PUBLIC ABSTRACT JSON EXECUTEJAVASCRIPT(STRING SCRIPT, BOOLEAN SHOULDYOU);

MAGNUS BAE

itera
MAKE A DIFFERENCE

OR

JAVASCRIPT IN THE JVM

IS IT A GOOD IDEA?

Who is this guy?

- Magnus Bae
- Consultant at Itera Norway
- Master of Science from NTNU
- Strongest in Java and JavaScript, but also experienced in Qt and C++, C, Objective-C, C#, and more.

Agenda

- Introduction to Nashorn
- Hello World!
- Examples in Java
- Examples in JavaScript
- Scripting
- Using frameworks
- Multithreading
- Summary
- Conclusions
- Q&A

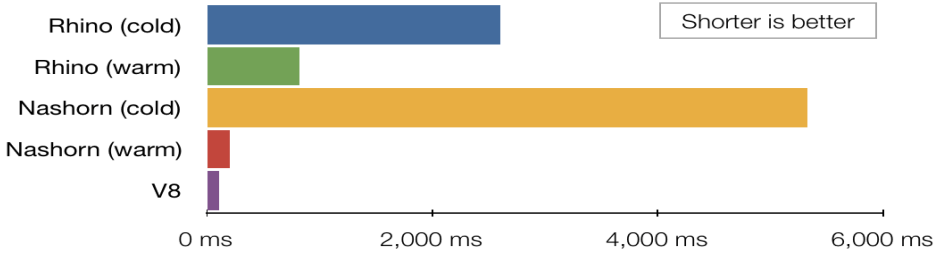
ANYWAYS

Rhino

- Part of Java 6 (released Desember, 2006)
- ECMAScript 3
- Sluggish performance

Nashorn

- Released with Java 8 (March, 2014)
- «Nashorn» is the German word for «Rhino»
- Supports ECMAScript 5.1
- Planned future support for ES6
- **Faster**

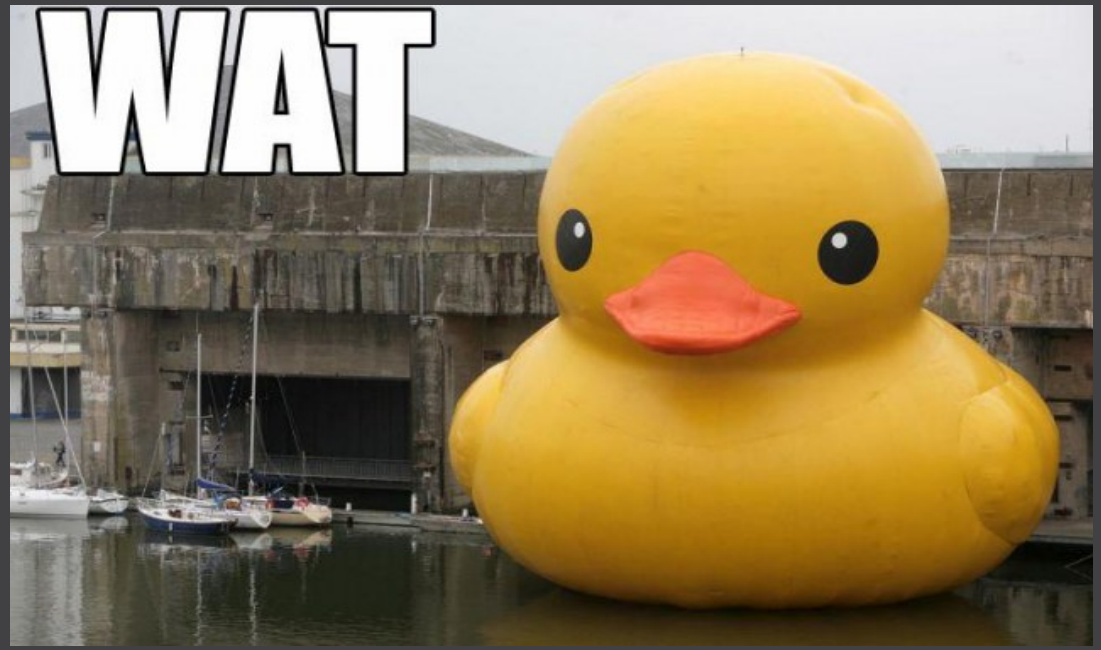


http://ariya.ofilabs.com/2014/03/nashorn-the-new-rhino-on-the-block.html

HI

console.log(‘Hello World’);

- Doesn’t actually work(!)


WAT

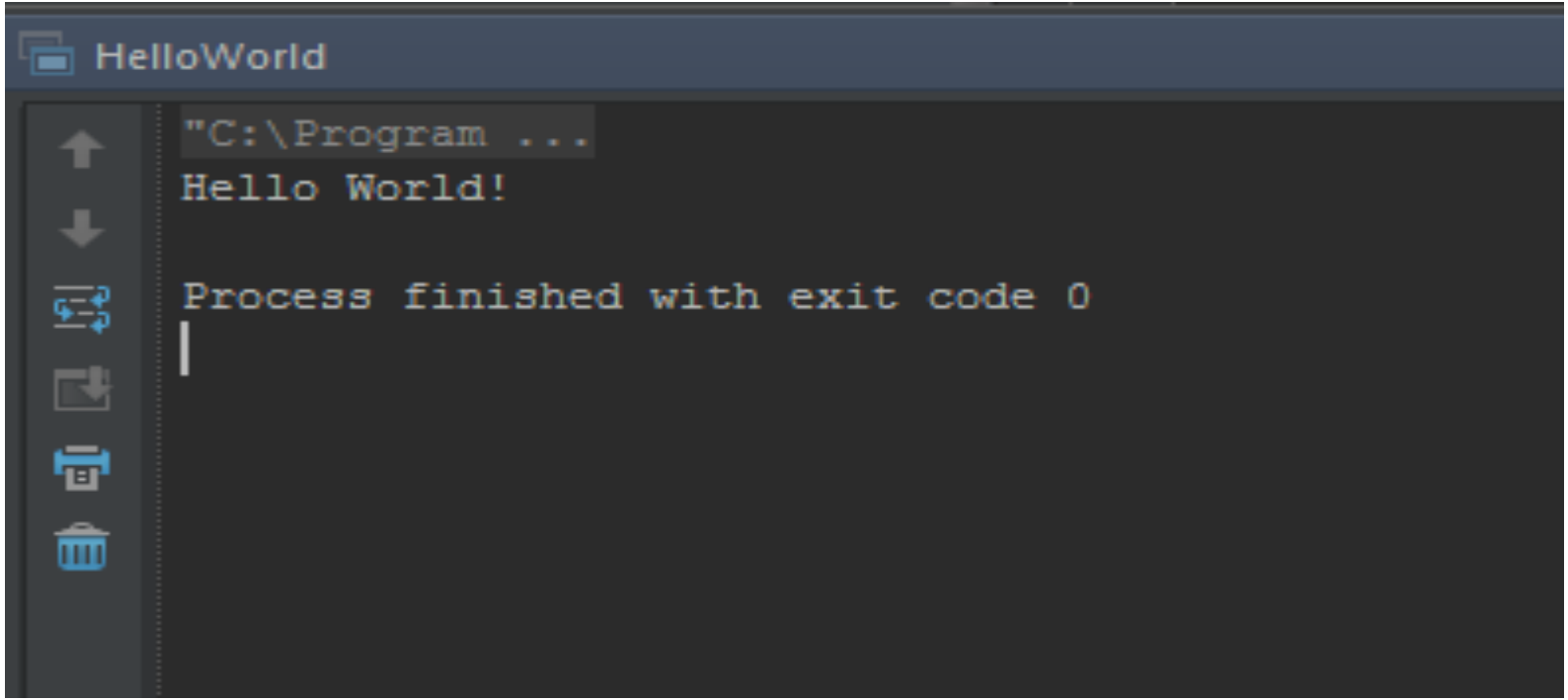
Let’s try that again

```
public class HelloWorld {

  public static void main(String[] args) throws ScriptException {
    ScriptEngine engine = new ScriptEngineManager().getEngineByName("nashorn");
    engine.eval("print('Hello World!');");
  }

}
```

Hello World


HelloWorld
"C:\Program ...
Hello World!
Process finished with exit code 0

But that's mostly Java?

- Yes, but…
- You can run pure JavaScript from the command line using «jjs»

```
~/git/jz/nashorn/src/main/javascript (master)
$ cat HelloWorld.js
print('Hello World!');

~/git/jz/nashorn/src/main/javascript (master)
$ jjs HelloWorld.js
Hello World!
```

SO WHAT ABOUT THAT CONSOLE.LOG()?

So what about that console.log()?

```
@Test(expected = ScriptException.class)
public void consoleLogFunctionIsNotNative() throws ScriptException {
   engine.eval("console.log('This text should not be output to the console');");
}
```

javax.script.ScriptException: ReferenceError: "console" is not defined in <eval> at line number 1

So what about that console.log()?

```
@Test
public void consoleLogCanBeInjected() throws ScriptException {
   String consolePrintString = "This text should be printed to standard out";
   String script = "console.log('" + consolePrintString + "');";

   engine.eval("var console = { log: print, dir: print };");
   engine.eval(script);

   assertEquals(consolePrintString, outContent.toString().trim());
}
```

GREAT,
I CAN LOG STUFF

NOW WHAT?

Possibilities with Nashorn

- Run Node.js applications in a Java environment
- Reuse JavaScript from a webapp
- Integrate JavaScript in your Java application
- Utilize or extend Java methods or classes in a JavaScript-application
- JavaFX applications

Possibilities with Nashorn

- Opens up scripting possibilities
  - Use ClassFilter for security when loading scripts from untrusted sources
    http://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html
- Increased code re-use. Eg. form validation.
- Server side rendering of React.js applications

EXAMPLES IN JAVA

eval()

```
String script = " (   function(){ return 42; }   ) (); ";

Object result = engine.eval(script);

assertTrue(result instanceof Number);

assertEquals(42, ((Number) result).intValue());
```

ScriptExceptions

```
@Test(expected = ScriptException.class)
public void returnWithoutFunctionThrowsException() throws ScriptException {
   engine.eval("return 42;");
}

//     javax.script.ScriptException: <eval>:1:0 Invalid return statement
//    return 42;
```

Invocation

```
String script = "function add(a, b){ return a + b; };";

engine.eval(script);
Invocable invocable = (Invocable) engine;

Object result = invocable.invokeFunction("add", 22, 20);

assertTrue(result != null);
assertTrue(result instanceof Number);

assertEquals(42, ((Number) result).intValue());
```

Watch out!

- If you are using an early version of Java 8 you might encounter issues with regards to return types of integer values (This is fixed in 8u40 and later)
- You might want to use java.lang.Number when you are expecting an integer or floating point value, followed by the appropriate getter (eg. intValue() or doubleValue())
- Alternatively you should be quite safe if you use the method outlined on the following pages

Invoke an interface

```
public interface Adder {
   int add(int a, int b);
}
```

Invoke an interface

```
String script = "function add(a, b){ return a + b; };";

engine.eval(script);
Invocable invocable = (Invocable) engine;
Adder adder = invocable.getInterface(Adder.class);

int result = adder.add(22, 20);

assertEquals(42, result);
```

itera
MAKE A DIFFERENCE

A RHINOCEROS WALKS INTO A CHINA SHOP

Porcelain

```
public class Porcelain {

  private final ScriptEngine engine;
  private final Invocable invocable;

  public Porcelain() {
    engine = new ScriptEngineManager().getEngineByName("nashorn");
    invocable = (Invocable) engine;
  }

  public Optional<Object> eval(String script) {
    Optional<Object> result;
    try {
      result = Optional.of(engine.eval(script));
    } catch (ScriptException e) {
      result = Optional.empty();
    }
    return result;
  }

  public Optional<Object> invokeFunction(String name, Object... args) {
    Optional<Object> ret;
    try {
      Object result = invocable.invokeFunction(name, args);
      ret = Optional.of(result);
    } catch (Exception e) {
      ret = Optional.empty();
    }
    return ret;
  }
...
}
```

Porcelain

```
String script = "function add(a, b){ return a + b; }; ";

porcelain.eval(script);

Optional<Object> result = porcelain.invokeFunction("add", 22, 20);

assertTrue(result.isPresent());
Object res = result.get();
assertTrue(res instanceof Number);
assertEquals(42, ((Number) res).intValue());
```

Porcelain

```
public class Porcelain {

  ( … )

public boolean load(String scriptPath){
  try {
    //Works in IntelliJ for this project. Path depends on which directory is "working directory"
    engine.eval(new FileReader("src/main/javascript/" + scriptPath));
    return true;
  } catch (ScriptException e) {
    return false;
  }
}

}
```

Porcelain

```
assertTrue(porcelain.load("SimpleScript.js"));

Optional<Object> result = porcelain.invokeFunction("returnFive");

assertTrue(result.isPresent());
Object res = result.get();
assertTrue(res instanceof Number);
assertEquals(5, ((Number) res).intValue());
```

EXAMPLES IN JAVASCRIPT

Running JavaScript

- «jjs» from the commandline
- *engine.eval("load(‘path/to/file.js’);");* will load and execute a file.

So will

- engine.eval(**new** FileReader("path/to/file.js"));

Examples

- The following code examples are adapted from http://www.oracle.com/technetwork/articles/java/jf14-nashorn-2126515.html

Calling Java

- Java classes can be accessed using their fully qualified name
- print(java.lang.System.currentTimeMillis());

Calling Java

```
var file = new java.io.File("path/to/file.js");

print(file.getAbsolutePath());

print(file.absolutePath);
```

- Nashorn infers properties from methods starting with get or set.

Calling Java

```
var stack = new java.util.LinkedList();

[1, 2, 3, 4].forEach(function(item) {
  stack.push(item);
});

print(stack);
print(stack.getClass());

//[4, 3, 2, 1]
//class java.util.LinkedList
```

Arrays and streams

```
var sorted = stack
  .stream()
  .sorted()
  .toArray();
print(sorted);
```

Converting from Java array

```
var jsArray = Java.from(sorted);
```

Converting to Java array

```
var javaArray = Java.to(jsArray);
```

Lambdas

```
javaMethodThatAcceptsLambdas(
  function(i) {
    return i % 2 == 0;
});
```

```
javaMethodThatAcceptsLambdas(
  function(i) i % 2 == 0);
```

Importing classes

- Oracle suggests that the following is the preferred way of importing classes and packages when using Nashorn:

```
var CollectionsAndFiles = new JavaImporter(
    java.util,
    java.io,
    java.nio);

with (CollectionsAndFiles) {
  var files = new LinkedHashSet();
  files.add(new File("Plop"));
  files.add(new File("Foo"));
  files.add(new File("w00t.js"));
}
```

http://www.oracle.com/technetwork/articles/java/jf14-nashorn-2126515.html

Importing classes

- Mozilla, however, discourages use of the **with** statement:

Using with is not recommended, and is forbidden in ECMAScript 5 strict mode. The recommended alternative is to assign the object whose properties you want to access to a temporary variable.

- That said, Nashorn does not run in strict mode by default.

Implementing interfaces

```
var iterator = new java.util.Iterator({
  i: 0,
  hasNext: function() {
    return this.i < 10;
  },
  next: function() {
    return this.i++;
  }
});

print(iterator instanceof Java.type("java.util.Iterator"));
while (iterator.hasNext()) {
  print("-> " + iterator.next());
}
```

```
/*
true
-> 0
-> 1
-> 2
-> 3
-> 4
-> 5
-> 6
-> 7
-> 8
-> 9
*/
```

Extending classes

```
var ObjectType = Java.type("java.lang.Object");

var MyExtender = Java.extend(ObjectType);

var instance = new MyExtender({
  someInt: 0
});
```

Extending classes

```
var ObjectType = Java.type("java.lang.Object");
var Comparable = Java.type("java.lang.Comparable");

var MyExtender = Java.extend(ObjectType, Comparable);

var instance = new MyExtender({
  someInt: 0,
  compareTo: function(other) {
    var value = other["someInt"];
    if (value === undefined) {
      return 1;
    }
    if (this.someInt < value) {
      return -1;
    } else if (this.someInt == value) {
      return 0;
    } else {
      return 1;
    }
  }
});
```

Extending classes

```
var ObjectType = Java.type("java.lang.Object");
var Comparable = Java.type("java.lang.Comparable");

var MyExtender = Java.extend(ObjectType, Comparable);

var instance = new MyExtender({
  someInt: 0,
  compareTo: function(other) {
    var value = other["someInt"];
    if (value === undefined) {
      return 1;
    }
    if (this.someInt < value) {
      return -1;
    } else if (this.someInt == value) {
      return 0;
    } else {
…
```

Extending classes

```
print(instance instanceof Comparable);
print(instance.compareTo({ someInt: 10 }));
print(instance.compareTo({ someInt: 0 }));
print(instance.compareTo({ someInt: -10 }));

//true
//-1
//0
//1
```

Gotcha!

- Extending classes from the same extender type results in different implementations being considered equal!
- Continuing with our previous extender class, *MyExtender*

```
var anotherInstance = new MyExtender({
  compareTo: function(other) {
    return -1;
  }
});

// Prints 'true'!
print(instance.getClass() === anotherInstance.getClass());
```

Extending classes – take 2

```
var CallableType = Java.type("java.util.concurrent.Callable");

var FooCallable = Java.extend(CallableType, {
  call: function() {
    return "Foo";
  }
});

var BarCallable = Java.extend(CallableType, {
  call: function() {
    return "Bar";
  }
});
```

itera
MAKE A DIFFERENCE

Extending classes – take 2

```
var foo = new FooCallable();
var bar = new BarCallable();

print(foo.getClass() === bar.getClass());        // 'false‘

print(foo.call());                               // ‘Foo'
print(bar.call());                               // ‘Bar'
```

itera
MAKE A DIFFERENCE

Gotcha - again

- You can still create a new implementation by passing an implementation-object to the constructor of the class.

```
var foobar = new FooCallable({
  call: function() {
    return “FooBar”;
  }
});

// ‘FooBar’
print(foobar.call());

// ‘true’
print(foo.getClass() === foobar.getClass());
```

itera
MAKE A DIFFERENCE

SCRIPTING

Scripting extensions

- Heredocs
- Shell invocations
- Enabled by applying the *-scripting* parameter when invoking jjs.
  - *jjs -scripting*
- Useful for creating JavaScript-driven shell scripts.
  - Set executable by starting file with
    #!/usr/bin/jjs *(substitute for you path to jjs)*
- Can **only** be used from the command line!

Heredocs

Unix multiline strings with support for embedded variables

```
var data = {
 foo: "bar",
 time: new Date()
};

print(<<EOF);
So...
    foo = ${data.foo} and the current time is
    ${data.time}
EOF
```

Shell invocations

- Scripting mode injects the following global objects
  - $ARG
  - $ENV
  - $EXEC()
  - $OPTIONS
  - $OUT
  - $ERR
  - $EXIT

- Read more here: https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/shell.html

Shell invocations

- Execute system commands by putting them in backticks
  - Eg. `ls -lsa` will list the contents of the current dir.

```
var lines = `ls -lsa`.split("\n");
for each (var line in lines) {
  print("|> " + line);
}

#run with jjs -scripting dir.js
```

Shell invocations

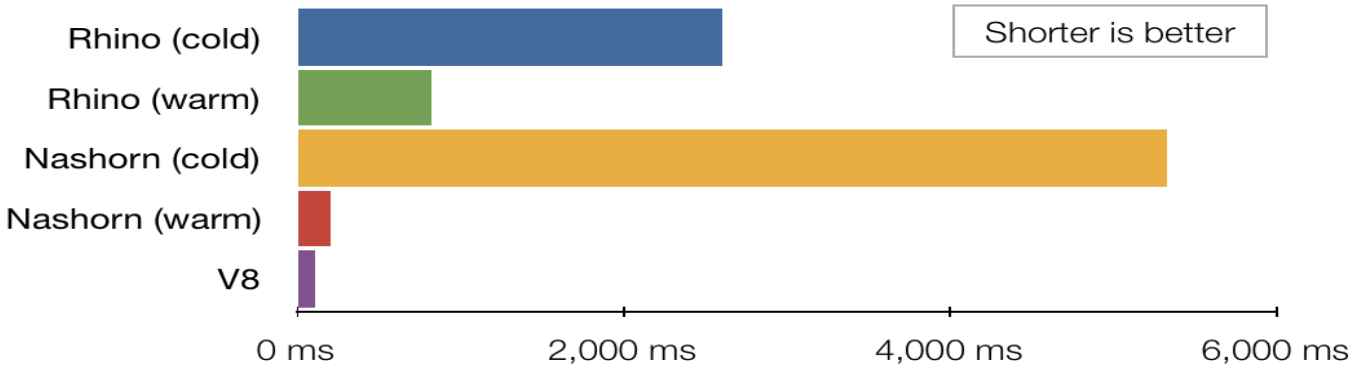
- Alternatively, execute commands by using the $EXEC()-function
  - Eg. $EXEC(‘ls -lsa’)

```
var lines = $EXEC("ls -lsa").split("\n");
for each (var line in lines) {
  print("|> " + line);
}

#run with jjs -scripting dir.js
```

PERFORMANCE

Performance



http://ariya.ofilabs.com/2014/03/nashorn-the-new-rhino-on-the-block.html

Benchmark.js

```
var suite = new Benchmark.Suite;

// add tests
suite.add('RegExp#test', function() {
  /o/.test('Hello World!');
})
```

Benchmark.js

```
var suite = new Benchmark.Suite;

// add tests
suite.add('RegExp#test', function() {
  /o/.test('Hello  World!');
})
.add('String#indexOf', function() {
  'Hello World!'.indexOf('o') > -1;
})
.add('lodash#sortBySin(n)', function(){
  _.sortBy([0,1,2,3,4,5],  function(n){
    return Math.sin(n);
  });
})
```

Performance with Node

$ node src/main/javascript/NodeBenchmark.js
RegExp#test x 9,288,051 ops/sec ±1.87% (94 runs sampled)
String#indexOf x 15,563,240 ops/sec ±1.53% (90 runs sampled)
lodash#sortBySin(n) x 481,666 ops/sec ±1.28% (93 runs sampled)
Fastest is String#indexOf

Performance with Nashorn

```
$ jjs src/main/javascript/Benchmark.js
RegExp#test x 5,503,565 ops/sec +/-5.94% (61 runs sampled)
String#indexOf x 23,423,545 ops/sec +/-0.39% (69 runs sampled)
lodash#sortBySin(n) x 48,411 ops/sec +/-13.51% (59 runs sampled)
Fastest is String#indexOf
```

Performance

Nashorn

RegExp#test x 5,503,565 ops/sec +/-5.94% (61 runs sampled)

*String#indexOf x **23,423,545** ops/sec +/-0.39% (69 runs sampled)*

lodash#sortBySin(n) x 48,411 ops/sec +/-13.51% (59 runs sampled)

Node

*RegExp#test x **9,288,051** ops/sec ±1.87% (94 runs sampled)*

String#indexOf x 15,563,240 ops/sec ±1.53% (90 runs sampled)

*lodash#sortBySin(n) x **481,666** ops/sec ±1.28% (93 runs sampled)*

FRAMEWORKS

I want to use framework x, can I?

- It depends
- Node dependent frameworks can be loaded using extra libraries
  - Avatar.js (Project Avatar)
  - jvm-npm (Nodyn)
- Browserify*
- Require.js**
- Some gotchas:
  - No console, window, etc.
  - No DOM (well, duh!)
  - Also missing some common functions like setTimeout() and setInterval(), you'll need to roll your own https://blogs.oracle.com/nashorn/entry/setinterval_and_settimeout_javascript_functions

Lodash

- Loads fine using the built in *load()* function
- Loads fine using *require()* and avatar.js
- Does not load using Browserify or jvm-npm (part of nodyn)
  - bundle.js:720 TypeError: Cannot read property "prototype" from undefined

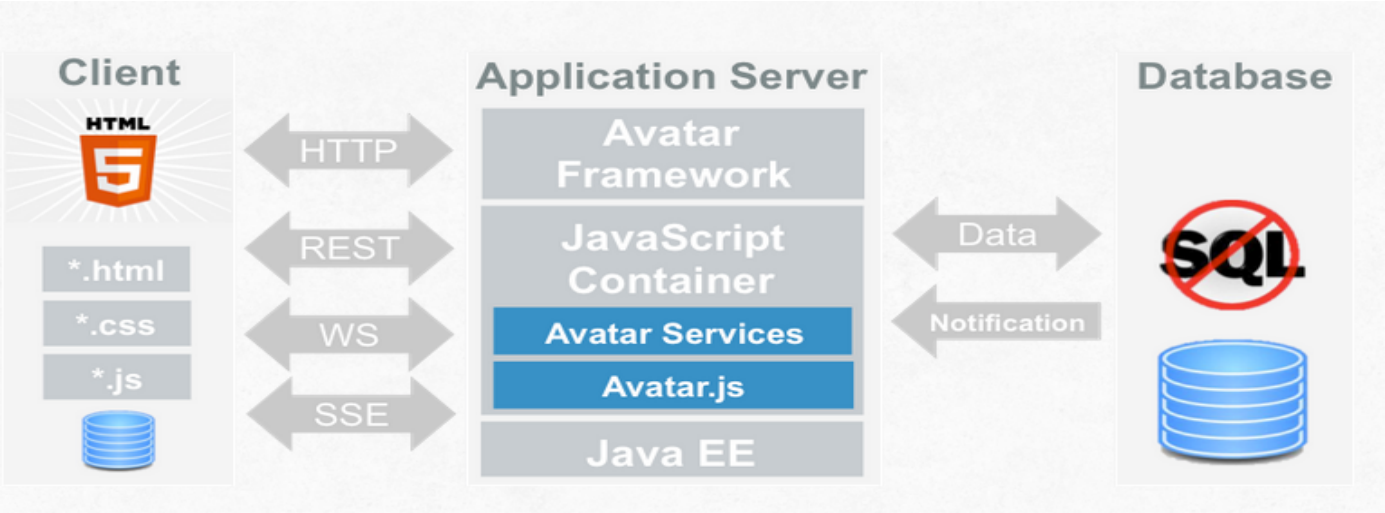
Promise

- https://github.com/then/promise
- Loads fine using Avatar.js
- Does not load using Browserify
  - bundle.js:424 ReferenceError: "setTimeout" is not defined

Avatar.js

- No sensible documentation
- Rumours that development has stopped (last build in March)
- Used to promote «Project Avatar» which allows you to run node applications on the Glassfish application server

Project Avatar



https://avatar.java.net/

Avatar.js

```
// load module 'http' (this is blocking) to handle http requests
var http = require('http');

// when there is a request we return 'Hello, World\n'
function handleRequest(req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, World\n');
}

// we listen on localhost, port 1337
// and give handleRequest as call back
// you see the non-blocking / asynchronous nature here
http.createServer(handleRequest).listen(1337, '127.0.0.1');

// logs to the console to reassure that we are on our way
console.log('Get your hello at http://127.0.0.1:1337/');
```

Avatar.js

$ java -jar lib/avatar-js.jar src\main\javascript\NodeHttp.js

//Get your hello at http://127.0.0.1:1337/

(J)avatar.js

```
public class AvatarRunner {
  public String runJs(final String javaScriptFile) throws Throwable{
    StringWriter scriptWriter = new StringWriter();
    ScriptEngine engine = new
ScriptEngineManager().getEngineByName("nashorn");
    ScriptContext scriptContext = engine.getContext();
    scriptContext.setWriter(scriptWriter);

    Server server = new Server(engine, new Loader.Core(), new Logging(false),
System.getProperty("user.dir"));
    server.run("src/main/javascript/" + javaScriptFile);

    return scriptWriter.toString();
  }
}
```

(J)avatar.js

```
@Before
public void setUp() throws Exception {
   avatarRunner = new AvatarRunner();
}

@Test
public void handlePromisesNpmDependency() throws Throwable {
   String result = avatarRunner.runJs("PromiseExample.js");

   System.out.println(result);
}
```

form validated

```
@Test
public void handleLodashNpmDependency() throws Throwable {
   String result = avatarRunner.runJs("LodashExample.js");

   System.out.println(result);
}
```

[1, 2, 3]

jvm-npm

- Allows usage of require: **load('lib/jvm-npm.js');**
- Best suited for own packages, but can also be used to require() npm-modules.

MULTITHREADING

Multithreading

- There are various ways of multithreading with Nashorn
  - Spawning a new Java-thread
    - Either from Java or
    - From JavaScript
  - Use Avatar.js
    - Except for the possibility of running multiple loops there isn't anything novel here, node is already multithreaded (except for your code)

Regular multithreading

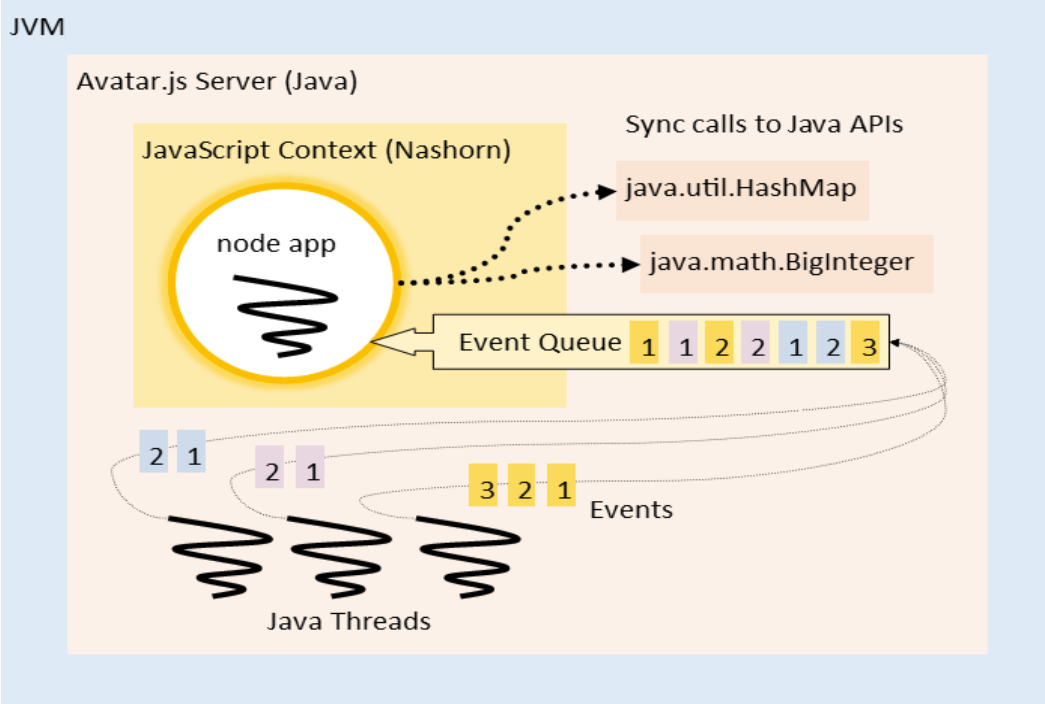
```
var Runnable = Java.type('java.lang.Runnable');
var Printer = Java.extend(Runnable, {
    run: function() {
        print('printed from a separate thread');
    }
});

var Thread = Java.type('java.lang.Thread');
new Thread(new Printer()).start();

new Thread(function() {
    print('printed from another thread');
}).start();
```

Example from: http://winterbe.com/posts/2014/04/05/java8-nashorn-tutorial/

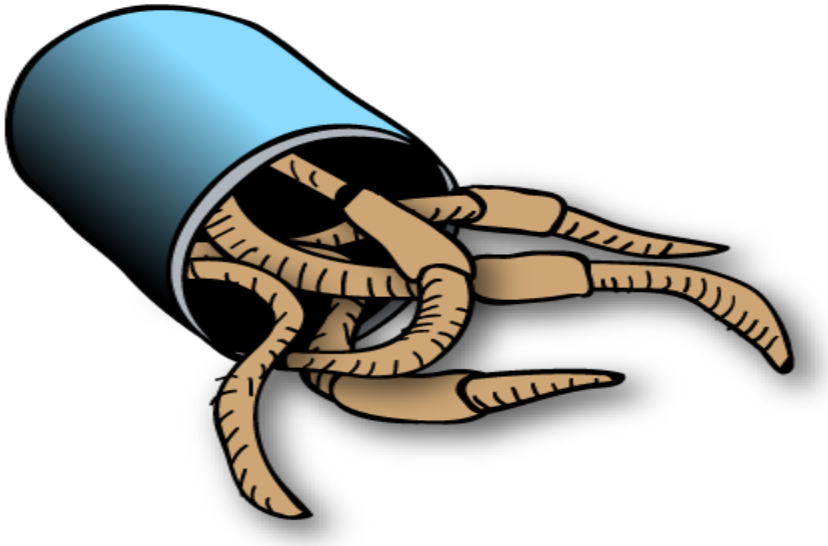
Multithreading in Avatar



https://avatar-js.java.net/index.html

Multithreading in Avatar

- It should also be possible to spawn threads inside code running on Avatar.js, but that is another can of worms.



Summary

- Nashorn is ECMAScript 5.1 compliant.
- Should be fast enough for most use-cases
- You can invoke JavaScript functions from Java
- You can invoke, implement, and extend Java classes in JavaScript
- You can pass JavaScript functions as lambdas to JavaScript functions
- You can use Java collections
  - And streams…
- Lists and arrays can be converted between Java versions and JavaScript versions using Java.to() and Java.from()

Summary

- Run JavaScript files from the command line using *jjs*
- Enable scripting functions with the -scripting parameter
- Load external dependencies from JavaScript with the *load()* function
- Load external dependencies from Java using the *eval()* method
- Run node applications using Avatar.js or Nodyn.
- Load dependencies using require with either browserify or jvm-npm, but be careful.
- You can invoke new threads from JavaScript

In conclusion

- Running JavaScript in the JVM opens up new possibilities, but it's not a drop-in replacement for fully fledged JavaScript environments like Node.js.
- In theory it should be easy to leverage the great amount of npm-modules out there, but even commonly used examples that are known to work in the browser fail.
- The most compelling use cases:
  - Opening up for scripting of applications
  - When you want a JavaScript backend, but need it to run in Java
  - Providing server side rendering of React-applications

QUESTIONS?

Code used in the presentation will be uploaded to:
https://github.com/magnusbae/jz-2015-nashorn

Code used in the presentation

https://github.com/magnusbae/jz-2015-nashorn

itera

MAKE A DIFFERENCE