
HAXE

# Haxe

- Это **не** Hack :)
- Haxe is an open source toolkit based on a modern, high level, strictly typed programming language, a cross-compiler, a complete cross-platform standard library and ways to access each platform's native capabilities

# Haxe

- Open source
- Высокоуровневый
- Со строгой типизацией
- Кросс-компилятор
- Кросс-платформенная стандартная библиотека
- Для каждой платформы есть способ использовать её возможности напрямую

# Платформы

- Flash, сразу в байткод, минуя ActionScript *(2005)*
- Neko VM, сразу в байткод (*2005)*
- JavaScript *(2006)*
- ActionScript 3 *(2007)*
- PHP *(2008)*
- C++ *(2009)*
- Java *(2012)*
- C# *(2012)*
- Python *(2015)*
- Lua *(2016)*
- PHP7 *(2016)*
- HashLink VM *(2016)*
- C (via HashLink) *(2016)*

# Кто использует Haxe

- BBC
- Disney
- Zynga
- Prezi
- Coca Cola
- Toyota

- **1995** — PHP 1.x
- **1998** — PHP 3.x
- **1999** — ES3
- **2000** — C# 1.x, PHP 4.x
- …
- **2004** — PHP 5.x
- **2005** — C# 2.x, **Haxe 1.x**
- …
- **2008** — C# 3.x, **Haxe 2.x** — *generics (только классы), anonymous functions, iterators, properties, ADT (algebraic data type), pattern matching, static extensions, anonymous structures, automatic type inference, macros, DCE*

- **2009** — Node.js, CofeeScript, ES5
- **2010** — C# 4.x, Rust
- **2011** — Dart
- **2012** — TypeScript, Elm, C# 5.x
- …
- **2015** — ES6, C# 6.x, PHP 7.x, **Haxe 3.x** — *generics (функции), array accessors, array comprehension, abstract types*
- **2016** — ES7
- **2017** — Kotlin, C# 7.x

# Для чего используют

- Игры (особенно кросс-платформенные, когда надо зашарить кодовую базу между Web / Mobile / Desktop)
- Web (можно было шарить кодовую базу между сервером и клиентом, ещё до того, как Node.js стал модным)
- Мобильные приложения
- Десктопные приложения
- CLI-приложения
- Кросс-платформенные библиотеки

# Зачем был написан Haxe

- Единый язык (это был 2005 год)
- Было:
  - PHP или Java для server-side
  - JS для dynamic html
  - ActionScript для графической части (на Flash)
- Хотели чтоб стало:
  - Haxe для всего

# Nicolas Cannasse / The Essential Guide to Open Source Flash Development

- Создать язык более мощный, чем ActionScript 2 или ActionScript 3
- Сделать так, чтобы было легко портировать приложения с ActionScript на Haxe
- Сделать язык, поддерживающий разработку для Flash 6, 7, 8 и 9
- Сделать язык, поддерживающий разработку для JavaScript / AJAX
- Сделать язык, поддерживающий разработку серверно-ориентированных программ, вместо PHP или Java

# О компиляторе

- Написан на OCaml
- Один frontend, множество backend
- Frontend - parsing, AST, type checking, macro, general optimizations, DCE
- Backend - имея на руках AST надо выдать код для целевой платформы (исходный код, байт код, может в будущем и машинный код)
- Есть режим сервера, для поддержки автодополнения кода в IDE (так же в этом режиме поддерживается кэш для для уменьшения времени компиляции)

# Компилятор — оптимизирующий

- Подстановка функций (инлайнинг)
- Свёртка констант
- Удаление мёртвого кода (DCE)
- С Haxe 3.3 — static analyzer (дополнительные compile-time оптимизации)

# Производительность (vs написанный руками код для целевой платформы)

- Flash — байткод сгенерированный Haxe более оптимизированный (и работает быстрей), чем байткод сгенерированный Action Script Compiler из Flex SDK либо из Flash
- JS — сравнимо с нативным кодом
- C++ — немного медленней, но тоже сравнимо с нативным кодом (только надо учитывать, что в Haxe есть GC)

# Рандомный кусок кода

```
var prev = clamp(last - 1);
var next = clamp(last + 1);
var newList : Array<Int> = Random.shuffle([prev, next]);

for (i in index ... (index + list)) {
    var sector = list[i % list];

    if (sector != last
        && sector != prev
        && sector != next
    ) {
        newList.push(sector);
    }
}
```

# JavaScript

```
var prev = (this.last - 1 + this.conf.sectors) % this.conf.sectors;
var next = (this.last + 1 + this.conf.sectors) % this.conf.sectors;
var newList = Random.shuffle([prev, next]);
var _g1 = this.index;
var _g = this.index + this.list.length;

while (_g1 < _g) {
    var i = _g1++;
    var sector = this.list[i % this.list.length];

    if (sector != this.last
        && sector != prev
        && sector != next
    ) {
        newList.push(sector);
    }
}
```

# C++

```
int prev = hx::Mod(((this->last - (int)1) + ((int)(this->conf-
>__Field(HX_("sectors", 0d, 96, dc, 5d), hx::paccDynamic)))), ((int)(this->conf-
>__Field(HX_("sectors", 0d, 96, dc, 5d), hx::paccDynamic))));
int next = hx::Mod(((this->last + (int)1) + ((int)(this->conf-
>__Field(HX_("sectors", 0d, 96, dc, 5d), hx::paccDynamic)))), ((int)(this->conf-
>__Field(HX_("sectors", 0d, 96, dc, 5d), hx::paccDynamic))));
::Array<int> newList = ::Random_obj::shuffle(::Array_obj<int>::__new(2)->init(0,
prev)->init(1, next));
{
    int _g1 = this->index; int _g = (this->index + this->list->length);
    while ((_g1 < _g)) {
        _g1 = (_g1 + (int)1); int i = (_g1 - (int)1);
        ::Array<int> sector = this->list; int sector1 = sector->__get(hx::Mod(i,
this->list->length));
        bool _hx_tmp1; bool _hx_tmp2;
        if ((sector1 != this->last)) { _hx_tmp2 = (sector1 != prev); } else
{ _hx_tmp2 = false; }
        if (_hx_tmp2) { _hx_tmp1 = (sector1 != next); } else { _hx_tmp1 = false; }
        if (_hx_tmp1) { newList->push(sector1); }
    }
}
```

# Кратко про язык

- Только про интересные особенности
- Если что, то мануал тут — http://haxe.org/manual/introduction.html
- Классы, интерфейсы, наследование, имплементация, проперти, дженерики и т.д. — всё как везде
- Языки похожие по внешнему виду — Action Script, Type Script

```
class B extend A implements C {
    public static inline var SOME_CONST = 42;
    public var someVar : Int = 0;
    public var propDefGetterNoSetter(default, null) : String = "42";
    public var propGetterSetter(get, set) : Int;

    public function new(someVar : Int) {
        this.someVar = someVar;
    }

    private function get_propGetterSetter() : Int {
        return someVar;
    }

    private function set_propGetterSetter(value : Int) : Int {
        someVar = value;
        return value;
    }
}
```

# В интерфейсе можно объявлять не только методы, но и переменные и свойства

```
interface Placeable {
    public var x : Float;
    public var y : Float;
}
```

# Всё — это выражение

- `var a = if (b > c) { d } else { e }`
- `var a = switch (b) { … }`

# Вывод типов

```
function foo() {
    return 42;
}

// тоже самое что и

function foo() : Int {
    return 42;
}
```

# Тип вычисляется не в момент декларации, а в момент использования

```
function foo() { return null; }
// тип будет Void -> Unknown<0>

var s : String = foo();
// тип foo() будет Void -> String

// var i : Int = foo();
// будет ошибка компиляции
```

# Enum / ADT

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

var color1 = Color.Red;
var color2 = Color.Rgb(50, 50, 50);
```

# Enum / ADT

```
switch (color) {
     case Red:
          trace("Red");
     case Green:
          trace("Green");
     case Blue:
          trace("Blue");
     case Rgb(r, g, b):
          trace('Rgb(${r},${g},${b})');
}
```

# Anonymous structure

```
var point = { x : 0.0, y : 12.0 };

// if (point.z > 0.0) { … }
// ошибка компиляции
```

```
// point.z = 42.0;
// тоже ошибка компиляции
```

# Typedef

```
typedef UserName = String;

typedef Point = {x:Float, y:Float };
```

# Abstract type

- В Нахе это несёт несколько иное значение, чем в других языках
- Абстрактный тип — это новый тип, на основе некоторого уже существующего

```
abstract BigDecimal(String) {
    inline function new(value : String) { this =
value; }

    @:from public static function fromInt(value : Int) {
return new BigDecimal(Std.string(value)); }

    @:op(-A) public static inline function neg(value :
BigDecimal) : BigDecimal { … }

    @:op(A + B) public static inline function add(a :
BigDecimal, b : BigDecimal ) : BigDecimal { … }

    @:arrayAccess public inline function arrayRead(k :
Int) { return this.charAt(k); }
}
```

# Abstract enum

```
@:enum
abstract TutorialStep(Int) {
    var None = 0;
    var MsgReachOppositeRow = 1;
    var TryMove = 2;
    var TryHorizontalWall = 3;
    var TryVerticalWall = 4;
    var MsgCantBlock = 5;
}
```

# Structural subtyping

```
typedef Iterator<T> = {
    function hasNext() : Bool;
    function next() : T;
}

function isEmpty<T>(it : Iterator<T>) {
    return !it.hasNext();
}
```

# Array comprehension

```
var a = [for (i in 0...10) i];
// [0,1,2,3,4,5,6,7,8,9]

var i = 0;
var b = [while(i < 10) i++];
// [0,1,2,3,4,5,6,7,8,9]
```

# Array comprehension

```
var a = [
    for (a in 1 ... 11)
        for (b in 2 ... 4)
            if (a % b == 0)
                a + "/" + b
];

// [2/2,3/3,4/2,6/2,6/3,8/2,9/3,10/2]
```

# Static extension

```
class StringExt {
    public static function dup(value : String,
count : Int) : String {
        var sb = new StringBuf();
        for (i in 0 ... count) { sb.add(value); }
        return sb.toString();
    }
}

using StringExt;
trace("Abc".dup(3));

// "AbcAbcAbc"
```

# Pattern matching (Basic)

```
enum Tree<T> {
    Leaf(v : T);
    Node(l : Tree<T>, r : Tree<T>);
}

switch (tree) {
    case Leaf(_): "0";
    case Node(_, Leaf(_)): "1";
    case Node(_, Node(Leaf("bar"), _)): "2";
    case _: "3"; // same as "default"
}
```

# Pattern matching (Variable capture)

```
switch (node) {
    case Leaf(s): s;
    case Node(Leaf(s), _): s;
    case _: "none";
}

switch (node) {
    case Node(leafNode = Leaf("foo"), _):
        leafNode;
    case x: x;
}
```

# Pattern matching (Structure)

```
switch (struct) {
    case { name: "haxe", rating: "poor" }:
        throw false;

    case { rating: "awesome", name: n }:
        n;

    case _:
        "no awesome language found";
}
```

# Pattern matching (Array)

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

# Pattern matching ("Or" patterns)

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

# Pattern matching (Guards)

```
switch (arr) {
    case [a, b] if (b > a):
        b + ">" + a;

    case [a, b]:
        b + "<=" + a;

    case _:
        "found something else";
}
```

# Patter matching (Extractors)

```
var e = TString(“fOo");

var success = switch(e) {
    case TString(_.toLowerCase() =>
"foo"): true;

    case _: false;
}
```

# Macros

- В compile-time можно менять AST программы
- Похоже на annotation processing в Java, только гибче

# Macros

```
macro static function dup(e : Expr) {
    return {
        expr: EBinop(OpAdd, e, e),
        pos: e.pos
    };
}

var x = 0; trace(dup(x++));

// преобразуется в
var x = 0; trace((x++) + (x++));
```

# Macro reification

```
macro static function dup(e : Expr) {
    return macro $e + $e;
}

// результат будет аналогичен
// предыдущему слайду
```

# Примеры

- **continuation**, **async** — async / await на макросах
- **HxSL** — шейдеры, пишутся на DSL, затем компилируются либо в Flash/AGAL, либо в GLSL
- **polygonal-printf** — printf-style форматирование, код генерируется в compile-time
- **json2object** — json-парсер генерится в compile-time под нужный объект, без рефлексии (значит что DCE не будет убирать ничего лишнего)
- Подстановка номера билда в код

# Haxe в настоящем

- Игры - Web / iOS / Android / Windows / macOS / Linux / etc
- Кросс-платформенные библиотеки
- В существующих проектах на любых языках, код для которых может генерировать Haxe