Definir el Estado del Arte (3 de Diciembre 2013)

Avance para el 19 de noviembre:

Nota: las paginas webs de este documento están consultadas durante el fin de semana de l 17/18 de noviembre

Ideas: Investigar en este orden V8 y Node.js al principio (debido a su centramiento en el motor). Al final Jquery y Kendo (debido a q están centrados en las interfaces, el ultimo usa jquery). Al medio QOOXDOO.

De cada uno investigar:

Que es?

Plataforma?

Sistema operativo

Arquitectura/diseño?

Características

Ejemplo de hola mundo para interiorizar

-------------------------------

**V8 Engine**: el código fuente se puede bajar desde un repositorio ~~SVN~~ (<http://code.google.com/p/v8/wiki/Source>) o git (se recomienda este ultimo)

En su diseño cuenta con 3 partes clave para su alto rendimiento:

1. Acceso rápido a sus propiedades (explicar ejemplo de los puntos en el video), uso de clases invisibles <https://developers.google.com/v8/design#prop_access>

JavaScript is a dynamic programming language: properties can be added to, and deleted from, objects on the fly. This means an object's properties are likely to change. Most JavaScript engines use a dictionary-like data structure as storage for object properties - each property access requires a dynamic lookup to resolve the property's location in memory. This approach makes accessing properties in JavaScript typically much slower than accessing instance variables in programming languages like Java

To reduce the time required to access JavaScript properties, V8 does not use dynamic lookup to access properties. Instead, V8 dynamically creates hidden classes behind the scenes. This basic idea is not new - the prototype-based programming language Self used maps to do something similar. (See for example, [An Efficient Implementation of Self, a Dynamically-Typed Object-Oriented Language Based on Prototypes](http://research.sun.com/self/papers/implementation.html)). In V8, an object changes its hidden class when a new property is added.

Ejemplo.

Función punto en javascript y v8 que utiliza clases ocultas

1. Generación dinámica de código

V8 compiles JavaScript source code directly into machine code when it is first executed. There are no intermediate byte codes, no interpreter. Property access is handled by inline cache code that may be patched with other machine instructions as V8 executes.

Al agregar atributos, no se compila todo de nuevo, sino que se van cambiando los trozos de código entremedio (o almenos eso entiendo del párrafo anterior)

En caso que exista una coincidencia, solo se retorna el valor. Esta ventaja es más notoria cuando hay muchas instancias de un objeto que desea acceder a las clases ocultas.

During initial execution of the code for accessing a property of a given object, V8 determines the object's current hidden class. V8 optimizes property access by predicting that this class will also be used for all future objects accessed in the same section of code and uses the information in the class to patch the inline cache code to use the hidden class. If V8 has predicted correctly the property's value is assigned (or fetched) in a single operation. If the prediction is incorrect, V8 patches the code to remove the optimisation.

1. Eficiente colector de basura

V8 reclaims memory used by objects that are no longer required in a process known as garbage collection. To ensure fast object allocation, short garbage collection pauses, and no memory fragmentation V8 employs a stop-the-world, generational, accurate, garbage collector. This means that V8:

* Detiene a ejecucon de los programas cuando comienza el ccilco de recoleccion de basura
* Procesa solo aquellas partes que quedan en la categrria d basura la mayor parte de los ciclos?
* Sabe exactamente donde estan los punteros (en memoria), lo cual evita falsos positivos que puedan desencadenar en problemas de memoria.
* stops program execution when performing a garbage collection cycle.
* processes only part of the object heap in most garbage collection cycles. This minimizes the impact of stopping the application.
* always knows exactly where all objects and pointers are in memory. This avoids falsely identifying objects as pointers which can result in memory leaks.

In V8, the object heap is segmented into two parts: new space where objects are created, and old space to which objects surviving a garbage collection cycle are promoted. If an object is moved in a garbage collection cycle, V8 updates all pointers to the object.

Una de las metas del proyecto es poder aumentar el uso de js en los navegadores.

Usado en chrome

Información: <https://developers.google.com/v8/intro>

Si se quiere probar: <https://developers.google.com/v8/build>

Si se quiere probar una aplicación stad alone <https://developers.google.com/v8/get_started>

V8 implements ECMAScript as specified in ECMA-262, 5th edition, and runs on Windows (XP or newer), Mac OS X (10.5 or newer), and Linux systems that use IA-32, x64, or ARM processors.

JavaScript is most commonly used for client-side scripting in a browser, being used to manipulate Document Object Model (DOM) objects for example. The DOM is not, however, typically provided by the JavaScript engine but instead by a browser. The same is true of V8—Google Chrome provides the DOM. V8 does however provide all the data types, operators, objects and functions specified in the ECMA standard.

V8 enables any C++ application to expose its own objects and functions to JavaScript code. It's up to you to decide on the objects and functions you would like to expose to JavaScript

**Node.js**

<http://www.nodebeginner.org/index-es.html>

<http://nodejs.org/api/>

<http://msdarkici.wordpress.com/2011/06/17/%C2%BFque-es-node-js/>

Existe una función que permite descubrir sistemas operativos (usable en acscb quizas con v8 y c++ standalone)

Node.js surge como una nueva forma de aprovechar la experiencia con Javascript, esta vez, del lado del servidor

En relación al hola mundo  many client connections can be handled concurrently. Node tells the operating system (throughepoll, kqueue, /dev/poll, or select) that it should be notified when a new connection is made, and then it goes to sleep. If someone new connects, then it executes the callback. Each connection is only a small heap allocation.

This is in contrast to today's more common concurrency model where OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use.

HTTP is a first class protocol in Node. Node's HTTP library has grown out of the author's experiences developing and working with web servers. For example, streaming data through most web frameworks is impossible. Node attempts to correct these problems in its HTTP [parser](https://github.com/joyent/http-parser/tree/master) and API. Coupled with Node's purely evented infrastructure, it makes a good foundation for web libraries or frameworks.

Modelo de ejecución

El desarrollador ve cada proceso del servidor Node.js como un único thread que sirve todas las peticiones que recibe, aquí todo es serializado en un único bucle de eventos, a diferencia de servidores como apache que generan un nuevo thread por cada petición, un servidor escrito en Node.js genera un nuevo evento dentro del mismo thread. Esto permite no preocuparse por memoria compartida u otros, disminuyendo el tiempo de desarrollo y los bugs.

El crear nuevos threads (como apache u otros) es un proceso mucho mas “pesado” para el S.O. que generar un nuevo evento en Node.js, es decir, la performance de un servidor web creado con Node.js cuando hay muchas peticiones es mejor que en un servidor normal.

Desde dentro Node.js contiene un pool autogestionado de thread, sobre el  se lanzan todas las tareas que requieren algún tipo de I/O o alguna tarea bloqueante. Así, Node.js es monothread para el desarrollador, todas las llamadas que impliquen una cierta espera (procesos bloqueantes) son invocados de forma asíncrona, proporcionando un manejador de callback o un suscriptor a eventos.

Ultima versión estable <http://blog.nodejs.org/>

**jQuery**

**Kendo**

**QOOXDOO**