Embedding V8 in the real world
Vladimir Mutafov & Stanimira Vlaeva
Software Engineers at Progress
NativeScript Core Team
A Story in 4 Parts
So...

NativeScript
Framework for building native Android and iOS apps with Angular, Vue or plain JS.
<table>
<thead>
<tr>
<th>Application Framework</th>
<th>NativeScript 'light'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data-binding, Navigation, ....</td>
</tr>
<tr>
<td></td>
<td>Angular</td>
</tr>
<tr>
<td></td>
<td>Vue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cross Platform Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layouts, UI Widgets, CSS, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Native API Access</th>
<th>NativeScript Android Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NativeScript iOS Runtime</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Native Code</th>
<th>Android</th>
<th>iOS</th>
</tr>
</thead>
</table>
Native API Access

Part 1 / 4
The Application Package
Android

Android Application

NativeScript Magic
Android

Android Application

JS code

(N) runtime
Executing JavaScript
V8
JavaScript Engine

Executes JS
Embedded in Chrome, Node, and NativeScript
Read more

A crash course in JIT compilers by Lin Clark

Life of a Script
by Sathya Gunasekaran & Jakob Kummerow
Android

Android Application

JS code  {N} runtime  V8
const recorder = new android.media.MediaRecorder();
Metadata Generator
Native Library

Metadata Generator

Runtime Binary
const recorder = new android.media.MediaRecorder();
Application launch

Initialize V8 ➔ Load Metadata ➔ Attach Callbacks
Callbacks
android.media.MediaRecorder

android -> Set as global object in the running V8 instance

android.media -> Package getter callback finds android.media in the android package metadata

android.media.MediaRecorder -> Package getter cb finds MediaRecorder in android.media MediaRecorder is a class -> a constructor function is returned
Constructor callback

`new android.media.MediaRecorder()`

Instantiates the native object in the Android world

How?
JNI
Java Native Interface

Allows V8 to send instructions to ART and vice versa.
The bridge between the two VMs.
new android.media.MediaRecorder()

Constructor callback

Instantiates the native object in the Android world
Creates a JS proxy object
Returns it back to the JS world
Field getter callback

Queries the original Java object for someRandomField
A slight complication...

`java.lang.String` !== `String`
Marshalling

Converts data from the Java world to the JS world and vice versa.

Java objects are proxied to special JS objects.
Method callback

Calls the method on the Java object

The result is marshallized and returned back to the JS world
const recorder = new android.media.MediaRecorder();

JavaScript Virtual Machine

Calls the constructor callback

NativeScript Runtime

Requests an instance of the class

Android OS
const recorder = new android.media.MediaRecorder();

JavaScript Virtual Machine

NativeScript Runtime

Returns a proxy object

Returns an instance of the class

Android OS
const result = recorder.doStuff();

JavaScript Virtual Machine

Calls the method callback

NativeScript Runtime

Calls the method on the native object

Android OS
const result = recorder.doStuff();

JavaScript Virtual Machine

NativeScript Runtime

Android OS

Returns the marshalled JS data

Returns the method call result
Objects lifecycle

Part 2 / 4
Garbage collection

Retrieves the memory of unused objects

Nondeterministic nature

Both the Android Runtime and V8 have GC
Synchronization by the NativeScript Runtime

Ensures no object is prematurely collected

Uses V8 finalizer callbacks

Stores strong/weak references to Java objects created with JS code
V8 marks an object for collection

Finalizer callback is triggered

Does the Runtime reference the object?

Tell V8 it's safe to delete the object

Make it a weak one

Is it a strong reference?

Tell V8 to keep the object

Does it contain a value?
If there is a strong reference, object is in use.

If there is only a weak reference, object can be collected.

Deleting an object depends on V8’s GC.
Challenges
Possible memory problems

The Java objects require several GC cycles to be collected.

Creating big Java objects through JS may lead to “out of memory” exceptions.
Forcing Garbage Collection

1. V8 GC
2. Android Runtime GC
3. V8 GC
releaseNativeCounterpart: fn
Multithreading

Part 3 / 4
JS in NativeScript -> Single Thread
JS in NativeScript -> Single Thread

= User Interface Thread
Jank

60 frames per second

1 second / 60 frames = 16.66 millisecond budget

Failing to meet the budget

==> frame rate drop
FRAME RATES FOR ANIMATION

15 FPS  30 FPS  60 FPS
No jank

Building UI
Animations
HTTP/network requests
Jank

Executing CPU-intensive operations.
The same happens in native Android apps.
Worker threads

Background threads in the JavaScript world

Based on the web workers API

No JS memory sharing
Worker thread = ???
Theory time!
Isolate

V8’s way to allocate and **isolate** memory for a code that’s running.

Isolates can run in parallel.
One isolate = multiple contexts.

No memory isolation.

Contexts can't run in parallel.
Worker thread = ???
Worker thread = Isolate
Let's talk about start up time...
File System Requests
Parsing & Compiling JS
require 'main.js' = 2142ms
Bundled app = fewer FS requests = faster launch time
What about 'Parse & Compile'?
How 8 Works

We take your JS → Parse it → turn that into an Abstract Syntax Tree

Get feedback for speculative optimisations → Generate bytecode

Optimize & Compile it

then run your optimized code!

x86, ARM, MIPS

By @addyosmani

JavaScript Start-up Performance by Addy Osmani
We must load the JS at some point...
Custom startup snapshots!
Creating custom snapshots
1. Load the snapshot binary
2. Set up the parameters for the new isolate
3. Create the new isolate

--> The context in the isolate will be a copy of the context in the snapshot.

Loading snapshots
Limitations

- Bare context
- No native APIs
- No require
- 3rd party-code
// Creating a snapshot throws an error.
// ReferenceError: android is not defined

const version =
    android.os.Build.VERSION.SDK_INT;

function doStuff() {
    console.log(version);
    ...
}

// Creating a snapshot works.
// The native getter is not evaluated immediately.

const getVersion = () =>
    android.os.Build.VERSION.SDK_INT;

function doStuff() {
    const version = getVersion();
    console.log(version);
    console.log(version);
    ...
}
Be lazy.