ejemplo en try.haxe.org .

Referencias

- "Do-while", manual de Haxe.

Control de flujo

El flujo o la ejecución de un bucle se puede controlar mediante el uso de las expresiones `break` y `continue` .

Descanso

`break` sale del loop actual. En caso de que el bucle esté anidado dentro de otro bucle, el bucle primario no se verá afectado.

```
for (i in 0...10) {
    for (j in 0...10) {
        if (j == 5) break;
```

```
        trace(i, j);
    }
}
```

Pruebe el ejemplo en try.haxe.org .

Continuar

`continue` omite la iteración actual del bucle en el punto de la expresión. En caso de que el bucle esté anidado dentro de otro bucle, el bucle primario no se verá afectado.

```
for (i in 0...10) {
    for (j in 0...10) {
        if (j == 5) continue;
        trace(i, j);
    }
}
```

Pruebe el ejemplo en try.haxe.org .

Referencias

- "Break", manual Haxe
- "Continuar", manual de Haxe.

Lea Bucles en línea: https://riptutorial.com/es/haxe/topic/4409/bucles

Capítulo 3: Derivación

Sintaxis

- si (*condición*) {...}
- if (*condición*) {...} else {...}
- if (*condición*) {...} else if (*condición*) {...} else {...}
- // Las llaves son opcionales para las declaraciones de una sola línea
  if (*condición*) ... else if (*condición*) ... else ...
- switch (*expresión*) { *patrón de* caso: ... predeterminado: ...}
- *condición* ? *expresión si es verdadera* : *expresión si es falsa* ;

Observaciones

Todas las expresiones de ramificación permiten devolver las expresiones evaluadas. Esto significa que los resultados de la bifurcación se pueden asignar a las variables. En este caso, **todas las expresiones que pueden evaluarse mediante una prueba de condición exitosa deben pasar la unificación de tipo** . Si no se da ninguna `else` expresión, se infiere que el tipo es `Void` .

Examples

Si / else if / else

```
if (a > b) {
    trace("You win!");
} else if (a == b) {
    trace("It's a draw!");
} else {
    trace("You lose!");
}

// Assigning the evaluated expression to a variable
var message =  if (a > b) {
    "You win!";
} else if (a == b) {
    "It's a draw!";
} else {
    "You lose!";
}
trace(message);
```

Referencia

- "Si", manual de Haxe.

Operador ternario

```
n % 2 == 0 ? trace("n is even!") : trace("n is odd!");

// Assigning the evaluated expression to a variable
var message = n % 2 == 0 ? "n is even!" : "n is odd!";
trace(message);
```

Referencia

- "Si", manual de Haxe.

Cambiar

```
switch (n % 2) {
    case 0: trace("n is even!");
    case 1: trace("n is odd!");
    default: trace("I don't know!");
}

// Assigning the evaluated expression to a variable
var message = switch (n % 2) {
    case 0: "n is even!";
    case 1: "n is odd!";
    default: "I don't know!";
}
trace(message);
```

Tenga en cuenta que **las expresiones del cuerpo del `case` nunca se cumplen** , **por** lo que el uso de la expresión de `break` en este contexto no es compatible con Haxe.

Referencia:

- "Switch", manual de Haxe

Lea Derivación en línea: https://riptutorial.com/es/haxe/topic/6265/derivacion

Capítulo 4: Enums

Sintaxis

- *identificador de* enumeración { *constructores* }

Examples

Visión general

Los tipos de enumeración de Haxe son **tipos de datos algebraicos** (ADT). Su uso principal es para describir estructuras de datos. Las enumeraciones se indican mediante la palabra clave `enum` y contienen uno o más **constructores de enumeración** .

```
enum Color {
    Red;
    Green;
    Blue;
    RGB(r : Int, g : Int, b : Int);
}
```

La enumeración anterior se puede instanciar de la siguiente manera:

```
var c1 = Color.Red;
var c2 = Color.RGB(255, 0, 0);
```

Pruebe el ejemplo en try.haxe.org .

Referencias

- "Enum instancia", manual de Haxe

Capturar valores enum

Los valores que se pasan como argumentos de constructor enum pueden capturarse en variables mediante el uso de **la coincidencia** de **patrones** .

Supongamos la siguiente enumeración:

```
enum Color {
    RGB(r : Int, g : Int, b : Int);
    HSV(h : Int, s : Float, v : Float);
}
```

El valor del canal rojo se puede capturar de la siguiente manera:

```
var color = Color.RGB(255, 127, 0);
var red = switch (color) {
    // Match the Color.RGB constructor and capture value into `r`
    case Color.RGB(r, _, _):
        // Return the captured red value
        r;
    // Catch-all for matching remaining constructors
    case _:
        // Return -1
        -1;
}
```

Pruebe el ejemplo en try.haxe.org .

Referencias

- "Coincidencia de patrones", manual Haxe
- "Captura de variables", manual de Haxe

Coincidencia de constructores de enumeración

Los constructores de enumeración pueden emparejarse utilizando **la coincidencia de patrones** .

Supongamos la siguiente enumeración:

```
enum Color {
    Red;
    Green;
    Blue;
    RGB(r : Int, g : Int, b : Int);
}
```

Los colores con solo un valor de canal verde se pueden combinar de la siguiente manera:

```
var color = Color.RGB(0, 127, 0);
var isGreenOnly = switch (color) {
    // Match Green or RGB with red and blue values at 0
    case Color.RGB(0, _, 0) | Color.Green: true;
    case _: false;
}
```

Pruebe el ejemplo en try.haxe.org .

Referencias

- "Coincidencia de patrones", manual Haxe
- "Enum de coincidencia", manual de Haxe
- "O patrones", manual de Haxe.

Lea Enums en línea: https://riptutorial.com/es/haxe/topic/4667/enums

Capítulo 5: La coincidencia de patrones

Observaciones

La coincidencia de patrones es el proceso de ramificación en función de los patrones proporcionados. Toda la coincidencia de patrones se realiza dentro de una expresión de `switch` , y las expresiones de `case` individuales representan los patrones.

Las reglas fundamentales de la coincidencia de patrones son:

- los patrones siempre se combinarán de arriba a abajo;
- el patrón superior que coincide con el valor de entrada tiene su expresión ejecutada;
- un patrón _ coincide con cualquier cosa, por lo que el `case _:` es igual al `default:`

Cuando se manejan todos los casos posibles, todas incidental de _ patrón o `default` no se requiere el caso.

Examples

Enumeración coincidente

Supongamos la siguiente enumeración:

```
enum Operation {
    Multiply(left : Int, right : Int);
}
```

La asignación de enumeración se puede realizar de la siguiente manera:

```
var result = switch(Multiply(1, 3)) {
    case Multiply(_, 0):
        0;
    case Multiply(0, _):
        0;
    case Multiply(l, r):
        l * r;
}
```

Referencias

- "Enum de coincidencia", manual de Haxe

Coincidencia de estructuras

Supongamos la siguiente estructura:

```
var dog = {
    name : "Woofer",
    age : 7
};
```

La asignación de enumeración se puede realizar de la siguiente manera:

```
var message = switch(dog) {
    case { name : "Woofer" }:
        "I know you, Woofer!";
    case _:
        "I don't know you, sorry!";
}
```

Referencias

- "Coincidencia de estructuras", manual Haxe

Juego de matrices

```
var result = switch([1, 6]) {
    case [2, _]:
        "0";
    case [_, 6]:
        "1";
    case []:
        "2";
    case [_, _, _]:
        "3";
    case _:
        "4";
}
```

Referencias

- "Array Matching", manual de Haxe.

O patrones

El | El operador puede usarse en cualquier lugar dentro de los patrones para describir múltiples patrones aceptados. Si hay una variable capturada en un o-patrón, debe aparecer en ambos sub-patrones.

```
var match = switch(7) {
    case 4 | 1: "0";
    case 6 | 7: "1";
    case _: "2";
}
```

Referencias

- "O patrones", manual de Haxe.

Guardias

También es posible restringir aún más los patrones con guardias. Estos están definidos por el `case ... if(condition):` sintaxis.

```
var myArray = [7, 6];
var s = switch(myArray) {
    case [a, b] if (b > a):
        b + ">" +a;
    case [a, b]:
        b + "<=" +a;
    case _: "found something else";
}
```

Referencias

- "Guardias", manual de Haxe.

Extractores

Los extractores se identifican por `extractorExpression => match` expresión `extractorExpression => match` . Los extractores constan de dos partes, que están separadas por el operador `=>` .

1. El lado izquierdo puede ser cualquier expresión, donde todas las apariciones del guión bajo _ se reemplazan con el valor coincidente actual.
2. El lado derecho es un patrón que se compara con el resultado de la evaluación del lado izquierdo.

Como el lado derecho es un patrón, puede contener otro extractor. El siguiente ejemplo "encadena" dos extractores:

```
static public function main() {
    switch(3) {
        case add(_, 1) => mul(_, 3) => a:
            trace(a); // mul(add(3 + 1), 3)
    }
}

static function add(i1:Int, i2:Int) {
    return i1 + i2;
}

static function mul(i1:Int, i2:Int) {
    return i1 * i2;
}
```

Actualmente no es posible utilizar extractores dentro de o-patrones. Sin embargo, es posible tener o-patrones en el lado derecho de un extractor.

Referencias

- "Extractores", manual Haxe

Lea La coincidencia de patrones en línea: https://riptutorial.com/es/haxe/topic/6436/la-coincidencia-de-patrones

Capítulo 6: Resúmenes

Sintaxis

- *identificador* abstracto (*tipo de subrayado*) {...}
- *identificador* abstracto (*tipo subyacente*) de *typeA* from *typeB* ... to *typeA* to *typeB* {...}

Observaciones

Un **tipo abstracto** es un tipo de *tiempo de compilación* que se resuelve al **tipo subyacente** en *tiempo de ejecución* . Esto significa que su tipo abstracto *no existe* en el código fuente generado por el compilador Haxe. En su lugar se colocan el tipo subyacente o los tipos definidos para la conversión implícita.

Los resúmenes se indican mediante la palabra clave `abstract` , seguida de un identificador y el tipo subyacente entre paréntesis.

Los resúmenes solo pueden definir campos de método y campos de propiedad no físicos. Los campos de método no incluidos se declaran como funciones estáticas en una **clase de implementación** privada, aceptando como un primer argumento adicional el tipo subyacente del resumen.

Tenga en cuenta que la sobrecarga del operador solo es posible para los tipos abstractos.

Examples

Resúmenes para la validación de datos.

El siguiente resumen define un `EmailAddress` tipo basado en la `String` tipo que va a utilizar una expresión regular para validar el argumento pasado como una dirección de correo electrónico. Si la dirección no es válida, se lanzará una excepción.

```
abstract EmailAddress(String) {
  static var ereg = ~/^[\w-\.]{2,}@[\w-\.]{2,}\.[a-z]{2,6}$/i;

  inline public function new(address:String) {
    if (!ereg.match(address)) throw "EmailAddress "$address" is invalid";
    this = address.toLowerCase();
  }
}
```

Utilice el resumen de la siguiente manera.

```
var emailGood = new EmailAddress("john@doe.com");
var emailBad = new EmailAddress("john.doe.com");
```

Pruebe el ejemplo en try.haxe.org .

Referencias

- "EmailAddress", libro de cocina de código Haxe

Sobrecarga del operador

La sobrecarga del operador solo es posible con tipos abstractos.

El siguiente resumen define un tipo `Vec2i` basado en el tipo `Array<Int>` . Este es un vector de dos componentes con valores enteros. La sobrecarga del operador es posible en los **metadatos del compilador** `@:op` . Solo se pueden sobrecargar los operadores numéricos disponibles; no se permite especificar operadores personalizados.

```
abstract Vec2i(Array<Int>) {
    public inline function getX() : Int {
        return this[0];
    }

    public inline function getY() : Int {
        return this[1];
    }

    public inline function new(x : Int, y : Int) {
        this = [x, y];
    }

    @:op(A + B)
    public inline function add(B : Vec2i) : Vec2i {
        return new Vec2i(
            getX() + B.getX(),
            getY() + B.getY()
        );
    }
}
```

Utilice el resumen de la siguiente manera.

```
var v1 = new Vec2i(1, 2);
var v2 = new Vec2i(3, 4);
v1 + v2;
v1.add(v2);
```

Pruebe el ejemplo en try.haxe.org .

Referencias

- "EmailAddress", libro de cocina de código Haxe

Lea Resúmenes en línea: https://riptutorial.com/es/haxe/topic/4162/resumenes

Creditos

S. No	Capítulos	Contributors
1	Empezando con haxe	5Mixer, ali_o_kan, Andy Li, Community, Domagoj, KevinResoL, YsenGrimm
2	Bucles	Domagoj
3	Derivación	Domagoj, Kev, Mark Knol
4	Enums	Domagoj
5	La coincidencia de patrones	Domagoj
6	Resúmenes	Domagoj