ScalaJS

czyli web development na nowo

przykładowy projekt:
`https://github.com/ghik/scalajs-demo`

https://github.com/AVSystem/Wyklady_2016

ScalaJS - co to?

- kompilator Scala -> JavaScript
- narzędzia i ekosystem:
  - linker/optymalizator
  - plugin SBT
  - wsparcie IDE
  - biblioteki w Scali (w tym standardowa)
  - statyczne API dla bibliotek natywnych (JS)
  - część standardowej biblioteki Javy
    - brakuje: API Javy 8, refleksja

Kompilator

phase name	id	description
parser	**1**	**parse source into ASTs, perform simple desugaring**
namer	**2**	**resolve names, attach symbols to named trees**
packageobjects	**3**	**load package objects**
typer	**4**	**the meat and potatoes: type the trees**
patmat	5	translate match expressions
superaccessors	6	add super accessors in traits and nested classes
extmethods	7	add extension methods for inline classes
pickler	8	serialize symbol tables
refchecks	9	reference/override checking, translate nested objects
uncurry	10	uncurry, translate function values to anonymous classes
tailcalls	11	replace tail calls by jumps
specialize	12	@specialized-driven class and method specialization
explicitouter	13	this refs to outer pointers
erasure	14	erase types, add interfaces for traits
posterasure	15	clean up erased inline classes
lazyvals	16	allocate bitmaps, translate lazy vals into lazifi
lambdalift	17	move nested functions to top level
constructors	18	move field definitions into constructors
flatten	19	eliminate inner classes
mixin	20	mixin composition
cleanup	21	platform-specific cleanups, generate reflective c
delambdafy	22	remove lambdas
icode	23	generate portable intermediate code
jvm	24	generate JVM bytecode
terminal	25	the last phase during a compilation run

frontend

backend

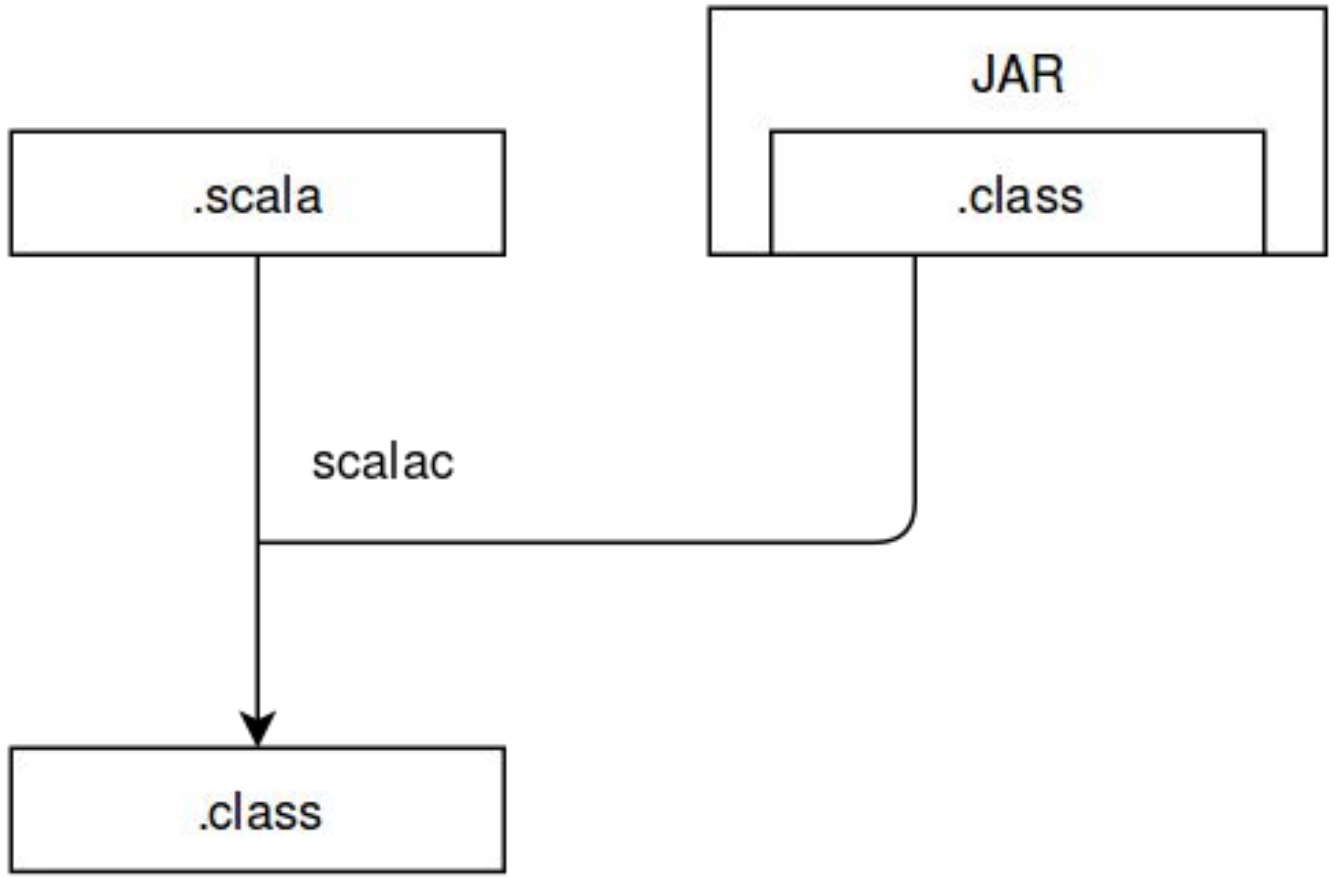
Kompilator + ScalaJS

phase name	id	description
parser	**1**	**parse source into ASTs, perform simple desugaring**
jspretyper	**2**	**capture pre-typer only tree info (for Scala.js)**
namer	**3**	**resolve names, attach symbols to named trees**
packageobjects	**4**	**load package objects**
typer	**5**	**the meat and potatoes: type the trees**
jsinterop	**6**	**prepare ASTs for JavaScript interop**
patmat	7	translate match expressions
superaccessors	8	add super accessors in traits and nested classes
extmethods	9	add extension methods for inline classes
pickler	10	serialize symbol tables
refchecks	11	reference/override checking, translate nested objects
uncurry	12	uncurry, translate function values to anonymous classes
tailcalls	13	replace tail calls by jumps
specialize	14	@specialized-driven class and method specialization
explicitouter	15	this refs to outer pointers
erasure	16	erase types, add interfaces for traits
posterasure	17	clean up erased inline classes
lazyvals	18	allocate bitmaps, translate lazy vals into lazifi
lambdalift	19	move nested functions to top level
constructors	20	move field definitions into constructors
flatten	21	eliminate inner classes
mixin	22	mixin composition
jscode	**23**	**generate JavaScript code from ASTs**
cleanup	24	platform-specific cleanups, generate reflective ca
delambdafy	25	remove lambdas
icode	26	generate portable intermediate code
jvm	27	generate JVM bytecode
terminal	28	the last phase during a compilation run

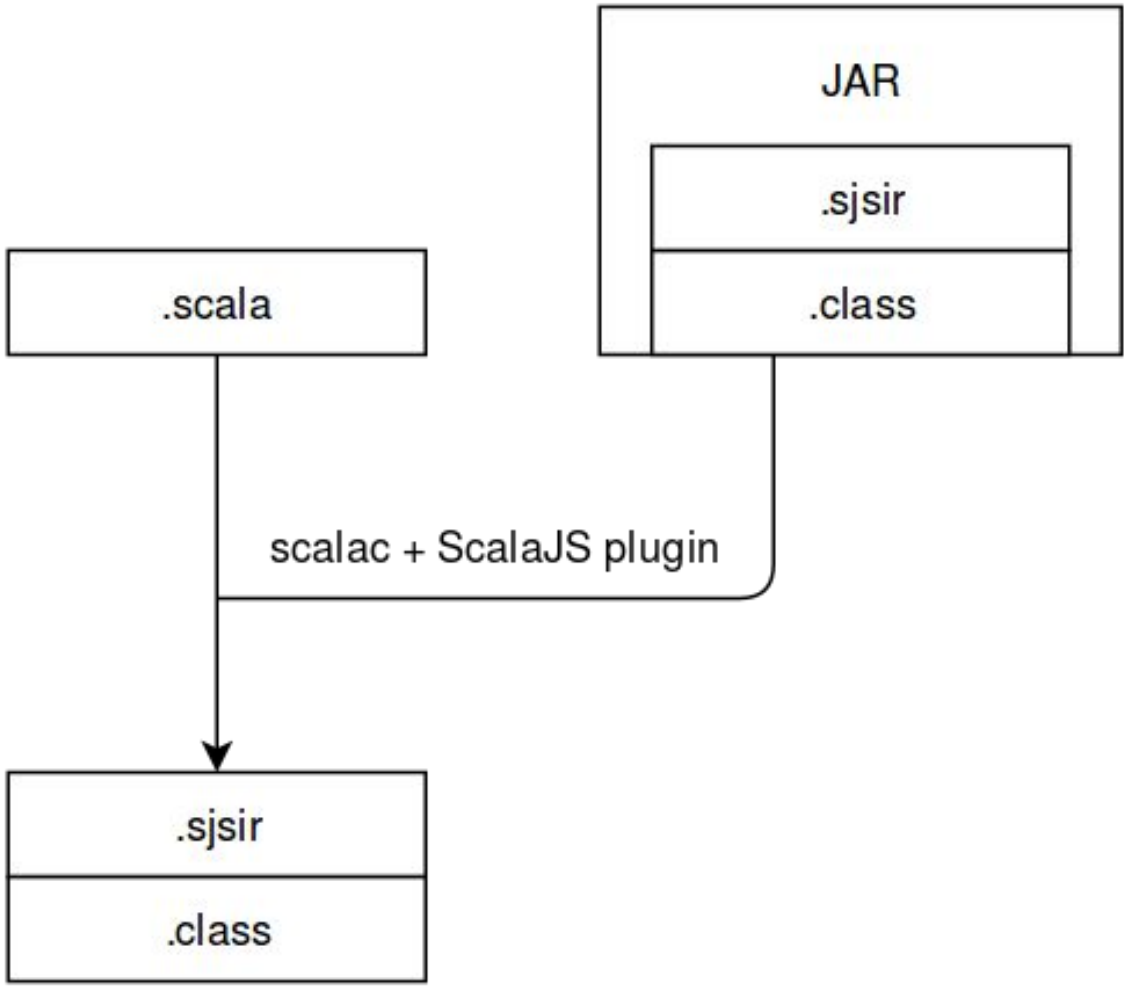
frontend

backend

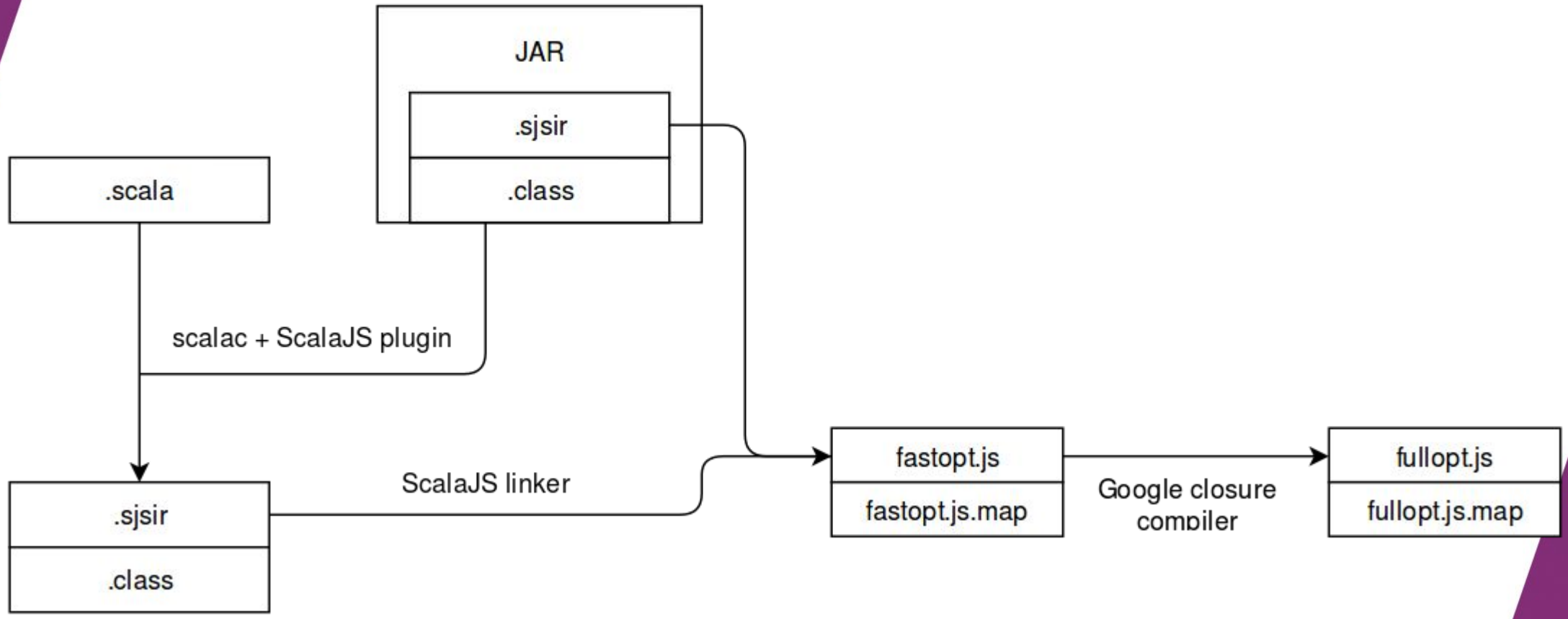
scalac



`scalac` + ScalaJS plugin



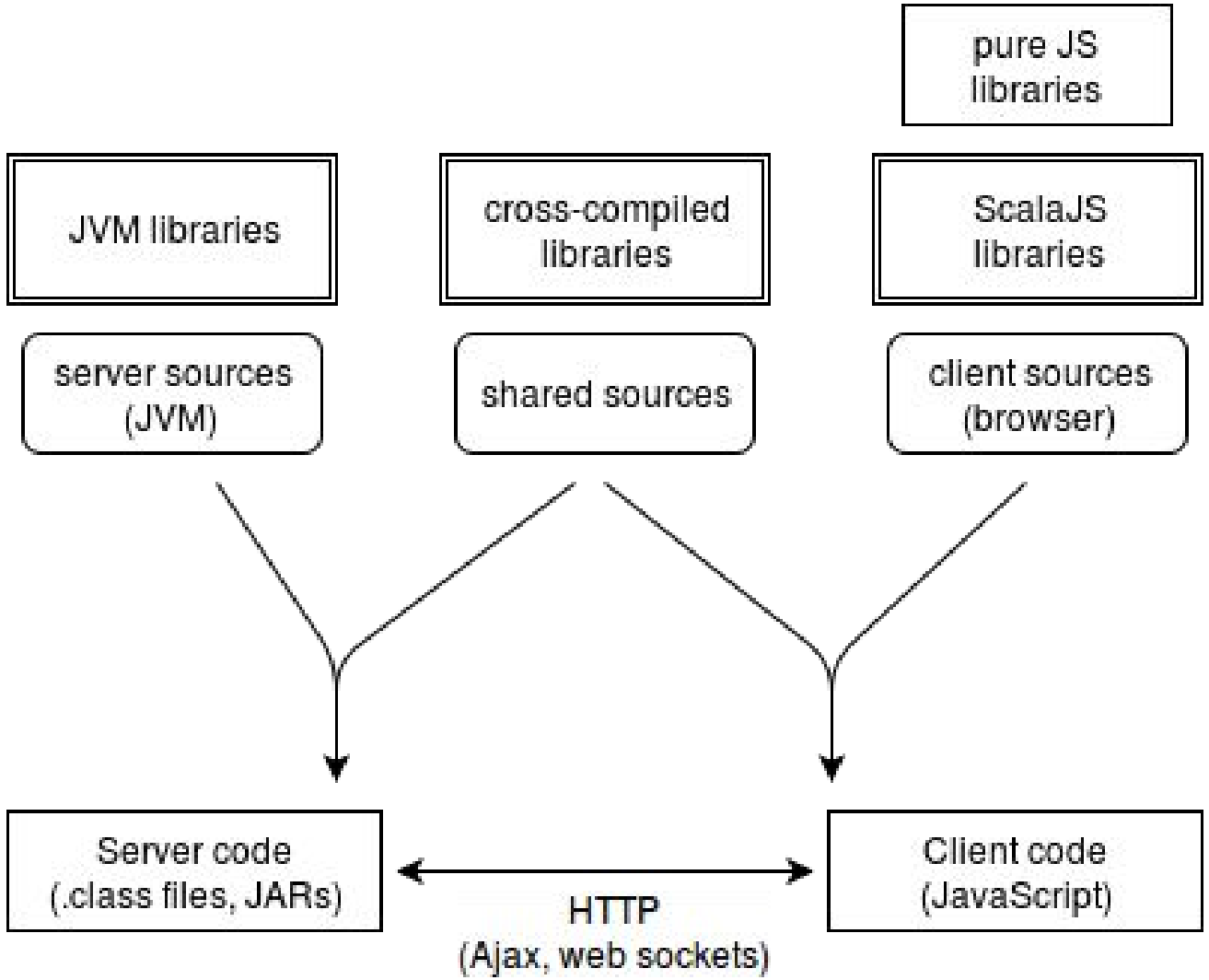
`scalac` + ScalaJS + linker



Struktura projektu

- Scala w back-endzie (JVM) i front-endzie (JS)
- współdzielenie kodu
- wsparcie SBT i IntelliJ IDEA

Struktura projektu



Rodzaje zależności w SBT

```
// Java library
libraryDependencies += "org.eclipse.jetty" % "jetty-server" % jettyVersion,

// Scala (JVM) library
libraryDependencies += "com.avsystem.commons" %% "commons-core" % avsCommonsVersion,

// ScalaJS or cross compiled library
libraryDependencies += "com.lihaoyi" %%% "upickle" % upickleVersion,

// JS native library
jsDependencies += "org.webjars" % "jquery" % jqueryVersion / "jquery.js" minified "jquery.min.js"
```

Scala <-> JavaScript interoperability

AVSYSTEM

- użycie bibliotek JavaScript z kodu Scali
- eksportowanie kodu napisanego w ScalaJS do użycia z JavaScriptu

Korzystanie z bibliotek natywnych

Przypadek: globalna funkcja napisana w JavaScripcie

```
<script type="text/javascript">

    function upperCaseThis(argument) {
        return argument.toUpperCase()
    }

</script>
```

Korzystanie z bibliotek natywnych

Przypadek: globalna funkcja napisana w JavaScripcie

```
function upperCaseThis(argument) {
    return argument.toUpperCase()
}
```

Rozwiązanie “siłowe” - surowy dostęp

```
js.Dynamic.global.upperCaseThis
  .asInstanceOf[js.Function1[String,String]]("something")
```

```
js.Dynamic.global.upperCaseThis("something").asInstanceOf[String]
```

scala.Dynamic

```scala
@native
sealed trait Dynamic extends Any with scala.Dynamic {
  /** Calls a method of this object. */
  @JSBracketCall
  def applyDynamic(name: String)(args: Any*): Dynamic = native

  /** Reads a field of this object. */
  @JSBracketAccess
  def selectDynamic(name: String): Dynamic = native

  /** Writes a field of this object. */
  @JSBracketAccess
  def updateDynamic(name: String)(value: Any): Unit = native

  /** Calls this object as a callable. */
  def apply(args: Any*): Dynamic = native
```

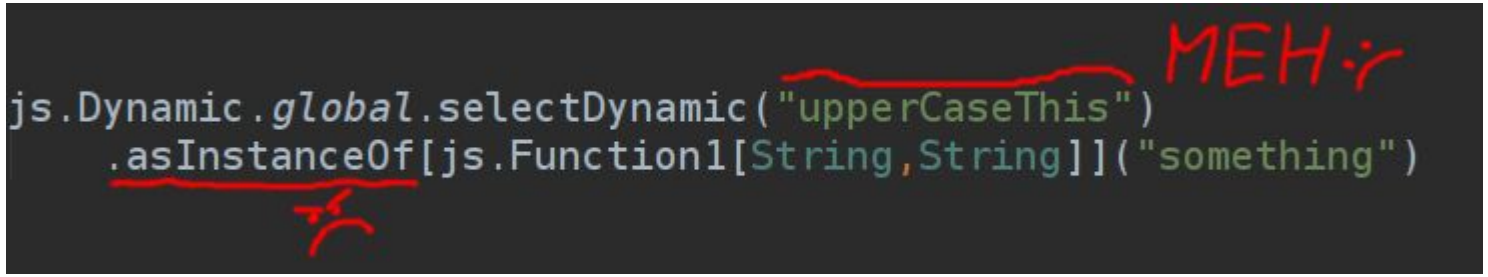
MAGIC

Korzystanie z bibliotek natywnych

Przypadek: globalna funkcja napisana w JavaScripcie

```
function upperCaseThis(argument) {
    return argument.toUpperCase()
}
```

Rozwiązanie “siłowe” - surowy dostęp



+ nie wymaga *glue code’u*

- brak statycznego typowania

Korzystanie z bibliotek natywnych

Przypadek: globalna funkcja napisana w JavaScripcie

```
function upperCaseThis(argument) {
    return argument.toUpperCase()
}
```

Rozwiązanie ładne - statycznie typowana fasada

```
@js.native
object Globals extends js.GlobalScope {
  def upperCaseThis(str: String): String = js.native
}

// użycie
Globals.upperCaseThis("something")
```

Korzystanie z bibliotek natywnych

Przypadek: klasa napisana w JavaScripcie

```
function Apple (type) {
    this.type = type;
    this.color = "red";
}
Apple.prototype.getInfo = function() {
    return this.color + ' ' + this.type + ' apple';
};
```

```
@js.native
class Apple(val `type`: String) extends js.Object {
  var color: String = js.native
  def getInfo: String = js.native
}
```

Korzystanie z bibliotek natywnych

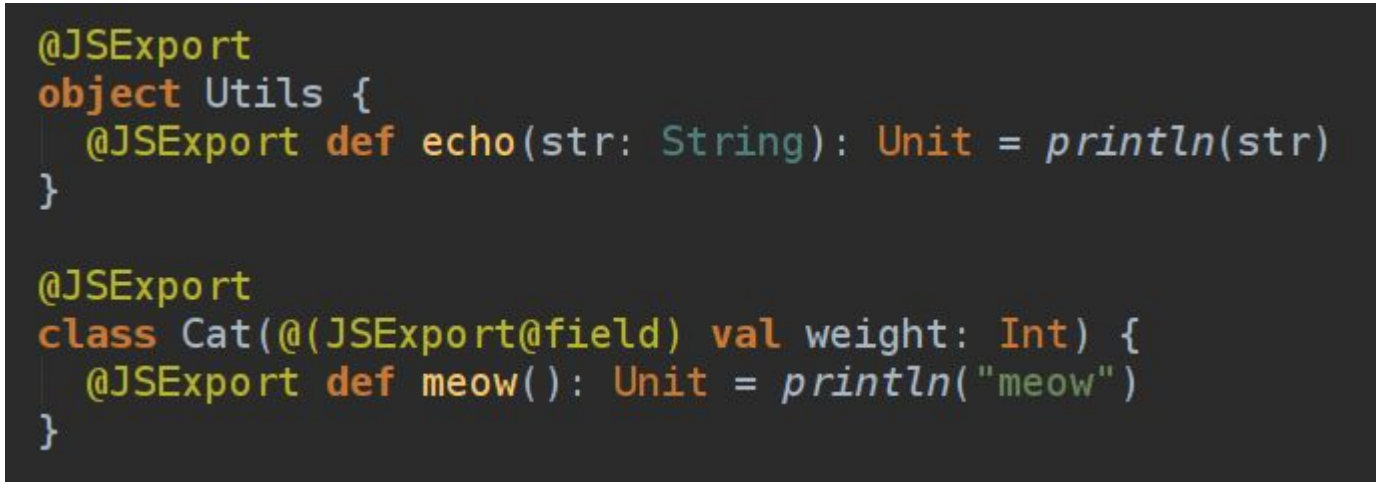
AVSYSTEM

Przypadek: obiekt JavaScript ze znanym API

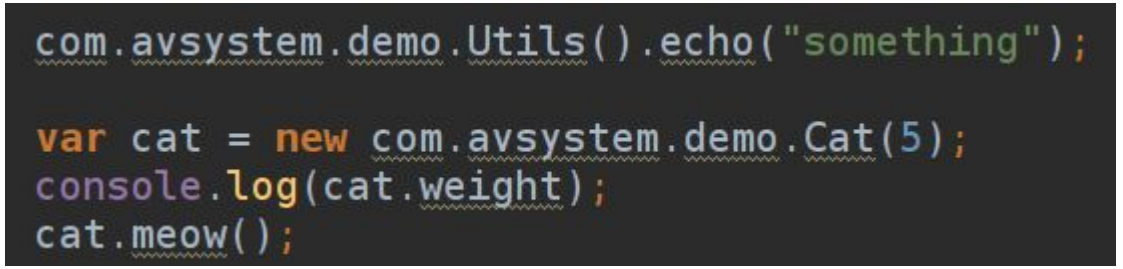
```
function Apple(type) {...}

function createApple(type) {
    return new Apple(type);
}
```

```
@js.native
object Globals extends js.GlobalScope {
  def createApple(`type`: String): Apple = js.native
}

@js.native
trait Apple extends js.Object {
  val `type`: String = js.native
  var color: String = js.native
  def getInfo: String = js.native
}
```

Eksportowanie kodu do użycia z JavaScriptu



```scala
@JSExport
object Utils {
  @JSExport def echo(str: String): Unit = println(str)
}

@JSExport
class Cat(@(JSExport@field) val weight: Int) {
  @JSExport def meow(): Unit = println("meow")
}
```



```javascript
com.avsystem.demo.Utils().echo("something");

var cat = new com.avsystem.demo.Cat(5);
console.log(cat.weight);
cat.meow();
```

Eksportowanie kodu do użycia z JavaScriptu

AVSYSTEM

```scala
@JSExport
@JSExportAll
object Utils {
  def echo(str: String): Unit = println(str)
}

@JSExport
@JSExportAll
class Cat(val weight: Int) {
  def meow(): Unit = println("meow")
}
```

```javascript
com.avsystem.demo.Utils().echo("something");

var cat = new com.avsystem.demo.Cat(5);
console.log(cat.weight);
cat.meow();
```

Mapowanie typów

- `Boolean`, `Byte`, `Short`, `Int`, `Double`, `String`, `Null` + odpowiedniki javove (`java.lang.Integer` etc.) są reprezentowane natywnie
- `Char`, `Long` - odrębna reprezentacja
- `Unit` = `undefined`

Specjalne typy natywne

Niejawne konwersje pomiędzy typami:

- `js.UndefOr[T]` <-> `Option[T]`
- `js.FunctionN` <-> `FunctionN`
- `js.ThisFunctionN` <-> `Function(N+1)`
- `js.Array[T]` <-> `mutable.Seq[T]`
- `js.Dictionary[T]` <-> `mutable.Map[String,T]`

- `js.Date`, `js.RegExp`

js.Function
js.ThisFunction

AVSYSTEM

```
@JSExport
val func: js.Function1[Apple,Unit] = (apple: Apple) => {
  apple.color = "green"
}

@JSExport
val thisFunc: js.ThisFunction0[Apple,Unit] = (apple: Apple) => {
  apple.color = "green"
}
```

this

```
var apple = new Apple("jonagold");
func(apple);
apple.thisFunc();
```

Wydajność

- porównywalna z czystym JavaScriptem
- czasami lepsza, np. `(a+b) | 0` zamiast `a+b`
- Google closure compiler: advanced optimizations

Podsumowanie

ScalaJS:

- wszędzie Scala!
- bogactwo języka
- koniec z głupimi pomyłkami pułapkami JS
- ten sam język i narzędzia w backendzie i frontendzie
- type-safe, nawet na styku JS-JVM
- szybkość