J2V8
A Highly Efficient JS Runtime For Java

R. Ian Bull
EclipseSource

@irbull
<table>
<thead>
<tr>
<th>Jochen (My Boss):</th>
<th>We are building a native widget toolkit for Android / iOS based on Javascript.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me:</td>
<td>Cool!</td>
</tr>
<tr>
<td>Jochen:</td>
<td>Our implementation with native widgets is 10x slower than the browser.</td>
</tr>
<tr>
<td>Me:</td>
<td>Not Cool!</td>
</tr>
<tr>
<td>Jochen:</td>
<td>Fix That!</td>
</tr>
</tbody>
</table>
Android and JS

• Javascript on Android is slow

• Nashorn is not available

• Compiled Rhino is not available

• Only option is non-optimized Rhino
What About **V8**?

- **V8** is a *highly performant* Javascript runtime
- Written entirely in **C++**
- Developed by Google & Workhorse behind Chrome

J2V8 Inspiration

• SWT is an open source widget toolkit for Java designed to provide **efficient, portable access** to the user-interface **facilities of the operating systems** on which it is implemented

• Create a thin JNI layer above V8

• Expose (some) V8 API in Java

• Complicated logic lives in Java
J2V8 Challenges

- **Two GCs** and **unmanaged** memory model in the middle

- V8’s API is **stack based**, once an Object goes out of scope, it can be collected
  - Makes it hard to **return** Objects to Java

- Java and Javascript both throw **exceptions**

- Unfamiliar with JS, JNI and C++
J2V8 Design

• Each V8 Object can be referenced using a Handle

• Each Object is stored in a V8 Persistent Object Store

• Objects must be explicitly freed
  • I would like some feedback on this point.

• Primitives where possible (no wrappers)

• Single Thread
public static void main(String[] args) {
    V8 v8 = V8.createV8Runtime();
    int result = v8.executeIntScript("1+1");
    System.out.println("Result: " + result);
    v8.release();
}

- A Runtime must be created
- Avoid unnecessary wrapping

int result = (int)(Integer)v8.executeScript("1+1");
Resources

```java
public static void main(String[] args) {
    V8 v8 = V8.createV8Runtime();
    String js = "var me = {First: 'Robert', Middle: 'Ian', Last: 'Bull', age: 38};";
    V8Object result = v8.executeObjectScript( js + "me;" );
    System.out.println(result.getString("Last") + ", " + result.getInteger("age");
    result.release();
    v8.release();
}
```

- V8Object creates a new JS Object in a **persistent store**
- Object are **lazily copied** to Java
- Objects must be explicitly **released**
V8Objects and V8Arrays

```java
V8Object result = v8.executeObjectScript(...);
for( String key : result.getKeys() ) {
    if (result.getType(key) == V8Value.INTEGER) {
        int value = result.getInteger(key);
    }
    ...
}
```

- **Types** can be inspected
- **Keys** can be fetched
- **Ranges of array** values can be loaded in bulk
  - More about that in the performance section
Building V8Objects and V8Arrays

```java
public static void main(String[] args) {
    V8 v8 = V8.createV8Runtime();
    V8Object me = new V8Object(v8)
        .add("first", "Robert")
        .add("Last", "Bull")
        .add("Age", 38);
    v8.add("irbull", me);
    v8.executeVoidScript( "... Script that operates on irbull ... ");
    me.release();
    v8.release(true);
}
```

- **Fluent** API for constructing objects
- JS Object is constructed **incrementally**
- V8Objects and V8Arrays can **contain** V8Objects and V8Arrays
- V8Objects and V8Arrays must be **released**
Lists and Maps

V8Object me = new V8Object(v8)
    .add("First", "Robert")
    .add("Last", "Bull")
    .add("Age", 38);
Map<String, Object> map = V8ObjectUtils.toMap(me);
System.out.println(map.get("Last") + "," + map.get("Age"));

• Utilities for integrating with Lists and Maps
  • Primitives are automatically wrapped
  • This performs a deep-copy
Calling JS Functions

String js = "var foo = function(x) {return 42 + x;};";
v8.executeVoidScript(js);
V8Array parameters = new V8Array(v8).push(3);
int result = v8.executeIntFunction("foo", parameters);
System.out.println(result);
parameters.release();

- JS Functions can be called on V8Objects or the global namespace
- Parameters are passed as V8Arrays
- Parameter Objects must be released
Java Callbacks

```java
public static class Printer {
    public void print(String string) {
        System.out.println(string);
    }
}

class V8 {
    public static void main(String[] args) {
        V8 v8 = V8.createV8Runtime();
        v8.registerJavaMethod(new Printer(), "print", "print",
                               new Class<?>[]{String.class});
        v8.executeVoidScript("print('Hello, World!');");
        v8.release(true);
    }
}
```

- Java methods can be registered as **callbacks** to JS
- Can be registered on a **V8Object** or **global namespace**
Java Callbacks (cont...)  

- Callbacks can be registered reflectively
  - Requires the Object, method name, and parameter types
- Or callbacks can implement `JavaCallback` or `JavaVoidCallback`
- Return results do not need to be released
Java Exceptions

```java
public static class Printer implements JavaVoidCallback {
    public void invoke(V8Array parameters) {
        Object arg1 = V8ObjectUtils.getValue(parameters, 0);
        if (arg1 == null)
            throw new NullPointerException("Naughty Developer!");
        System.out.println(arg1);
    }
}

public static void main(String[] args) {
    v8.registerJavaMethod(new Printer(), "print");
    v8.executeVoidScript("try {
        print(null);
    } catch (e) {
        print(e);
    }");
}
```

- Exceptions in Java callbacks are **converted** to JS exceptions
- **Java message** becomes JS Exception
JS Exceptions

• All exceptions are **Runtime Exceptions**

• **V8ScriptExecution** exceptions are thrown for JS exceptions

• **V8ScriptCompilation** exceptions are thrown if the script doesn’t compile
Debugger Support
Library Loading

- J2V8 includes the native library in the Jar

- Inspired by SWT, J2V8 extracts the native library and dynamically loads them

  - First looks in java.library.path
  
  - Checks user.dir/jni (for development purposes)

  - Unpacks the native libs into user.home/j2v8

- Existing native libraries are overwritten
Using J2V8

• J2V8 is available in **Maven Central**

• Currently **5 variants** are available:

  - com.eclipsesource.j2v8.j2v8_win32_x86:2.0
  - com.eclipsesource.j2v8.j2v8_macosx_x86_64:2.0
  - com.eclipsesource.j2v8.j2v8_android:2.0
  - com.eclipsesource.j2v8.j2v8_android_armv7l:2.0
  - com.eclipsesource.j2v8.j2v8_android_x86:2.0

• j2v8_android:2.0 contains both **x86** and **armv7l**, and the correct library will be selected at runtime
Performance

```javascript
var data = [];
for (var i = 0; i < 20000; i++) {
    data[i] = i;
}

for (var i = 0; i < data.length; i++) {
    for (var j = 0; j < data.length; j++) {
        if (compare.compare(data[i], data[j])) {
            var tmp = data[i];
            data[i] = data[j];
            data[j] = tmp;
        }
    }
}

data;
```
Performance

- **Best** option for Android

<table>
<thead>
<tr>
<th>Array Usage</th>
<th>V8</th>
<th>Nashorn</th>
<th>Rhino</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Array Usage</td>
<td>V8</td>
<td>Nashorn</td>
<td>Rhino</td>
</tr>
</tbody>
</table>
JNI Bottleneck

- **JNI** is slow :(

**Callback on each iteration**

<table>
<thead>
<tr>
<th>V8</th>
<th>Nashorn</th>
</tr>
</thead>
</table>

**int[] return**

<table>
<thead>
<tr>
<th>V8</th>
<th>Nashorn</th>
<th>V8 Bulk</th>
</tr>
</thead>
</table>
Future Work

• Script API (JSR 223)

• Advanced **exception handling** between Java and JS

• **Batch** callbacks for better JNI performance

• Properly version the native libraries

• **Debugger** integration with Chrome Dev Tools

• One thread per V8Runtime
J2V8

- Open Source Java bindings for V8
- Licensed under the **EPL**

[GitHub Link](https://github.com/eclipsesource/j2v8)