Dart

Vijay Menon
Google Seattle

What is Dart?

Terse Java / C# style OO language

Open source, but primarily used at Google for large-scale web apps

Ads, Google Fiber

General client-side language

- **Modern browsers via compilation to JavaScript**
- iOS and Android (http://flutter.io)
- Embedded devices (http://dartino.org)

Optionally typed

```
class Point {
 var x, y;
 Point(this.x, this.y);
 operator +(other) => new Point(x + other.x, y + other.y);
 toString() => "($x,$y)";
}

main() {
  var p = new Point(2, 3);
  print(p + new Point(4, 5));
}
```

Optionally typed

```
class Point {
 num x, y;
 Point(this.x, this.y);
 Point operator +(Point other) => new Point(x+other.x,y+other.y);
 String toString() => "($x,$y)";
}

void main() {
  Point p = new Point(2, 3);
  print(p + new Point(4, 5));
}
```

Type “checking” in Dart

Static checking of types

- Static lint to detect potential errors
- Tooling (completion, refactoring)
- Optional: program semantics well-defined regardless of static errors

Runtime checking

- Checked mode (static type annotations -> runtime assertions)
- Production mode (static type annotations are ignored)

Deliberately unsound

Unsound?

Philosophy:

- Don’t make programmers fight the type system
- Type common patterns in the browser DOM

Examples:

List<**CanvasElement**> canvases = document.querySelectorAll(queryString);

button.addEventListener(eventName, (**MouseEvent** e) => …);

Types can lie!

Example:

```
    List<num> list = [“hello”, “world”];

    print(list[0] + list[1]);
```

No static errors

No checked mode errors (even though generics are reified)

Dart subtyping is circular: *`List<num> <: List<dynamic> <: List<num>`*

Does it work?

Millions of lines of code

Does it work?

Millions of lines of code

But … confusing to programmers

- They *think* Dart is a statically-typed language
- They *think* the type system is sound
- Confused when they hit examples like the previous
- Confused that static types don’t help performance
    - Why doesn’t the compiler know this is a String? I just told it that!

An implementation cost

Why does this Dart:

```
    int x = a.bar;

    b.foo(“hello”, x);
```

not just map to this JavaScript?

```
    var x = a.bar;

    b.foo(“hello”, x);
```

`Answer: Dart has stricter dispatch - but there is a cost`

Field does not exist? Too many args?

Why does this Dart:

```
    int x = a.bar;       // Runtime error
    b.foo(“hello”, x); // Runtime error
```

not just map to this JavaScript?

```
    var x = a.bar;       // Silently return undefined
    b.foo(“hello”, x); // Silently drop unused arguments
```

Different dynamic dispatch

Dart-to-JavaScript translates this:

```
    int x = a.bar;

    b.foo(“hello”, x);
```

to roughly this JS in the general case:

```
    var x = getInterceptor(a).get$bar(a);

    getInterceptor(b).foo$2(b, “hello”, x);
```

Aggressive whole-program optimization required to remove this overhead

New project: strong mode

Can we retrofit a sound type system onto Dart?

- Better static type errors for users
- Better code generator

Semantic constraint - consistency with existing Dart semantics:

- Proper subset of existing Dart
- Correct strong mode execution
- -> Correct checked mode execution
- -> Correct production mode execution

Strong mode

Enforce the types

- Require static type verification
- Runtime checks on potentially unsafe operations (e.g., downcasts)

… and use them

- Assume static types are correct
- Fast dispatch for typed code
- Allow **dynamic** type (triggers slow dispatch path)

Using types

If this type checks in Strong Dart and we know the types of a and b:

```
int x = a.bar;

b.foo(“hello”, x);
```

generate the following JavaScript:

```
var x = a.bar;     // Statically verified a has bar

b.foo(“hello”, x); // Statically verified b has matching foo
```

What’s in strong mode?

Stricter subtyping rule : Partial order on types

- Generics
- Functions

Type inference : Preserve terseness of language

Preserve dynamic type : Checked at runtime

Generic methods : Stronger typing on polymorphic code

Challenges

Types are reified at runtime

- **is** (instanceof) and **as** (cast) operations can differ

A handful of common unsound patterns

- covariant parameter types for function subtyping

Millions of lines of code

- Initially focusing on core unsoundness in the language
- For later: implicit casts, variance on generics

Initial feedback

Initial strong mode implementation

- Strong mode checking integrated into Dart tooling / IDEs
- Dart Dev Compiler
- Focus on fast iteration, readable output (i.e., no whole program analysis)

Users seem to like it!

- Appetite for stronger static checking greater than we anticipated
- Willing to give up flexibility … mostly
- Not so much push back on code changes

Learn more

Dart: http://dartlang.org

Dev compiler: https://github.com/dart-lang/dev_compiler

Strong mode: https://github.com/dart-lang/dev_compiler/blob/master/STRONG_MODE.md

People:

- Leaf Petersen
- Stephen Adams
- … and several others in Seattle, Portland, Mountain View, Aarhus