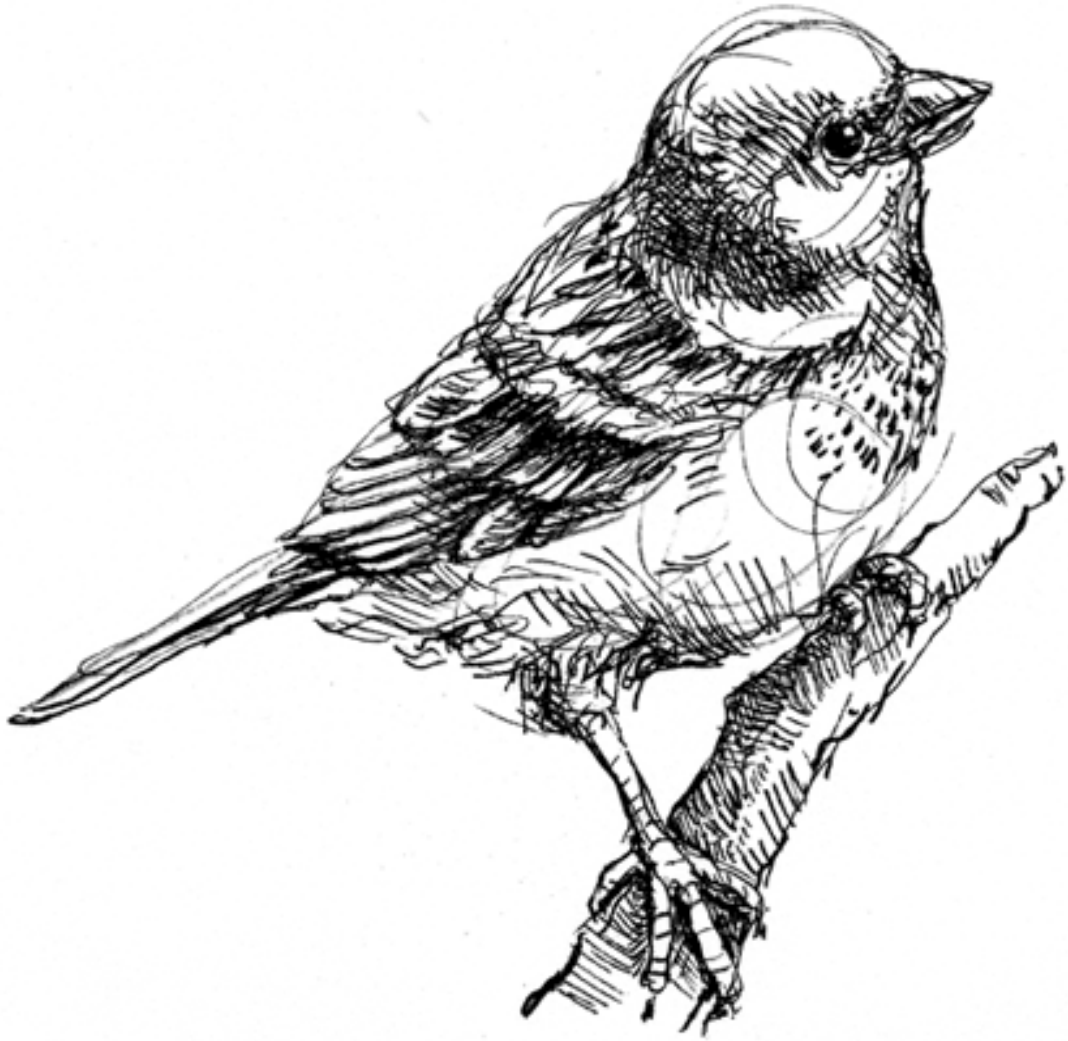
Un nuevo lenguaje para construir aplicaciones Web Estructuradas



¿Qué es Dart?

Germán Quiróz Bogner

¿Qué es Dart?

Basado en Material escrito por Seth Ladd & Kathy Walrath.
Traducido y actualizado por: Germán Quiróz Bogner y Fernanda Goñi.
Santa Fe de la Vera Cruz, Santa Fe, Argentina.

Marzo 2014.

Tabla de Contenidos

¿Qué es Dart?

Dart es un nuevo lenguaje desarrollado por Google que está llamando la atención en los círculos de desarrolladores de aplicaciones web.

¿Por qué Google Creó Dart?

El objetivo de Google es que tu puedas crear grandes aplicaciones web. Grandes aplicaciones web mejoran la web, y cuando la web mejora, todo el mundo gana.

Los ingenieros de Google han estado pensando en las aplicaciones web durante mucho tiempo. Han escrito un montón de aplicaciones web a gran escala, complejas y ampliamente utilizadas (crearon GMail, Google+ y Google Docs), así que están muy familiarizados con los retos de las arquitecturas de las aplicaciones web.

También han escrito un navegador (Chrome) y un motor de JavaScript (V8), por lo que han pensado mucho sobre cómo hacer que las aplicaciones web funcionen más rápido.

Básicamente, crearon Dart porque creen que va a ayudar a traer más aplicaciones muy buenas para la web, y creen que debería ser más fácil para crear aplicaciones web más complejas.

¿Puede la Web realmente necesitar otro lenguaje?

Los desarrolladores de Aplicaciones de todas las plataformas deben ser capaces de construir para la web moderna, pero los desarrolladores web no endémico tienen diferentes expectativas y necesidades de su lenguaje, herramientas, y la plataforma de desarrollo de lo que está disponible hoy en día. Los ingenieros de Google creen que hay espacio para una nueva plataforma, una que no está cubierta de 15 años de polvo, que es familiar para los desarrolladores de diferentes orígenes, y que está estructurado para las largas y complejas aplicaciones que los usuarios están demandando.

Ellos no creen que JavaScript se va a acabar. Al contrario, Google trabaja de manera activa junto al TC39 para mejorar JavaScript, y nuevas características que comienzan a implementarse en V8 y Chrome.

Sin embargo, el compromiso en mejorar JavaScript no les impide en pensar en otras soluciones. Por ejemplo, Dart aprovecha el continuo trabajo sobre JavaScript, ya que los programas en Dart compilan a JavaScript para correr sobre toda la web moderna.

Ellos creen que Dart es una plataforma irresistible y familiar que conoce las necesidades de los desarrolladores de diferentes orígenes y experiencias, incluyendo a los endémicos desarrolladores web. Dart brinda nuevas y frescas ideas sobre hacia dónde va la programación web, y qué innovaciones tanto en Dart como JavaScript pueden ayudar a empujar la web de los desarrolladores de aplicaciones hacia los usuarios.

Muéstrame el Código

Suficiente charla, vamos a ver algo de código. Si sabes JavaScript o Java, el código de Dart debería resultarte familiar. He aquí un ejemplo de una pagina web sencilla en Dart:

hola.dart

```
#import('dart:html');

main(){
      document.query('#status').text = 'Hola, Dart';
}
```

hola.html

```
...
<h2 id="status"></h2>
<script type="application/dart" src="hola.dart"></script>

<!--
  Si el navegador no tiene embebida la VM de Dart,
tu puedes compilar el código de Dart a JavaScript.
-->
<script type="text/javascript" src="hola.dart.js"></script>
...
```

Ahora vamos a ver un poco del código de Dart que usa funciones:

```
enviar(msj, destinatario, emisor, [rate='Primera Clase']) {
      return '$emisor dice $msj a $destinatario via $rate';
}

main() => print(enviar('hola', 'German', 'Fer'));
```

> "German dice hola a Fer via Primera Clase"

El uso de `=>` en la sintaxis implementado a `main()` esta bueno, es una manera compacta de implementar una función que evalúe y retorne un expresión simple. Sin el uso de `=>`, la implementación del método `main()` luciría de la siguiente manera:

```
main() {
      print(enviar('hola', 'German', 'Fer'));
}
```

En el método `enviar()` implementado abajo, *`rate`* es un parámetro opcional con un valor por defecto. Este método muestra la interpolación de String trabajando `($var)`.

Aqui un poco de código Dart un poco más orientado a objetos:

```
class Punto {
```

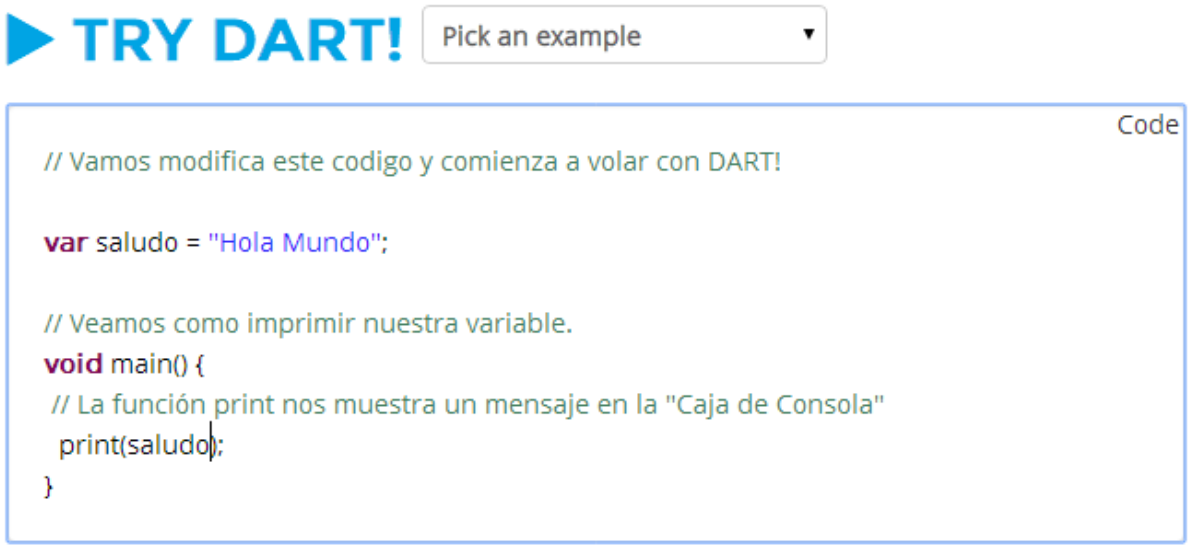
```
        Punto(this.x, this.y);
        distanciaA(otro) {
              var dx = x - otro.x;
              var dy = y - otro.y;
              return Math.sqrt(dx*dx+dy*dy);
        }
        var x, y;
}

main() {
        var p = new Punto(2, 3);
        var q = new Punto(3, 4);
        print('Distancia de p a q = $p.distanciaA(q)');
}
```

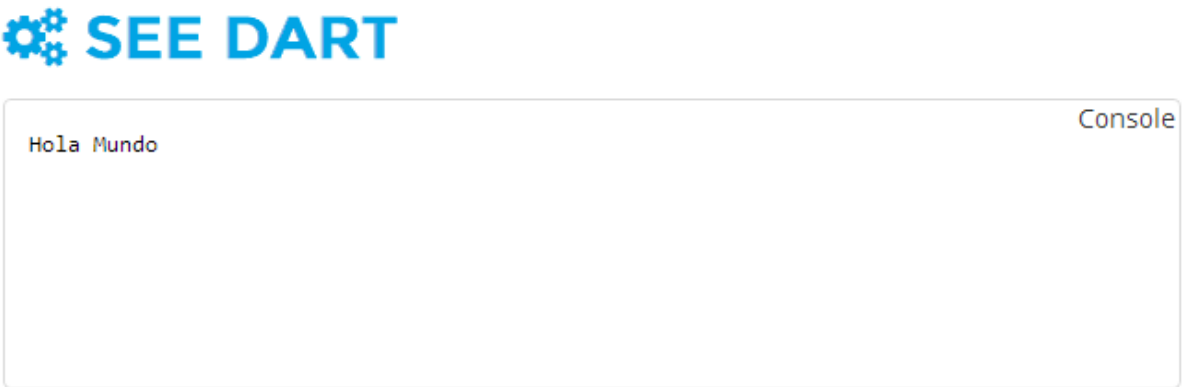
Éste código puede parecer un poco familiar si tu has usado un lenguaje basado en clases con anterioridad.

¿Cómo puedo jugar con Dart?

La manera más fácil de probar Dart es empleando Dartboard, un modo de ejecutar el código de Dart en cualquier navegador moderno. Dartboard esta integrado en el sitio de Dart, en www.dartlang.org. Aunque también se puede usar Dartboard yendo directamente a try.dartlang.org. Aquí hay una imagen de lo que verás en el sitio:



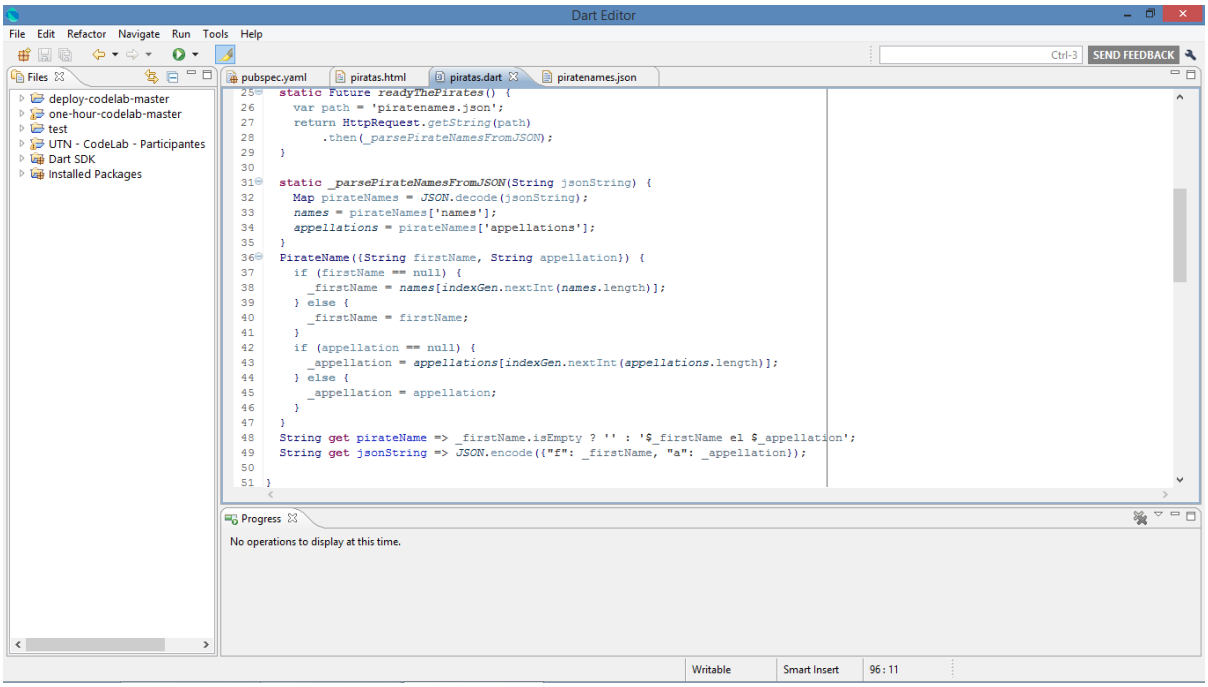
Cambia el código como quieras, automáticamente irá compilando tu código y mostrará el resultado a la derecha, se verá algo así:



Como todas las cosas relacionadas con Dart, Dartboard sigue cambiando. Para la última información se recomienda ver el tutorial de Dartboard.

¿Qué tal un editor Real?

Cuando superes Dartboard, intenta Dart Editor. Es un editor descargable para Windows, Mac y Linux que te permite escribir, modificar y ejecutar aplicaciones web Dart. Dart Editor puede ejecutar tu código Dart vía la VM (Máquina Virtual), o puede compilarlo a JavaScript y probarlo en tu navegador. Aquí hay una imagen de cómo luce actualmente Dart Editor:



Para ejecutar cualquier programa en Dart Editor, solamente clickea el botón Run mientras seleccionas algun item en esa galería de programas. Si tu programa es una aplicación web, Dart Editor presenta Dartium (Chromium con una Dart VM integrada) para ejecutar el programa. Cuando estés listo para pasar del desarrollo a producción Dart Editor puede compilarlo a JavaScript haciendo que tu aplicación sea accesible para toda la web moderna.

Dart Editor tiene diversas características para ayudarte con el código Dart, se esperan más características pronto. Como puedes ver en la captura de pantalla Dart Editor resalta la sintaxis de Dart. Dart Editor también soporta autocompletado, y rápidamente puede llevarte donde los tipos y otras APIs son declaradas. También puedes obtener una rápida descripción de tus clases, métodos y campos.

Puedes ver y descargar el tutorial de Dart Editor aquí: https://www.dartlang.org/codelabs/darrrt/

¿Qué hay de nuevo con Dart?

Bien ahora que ya estás familiarizado, vamos a hablar del lenguaje. Vamos a meternos dentro de las caracteristicas de Dart y su sintexis en detalle -- puedes leer sobre todas las caracteristicas y detalles en la sintaxis en ww.dartlang.org --, bien aqui están algunas caracteristicas interesantes de DART:

Tipos Opcionales: Tu puedes usar tipos o no, depende prácticamente de ti. Los

tipos en el código de Dart no cambian la manera en que se ejecutan tus aplicaciones, pero su uso puede ayudar a otros desarrolladores a leer tu código al programar herramientas. Es probable que no pongas atención a los tipos mientras desarrollamos un prototipo, pero cuando deberías hacerlo cuando estés trabajando en una implementación en producción. Un patrón emergente es ir colocando tipos a las interfaces y métodos, y omitir los tipos dentro de los métodos.

Snapshots: Actualmente, los navegadores necesitan analizar el código fuente de una aplicación web antes de que ésta se pueda ejecutar. Dart permite la implementación de Snapshots-- esto sería una grabación de todo su estado en un punto de tiempo -- lo que significa un incremento considerable en la velocidad de carga. En un inicio, una aplicación web con 54,000 líneas de código en Dart inicia en unos 640ms sin el uso de snapshots. Con el uso de éste método, puede iniciar en 60ms. Cuando tu pograma Dart corre sobre la Dart VM, se puede apreciar como éste inicia en un tiempo con mejor performance, gracias a el uso de snapshots.

Aislamientos: Dart soporta ejecución simultánea gracias al uso de aislamientos(Isolates), se puede pensar que es el proceso sin el uso de sobrecargas. Cada aislación tiene su propia memoria y código, que no pueden ser afectados por otra aislación. La única manera en que una aislación puede comunicarse con otra aislación es mediante mensajes. Las aislaciones permiten que una aplicación web utilice a las computadoras con procesadores multi-núcleos de manera efectiva. Otro uso para las aislaciones: correr código desde diferentes orígenes en la misma página, sin comprometer la seguridad.

Interfaces con implementaciones por defecto: Ignora ésta parte si tu nunca haz usado un lenguaje con características de uso de clases, interfaces o protocolos. ¿Sigues aquí? OK. Si tu echas un vistazo a las librerías de Dart, verás que ellas usan interfaces en casos donde algún otro lenguaje usa clases --por ejemplo, para Dart y HashMap. Esto es posible porque un interface de Dart puede tener una implementación por defecto-- una clase (usualmente privada) que es la manera por defecto de crear objetos que implementen la interface. Por ejemplo, a pesar de que Stopwatch es una interface, puedes llamar a `new Stopwatch()` para obtener una implementación por defecto de Stopwatch. Si echas un vistazo a la documentación de la API o el código fuente, puedes ver que la implementación por defecto de Stopwatch es una clase llamada StopwatchImplementation.

Genéricos, pero fáciles: Los genéricos se han hecho antes, pero han sido un poco confusos.
Dart toma un nuevo enfoque mediante el diseño de un sistema de genéricos que sea más comprensible. La desventaja es tener que sacrificar un poco de exactitud, pero los ingenieros de Google creen que permite a los desarrolladores ser más productivos y superar los diseños de lenguajes como Ivory.

Librería HTML: Dart da una nueva mirada a la manera en que debe manejarse el HTML DOM. (DOM es la forma abreviada para Document Object Model; es la interfaz que te permite, mediante programación, actualizar el contenido, la estructura y el estilo de una página web). Gracias a la creación de una librería nativa en Dart (dart:html) se puede acceder y manipular el DOM; se crearon elementos, atributos y nodos para que se sienta natural la forma de trabajar con ellos.

¿Por qué Dart luce tan familiar?

Vanguardismo y un lenguaje único pueden ser hermosos, pero quizás esto tendría unos cinco usuarios. Dart está diseñado para aprobación masiva, así que tiene que resultar familiar para ambos, tanto usuarios de lenguaje de scripting (similar a JavaScript) como de lenguajes estructurados (similar a Java).

Aún así Dart tiene en la mira algunas características únicas para el lenguaje prevaleciente. Por ejemplo, interfaces con implementos predeterminados ayudan disimulando detalles de implementación. El tipo de variables opcional es otra característica nueva para la web, debería ayudar dando mayor claridad al desarrollador acotando su intención junto con las interfaces y librerías.

¿Qué hay en la Plataforma de Dart?

Dart es más que sólo un lenguaje, es una completa plataforma para los desarrolladores web modernos.

Especificaciones del Lenguaje: El lenguaje Dart es familiar, con algunas nuevas características como el tipeo opcional y los aislamientos.
Librerías: Las librerías del núcleo proveen funcionalidad incluyendo 'collections', 'dates', y 'math', como también enlaces HTML, I/O del lado del servidor como sockets e incluso JSON.
Compilador a JavaScript: Tu puedes compilar tus programas en Dart a JavaScript que puede correr en la totalidad de la web moderna.
VM: La máquina virtual está construida desde cero para correr el código de Dart de forma nativa. La VM corre sobre la consola de las aplicaciones del lado servidor y puede ser embebida dentro de navegadores para correr aplicaciones del lado cliente.
Integración con Chromium: La Dart VM ha sido embebida dentro de del build de Chromium, se le dio el nick de Dartium, haciendo alusión a que las aplicaciones Dart pueden correr de forma nativa sin compilar a JavaScript.
Editor de Dart: Este editor liviano, completo con sintaxis destacada y completado de código, puede lanzar tu script en la VM o como una aplicación web en Dartium. Incluso puede compilar tus aplicaciones a JavaScript y correrlas en otro navegador.

¿Puedo usar Dart para mi Aplicación Web hoy?

Dart sigue cambiando, lo que te asegura de que tengas un lenguaje optimizado y actualizado permanentemente. Esto es gracias a que Dart tiene un código libre y abierto a debate; de ésta manera se incentiva a investigar Dart y proveer una retroalimentación. Con un lenguaje con librerías establecidas, está calificado para la producción de aplicaciones web Dart en todos los navegadores modernos gracias a su compilador Dar2JS, y se espera que pronto los navegadores modernos soporten Dart de forma nativa.

¿Cómo esperas que las personas usen Dart?

Puedes usar Dart para construir complejas aplicaciones de alto rendimiento para la web moderna. El lenguaje de Dart está diseñado para trabajar en el cliente y el servidor, lo que significa que puedes usarlo para implementar y completar, de punta a punta, la aplicación.

También podés usar Dart para ascender tu desarrollo, a medida que tu programa crece de un pequeño grupo de funciones a una larga colección de clases. Por ejemplo, el desarrollo web fomenta iteraciones de programación extremadamente pequeñas. Así

mismo, mientras refinas tu idea, y tu programa crece en alcance y complejidad, probablemente tu código necesite más modularidad y encapsulación. Dart te posibilita pasar fácilmente de funciones a clases, y de un código no tipeado a uno tipeado.

En cuanto al deploy, tienes dos opciones: compilando a JavaScript, o usando Dart VM. Para código del lado cliente, compilarlo a JavaScript le da a tu código de Dart la posibilidad de ejecutarse en navegadores modernos. Una futura versión de Chrome saldrá con Dart VM incorporada, permitiendo ejecutar directamente tu código Dart, sin la necesidad de que sea primero compilado a JavaScript.

¿Cómo puedo compilar a JavaScript?

Para poder compilar nuestro código Dart a JavaScript utilizaremos la herramienta que nos provee el SDK de Dart, Dart2JS. Éste compilador nos brinda ciertas características muy ventajosas, como por ejemplo que el código JavaScript generado a partir de un código Dart corre mucho más rápido que si hubiésemos escrito ese código directamente en JavaScript gracias a la capacidad de minificación que incrementa la performance del código.

¿Qué librerías están disponibles?

Veamos un poco sobre las librerías más importantes en Dart:

dart:core

Ésta es la librería principal que todas tus aplicaciones dart debe contener, se carga de manera automática por lo que no debemos especificar su importación, ella nos permite el uso diferentes características, tipos y funciones. Echemos un vistazo a algunos:

- Números y booleanos.
- Strings y Expresiones Regulares.
- Collections.
- Fecha y Hora ('Date' y 'Time').
- Uri.
- Errores.

dart:html

Esta librería nos proporciona todas las herramientas para el manejo del DOM y la interacción con la UI. Por ejemplo:

- Obtener objetos globales ('document', 'window').
- Encontrar elementos HTML (Elementos 'query()' y 'queryAll()').
- Agregar y quitar eventos (Propiedad 'on' de los elementos).

dart:io

Esta es una librería que trabaja del lado del Servidor y podemos utilizarla siempre que la VM de Dart esté corriendo sobre uno. Unos ejemplos de que podemos hacer:

- Leer y Escribir información (InputStream, OutputStream).
- Abrir y Leer Archivos ('File', 'Directory')
- Conectar a Sockets en red ('Socket').

Puedes ver la documentación completa de cada una de éstas librerías en api.darlang.org
La lista de librerías listadas en la documentación es:

- **dart:async** (Nos provee el soporte para trabajar de manera asíncrona).
- **dart:collection**
- **dart:convert** (Codifica y decodifica para convertir entre diferentes representaciones de información, incluyendo JSON y UTF-8).
- **dart:core**

- **dart:html**
- **dart:indexed_db**
- **dart:io**
- **dart:isolate**
- **dart:js**
- **dart:math**
- **dart:mirrors**
- **dart:svg**
- **dart:typed_data**
- **dart:web_audio**
- **dart:web_gl**
- **dart:web_sql**

Muéstrame más código

Se ha dado una información general sobre las más interesantes características de Dart, y se ha mostrado algo de código Dart; ahora se elaborarán, en algunas de esas características -- tipeos, genéricos, aislaciones y manipulación del DOM -- y se mostrará aún más código.

Tipos

Anteriormente se ha mostrado este ejemplo:

```
class Punto {
        Punto(this.x, this.y);
        distanciaA(otro) {
                var dx = x - otro.x;
                var dy = y - otro.y;
                return Math.sqrt(dx*dx+dy*dy);
        }
        var x, y;
}

main() {
        var p = new Punto(2, 3);
        var q = new Punto(3, 4);
        print('Distancia de p a q = $p.distanciaA(q)');
}
```

Quizás notes que el código define y usa una clase, ésta no tiene un tipo; éstos son opcionales en Dart, no modifican la manera en la que el programa se ejecuta, pero hacen que el código sea más entendible por herramientas (como depuradores e IDEs) y desarrolladores (como si fuera tu reemplazo cuando te vas a un proyecto mucho mejor). Piensa a los tipos como anotaciones o documentación que puede ayudar a herramientas y humanos a entender tu intención mejor y a detectar errores más rápido.

Un buen lugar para empezar a agregar tipos es en los métodos, las firmas e interfaces -- el área superficial de tu programa -- . Mientras más personas sean adheridas a tu proyecto, y más clases sean generadas, será útil saber qué tipos son usados (y devueltos) por métodos.

```
class Punto {
      Punto(this.x, this.y);
      num distanciaA(Punto otro) {
            var dx = x - otro.x;
            var dy = y - otro.y;
            return Math.sqrt(dx*dx+dy*dy);
      }
      num x, y;
}

main() {
      var p = new Punto(2, 3);
      var q = new Punto(3, 4);
      print('Distancia de p a q = $p.distanciaA(q)');
}
```

Nota como no se han agregado tipos al cuerpo del método de "distanciaA()". Un patrón emergente a la hora de leer un código en Dart es usar tipos en métodos al declararlos pero no en el cuerpo del método, en su scope. El scope o cuerpo de un método cuerpo debería ser lo suficientemente pequeño para ser entendido fácilmente, y se esperan herramientas para realizar una inferencia de tipos variables en el mismo punto. Sin embargo, si prefieres usar tipos en todos lados, eres libre de hacerlo para cambiar de **var dx** a **num dx**, y así sucesivamente.

Genéricos

Si tu nunca has usado tipos genéricos antes -- o tal vez si -- ver algo como List<E> en la documentación de la API, puede resultar un poco aterrador. Pero no te preocupes, los genéricos en Dart son muy fáciles.

Por ejemplo, si no te importa que tipos de objetos hay en la lista, entonces, puedes crear una lista como esta:

```
new List()
```

Si sabes que tu lista sólo tendrá un tipo de objeto en ella -- sólo restricciones, por ejemplo -- entonces puedes (pero no necesariamente tienes que hacerlo) declararlo cuando crees la Lista:

```
new List<String>()
```

¿Por qué molestarse con tanta ceremonia? Especificando qué tipos puede tener tu colección, es una buena manera de dar conocimiento a tus colegas programadores, y a tus herramientas que expectativas tienes. Así las herramientas en tiempo de ejecución (runtime) pueden detectar errores más rápidamente.

(**Nota:** `new List()` es la manera acortada para `new List<Dynamic>()`. Dynamic es el tipo usado detras de escena para las variables definidas).

Aquí hay algunos ejemplos de como Dart adopta diferentes tipos y colecciones de variables indefinidas. Si estás familiarizado con genéricos en Java, presta mucha atención a los últimos dos ejemplos.

```
main() {
       print(new List() is List<Object>);
       print(new List() is List<Dynamic>);
       print(new List<String>() is List<Object>);
       print(new List<Object>() is! List<String>); //No todos los objetos son String
       print(new List<String>() is! List<int>);    //Los String no son int
       print(new List<String>() is List);          //Todas las listas de String son
                                                   //List
       print(new List() is List<String>);          //Es correcto pasar de una List a
                                                   //un método
                                                    //que espera List<String>
}
```

Estos ejemplos resaltan la variante cogenérica de Dart, porque tu puedes correr tu código sin tipos a lo salvaje, el lenguaje Dart te permite tratar con Listas indefinidas como con Listas definidas y pasar de una a otra sin problemas.

Así como se puede ver en el código anterior, los tipos parametrizados de Dar son materializados. Esto significa que los genéricos no se pierden en el tiempo de ejecución, lo que sucede es que Dart realmente conoce que un List<String> es una "Lista de Strings".

Manipulando el DOM

Recorrer el DOM con Dart tiene muchas ventajas. Tu puedes usar elementos y nodos con los objetos de Dart, puedes iterar a través de nodos hijos de la misma manera que lo haces con otras collections de Dart.

Por ejemplo, aquí hay un poco de código para encontrar un elemento específico y entonces realizar algunas operaciones sobre él y su elemento hijo.

```
#import 'dart:html';

main() {
       //Encontrar el elemento.
       var elemen = document.query('#id');

       //Agregarle eventos.
       elemen.on.click.add((event) => print('click'));

       //Seteando un atributo.
       elemen.attributes['name'] = 'valor';

       //Agregar un elemento hijo.
       elemen.elements.add(new Element.tag("p"));

       //Agregar una clase de CSS al elemento hijo.
       elemen.elements.forEach((e) => e.classes.add("important"));
}
```

Aislaciones

A pesar de que Dart tiene un único hilo de procesamiento, tu puedes tomar ventaja de las máquinas multinúcleo usando aislaciones. Una aislación provee memoria individual entre diferentes partes de un programa en ejecución. Cada aislación puede correr en un hilo separado de procesamiento, administrado por la Dart VM.

Las aislaciones se comunican mediante el envío de mensajes a través de puertos. Los mensajes son copiados de manera que una aislación no puede modificar el estado de

los objetos de otra aislación directamente.

Para ilustrar cómo trabaja una aislación , vamos a construir un simple echo service. Primero necesitamos definir `echo()`, una función para correr una aislación. Cada aislación tiene un puerto, el cual podemos usar para recibir mensajes. Nosotros vamos a responder un mensaje a través del puerto de respuesta que nos provee.

El método `main()` crea una nueva aislación para la función `echo()` con `spawnFunction()`. Usa el SendPort para enviar mensajes a la aislación, y escucha por alguna respuesta usando `then()`.

```
#import 'dart:isolate';

	echo() {
		port.receive((msj, respuesta) => respuesta.send("Echo: $msj"));
	}

	void main() {
	SendPort sendPort = spawnFuction(echo);
	sendPort.call("Hola, Dart!").then((response) => print(respuesta));
	}
```

Y la salida sería:

Echo: Hola, Dart!

Los procedimientos han sidos testeados en la Dart VM. In JavaScript, las aislaciones se compilan a web workers de esa manera pueden correr en procesos separados. Las aislaciones están en permanente actualización, así que estén atentos a la documentación de las mismas.

¿Dónde puedo aprender más?

El principal sitio a donde dirigirse por información de Dart es dartlang.org.

Estos son algunos sitios para obtener noticias sobre Dart:

- Noticias oficiales: news.dartlang.org
- Google+:
  - Dart: Structured web apps (Página no oficial)
  - #dartlang (hashtag para encontrar contenido y post en Google+)
- Twitter:
  - @dart_lang (Tweets Oficiales de Dart)
  - #dartlang (hashtag para encontrar tweets de Dart, también #dart)
- Blogs: Dartosphere (Un simple feed de muchos Blogs de Dart)

Si tu quieres ver como Dart está implementado o tu quieres comenzar como contribuidor de el projecto de Dart, visita:

code.google.com/dart

Información de contacto:

Material Original:

Seth Ladd: www.google.com/+SethLadd
Kathy Walrath: www.google.com/+KathyWalrath

Traducción y Actualización:

Germán Quiróz Bogner: www.google.com/+GermanQuirozBogner