Scala.js for JavaScript developers

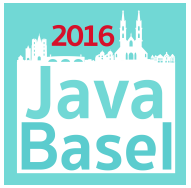
*"I can honestly say if someone had shown me the Programming in Scala book by by Martin Odersky I'd probably have never created Groovy." by **James Strachan**.*

*"If I were to pick a language to use today other than Java, it would be Scala" by **James Gosling***

Vitaly Tsaplin @ Adobe

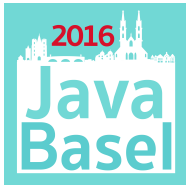
Agenda

- Scala.js? Are you serious?
- Scala fundamentals
- JavaScript interoperability
- Demo project
- QA



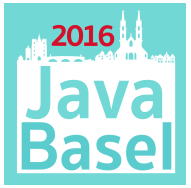
What is Scala.js?

JavaScript: Easy to prototype



- Easy to learn
- Runs virtually everywhere
- No need to compile anything
- Zoo of frameworks

...but hard to scale



- Hard to reason about the code
- Refactoring is deadly
- Linting is not the same as static analysis
- Requires more tests
- Code completion is slow and unreliable

Breaking the language barrier

- Virtual Machine written in JavaScript
  - DoppioJVM
- Source-to-source compiler
  - TypeScript
  - Google Web Toolkit
  - Scala.js (our hero)

Why Scala.js?

- Mature, extensible and concise language
- Has many advanced features
- Object-oriented and functional
- Typesafe
- Code completion
- Refactoring
- Excellent tooling
- Interoperability with JavaScript
- Shared code

Scala.js facts

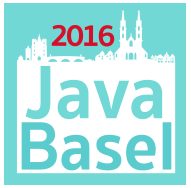
- Incremental compilation is fast (1-2s)
- The generated code size is starting 45kB gzipped
- Existing Scala libraries can be leveraged

Show me the code

```
// ES6
var xhr = new XMLHttpRequest();
xhr.open("GET",
      "https://api.twitter.com/1.1/search" +
      "tweets.json?q=%23scalajs");
xhr.onload = (e) => {
      if (xhr.status === 200) {
            var r = JSON.parse(xhr.responseText);
            $("#tweets").html(parseTweets(r));
      }
}
```

Show me the code

```
// Scala
val xhr = new XMLHttpRequest()
xhr.open("GET",
      "https://api.twitter.com/1.1/search" +
      "tweets.json?q=%23scalajs");
xhr.onload = { (e: Event) =>
      if (xhr.status == 200) {
            val r = JSON.parse(xhr.responseText)
            $("#tweets").html(parseTweets(r))
      }
}
```



Scala fundamentals

Variables

```
// ES6
let x = 5;
const y = "Constant";

// Scala
var x = 5
val y = "Constant"
```

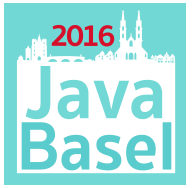
Primitive types

Scala	JavaScript
String	string
Boolean	boolean
Int, Short, Byte	number
Double, Float	number
Unit	undefined
Null	null

Calculations

```
// ES6
const x = 5 / 3;// == 1.6666666666666667

// Scala
val x = 5 / 3   // == 1
val y = 5.0 / 3 // == 1.6666666666666667
val z = 5 / 3.0 // == 1.6666666666666667
```

Type conversions




```
val x: Double = 3        // Ok!
val y: Int = 3.5         // Compile error
val z: Int = 3.5.toInt   // Ok!
val a: Int = x           // Compile error
val b: Int = x.toInt     // Ok!
```

String interpolation

```
// ES6
var name = "James";
console.log(`Hello, ${name}`);

// Scala
val name = "James"
println(s"Hello, $name")

val age = 25
val name = "James"
println(f"$name%s is $age%d")
```

Functions

```
// ES6
function mult(x, y = 42.0) {
      return x * y;
}

// Scala
def mult(x: Double, y: Double = 42.0): Double =
      x * y
```

Anonymous functions

```
// ES6
const f = (x, y) => x + y;

// Scala
val f = (x: Double, y: Double) => x + y

val p = Array("Fox", "jumped", "over", "me")
val l = p.map(s => s.length).sum
```

Named parameters

```
// ES6
function vec({x = 0, y = 0}) {
     return new Vec(x, y);
}
const v = vec({x: 8});

// Scala
def vec(x: Int = 0, y: Int = 0): Vec = {
     new Vec(x, y)
}
val v = vec(y = 42)
```

Rest parameters

```
// ES6
function sum(...args) {
      return args.reduce((a, b) => a + b, 0);
}

// Scala
def sum(args: Double*): Int =
      args.foldLeft(0)((a, b) => a + b)
```

If-Else expression

```
// ES6
if (name === "") {
      return 0;
} else {
      return 1;
}
const res = (name === "") ? 0 : 1;

// Scala
val res = if (name == "") 0 else 1
```

For loop

```
// ES6
let x = 0;
for (let i = 0; i < 100; i++)
     x += i * i;

// Scala
var x = 0
for (i <- 0 until 100)
     x += i * i
```

For loop

```
// ES6
const p = ["Fox", "jumped", "over", "me"];
for (let s of p) {
     console.log(`Word ${s}`);
}

// Scala
val p = Array("Fox", "jumped", "over", "me")
for (s <- p) {
     println(s"Word $s")
}
```

Pattern matching

```
// ES6
const animal = "Dog";
let description;

switch (animal) {
      case "Cat":
            description = "It's feline!";
            break;
      case "Dog":
      case "Wolf":
            description = "It's canine!";
            break;
      default:
            description = "It's something else";
}
console.log(description);
```

Pattern matching

```
// Scala
val animal = "Dog"
val description = animal match {
      case "Cat =>
            "It's feline!"
      case "Dog" | "Wolf" =>
            "It's canine!"
      case _ => "It's something else"
}

println(description)
```

Classes

```
// ES6
class Circle extends Shape {
      constructor(x, y, r) {
            super(x, y);
            this.r = r;
      }
      draw() {
      }
}
const c = new Circle(5, 5, 42);
```

Classes

```
// Scala
class Circle(x: Int, y: Int, val r: Int)
          extends Shape(x, y) {
     override def draw(): Unit = {
     }
}
val c = new Circle(5, 5, 42)
```

Case classes

```
// ES6
const p1= {first: "James", last: "Bond"};
const p2 = Object.assign({}, p1, {first: "John"});


// Scala
case class Person(first: String, last: String)
val p1 = Person("James", "Bond")
val p2 = p1.copy(first = "John")
```

Objects

```
// ES6
const RandomGen = {
      _rnd() {
            return Math.random();
      },
      getRandomNumber() {
            return this._rnd();
      }
};
const r = RandomGen.getRandomNumber();
```

Objects

```
// Scala
object RandomGen {
     private val rnd = new Random()
     def getRandomNumber: Double =
          rnd.nextDouble()
}
val r = RandomGen.getRandomNumber
```

Traits

```
// ES6
class Circle extends Shape {
        constructor(x, y, r) {
                super(x, y);
                this.r = r;
        }
};
const Clickable = {
        onClick() {
                console.log("Clicked!");
        }
};
class ClickableCircle extends Circle {}

Object.assign(ClickableCircle.prototype, Clickable);
const cc = new ClickableCircle(0, 0, 42);
cc.onClick();
```

Traits

```
// Scala
class Circle(x: Int, y: Int, val r: Int)
            extends Shape(x, y)
trait Clickable {
      def onClick(): Unit = {
            println("Clicked!")
      }
}
class ClickableCircle(x: Int, y: Int, r: Int)
            extends Circle(x, y, r) with Clickable
val cc = new ClickableCircle(0, 0, 42)
cc.onClick()
```

Option

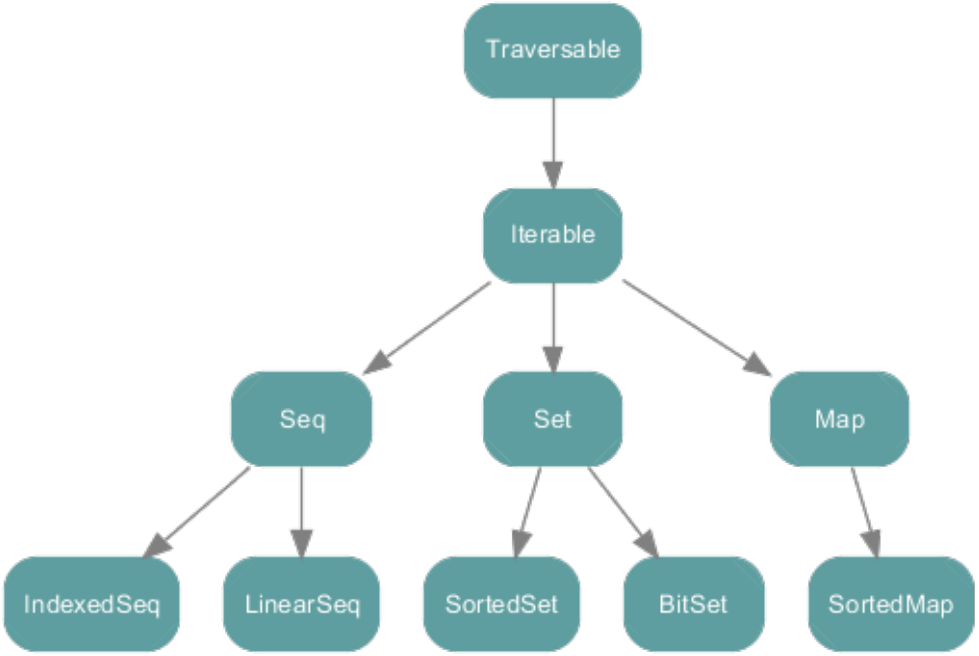
```
// ES6
function log(msg, context) {
      let s;
      if (context !== undefined)
            s = `[${context}] ${msg}`;
      else
            s = msg;
      console.log(s);
};
log("First message");
log("Second message", "debug");
```

Option

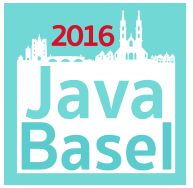
```
// Scala
def log(msg: String,
                context: Option[String] = None): Unit = {
        val s = context match {
                case Some(c) => s"[$c] $msg"
                case None => msg
        }
        println(s)
}
log("First message")
log("Second message", Some("debug"))
```

Collections

- Immutable and mutable implementations
- Scala always use immutable collections by default



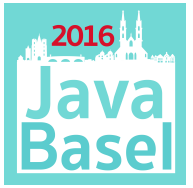
Implicit conversions



- Use conversion functions available in scope
- Safe alternative to monkey patching

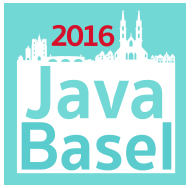
```
// ES6
String.prototype.toDate = function() {
       return convertToDate(this);
}
"2015-10-09".toDate();

// Scala
implicit class StrToDate(val s: String) {
       def toDate = convertToDate(s)
}
"2015-10-09".toDate
```

Future and Promise



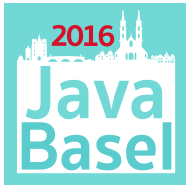
- Future is a read-only placeholder object
- Promise is a writable, single-assignment container

Future and Promise




```
// ES6
function onLoadPromise(img) {
    if (img.complete) {
        return Promise.resolve(img.src);
    } else {
        const p = new Promise((success) => {
            img.onload = (e) => {
                success(img.src);
            };
        });
        return p;
    }
}
const img = document.querySelector("#mapimage");
onLoadPromise(img).then(url =>
    console.log(`Image ${url} loaded`)
);
```

Future and Promise

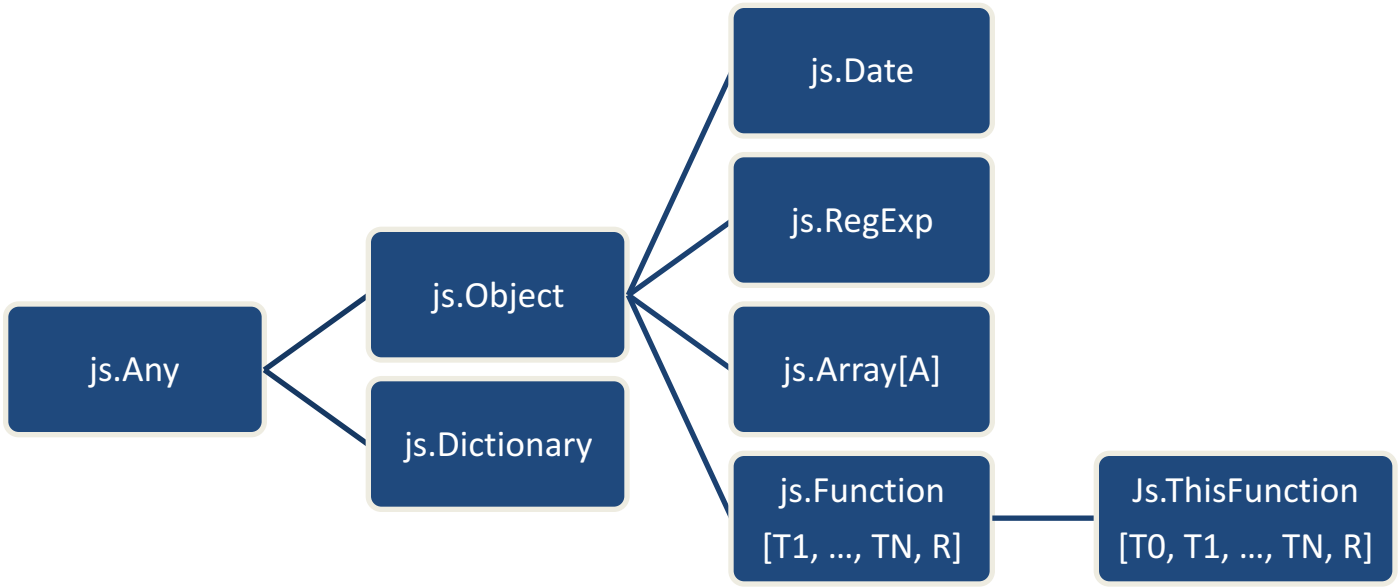



```
// Scala
def onLoadFuture(img: HTMLImageElement) = {
    if (img.complete) {
        Future.successful(img.src)
    } else {
        val p = Promise[String]()
        img.onload = { (e: Event) =>
            p.success(img.src)
        }
        p.future
    }
}
val img = dom.document.querySelector("#mapimage")
        .asInstanceOf[HTMLImageElement]
onLoadFuture(img).foreach {
    url => println(s"Image $url loaded")
}
```

JavaScript interoperability

Pre-defined JavaScript types

- Primitive types are represented by natural equivalents
- Other types have dedicated definitions



Function types

- *js.FunctionN[T1, ..., TN, R]*

```
// Scala
val f: js.Function1[Double, Double] =
        { (x: Double) => x*x }

// ES6
var f = function(x) {
    return x*x;
};
```

Function types

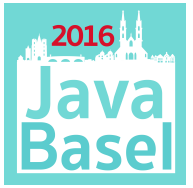
- *js.ThisFunctionN[T0, T1, ..., TN, R]*

```
// Scala
val f: js.ThisFunction1[js.Object,
      js.Number, js.Number] = ???
val o = new js.Object
val x = f(o, 4)
```

Type correspondence

- Primitive types are mapped directly
- Function types
  - *js.FunctionN <–> scala.FunctionN*
- Collection types
  - *js.Array[T] <–> mutable.Seq[T]*
  - *js.Dictionary[T] <–> mutable.Map[String, T]*

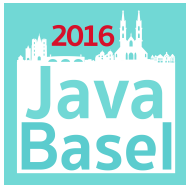
Dynamically typed interface



- Allows to read and write any property
- Provides an entry point to the global scope

```
val document = js.Dynamic.global.document
val parent = document.getElementById("parent")
val p = document.createElement("p")
p.innerHTML = "Hello world!"
parent.appendChild(p)

js.Dynamic.literal(foo = 42, bar = "foobar")
js.Dynamic.literal("foo" -> 42, "bar" -> "foobar")
```

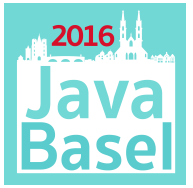
Facade types for JavaScript APIs



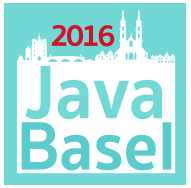
- Inherit directly or indirectly from *js.Any*
- All concrete definitions must have *js.native* as body

```
trait Window extends js.Object {
    val document: HTMLDocument = js.native
    var location: String = js.native
    def innerWidth: Int = js.native
    def innerHeight: Int = js.native
    def alert(message: String): Unit = js.native
}
```

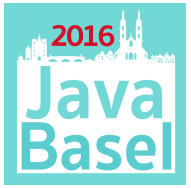
Exporting Scala.js APIs




```
package example

import scala.scalajs.js;
import js.annotation.JSExport;

@JSExport object HelloWorld {
    @JSExport def main(): Unit = {
        println("Hello world!")
    }
}

HelloWorld().main();
```



Demo project



QA