Google Dart に就いて（速報）

2011 年 10 月
㈱クレス

10 月 10 日の月曜日に Google は Dart という「ウェブ・プログラミングの為の構造化された新しいスクリプト言語」を発表した。これはデンマークの Goto 会議の基調プレゼンテーションで Google のソフトウエア技術者の Lars Bak（VM の専門家）と Gilad Bracha（オブジェクト指向言語の専門家で Java 仕様書作成者のひとり）が紹介したものである。ただし Dart を発表するという情報は 1 ヶ月前から流れていた。

この言語はこれまでの JavaScript の問題点を解決しようとしており、将来的にはウェブ・アプリケーションの核となることを目指したものである。現時点ではこの言語は未だ開発段階で、アーリー・プレビュー（概要を知ってもらいデベロッパたちからの意見を収集する）として Dart のサイトからその情報を得ることが出来る。これが本当に JavaScript キラーになり得るか、あるいはこれは本当にオープンなものかに関しては、ネット上でいろんな意見が交わされている。

Dart は構造化されたウェブ・アプリケーション開発の為の新しいクラス・ベースのプログラミング言語である。シンプルであること、効率が良いこと、及び規模対応性（スケーラビリティ）があることなどを目標にして開発された Dart 言語は、強力で新しい機能たちを馴染みのある言語構成と組み合わせ、すっきりした読みやすい文法になっている、と Google は説明している。

1. Google の JavaScript 戦略

歴史的には Dart はこれまでの数年間にわたる Chrome ブラウザの為の V8 JavaScript エンジンの開発の経験から昨年生まれた Spot と呼ばれる言語がもとになっている。その時の目標は JavaScript の経験をもとにしたウェブの為の新しいシンプルな言語だった。リークされた Google 社内メモによれば、昨年 11 月の 10-11 日に、Google 内のクライアント・サイドの技術者たちからなる幾つかのチームが集まって会合を持ち、JavaScript の将来に関する「共通ビジョン」で合意した。その共通ビジョンとは Spot がベースになっている Dash と呼ばれる新しいプログラミング言語開発であり、これがその後 Dart という名前に変更されている。このときの Dash の目標はこのオープン・ウェブ・プラットホーム上でウェブ開発の共通語 (lingua franca)として最終的に JavaScript に置き換わることだった。しかしながら Dart が失敗するリスクを最小化する為に、Google はヘッジを行っている。彼らの社内メモには以下のように記されている：
JavaScript はこの言語を単に改良するだけでは解決できない基本的な問題が存在する。従って我々は JavaScript の将来に対し次の 2 本立ての戦略をとることにする：

- Harmony（ロー・リスクだが成果も少ない）：EcmaScript 標準化機関である TC39 とともに JavaScript 改善の作業を継続する
- Dash（ハイ・リスクだが結果も大きい）：Dash と呼ばれる新しい言語を開発する

2. Google Dart の概要

Goto 会議での彼らのプレゼンテーションでは、まず彼らはこれまでのウェブ技術の良い点と問題点とあげ、Dart をこのギャップを埋めるものだと主張している。また Dart は JavaScript のキラーではないが、現在ばらばらになってしまっているモバイルのプラットホーム環境に対処するひとつの手段だと述べており、Chrome に加えて Android を含むモバイルの世界を強く意識していることも注目される。

良い点：小規模アプリケーションが簡単に作れる	問題点：大規模アプリケーションには不十分
・プラットホームに依存しない ・アプリケーションをインストールしなくて良い ・段階的な開発に対応 ・...そして広く普及している	・プログラム構造が判りづらい ・スタティック型が存在しない ・ライブラリ対応していない ・サポートの為のツールが弱い ・立ち上げ性能が良くない

Dart はシンプルで一般的なオブジェクト指向言語だと彼らは言う：

- クラス・ベースでインターフェイスによる単一継承
- オプションとしてのスタティック型
- きちんとした静的スコープ
- 単一スレッド
- 馴染みのある（特に Java 等のオブジェクト指向の言語の経験者にとって）文法

Dart のサイトの技術概要のページでは、Dart 言語の特徴を以下のように示している：

- クラス：クラスとインターフェイスは、効率的に API を定義する為の良く理解されたメカニズムとなる。これらによりメソッドとデータのカプセル化と再利用が可能になる。
- オプションとしての型指定：Dart のプログラマたちは自分たちのプログラムにオプション的に静的型を付加できる。即ち変数とメソッド呼び出しにオプション的にデータ型を指定できる。プログラマの好みにより、あるいはアプリケーション開発段階により、そのプログラムはシンプルで型なしの実験的なプロトタイプから型づけをした複雑でモジュール構造のアプリケーションに移行できる。型がプログラマの意図を記すことになる（セルフ・ドキュメンテーション）ので、そのプログラムで何が起きるのかを説明するのに必要なドキュメンテーションが少なくなり、デバッグ段階では型チェックのツールが使える。
- ライブラリ：デベロッパたちは実行時に変わらないことが保障されているライブラリの作成と使用が可能になる。従って独立して開発されたコード片は共有されたライブラリに依存できる。これらのライブラリには HTML 5 の DOM 仕様書原案に基づくライブラリも含まれる。
- ツール：Dart はこの言語をサポートする為に開発された実行環境、ライブラリ、及び開発ツールで構成される。これらのツールにより生産的でダイナミックな開発が出来るようになり、例えば編集と実行再開デバッギング、そしてあるプログラムのアウトラインを書き、実行し、実行しながら未完の個所を埋めてゆくといったプログラム作成スタイルがとれる。

また Dart の設計目標を次のように示している：

- 構造化されているものの柔軟なウェブの為のプログラミング言語とする
- プログラマたちにとって馴染みがあり自然なものと感じるようにし、従って学習し易いものにする
- Dart の総ての言語構成物が高い性能を持ち、早いアプリケーションの立ち上げが確保されるようにする
- 電話機、タブレット、ノートブック、そしてサーバ等ウェブ上の総ての機器にとって相応しいものにする
- 総ての主要ブラウザにわたって Dart が高速で走るようにするツールたちを用意する

クラスとインターフェイス

Dart のインターフェイスとクラスにより、再利用可能で拡張性あるビルディング・ブロック（基礎的要素となるプログラム要素）のセットを持つことができる。インターフェイスはメソッドと定数の基本的なセットを定義し、これは他の複数のインターフェイスを継承できる。クラスのほうは複数のインターフェイスを実装できるが、単一のスーパークラスからのみ継承が可能である。

以下はその例である:

```
interface Shape {
  num perimeter();
}

class Rectangle implements Shape {
  final num height, width;
  Rectangle(num this.height, num this.width);  // Compact constructor syntax.
  num perimeter() => 2*height + 2*width;       // Short function syntax.
}

class Square extends Rectangle {
  Square(num size) : super(size, size);
}
```

Shape(形状)というインターフェイスは primeter(周辺長)を得るメソッドのみを持つ。このインターフェイスを実装した Rectangle(矩形)というクラスは height(縦長)と width(横長)という変数を持つ。このコンストラクタはこれらの変数を指定する。これは非常にコンパクトな文法である。this.id という記述はインスタンス変数 id にセットするという意味である。perimeter のメソッドは縦長と横長の 2 倍を加えたものを返す。これも短い記述で済む。form => e という記述は form {return e;}と等価である。Square(正方形)というクラスは Rectangle を継承したもので、これのコンストラクタは Rectangle クラスの縦長と横長に同じ size という値をセットする。

オプショナルな型指定

Dart では、プログラマのオプションで、静的及び動的なチェックが出来る。実験中はそのプログラマはシンプルなプロトタイプの為に型指定しないコードが書ける。そのアプリケーションが大きくなってゆきより安定化されれば、必要に応じデバッグ支援の為あるいは構造を規定する為に型を付加できる。

例えば以下は、x と y というパラメタと倍数倍する scale()と原点からの距離を返す distance()という 2 つのメソッドを持った新しい Point クラスを作るための型指定なしの Dart のコードである。

```
class Point {
  var x, y;
  Point(this.x, this.y);
  scale(factor) => new Point(x*factor, y*factor);
  distance() => Math.sqrt(x*x + y*y);
}

main() {
  var a = new Point(2,3).scale(10);
  print(a.distance());
}
```

これに対し x、y、及び factor が確実に num 型で、Point が確実に num 型の 2 つの変数を含むようにするために型指定をした場合は以下のようになる:

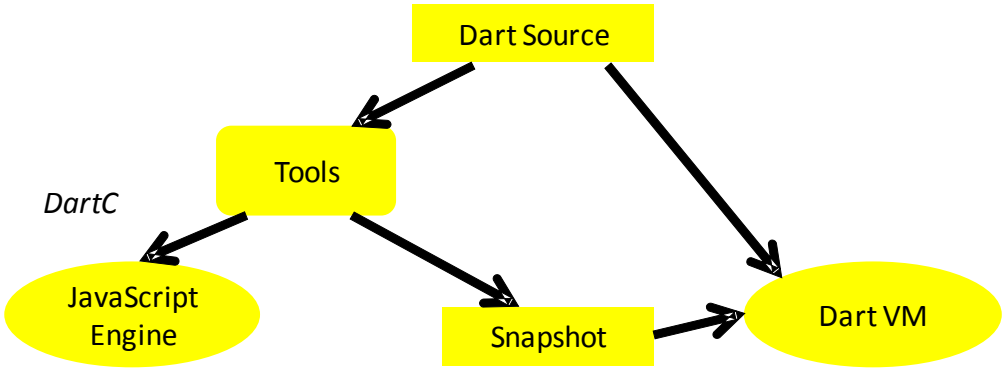
```
class Point {
  num x, y;
  Point(num this.x, num this.y);
  Point scale(num factor) => new Point(x*factor, y*factor);
  num distance() => Math.sqrt(x*x + y*y);
}
```

```
void main() {
  Point a = new Point(2,3).scale(10);
  print(a.distance());
}
```

num はインターフェイスであり、64 ビット倍精度浮動小数点あるいは可変長整数である。

Dart の実行

Dart の実行には 2 つの手段がある。ひとつはネーティブな VM（仮想マシン）上での実行であり、もうひとつは Dart のコードを JavaScript に変換するクロス・コンパイラ(DartC)を使って JavaScript エンジン上で実行させる方法である。後者の場合は、Dart でウェブ・アプリケーションを書いて、それをどのブラウザ上ででもコンパイルし実行させることが出来る。現在 Dart の VM は Google の Chrome ブラウザには未だ組み込まれていないが、それは計画中だという。



上図で注目すべきことは Snapshot と呼ばれる処理である。これは Dart のアプリケーションをロードした後でヒープを直列化する処理であり、これによりロード時間の大幅な短縮がなされている。例えば 54173 行からなる Dart のコードをロードするには 640ms かかるが、同じアプリケーションを Snapshot からロードすると 60ms で済んでしまい、10 倍以上の高速化が図れる。通常の JavaScript のコードを JavaScript エンジンがロードする時間も Snapshot なしの Dart コードのロード時間並みだと Google の Bak が述べている。

Google はまたサーバ上での Dart VM による実行も考えている。これによりフロント・エンドとバック・エンドの双方が同じプログラミング言語で書かれた「Google 規模」のウェブ・アプリケーションが可能になる（JavaScript と Node.js で書かれたアプリケーション専用サーバのように）。

新しい MIME タイプ

HTML ページに Dart のプログラムを直接埋め込む、あるいは外部ファイルを取り込むために#import あるいは #source ステートメントが使えるようになる。その為に“application/dart”という新しい MIME タイプが提案されている：

```
<html>
 <body>
   <script type="application/dart">
     main() {
       Element element = document.getElementById('message');
       element.innerHTML = 'Hello from Dart';
     }
   </script>
```

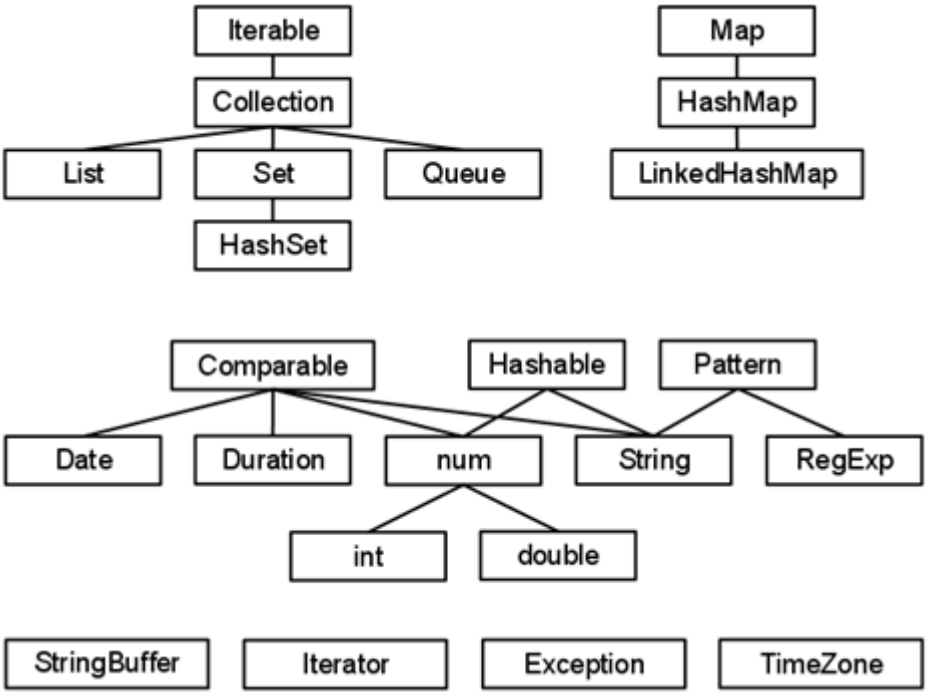
```
    <div id="message"></div>
  </body>
</html>
```

ライブラリ

ウェブとウェブ・サーバ開発をサポートする為に、Dart では以下のようなライブラリが用意される。

- コア・ライブラリ：共通なデータ構造と操作をサポートする為のインターフェイスたちで構成される
- DOM ライブラリ：HTML5 DOM の為のインターフェイスたち、W3C/WHATWG で規定されている HTML5 標準案にほぼ基づいている。これらのインターフェイスは HTML5 標準と並行して進化することになる

下図は現時点でのコア・ライブラリのインターフェイス階層である（変更の可能性あり）：

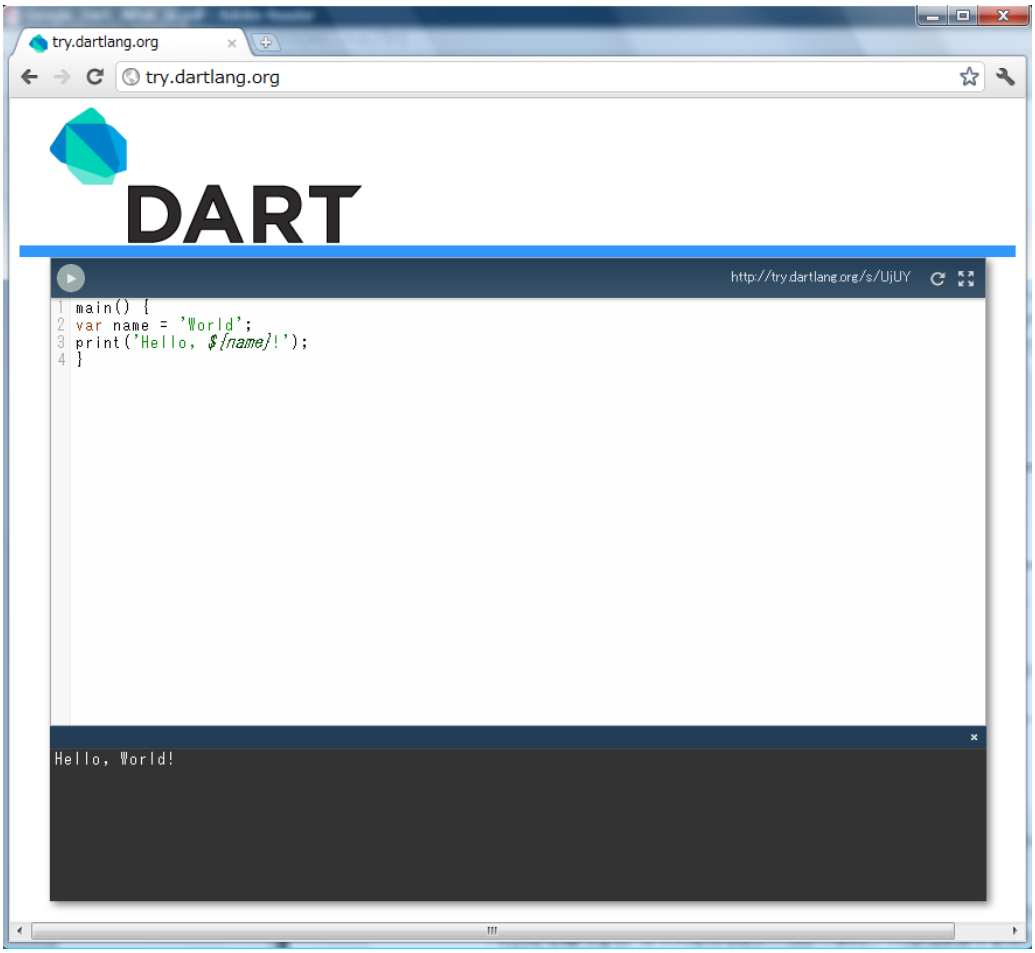


3. チュートリアル

以下は Dart のサイトにあるチュートリアルを訳したものである。Dartboard（ダーツの的）という簡単なオンラインの開発環境アプリケーションを使うと、自分のブラウザ・ウィンドウ内で短い Dart プログラムを編集・実行可能なので、それを使って以下のチュートリアルを試してみると良い。Dartboard は現時点では Chrome、Safari 5+、及び Firefox 4+で実行できる。IE9 にも間もなく対応するとのことである。

Dartboard の使い方

下図を見て頂きたい：



Google Chrome のアドレス・バーに http://try.dartlang.org/を入力し、編集画面上にプログラムを貼り付け編集する。次に左上の実行ボタンを押すとコンパイルと実行が行われる。

- 警告は黄色のフラグで、エラーは赤のフラグで知らせる。そのフラグの上にマウスを置くとその問題の記述が表示される。
- Dartboard に当初から表示されていたコードを編集するときは、Dartboard の上のほうにあるリロードのボタンを押す。
- 通常のキーボードのショートカットが使える。例えば Ctrl-Z で最後の変更を戻す。Shift-Ctrl-Z はそれを再適用する。

Hello World

```
main() {
  var name = 'World';
  print('Hello, ${name}!');
}
```

このサンプルでは幾つかの基本的なDartの特徴と規約を知ることが出来よう:

- トップ・レベルのmain()関数
  クラスで囲まれていないmain()関数かあることで、Dartはそのコードの開始場所を知る。
- varで宣言された変数
  Dartプログラム内で変数を作りたいときは何時でもvarキーワード、finalキーワード、あるいは型の名前を使用してその変数を宣言しなければならない。
- print()による出力
  Dartのprint()関数はテキストをコンソールに送信する。
- 文字列リテラル
  文字列をマークするにはシングル・クオートまたはダブル・クオートのどれかを使用する。'World'と"World"は等価である。
- ${expression}による文字列の挿入
  Dartでは文字列リテラルのなかに式を埋め込むことが出来る。この例のようにその式が単に変数のときは中カッコ({})を省略できる。文字列を作るもう一つの方法はプラス(+)演算子を使うことである。例えば以下の3つのステートメントは等価である:

```
print('Hello, ${name}!');
print('Hello, $name!');
print('Hello, ' + name + '!');
```

クラス

クラス・ベースの言語を知っている人であれば、クラスの生成と使用の為のこの文法は多分馴染みのあるものであろう。そうでない人はこのサンプルでクラスに就いて知る必要がある。

クラスはDartでは重要である。クラスはどのようにオブジェクトを生成するかということである。Dartのプログラムの総てのオブジェクト型はクラスかインターフェイスかのどちらかである。

```
class Greeter {
  var prefix = 'Hello,';

  greet(name) {
    print('$prefix $name');
  }
}

main() {
  var greeter = new Greeter();
  greeter.greet("Class!");
}
```

このサンプルはDartのクラスの幾つかの基本的な特徴を示すものである。Greeter(挨拶状)というクラスはprefix(前置文)というインスタンス変数と、それにname(名前)をつないだgreet(挨拶)というインスタンス・メソッドからなる。

- classステートメント
  この例ではデフォルトのスーパークラスObjectを持つGreeterという名前のクラスを定義している。Dartでは、総

てのクラスは直接あるいは間接的に Object がおおもとになっている。非 Object スーパークラスを指定したいときには、extends キーワードを使用する。

- インスタンス変数
  正規変数と同じように、インスタンス変数を生成するときには var、final、または type キーワードが必要である。各 Greeter オブジェクトは'Hello,'に初期化された prefix という名前の変数のコピーを持つ。インスタンス変数に直接（例えば greeter.prefix = 'Hi,'）、あるいはコンストラクタで、あるいはセッタ・メソッドをつかって値をセットできる。（セッタとゲッタのメソッドは次のサンプルで説明する）
- コンストラクタ
  あるクラスのインスタンスを生成するには、new キーワードとそれに続くそのクラスのコンストラクタを呼び出す--この場合は new Greeter()。このコードでは Greeter コンストラクタが定義されていないので、そのときはそのスーパークラスの引数なしのコンストラクタが呼び出される。Greeter のスーパークラスは Object なので、new Greeter()というコードは Object()を呼び出す。
- インスタンス・メソッド
  greet()メソッドは Greeter オブジェクトに結び付けられたある関数を定義している。
- 型の無指定
  Dart では型の指定はオプショナルである。型は自分のコードの作成とメンテナンスには役に立つが、それは Dart のプログラムがどのように振る舞うかを変えるものではない。

クラスに関する更なる説明

Dart のクラスには多くの特徴が付加されている。最も一般的に使われる 2 つは名前つきのコンストラクタと、ゲッタ/セッタである。

- 名前付きのコンストラクタ
  あるクラスに対して複数のコンストラクタが必要な場合、名前付きコンストラクタ、例えば Greeter.withPrefix()が定義できる。あるコンストラクタを定義するときは、デフォルトの引数なしのコンストラクタはその為には生成されない;その場合には例えば次のようにそれを付加しなければならない:

```
// Greeter クラス内では:
Greeter();
Greeter.withPrefix(this.prefix);

// Greeter オブジェクトを生成するコード内では:
var greeter = new Greeter.withPrefix('Howdy,');
```

- セッタとゲッタ
  セッタとゲッタはインスタンス変数を直接おもてに出すことなくデータにアクセスするひとつの手段である。以下は Greeter クラスのなかの prefix データの為のゲッタとセッタの例である:

```
class Greeter {
  String _prefix = 'Hello,';          // Hidden instance variable.

  String get prefix() => _prefix;     // Getter for prefix.

  void set prefix(String value) {     // Setter for prefix.
    if (value == null) value = "";
    if (value.length > 20) throw 'Prefix too long!';
    _prefix = value;
  }

  greet(name) {
    print('$prefix $name');
  }
}
```

```
main() {
  var greeter = new Greeter();
  greeter.prefix = 'Howdy,';          // Set prefix.
  greeter.greet('setter!');
}
```

prefix()にある特別な=>文法に注意されたい。この簡略記述は、そのメソッドが=>に続く記述の値を返すことを意味する。なお、Dart には public キーワードは存在しない。Dart ではアンダスコア("_")で始まる名前を除いて総ての型は public である。例えば Greeter は public だが_Greeter は private である。

インターフェイス

あるクラスが提供するメソッドたちを定義した型であるインターフェイスは、Dart では重要である。実際 Dart のコア・ライブラリ(Dart Core Library)の多くはインターフェイスとして定義されている。インターフェイスあるいはプロトコルが導入されている言語を使用したことのない読者は、コンサンプルを見て頂きたい。

```
class Greeter implements Comparable {
  String prefix = 'Hello,';
  Greeter() {}
  Greeter.withPrefix(this.prefix);
  greet(String name) => print('$prefix $name');

  int compareTo(Greeter other) => prefix.compareTo(other.prefix);
}

void main() {
  Greeter greeter = new Greeter();
  Greeter greeter2 = new Greeter.withPrefix('Hi,');

  num result = greeter2.compareTo(greeter);
  if (result == 0) {
    greeter2.greet('you are the same.');
  } else {
    greeter2.greet('you are different.');
  }
}
```

- インターフェイスの実装
  Greeter クラスはコア・ライブラリの Comparable インターフェイスを実装しており、2 つの Greeter オブジェクトを比較出来るようにしている。このインターフェイスを実装するには 2 つのステップを踏む:クラス・ステートメントのなかに Comparable 実装を付加(行 1)、及び Comparable が必要とするメソッドのみ compareTo()の定義の追加(行 7)
- int と num インターフェイス
  int と double はプリミティブな型と読者は考えるかもしれないが、これらは実際はインターフェイスであって num インターフェイスを継承したものである。このことは、int と double の変数も num であることを意味する。
  重要:num は初期化をすること。これらはオブジェクトであるので、これらの初期値は 0 ではなくて null である。
  使用上は int と double は読者が多分馴染んでいるプリミティブ型のように感じられる。例えば、これらの値をセットするのに文字列を使うことが出来る:

```
int height = 160;
double rad = 0.0;
```

インターフェイスに関する更なる説明

Dart では、そのインターフェイスを実装したクラスを探さなくても、しばしばあるインターフェイスから直接オブジェクトを生成できる。これは多くのインターフェイスがファクトリ・クラスを持っているからである。ファクトリ・クラスはそのインターフェイスを実装したオブジェクトを生成するクラスである。例えば、皆さんのコードが new Date.now()と書かれているときは、Date インターフェイスの為のファクトリ・クラスが現在の時刻を示すオブジェクトを生成する。

4. 参考報道

Google の Dart はプログラムを生彩に富んだものにするかも(Gigaom)

Google's Dart may be the code to make web apps shine

http://gigaom.com/cloud/googles-dart-may-be-the-code-to-make-web-apps-shine/

By Stacey Higginbotham

Oct. 10, 2011, 11:00am PT

Google は 2011 年 10 月 10 日にウェブ・アプリケーション構築専用のプログラミング言語を作る取組みを発表した。コンピュータのハードウエアがより接続された世界とモバイルの世界の為に進化させていると同じように、Google はソフトウエアにおいても同時進化を推進させようとしている。

Google today unveiled its effort to create a programming language solely for building web apps. Much like there's a shift in computer hardware to take advantage of a more connected and mobile world, Google is attempting to push a concurrent shift in software.

Dart は何で、どう違うのか？

Dart: What it is and what it isn't

Google は昨年、データ・センタ内のサーバ上で走るウェブ・アプリケーションのプログラム作成を容易なのにするという目標のもとに Dash とい名前で Dart 開発に着手している。その結果としての 10 月 11 日発表の言語は JavaScript キラーになることを示唆しているが、これが本当にスクリプト言語の王者で、ウェブ上の静的及び動的なプログラミングの間を埋める共通語としての JavaScript を超えるようになるかどうかはこれからのことである。

Google started building Dart last year under the name Dash, with the goal of making it easier to program web applications that run on servers inside a data center. The resulting language unveiled Monday is suggested to become a JavaScript killer, although we'll have to see if it can take over the king of scripting languages and the lingua franca of bridging the static and dynamic programming divide on the web.

Google は Dart の目標をかなり明確に述べている。そのアイデアになっているのは、簡単で迅速に作成できるものの、ウェブ規模のアプリケーションに対応するに十分強力なものであり、またオリジナルのプログラムを書いた人以上により容易にメンテナンス出来るスクリプト言語を作るというものである。JavaScript のようなユーザ・フレンドリな言語にするというアイデアにより、ホビー・レベルの開発者が迅速かつ簡単に開発できるようになるが、一方アプリケーションの規模が大きくなればより大きなプログラムと人員が必要になるので、デベロッパたちがプログラムを作成する為の行当たりのコストが下がることにもなる。

Google states its goals for Dart fairly clearly. The idea is to build a scripting language that can be built easily and quickly, but is also powerful enough to support a webscale applications and be maintained easily by more than just the author of the original code. The idea is that user-friendly languages like JavaScript allows a hobbyist-level developer to build out something quickly and easily, but the way the developer does writes the code can create hidden costs down the line as the application scales and requires more code and more people.

JavaScript ではひとりのプログラマが単一の統一化されたプログラムを書かねばならないが、Java や C++のような他の言語の場合はよりモジュール化が可能である。オブジェクト指向のプログラミングにより、Java のプログラマはモジュールの再利用あるいは以前書かれたプログラムを再利用してプログラムを書くことが可能である。他のプログラマはそれを付加したり、それがどのように作られているかを簡単に調べることが出来る。従って Java や他のオブジェクト指向のプログラミングは LEGO を組み立てるのと似ているのに対し、JavaScript や他のスクリプト言語は張りぼてを作るのに似ている。前者ではそのプログラムのメンテナンスをしたり作り変える際は、それが何でどのように作られているかを知ることが出来る。後者では、結果を知ることが出来ても、そのもとになっている構造を知ることは殆ど不可能である。

JavaScript requires a programmer to deliver a single chunk of unified code, whereas other languages such as Java and C++ allow for more modularity. Thanks to object-oriented programming, a programmer working with Java can reuse modules or previously written code to build a program. Other people can add to it and easily see how it was made. So where Java and other object-oriented programming are akin to building something with LEGO, JavaScript and other

scripting languages are more like using paper-mache. In one, you can see what it is and how it was built in order to maintain or replicate it. In the other, you see the result but figuring out the underlying structure is almost impossible.

早いことは良いこと。

Fast is beautiful

Dart はスナップショット(snapshot)を使ってオブジェクト指向のモジュール化で JavaScript によるコーディングをし易くしている。Dart のプロジェクト・リーダの Lars Bak は、CNet のインタビューで、スナップショットと呼ばれるコンセプトを説明している:

Dart hopes to offer the ease of JavaScript coding with the modularity of object-oriented programming languages by using snapshots. In a post over at CNet, based on an interview with Lars Bak, the project leader for Dart, Bak explains a concept called snapshotting:

> Google は Chrome ブラウザに直接 Dart を組み入れる最善な手段を評価しており、これは Bak が熱心になっていることである。ひとつの理由は:これは「スナップショット(snapshotting)」技術が使えるようにすることで、劇的にウェブ・アプリケーションの立ち上がり時間を改善する。スナップショットというのは、あるアプリケーションにたいしこれを単一のデータ・ブロックに「直列化(serializing)」する。
>
> Google is evaluating the best way to integrate Dart directly into its Chrome browser, something Bak is keen on. One reason: it will enable a "snapshotting" technology that dramatically improves a Web app's startup time. Snapshotting involves taking an application and "serializing" it into a single block of data.
>
> スナップショットのテストでは、5,000 行の Dart のプログラムのロード時間はスナップショットしないときの 640ms にたいし 60ms に短縮された、と Bak はいう。彼によれば、これまでの JavaScript のプログラムの場合、スナップショットなしの Dart 相当の時間でロードする。彼は「Dart を直接ブラウザに統合すると、Dart に適用できる多くの最適化が可能になる」という。
>
> In one test of snapshotting, a 55,000-line Dart program loaded in 60 milliseconds compared to 640 milliseconds without it, Bak said. A conventional JavaScript program would load in comparable time as Dart without snapshotting, he said. "I can see a lot of optimizations that'll be applicable to Dart" when it's integrated directly into a browser, he added.
>
> ロード時間を Bak がいうように速くできれば、スナップショットがデベロッパたちにとって Dart 中毒になりそうなものの最初になろう。従って Google が Dart フレームワークを Chrome ブラウザに統合すれば、ウェブ・ページ開発に Dart を使っているひとたちにとって、他のサイトでは JavaScript がロードされるのを待っている一方で、自分たちのウェブ・ページが非常に高速にロードされることになる。Google は Dart がデベロッパたちの注目を集めない場合の為に JavaScript 開発を明らかに推進し続ける計画だが、デベロッパたちが Dart に取りつかれてしまうように、それが新しいコンピューティング時代の為のより高速なウェブ・ページを推進し続けることを明確に期待している。Axel Rauschmayer は自分の詳細なポストでどうしてデベロッパたちやブラウザのメーカたちが Dart を回避したいと思うかに関して書いている。
>
> Snapshots may also be the first hit of the Dart drug that gets developers addicted, if it can speed up load times like Bak says. So if Google integrates the Dart framework into its Chrome browser, that means that folks using Dart to build their web pages will see blazing fast load times while other sites wait for their JavaScript to load. Google still apparently plans to keep pushing JavaScript development in case Dart doesn't catch developers' eyes, but it's clearly hoping it can continue pushing faster web pages for the modern computing era in a way that developers will latch onto. Axel Rauschmayer offers an in-depth post on why developers and browser makers might want to duck and avoid Dart.

それ以外にも利点がある

But wait, there's more!

もうひとつの Dart の目標として述べられていることは、デベロッパたちがフロント・エンドとバック・エンドの双方で同じようにプログラムが作成できるようなフレームワークを作ることである。デベロッパたちにとってそのほうが楽なので、Node.js のような新しい別の言語とフレームワークでもこのような一様なプログラムが可能になっていて、プログラムの

作成と維持がしやすく、またウェブ・サイトが高速化される。そのようなデベロッパたちにとって楽なことにより、Google はデベロッパたちに Dart を試させようとしている。

Another stated goal of Dart is to build a framework that allows developer to code both the front end and the back in the same way. A new crop of languages and frameworks are being built, such as Node.js that allow this homogenous programming, because it makes life easier on developers., which combined with ease of building and maintaining code, and faster web sites, is just another way Google wants to entice developers into giving Dart a try.

Google はこのツールをオープン・ソースとして利用できるようにしており、これは Google が世界中にこれを推進することをコミットしていることを示している。Google は、ウェブ・アプリケーションの為のプログラミング・ツールを再構築しようとすると、Fortune 500 社の為のプログラムを開発しているような人たちの為に空き時間にアプリケーションを作っている人たちが広く使われるようにならないと駄目なことを理解している。そうすれば Google は支持層を確保し、ウェブ・ベースのアプリケーション開発の為の標準にしたいと思っている標準化作業で潜在的な支配力を持つことが出来る。

Google is making the tools available via open source, which does show how committed Google is to pushing this out to the rest of the world. Google understands that any effort to rethink the underlying programming tools for web applications needs broad adoption from those building apps in their spare time to those hired to code for the Fortune 500. It also helps Google get in on the ground floor and potentially dominate a standards effort around what it hopes will become a standard for developing web-based applications.

従って Dart がスピードのニーズ、そしてオブジェクト指向言語のモジュール性をもったスクリプト言語のニーズに対応してゆけるかどうかが注目される。それが可能なら Dart は JavaScript キラーになる可能性がある。そうでないと、これは単にウェブの高速化のための Google によるひとつの試みに終わってしまう。

So we'll see Dart can deliver on the need for speed and the desire for a scripting language with the modularity of an object-oriented language. If it can, it may indeed be a JavaScript killer. Otherwise, it's just another Google attempt to make the web faster by rethinking the way things are done.

Google Dart のデビューは JavaScript の政変かと話題に(PhysOrg)

Google Dart debut sparks chatter of JavaScript coup

October 12, 2011 by Nancy Owano

http://www.physorg.com/news/2011-10-google-dart-debut-chatter-javascript.html

10 月 10 日の月曜日に Google がデベロッパ向けに新しいプログラミング言語を発表したとの報道は、どうやって Google は JavaScript 政変を始めるのかとの議論が技術ブログ上で展開されている。Google は JavaScript ではなくて Dart をウェブ・アプリケーションを書くための最終的な共通語としようとしている、との憶測に対し、Google の Dart 開発チームはそのような動機は正確ではないと話している。JavaScript は普及しており、今後もそうだというのが彼らの答えである。

(PhysOrg.com) -- When the news appeared earlier this week that Google was unveiling a new programming language, Dart, for developers. tech blogs ignited with talk of how Google is staging a JavaScript coup. The assumption was that Google wants Dart, not JavaScript, as the eventual lingua franca for writing Web applications, a motive that Google's Dart team says is not accurate. JavaScript has been around and will continue to be, was their response.

Dart 開発チームの 2 人の技術者たちは月曜日にデンマークの Goto 会議の基調講演でこの新しい言語を紹介している。「本日我々はウェブ・アプリケーション構築のためのクラス・ベースでオプション的な型付け(class-based optionally typed)のプログラミング言語の初期プレビューを発表している。」

Two engineers on the Dart team introduced the new language at the opening of the Goto Conference in Denmark earlier this week."Today we are introducing an early preview of Dart, a class-based optionally typed programming language for building Web applications."

詳細は dartlang.org という新しいウェブ・サイト上で知ることが出来る。Google はプログラマたちにこの新しいプログラミング言語は新しいものと馴染みなもののブレンドで容易に理解できるものだと知らせている:即ちこの新しい言語は馴染みのある構成であることとコードが簡単に書けることが特徴である。「Dart のプログラムはひとつまたは 2 つの言語を知っているプログラマたちにとって馴染み易いものであり、クラスとクローズのような時の試練を経た機能が使える」ことを Google の Dart は保証している。

The details are available on a new web site, dartlang.org. Google is telling programmers that the new language is an easily digestible blend of the new with the familiar—that is, new language features with familiar constructs and easy to define code. Google's Dart assures developers that “Dart code should look familiar if you know a language or two, and you can use time-tested features such as classes and closures.”

Dart を既に試してみた人たちによれば、簡単で馴染みがある部分があり、この言語は Java のように見え、機能し、動作するし、そのプログラムを JavaScript に変換するツールが用意されている。

According to those who have already examined Dart, there is a comfortable familiar part, in, as one site puts it, the language looks, acts, and runs like Java and has a tool that converts the code into JavaScript.

Dart 担当のソフトウエア技術者の Lars Bak によれば、Dart のプログラムは 2 つの実行手段があり、ひとつはネーティブな VM 上での動作で、もうひとつは Dart のプログラムを JavaScript に翻訳するコンパイラを使った JavaScript エンジンのもとでの動作である。Dart のプログラムは殆どのブラウザ上で使えるようになる。

According to Dart software engineer Lars Bak, Dart code can be executed in two different ways: either on a native virtual machine or on top of a JavaScript engine by using a compiler that translates Dart code to JavaScript. One can use the same Dart code in most modern browsers.

Dart には基本的なライブラリたちと、Dart のプログラムのチェック、コンパイル、及び実行の為のツールたちのセットとして提供され、これらは最終的にはプログラマたちからのフィードバックを基に更に発展して行く、と Bak は述べた。Google の“Dartboard”というアプリケーションには Dart で書かれたサンプルたちが含まれている。

Dart comes with a set of basic libraries and tools for checking, compiling, and running Dart code, which will evolve as programmers send in their feedback, Bak said. Google’s app, “Dartboard” has examples of Dart code.

Google はデベロッパたちに Dart は未だ開発の初期段階にあり、デベロッパたちがどう考えるかのコメントを歓迎するとしている。Dart は BSD ライセンスのもとでオープン・ソース化されている。この言語と暫定的なツールたちは dartlang.org から取得できる。

Google is telling developers that Dart is still in the early stages of development and that it invites comments on what developers think of it so far. Dart has been made open source under a BSD license. The language and preliminary tools are available on dartlang.org.

一方、プログラミングの生産性と良い結果で差別化出来る言語として Dart がどこまで成功できるかどうかは、最終的にはデベロッパたちが判断することになる。デンマークでの会議では、Dart のウオッチャたちは Dart の基調講演の後で、Dart は未だ技術プレビューの段階でしかなく、いろんな機能はフィードバックを受けて変更される可能性があることを強調した。この会議でもう一つ強調されたことは、Dart はそれ自体は「もうひとつの JavaScript」として設計されたものではなく、むしろ現在ばらばらになってしまっているモバイルのプラットホーム環境に対処するひとつの手段だということである。

Meanwhile, developers will ultimately decide to what extent Dart succeeds as a language that can make a difference in programming productivity and good results. Dart watchers at this week's conference in Denmark, following the Dart keynote, underscored that Dart is only in technology preview stage and that features may change as feedback comes in. Another point underscored at the conference is that Dart is not designed to be "another JavaScript" per se but rather a way to address what is now a fragmented mobile platform environment.

More information: Google's blog post: http://googlecode. … red-web.html

参考:
http://gototoday.dk/2011/10/10/lars-bak-on-dart/
GoTo 会議の報告