JavaScript

...

De la especificación a la implementación y una vuelta

Daniel Minor
Staff SpiderMonkey Engineer / Mozilla
https://github.com/dminor

Una breve introducción

Soy un desarrollador de software para Mozilla, en el equipo de SpiderMonkey

Mozilla

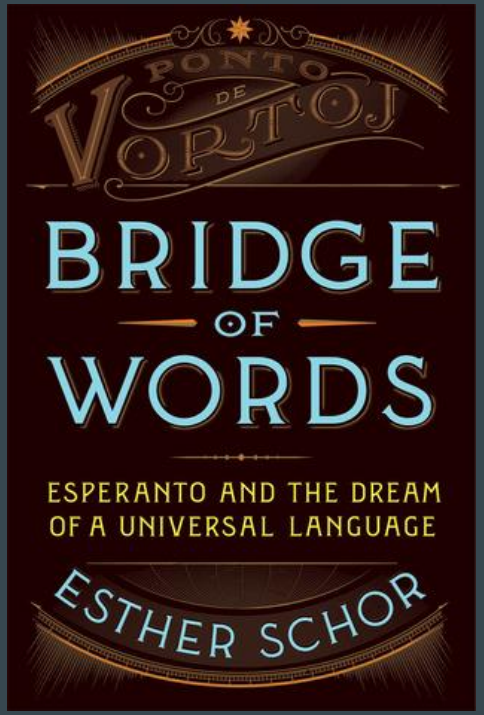
- Una organización sin ánimo de lucro
- Desarrollamos el navegador de web Firefox

SpiderMonkey

- Nuestro motor del lenguaje de JavaScript en Firefox
- Un motor es un intérprete de JavaScript dentro de una aplicación

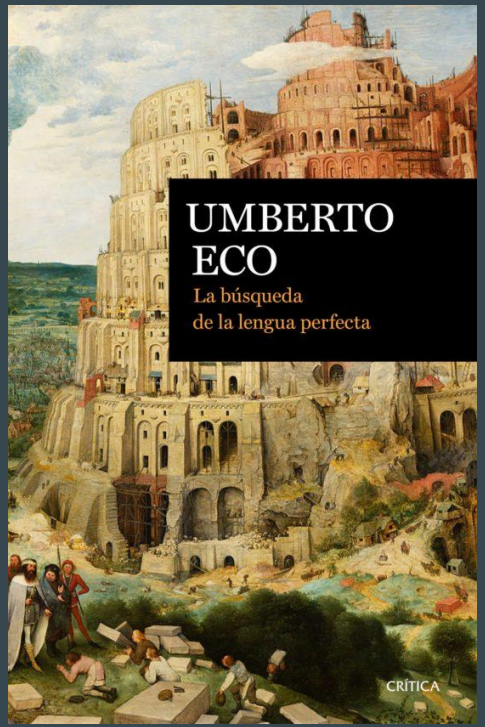
¿Por qué nos interesan los lenguajes?

El aspecto humano: la búsqueda de la lengua perfecta

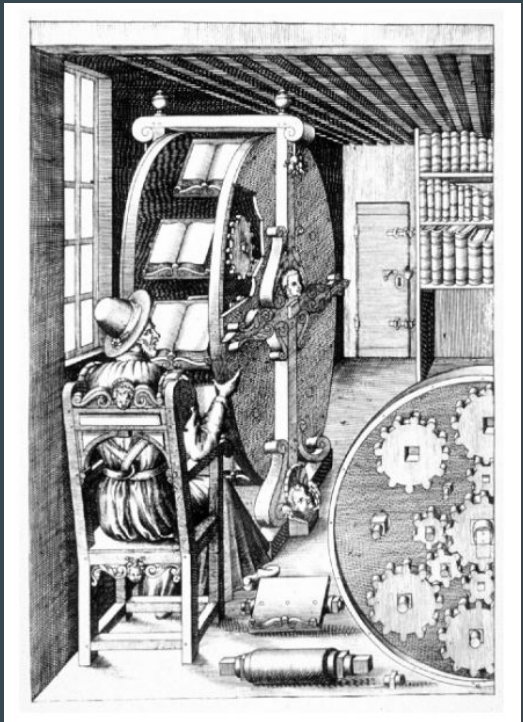


Siglos de

- Filosofos
- matemáticos
- soñadores
- y una miscelánea de locos



Conceptos técnicas: los intérpretes son programas fascinantes



Llenos de conceptos profundos de computación y matemáticas:

- Lógica y lenguajes formales
- Parsers y árboles de sintaxis abstracta
- Sistemas de tipos
- Compilación just in time (JIT)


It's alive! It's alive!
VINTAGEGAL

La Agenda de hoy

Quiero que vayamos con una propuesta paso a paso de la idea al estándar de JavaScript:

- Cómo funciona TC39, el comité de estandarización
- Las etapas de una propuesta en el comité
- La implementación de una propuesta
- Cómo pueden contribuir en el proceso

TC39

¿Qué es TC39?

Comité Técnico 39 de Ecma International

Es responsable de ECMAScript, el estándar para JavaScript

Un estándar es un acuerdo de la funcionalidad de un sistema:

- Permite que se puedan crear implementaciones independientes
- Evita problemas con derechos de propiedad intelectual

¿Cómo funciona el comité?

Está formado por un grupo de empresas* que cumplen con dos requisitos:

- están interesados en el futuro de JavaScript
- y que pueden pagar por la afiliación :)

Cada empresa puede enviar sus delegados a las reuniones del comité

En dichas reuniones las decisiones se toman por consenso

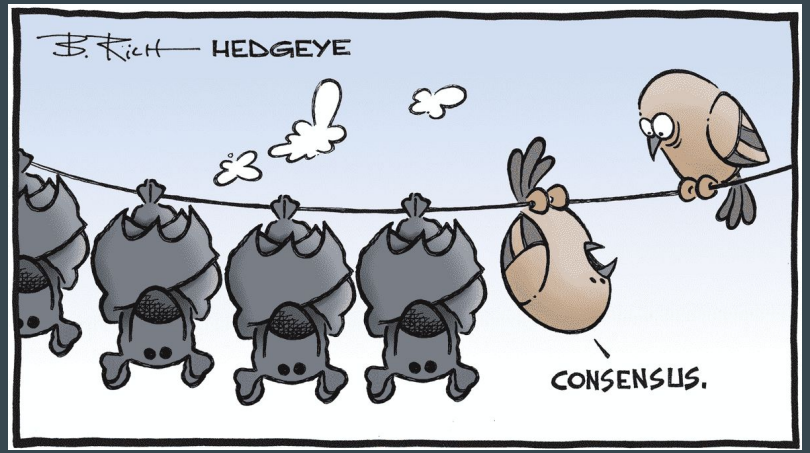
* y organizaciones sin ánimo de lucro, como Mozilla, OpenJS y universidades

¿Cómo funciona el Consenso?

Cada delegado tiene el poder de bloquear una propuesta, pero –

¡Con un gran poder viene una gran responsabilidad!

El bloqueo de una propuesta en el comité es un hecho trascendental



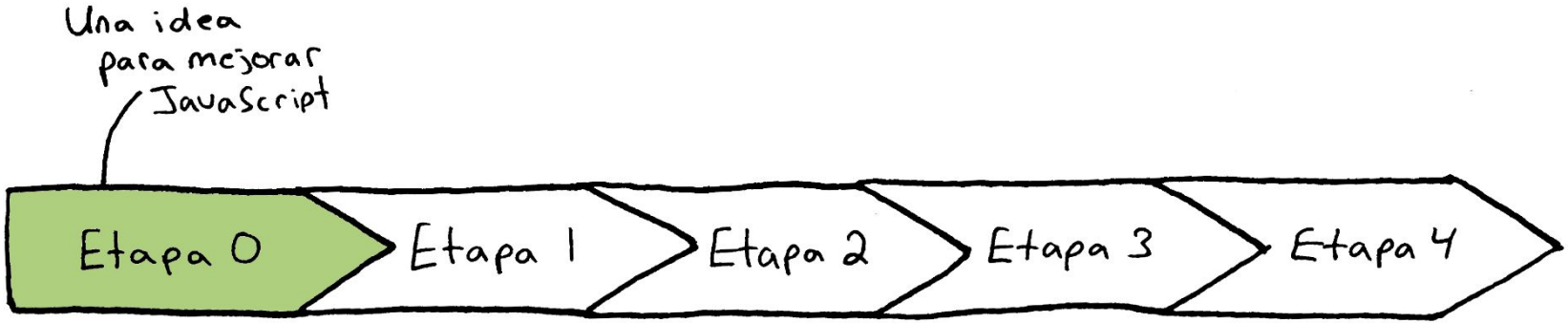
¿Entonces, cómo funciona el comité?

El comité funciona con mucho tacto, por

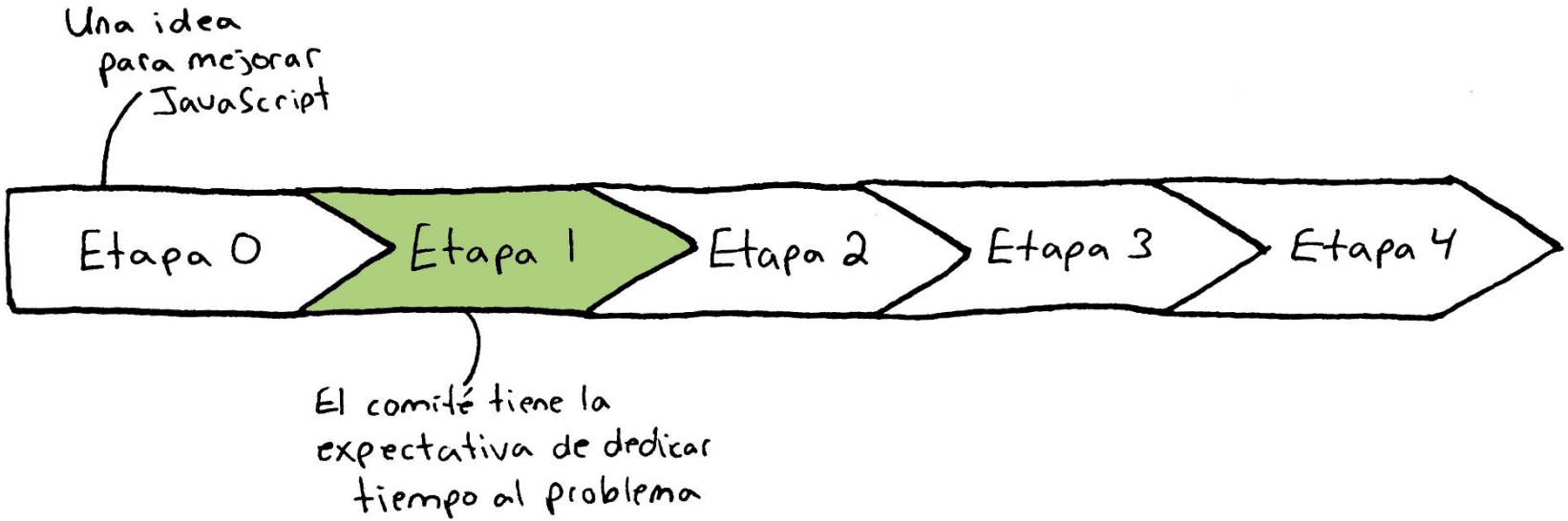
- sugerencias y modificaciones de las propuestas
- acuerdos de compromiso
- conversaciones fuera de las reuniones del comité

Las etapas de una propuesta

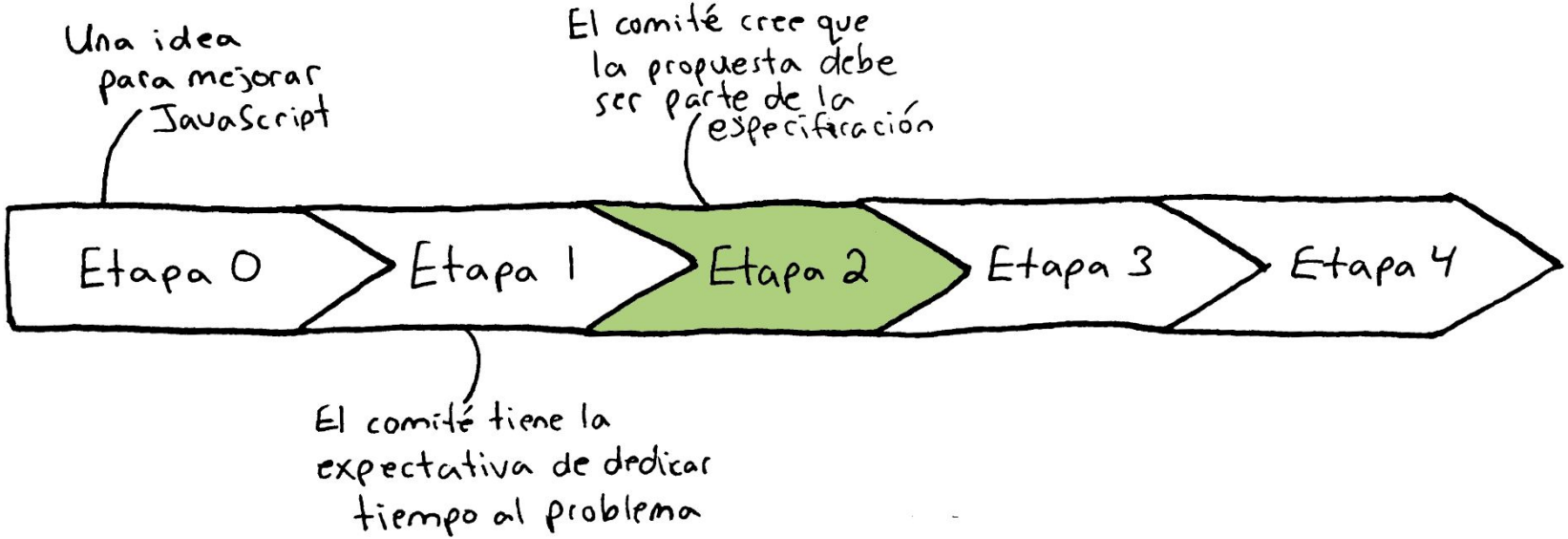
Las cinco etapas de una propuesta



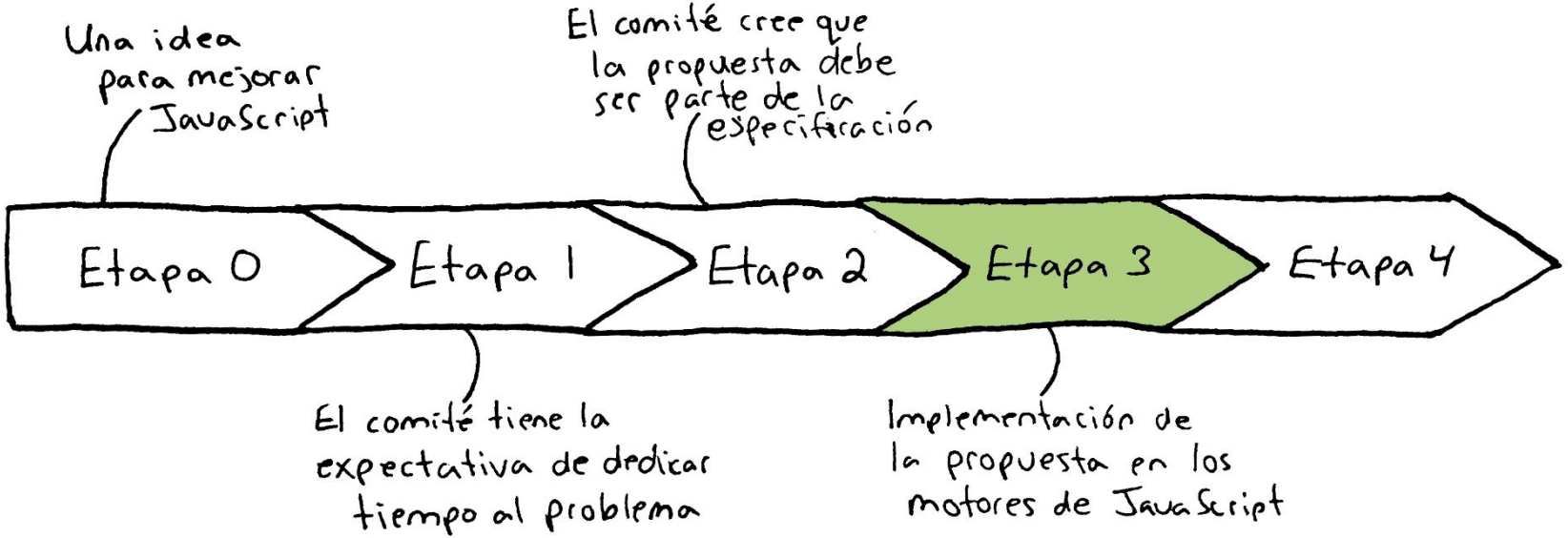
Las cinco etapas de una propuesta



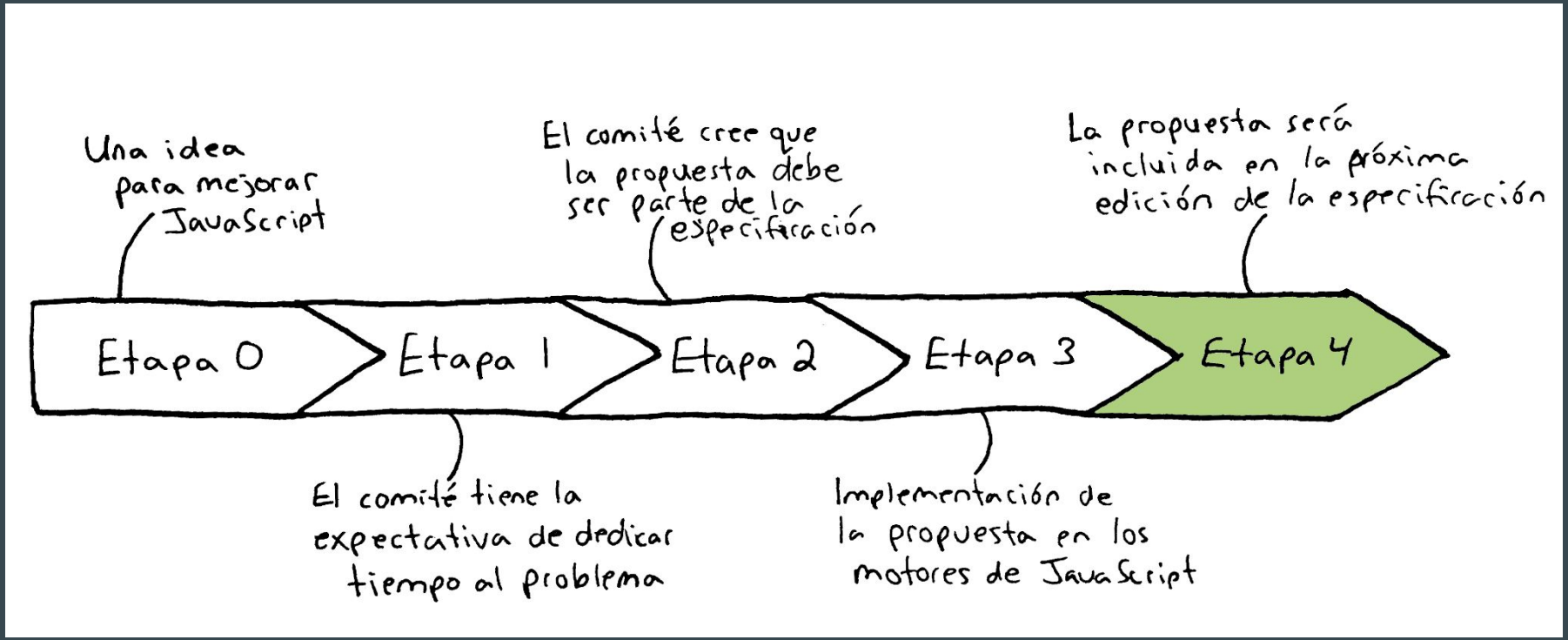
Las cinco etapas de una propuesta



Las cinco etapas de una propuesta



Las cinco etapas de una propuesta



¿Por qué una propuesta no avanza?

Pueden existir muchas razones:

- La idea no está clara
- No hay una justificación suficiente para el cambio
- El propósito de la propuesta ha cambiado desde etapas anteriores
- No se puede implementar la propuesta eficientemente
- etc, etc.

Cada delegado tiene su propio punto de vista sobre si una propuesta debería avanzar

El punto de visto de SpiderMonkey

Somos conservadores con nuevos features:

- Tenemos que pagar el costo de mantener un feature nuevo en el motor

Nuestros usuarios son todas las personas que usan Internet:

- No sólo desarrolladores de JavaScript
- No sólo personas que viven en Norteamérica o Europa
- La compatibilidad web es muy importante para nosotros

Por ejemplo: Array.GroupBy

La propuesta fue creada para facilitar la creación de grupos en Arrays

```
const array = [1, 2, 3, 4, 5];

array.group((num, index, array) => {
  return num % 2 === 0 ? 'par': 'impar';
});

// =>  { impar: [1, 3, 5], par: [2, 4] }
```

Una breve historia de Array.GroupBy

Comenzó con la idea de añadir Array.filterReject

- Evitar confusión si Array.filter mantiene o descarta elementos
- Avanzó a la etapa 1 para explorar cómo mejorar la filtración de un Array

El “champion” presentó dos ideas al comité para avanzar a la etapa 2

- Array.filterReject
- Array.groupBy

Al comité le gustó más la idea de Array.groupBy y avanzó a las etapas 2 y 3

La implementación en SpiderMonkey

Tuvimos la primera implementación de la propuesta

- Fue desarrollada por voluntarios de la Universidad de Bergen 🥰

Para esta propuesta, casi todo el código estaba desarrollado en JavaScript

- Las partes de SpiderMonkey que están desarrolladas en JavaScript se llama código “self-hosted”
- Muchas veces la implementación de una propuesta solo necesita conocimiento de JavaScript

Problemas con compatibilidad de web

Después de un rato encontramos un problema serio en Firefox:

- No se puede abrir PagerDuty.com :(
- Rápidamente, descubrimos que fue por Array.GroupBy!

PagerDuty

PagerDuty usa la librería Sugar.js

Muchas versiones de Sugar.js modifican el prototipo de Array con una implementación incompatible de GroupBy

Eso se llama “monkey patching” en inglés

Un cambio de nombre: a `group`

Aunque el problema ya habían arreglado en versiones nuevas de Sugar.js:

- No podemos romper la web para usuarios que usan sitios con un versión afectada de Sugar.js

Más de 1000 sitios tenía este problema

- Intentaba contactar todos, pero no tenía una respuesta suficiente

Por eso, el nombre fue cambiado a`group`

- Este principio de “Don’t break the web” es muy importante para el comité

Y aún más problemas con compatibilidad de web

Después de otro rato, encontramos aún más problemas con `group`:

- LastPass no funciona con el nombre `group`
- Tampoco IBMCloud ni alltron.ch ni ... ?

Los síntomas en todos los casos son que la página web no termina de abrir

- No era muy evidente que Array.group era el problema

Y ahora?

- No queremos cambiar el nombre otra vez, y encontrarnos con que habrán más problemas
- Una nueva idea es añadir un método estático de Array
- Quizás la propuesta vaya a volver a la segunda etapa por eso

```
const array = [1, 2, 3, 4, 5];

Array.group(array, (num, index, array) => {
  return num % 2 === 0 ? 'par': 'impar';
});

// =>  { impar: [1, 3, 5], par: [2, 4] }
```

La implementación de una propuesta

Por ejemplo: la implementación de ‘decorators’

Mi primer proyecto grande en SpiderMonkey

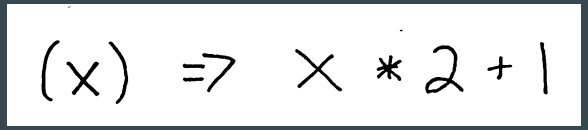
Un buen ejemplo de trabajar con el frontend de un motor:

- Tokenizer
- Parser
- Generación de bytecode

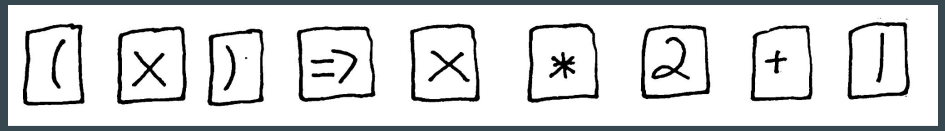
El tokenizer

Un ‘token’ es una parte indivisible en el input de un intérprete

El ‘tokenizer’ divide el texto del input:

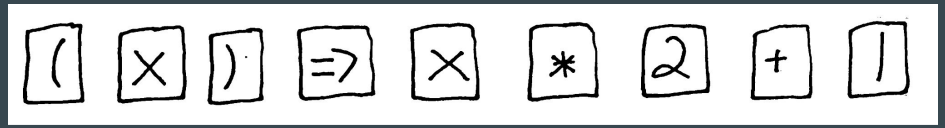


en una lista de tokens:

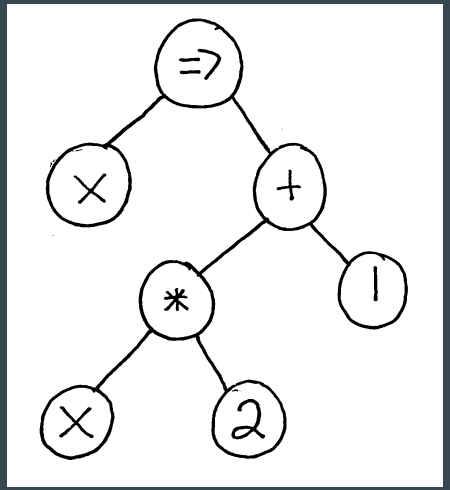


El parser

El parser usa esta lista de tokens y la gramática del lenguaje

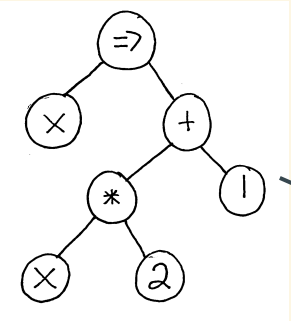


para construir un árbol de sintaxis abstracta:



Generación de ‘bytecode’

AST (entrada)




```
00000:   1  Int8 2                    # 2
00002:   1  GetArg 0                  # 2 x
00005:   1  Mul                       # (2 * x)
00006:   1  One                       # (2 * x) 1
00007:   1  Add                       # ((2 * x) + 1)
00008:   1  Return                    #
```

Bytecode (salida)

Generación
de bytecode

Intérprete de bytecode
(consumidor)

¿Qué es un decorator?

```
// El ejemplo icónico de un decorator, va a escribir ¡saludos!
// cuando la clase es creada...
function decorator(value, { kind, name }) {
 console.log("¡saludos!");
}

class Class {
 @decorator method(x) { return x + 1; }
}
```

¿Qué es un decorator?

```
// También podemos reemplazar el método con algo nuevo...
function decorator(value, { kind, name }) {
 return (x) => x * 2 + 1;
}

class Class {
 @decorator method(x) { return x + 1; }
}
```

Un token nuevo para decorators

Los tokens están en una lista en TokenKind.h en el código de SpiderMonkey

- Más o menos solo tuve que añadir el token para `@` allí

También hice cambios en el ‘build system’ para controlar el feature

```
MACRO(String, "string literal")                                  \
MACRO(BigInt, "bigint literal")                                  \
IF_DECORATORS(MACRO(At, "'@'"))                                  \
                                                                 \
/* start of template literal with substitutions */               \
MACRO(TemplateHead, "'${'")                                      \
/* template literal without substitutions */                     \
MACRO(NoSubsTemplate, "template literal")                        \
```

Cambios del parser para decorators

Tuve que construir una lista de los decorators para cada propiedad, método y clase en la implementación del parser en Parser.cpp.

```
#ifdef ENABLE_DECORATORS
template <class ParseHandler, typename Unit>
typename ParseHandler::ListNodeType
GeneralParser<ParseHandler, Unit>::decoratorList(YieldHandling yieldHandling) {
  ListNodeType decorators =
      handler_.newList(ParseNodeKind::DecoratorList, pos());

  // Build a decorator list element. At each entry point to this loop we have
  // already consumed the |@| token
  TokenKind tt;
  for (;;) {
    if (!tokenStream.getToken(&tt, TokenStream::SlashIsInvalid)) {
      return null();
    }

    Node decorator = null();
    if (tt == TokenKind::LeftParen) {
      // Handle DecoratorParenthesizedExpression
      decorator = exprInParens(InAllowed, yieldHandling, TripledotProhibited);
```

Generación de ‘bytecode’ para decorators

Es la parte más complicada y todavía estoy trabajando con esa :)

```
// 5. For each element decorator of decorators, do
for (ParseNode* decorator : decorators->contents()) {
  //     5.a. Let decorationState be the Record { [[Finished]]: false }.
  if (!emitDecorationState()) {
    return false;
  }

  // Prepare to call decorator
  CallOrNewEmitter cone(bce_, JSOp::Call,
                        CallOrNewEmitter::ArgumentsKind::Other,
                        ValueUsage::WantValue);

  // DecoratorMemberExpression: IdentifierReference e.g. @dec
  if (decorator->is<NameNode>()) {
    if (!cone.emitNameCallee(decorator->as<NameNode>().name())) {
```

Generación de ‘bytecode’ para decorators

A modo de resumen, para cada decorator hay que:

1. Generar bytecode para invocar al decorator con los argumentos necesarios
2. Generar bytecode para examinar el resultado de la invocación:
    a. Si es ‘undefined’, puede descartar el resultado
    b. Si es una función, tiene que almacenar el resultado en algún lugar
        i. Ejemplo: reemplazar un método con el resultado
    c. Si es algo diferente, es un TypeError

¿Cómo puedes contribuir?

¿Por qué pensarías en contribuir?

Mozilla es una organización sin ánimo de lucro y desde el principio, hemos tenido un manifiesto que es nuestro guía

Este principio en el manifiesto de Mozilla siempre me llama la atención:

Principio 2

> Internet es un recurso público mundial que debe permanecer abierto y accesible.

JavaScript: también un recurso público mundial

¡JavaScript es importante para todas las personas que usan internet!

- No solo los desarrolladores de JavaScript
- No solo empresas que son de Norteamérica o Europa

A menudo, muy pocas personas toman las decisiones sobre el futuro de JavaScript

¡Necesitamos las ideas y participación de más personas!

¿Cómo puedes contribuir al estándar?

Las reuniones del comité de TC39 están cerradas al público, pero no importa

Todas las propuestas están en GitHub: https://github.com/tc39/proposals

También el trabajo técnico importante ocurre allí

- Por favor, lee las propuestas
- Abre issues con comentarios o dudas

¿Cómo puedes contribuir al estándar?

Hay reuniones abiertas: https://github.com/tc39/js-outreach-groups

- Educators
- Tools and transpilers

Puedes contactar al comité en este canal de Matrix:

- https://matrix.to/#/#tc39-general:matrix.org

¿Tienes ganas de escribir código?

Puedes crear un ‘polyfill’ para una nueva propuesta

Puedes contribuir una implementación para Babel

Puedes escribir nuevas pruebas en https://github.com/tc39/test262

¿Cómo puedes contribuir a SpiderMonkey?

Muchas de las propuestas de TC39 son implementadas por voluntarios

Puede ser una buena manera de aprender cómo funciona un motor de JavaScript

En Bug 1435811 de bugzilla.mozilla.org:

- Hay una lista de todas las propuestas en la tercera etapa
- Puedes buscar las propuestas que todavía necesitan una implementación

¿Cómo puedes contribuir a SpiderMonkey?

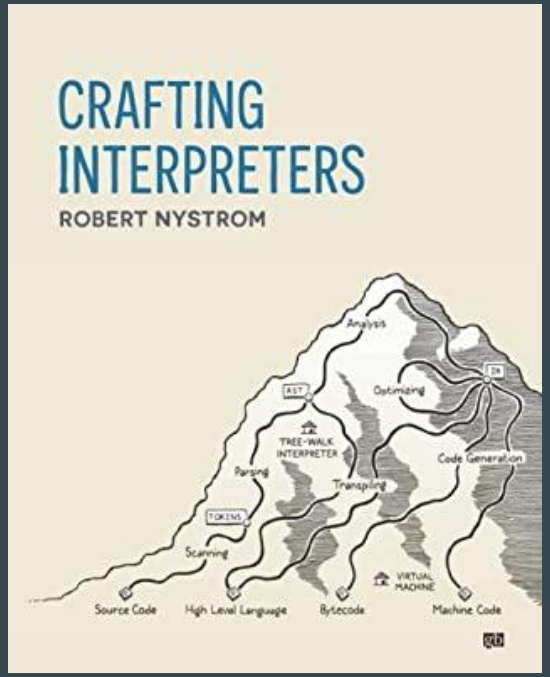
Conocimiento necesario para trabajar con el frontend de SpiderMonkey:

- C++ y JavaScript
- Parsers de descenso recursivo
- Máquinas virtuales

No tienes que ser experto :)

Puedes contactarnos en nuestro canal de Matrix:

- #spidermonkey:mozilla.org



https://craftinginterpreters.com/

¡Muchas gracias!

Muchísimas gracias al equipo de JSConf Chile por la oportunidad de compartir mis experiencias y pensamientos

Gracias a todas las personas que me ayudaban preparar la presentación

Y gracias a ustedes por su tiempo y atención :)

¿Dudas o comentarios?

- dminor@mozilla.com
- https://github.com/dminor
- https://www.linkedin.com/in/daminor/