# Streams API

09/12/2015

youenn/calvaris

# What is it?

- Enabling I/O processing
  - Read chunks asynchronously
  - Write chunks asynchronously
  - Pipe from a stream to another
  - Automatic transformations
- Any kind of chunk
  - Strings
  - ArrayBuffers
  - Potatoes
  - Any JSValue cocktail

# What is it good for?

- Get me this video segment ASAP
  - Without streams API: download it, then read it
  - With the streams API + MSE: start downloading it and pipe it to MSE
    - Using SourceBuffer.appendStream
- Get me this WebSocket-like connection on HTTP
  - ReadableStream/WritableStream to receive/send messages
  - With HTTP/2, just one TCP/IP connection for both WebSocket-like channels and regular HTTP content
- Any I/O use-case actually…
  - Wrapping of all data sources in a single clean model
    - HTTP, WebRTC, file system

# What is in the spec?

- Stable
  - ReadableStream
    - Except pipe operations (related to WritableStream)
- Beta
  - WritableStream
- Experimental
  - TransformStream
  - ReadableByteArrayStream
    - May be almost merged with ReadableStream

# What it is good for/bad for, internals?

- Promise based
- Async is good but
  - Still a bit expensive
- Use it for arrays, objects
  - Probably not to pass one byte after one byte

# Where will it be?

- Fetch API
  - Retrieve data progressively
  - Send data progressively
- MSE API
  - Append stream
- WebRTC
  - Plan to use it

# ReadableStream API remarks

- ReadableStream : locked, cancel, getReader, pipeThrough, pipeTo, tee.
- Underlying source : start, pull, cancel.
- Strategy : highWaterMark, size
- Controller: enqueue, close, error, desiredSize
- Reader : closed, cancel, read and releaseLock

# How is it working?

```
function makeReadableBackpressureSocketStream(host, port) {
  const socket = createBackpressureSocket(host, port);

  return new ReadableStream({
    start(controller) {
      socket.ondata = event => {
        controller.enqueue(event.data);

        if (controller.desiredSize <= 0) {
          // The internal queue is full, so propagate
          // the backpressure signal to the underlying source.
          socket.readStop();
        }
      };

      socket.onend = () => controller.close();
      socket.onerror = () => controller.error(new Error("The socket errored!"));
    },

    pull() {
      // This is called if the internal queue has been emptied, but the
      // stream's consumer still wants more data. In that case, restart
      // the flow of data if we have previously paused it.
      socket.readStart();
    },

    cancel() {
      socket.close();
    }
  });
}
```

```
httpResponseBody.pipeThrough(decompressorTransform)
                .pipeThrough(ignoreNonImageFilesTransform)
                .pipeTo(mediaGallery);
```

```
function readAllChunks(readableStream) {
  const reader = readableStream.getReader();
  const chunks = [];

  return pump();

  function pump() {
    return reader.read().then(({ value, done })=> {
      if (done) {
        return chunks;
      }

      chunks.push(value);
      return pump();
    });
  }
}
```

# Where is it?

- Chrome
  - Shipped
    - ReadableStream tied to the Fetch API response
  - Ongoing
    - ReadableStream created by scripts
    - ReadableStream for progressive uploads using fetch
- Mozilla
  - Will start 25/12/2015 (roughly)
- IE
  - Support of an earlier version of stream/XHR as producer
- WebKit
  - ReadableStream fully implemented
    - pipeTo to be broken by the spec
  - WritableStream fully implemented

# Implementation Story

# First Approach – Initial steps

- C++ implementation
- Regular WebIDL to bind API with JavaScriptCore
  - Needed improved promise binding code
- Initial prototype supporting byte arrays
  - Nicely working
  - Not too complex

# First Approach – second steps

- Support of any JavaScript value
- WebIDL
  - Starting to add special cases in the binding generator
- Adding a lot of JS code in WebCore/bindings/js
  - Storing JS values, making them not collectable
  - Calling JS functions
  - Handling of asynchronous behavior, JS promises
- Overall conclusion
  - Code difficult to relate with the specification
  - Difficult to keep proper reference counting
  - Templates to add further specialization for byte array

# Second Approach – JS Builtins

- JS Builtin is a JavaScriptCore feature
  - Introduced a few years ago
  - Promise code is mostly JS builtin
- Enable JS Builtin into WebCore
  - Integrate it with WebIDL binding generator
- Streams API implementation
  - WebIDL code
  - JavaScript code
  - Some limited C++ code
    - 80 lines
    - Except for automatically generated code

# JS Builtins tied to WebIDL

- WebIDL

```
[
    Conditional=MEDIA_STREAM,
] partial interface Navigator {
    [JSBuiltin] void webkitGetUserMedia(Dictionary object, any successCallback, any errorCallback);
};
```

- JavaScript built-in

```
function webkitGetUserMedia(options, successCallback, errorCallback)
{
    "use strict";

    // FIXME: We should raise a DOM unsupported exception if there is no navigator and properly detect whether method is not called on a Navigator
    if (!(this.mediaDevices && this.mediaDevices.@getUserMediaFromJS))
        throw new @TypeError("The implementation did not support the requested type of object or operation.");

    if (arguments.length < 3)
        throw new @TypeError("Not enough arguments");

    if (options !== Object(options))
        throw new @TypeError("Argument 1 (options) to Navigator.webkitGetUserMedia must be an object");

    if (typeof successCallback !== "function")
        throw new @TypeError("Argument 2 ('successCallback') to Navigator.webkitGetUserMedia must be a function");
    if (typeof errorCallback !== "function")
        throw new @TypeError("Argument 3 ('errorCallback') to Navigator.webkitGetUserMedia must be a function");

    this.mediaDevices.@getUserMediaFromJS(options).then(successCallback, errorCallback);
}
```

# JS Builtins calling C++ methods

- Private keyword

```
interface RTCPeerConnection {

  // Private functions called by runQueuedOperation() (RTCPeerConnectionInternals.js)
  [Private] Promise queuedCreateOffer(optional Dictionary offerOptions);
  [Private] Promise queuedCreateAnswer(optional Dictionary answerOptions);
  [Private] Promise queuedSetLocalDescription(RTCSessionDescription description);
  [Private] Promise queuedSetRemoteDescription(RTCSessionDescription description);
  [Private] Promise queuedAddIceCandidate(RTCIceCandidate candidate);

  [Private] Promise privateGetStats(optional MediaStreamTrack selector);

  [JSBuiltin] Promise createOffer(optional Dictionary offerOptions);
```

```
// @conditional=ENABLE(MEDIA_STREAM)

function createOffer()
{
    "use strict";

    return @createOfferOrAnswer(this, this.@queuedCreateOffer, "createOffer", arguments);
}
```

# JS Builtins misc

- Conditional compilation
  - @conditional
- Constructor as JS built-in
- @assert
- JS built-in functions (helper routines) attached to the global object
  - @internal

# JS Builtins build

- Update CMakeLists.txt
  - Add IDL file in WebCore_NON_SVG_IDL_FILES
  - Add JS file(s) in WebCore_BUILTINS_SOURCES
    - 1 file for WebIDL tied routines (stored in  the prototype)
    - 0/1/+ files for helper routines (stored in the global object)
- Update Source/WebCore/bindings/js/WebCoreBuiltins.h and Source/WebCore/bindings/js/WebCoreBuiltins.cpp
  - When adding a new JS built-in file
  - To be automated soon hopefully

# Overall experience

- Easier to write JS code then to write C++ binding code
  - No more crashes, no more memory leaks, no more ref-counting cycles
- Performances is not really an issue
  - Apple made measurements on the Promise implementation and saw some improvements
- Everything is nice, except…
  - No JS built-in code debugger
    - Back to console.log(“potato 1”);
  - JS builtin security issues

# Security issues

- JS builtins run in the same world with the same GlobalObject as user scripts
  - Modifying a prototype or a user object may affect JS built-in code
- First possibility:  JS built-in code may break

```
this.mediaDevices.@getUserMediaFromJS(options).then(successCallback, errorCallback);
```

  - What if mediaDevices is overridden by user scripts?
  - What if mediaDevices prototype is changed.

# Security issues – leaking information

```
function enqueueOperation(peerConnection, operation)
{
    "use strict";

    if (!peerConnection.@operations)
        peerConnection.@operations = [];

    var operations = peerConnection.@operations;

    function runNext() {
        operations.shift();
        if (operations.length)
            operations[0]();
    };

    return new @Promise(function (resolve, reject) {
        operations.push(function() {
            operation().then(resolve, reject).then(runNext, runNext);
        });

        if (operations.length == 1)
            operations[0]();
    });
}
```

```
const goodOldPush = Array.prototype.push;
Array.prototype.push = function () {
    makeMyEvilThingWithPotatoes(arguments);
    return goodOldPush.apply(this, arguments);
};
```

```
const goodOldThen = Promise.prototype.then;
Prototype.prototype.then = function () {
    stealingYourPotatoes(arguments);
    return myEvilPromise(this, arguments);
};
```

# Security issues – current rules

- Do not use functions under the control of the user

```
operations.push(...)
operations.@push(...)
```

- Do not use prototype of objects under the control of the user

```
function processValue(promise) {
    promise.then(...)
}
function processValue(promise) {
    promise.@then(...) // ok but it may break
}
function processValue(promise) {
    @Promise.prototype. @then.@call(...) // ok but so unreadable
}
```

- Beware of Promise
  - Might want to use InternalPromise if you are doing chaining

# Security issues – we need something better

- So easy to fall into that trap
  - Not so easy to find the holes
- How can we improve the situation?
  - Testing tool to catch these errors
  - JS builtin check style?
  - JS proxy object in Debug mode to control how are accessed objects
  - Sanitizers
- Should we change the infrastructure?
  - Run the built-ins in a more secure environment
    - Chrome is doing that in a completely separate world
      - Cannot pass promises, JS objects.... Between the worlds
- Input most welcome

# Tentative conclusion

- Streams API is
  - Very maintainable
  - Fast enough (more study needed)
- But
  - Potential security issues
    - We fixed the ones we know of
- Need to improve JS built-in tooling
- Think about JS built-ins when adding WebCore/JSC specific code
  - Like in WebCore/bindings/js