

# Running Scala in the Browser

## Cross-Compiling Scala to JavaScript

Wolfgang Kuehn

EnBW Trading

wo.kuehn@enbw.com

### Abstract

This paper describes one possibility to cross-compile the Scala programming language to JavaScript. It discusses the required steps and first findings, based on an ELIZA demo.

**Categories and Subject Descriptors** D.3.4 [Programming Languages]: Processors

**General Terms** Algorithms

**Keywords** Cross-Compiler, Scala, JavaScript

## 1. Introduction

We present a cross compiler for Scala to JavaScript which enables you to execute code written in Scala directly in a web browser. Event handling and DOM access according the w3c specification is demonstrated. Results, which are based on an Eliza implementation in Scala, are discussed. We describe the steps needed to extend an existing Java to JavaScript cross-compiler to handle Scala as input.

## 2. ELIZA

ELIZA [1] was a computer program and an early example of primitive natural language processing. ELIZA makes a good case to demonstrate our approach, because it is an excellent fit for Scala through functional programming, and is complex enough to draw substantial conclusions. We chose the ELIZA implementation by Paniz [8]. Figure 1 shows a screen shot of ELIZA.

### 2.1 Event Handling and DOM Access

The demo uses the w3c specifications DOM-EVENTS [11] for event handling and DOM-HTML [10] for DOM access, see Figure 2. The only thing we need from the DOM are two declared text area elements in the HTML code:

[Copyright notice will appear here once 'preprint' option is removed.]

## Welcome to ELIZA, your personal psycho therapist

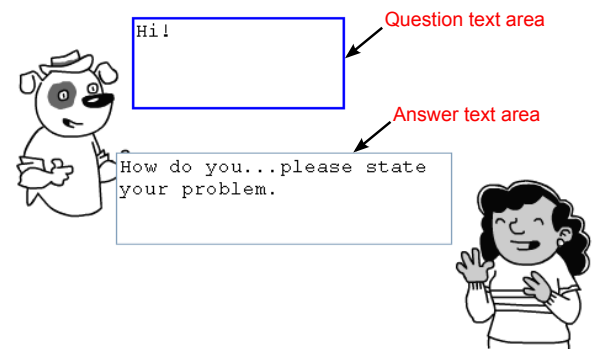


Figure 1. Screen Shot ELIZA

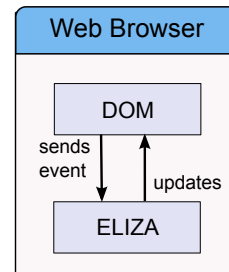


Figure 2. Event Handling and DOM Access

```
<textarea id="Question"/>
<textarea id="Answer" />
```

First thing in coding the example we have to import the needed classes:

```
import org.w3c.dom.events.KeyboardEvent
import org.w3c.dom.html.HTMLTextAreaElement
...
```

We are lazy and define a shortcut to access DOM elements by id:

```
def $(id: String) = document.getElementById(id)
```

Fetch text areas for question and answer:

```
val question = $("Question").
  asInstanceOf[HTMLTextAreaElement]
val answer = $("Answer").
  asInstanceOf[HTMLTextAreaElement]
```

Define the entry method (see Section 3.2) which is called after the document has loaded. There we set initial values for question and answer, and we bind the keyDownHandler to the question element.

```
def main(args: Array[String]) {
  question.setValue("Hi!")
  answer.setValue("")
  setEventHandler(question, "keydown", keyDownHandler);
}
```

The keyDownHandler delegates to ELIZA and displays the answer:

```
def keyDownHandler(evt: Event) {
  if (evt.asInstanceOf[KeyboardEvent].
    getKeyIdentifier() == "Enter") {
    answer.setValue(eliza.eval(question.getValue()))
  }
}
```

Because we want to use the w3c interfaces, we have to wrap handlers by instances of EventListener

```
def setEventHandler(elem: Element, evtType: String,
  handler: Event => Unit) {
  val listener = new EventListener {
    override def handleEvent(evt: Event) =
      handler(evt)
  }
  elem.asInstanceOf[EventTarget].
    addEventListener(evtType, listener, false);
}
```

## 2.2 Running and Building ELIZA

The ELIZA demo is available online ScalaDemo [4] at <http://www.J2JS.com/scala-demo>. You can change the code, for example the data tables, and easily build the demo according the instructions given in the online demo. We measure the following quantities while cross-compiling the demo:

Number of compiled classes	179	
Number of assembled methods	978	
Size of JavaScript assembly	229	kByte
Time to cross compile	20	sec

## 3. Java to JavaScript Cross Compiling

### 3.1 JavaScript Cross Compilers

There are several frameworks to produce JavaScript from stronger typed languages. Examples are GWT [2] and Java2Script [5] taking Java source code as input, or XMLVM [12] and J2JS [3] translating Java byte code. Apart

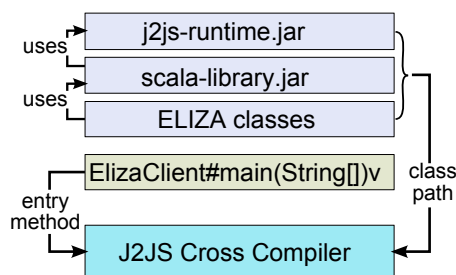


Figure 3. Input for Cross Compiler

from the Java to JavaScript frameworks, there are other tools cross compiling a stronger typed language (Java, C#) to a weaker typed language (JavaScript, ActionScript/Flex).

Because Scala compiles to Java Byte Code and integrates so well with the Java Runtime environment, we chose J2JS to venture into cross-compiling Scala.

We should mention that, GWT being the most popular cross-compiler, and because the widespread adoption of Scala, there are efforts ScalaGWT [9] to join GWT and Scala through an intermediate format.

### 3.2 About J2JS

The J2JS cross compiler is basically a de-compiler, where the output language is not Java but JavaScript. It emits as much control structure offered by JavaScript as possible. Through node reduction on a control-flow-graph it converts goto instructions to if-else or while instructions, exception tables to try-catch statements, etc.

The cross compiler takes class files or jars and a single entry method as input (see Figure 3) and discovers, by static code analysis, all class members referenced by the entry method. It then translates the methods to JavaScript and packages them into an assembly. This assembly can now be loaded by the web browser.

## 4. Adjusting the Cross Compiler

A first take to cross-compile sample Scala code with the existing J2JS compiler failed. This was not a big surprise, because J2JS was geared towards byte code produced from Java sources. The remaining sections describe the problems we encountered and their solutions.

### 4.1 Signatures

Java and Java byte code reference methods by signatures: A class may not declare two methods with the same signature. In Java, the signature of a method consists of the name of the method and the number and type of formal parameters, see JVM Spec [6] Section 2.10.2. However, Java byte code includes the return type of the method, see JVM Spec [6] Section 4.3.3. For the JVM, it is therefore perfectly legal for a class to have the following methods:

```
Object mymethod(int i, double d)
String mymethod(int i, double d)
```

The existing J2JS compiler ignored the return type. Because Scala makes extensive use of overloaded methods by return type, the compiler had to be adjusted to honor the return type in method signatures.

## 4.2 Java Runtime

The wrapper classes for the primitive types such as `java.lang.Double` had the `toString()` and `equals()` methods missing. These had to be added to make ELIZA compile.

## 4.3 Patching Scala Classes

No Scala class needed to be patched. One almost exception was `scala.Console`, which uses `java.lang.ThreadLocal`. Implementing threads for browsers does not make sense, so the `Console` class could be easily patched to do without `ThreadLocal`. We decided against it. Please use `java.lang.System.out` instead.

## 4.4 Exception Handling

The current Scala compiler (Version 2.77) emits multi-entrant catch blocks. For example the method

```
0 def insertAll(n: Int, iter: Iterable[A]) {
1   try {
2     var elems = iter.elements.toList
3     while (!elems.isEmpty) {
4       elems = elems.tail
5     }
7   } catch {
8     case ex: Exception => throw null
9   }
10 }
```

has one exception handler and is compiled to something like (see JVM Spec [6] Section 7.12 for details)

```
PC    Instruction
0     do first assignment at line 2
12    do while loop at line 3 to 5
30    return
31    handle exception at line 8
```

and the exception table

```
Code from PC=0 to PC=11 is handled by PC=31
Code from PC=12 to PC=30 is handled by PC=31
```

This is perfectly legal byte code, but cannot be transformed back to a `try-catch` statement where each exception handler is only entered once by the complete `try` block.

The simple solution is to join all contiguous elements with the same handler PC to get the exception table

```
Code from PC=0 to PC=30 is handled by PC=31
```

For the Scala Compiler people: It is best not to emit multi-entrant catch blocks in the first place.

## 5. Future Work

The generated JavaScript code can be of poor performance, especially on MS-IEExplorer. This is probably due to Scala's boxing and its extensive use of final anonymous methods. Currently J2JS does not optimize for final methods. We believe that a performance boost by a factor of five through moderate optimization is realistic.

## References

- [1] ELIZA in Wikipedia. <http://en.wikipedia.org/wiki/ELIZA>
- [2] Google Web Toolkit <http://code.google.com/webtoolkit>
- [3] J2JS site. <http://www.J2JS.com>
- [4] Online ELIZA Demo with sources. <http://www.J2JS.com/scala-demo>
- [5] Java2Script site. <http://j2s.sourceforge.net>
- [6] The Java Virtual Machine Specification, Second Edition. [http://java.sun.com/docs/books/jvms/second\\_edition/html/VMSpecTOC.doc.html](http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html)
- [7] Maven build tool. <http://maven.apache.org>
- [8] Panitz, S. E. 2006. Grundlagen der Kuenstlichen Intelligenz. <http://www.cs.hs-rm.de/~panitz/ki/skript.pdf>
- [9] Scala + GWT. <http://scalagwt.gogoego.com/index.html>
- [10] Document Object Model (DOM) Level 2 HTML Specification. <http://www.w3c.org/TR/2003/REC-DOM-LEVEL-2-HTML-20030109>
- [11] Document Object Model (DOM) Level 2 Events Specification. <http://www.w3c.org/TR/2000/REC-DOM-LEVEL-2-Events-2000113>
- [12] XMLVM. <http://xmlvm.org>