Parameter Passing & Delegate Covariance: What's Cooking?

2020-11-21

https://github.com/Geod24/DConf2020

Introduction

- Mathias Lang (@Geod24) 🇫🇷
- Former Sociomantic 🇩🇪
- Current CTO @ BPFK 🇰🇷

BPF Korea

- Blockchain project: https://bosagora.io/
- Creating a decentralized ledger
- 9 D developers (2 exp / 7 new)
- Open source: https://github.com/bpfkorea/agora

Why D?

- Strongly typed system programming language
- Best C++ integration, great C integration
- Amazing for prototype to functional

Pain points

- Debugging experience / tools (DUB!)
- Control over types (e.g. unpreventable moves)
- **Supporting multiple attributes**

Composition via delegates

- Generic (3rd party records)
- User customizable
- Easy to compose
- Defer aggregation / allocation logic to the caller

Examples

- Hashing
- Custom serializer
- String formatting (pretty printer)

Use case #1: Hashing

```
/// Our entry point
public Hash hashFull (T) (in T record);
/// Forwards to:
public void hashPart (T) (in T record, scope HashDg state) {
    static if (hasComputeHashMethod!T)
        record.computeHash(state);
    else {
        // Handles native types or static assert
    }
}
alias HashDg = void delegate(in ubyte[]) /*pure*/ nothrow @safe @nogc;
```

Source agora.common.Hash

Use case #1: Hashing

```d
struct Input
{
    SenderInfo sender;
    Hash commitment;
    Signature sig;

    void computeHash (scope HashDg state) const nothrow @safe @nogc
    {
        // Ignore `sig`, do not double hash `commitment`
        hashPart(this.sender, state);
        state(this.commitment[]);
    }
}
```

Use case #2: Serializer

```
/// Simple interface that returns bytes
ubyte[] serializeFull (T) (in T record);
/// Forwards to:
void serializePart (T) (in T record, scope SerializeDg dg);

/// Deserialization part:
T deserializeFull (T) (in ubyte[] data) @safe;
/// Forwards to:
T deserializePart (T) (scope DeserializeDg dg) @safe;

/// Delegate types
alias SerializeDg = void delegate(in ubyte[]) @safe;
alias DeserializeDg = const(ubyte)[] delegate(size_t size) @safe;
```

Source: agora.common.Serializer

Use case #3: Formatter

```d
/// Allocates, Phobos-style interface
string format (A...) (in char[] fmt, A args);

/// Actual implementation
void sformat (A...) (scope FormatterSink sink, in char[] fmt, A args);

/// Write to a pre-allocated `buffer`, never allocates
char[] snformat (A...) (char[] buff, in char[] fmt, in A args);
```

Source: ocean.text.convert.Formatter

One delegate to rule them all

```
string format (Args...) (in char[] fmt, in Args args)
{
    chap[] result;
    scope FormatterSink sink = (in char[] s) {
        result ~= s;
    };
    sformat(sink, fmt, args); // The magic part
    return result.assumeUnique();
}
```

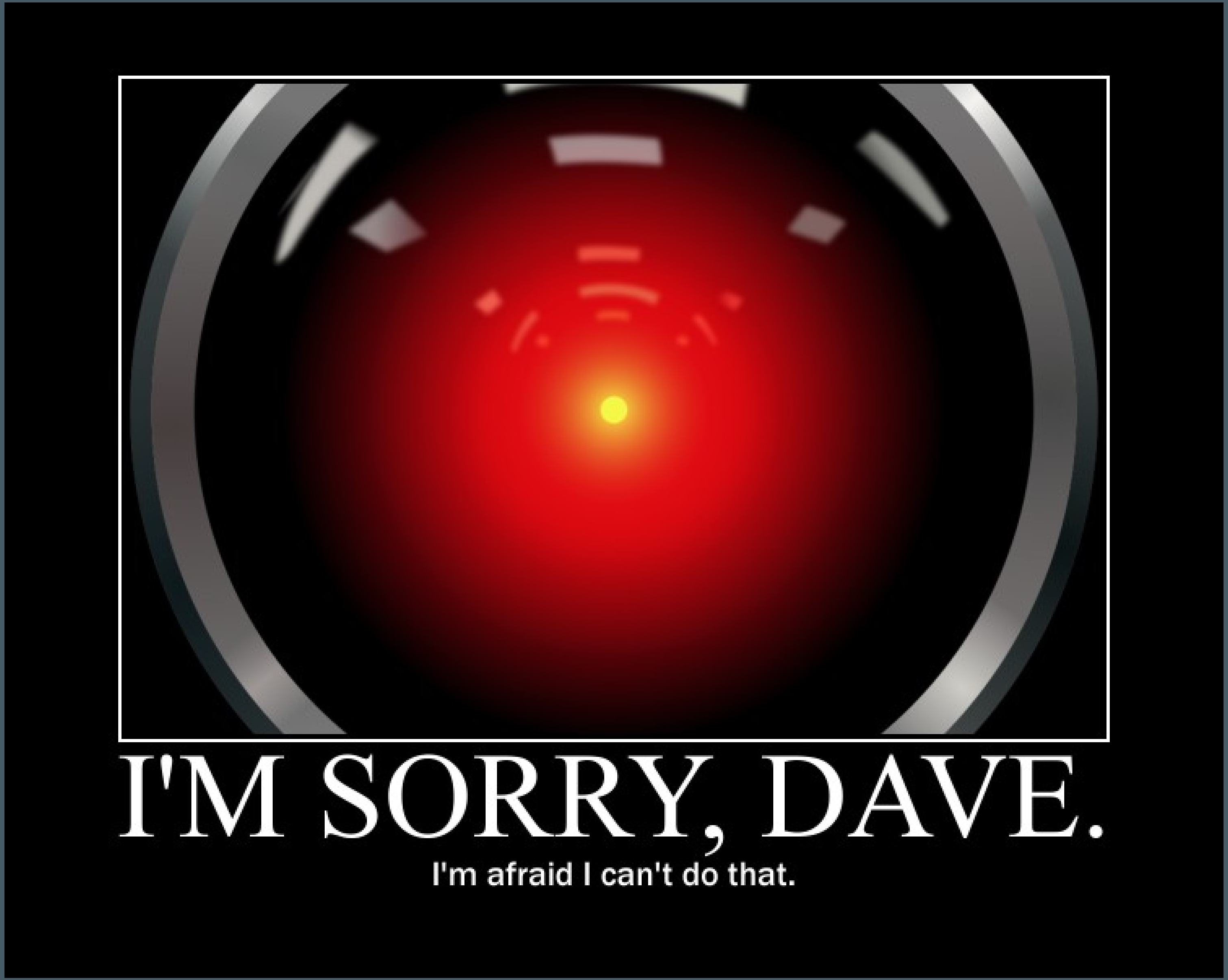
Composing delegates

```d
void safe_sformat (Args...) (scope FormatterSink sink,
    in char[] fmt, in Args args)
{
    scope FormatterSink wrapper = (in char[] s)
    {
        if (canFind(s, "password"))
            throw new Exception("Credential leaked");
        sink(s);
    };
    sformat(wrapper, fmt, args);
}
```

Composing in the client

```
struct SenderInfo {
    /// ....

    void computeHash (scope HashDg state) const nothrow @safe {
        Buffer buff;
        scope HashDg safer = (in ubyte[] a) {
            if (!buff.canFit(a))
                buff.dump(state);
            if (buff.append(a))
                state(a);
        };
        static foreach (field; this.tupleof)
            hashPart(field, safer);
        buff.dump(state);
    }
}
```


I'M SORRY, DAVE.
I'm afraid I can't do that.

Oops

```d
/// We want this
void computeHash (scope HashDg state) const @safe
{
    scope HashDg safer = (in ubyte[] a) {
        if (containsPrivateKey(a))
            throw new Exception("Credential leaked");
        state(a);
    };
    static foreach (field; this.tupleof)
        hashPart(field, safer);
}

/// But got this
alias HashDg = void delegate(in ubyte[]) /*pure*/ nothrow @safe @nogc;
```

How `inout` solves this

```
inout(char)[] strip(return inout(char)[] input);

class Container {
    inout(T)[] opSlice(size_t lower, size_t upper) inout return;
}
```

Argument-dependent attributes

```
struct Struct
{
    void toString (scope void delegate(in char[]) sink) const
        @safe(sink) pure(sink) nothrow(sink) @nogc(sink)
    {
        sink("Hello World");
    }
}
```

ADAs are optional

```d
struct Struct
{
    void toString (scope void delegate(in char[]) @safe sink) const
        @safe pure(sink) nothrow(sink) @nogc(sink)
    {
        sink("Hello World");
    }
}
```

ADAs support multiple delegates

```
struct Struct
{
    void toString (
        scope void delegate(in char[]) sink1,
        scope void delegate(in char[]) sink2,
        ) const
        @safe(sink1, sink2)
    {
        sink1("Hello");
        sink2("World");
    }
}
```

ADAs are composable

```
struct Struct
{
    void fwd (scope void delegate(in char[]) sink)
        @safe(sink) pure(sink) nothrow(sink) @nogc(sink) const
    {
        this.toString(sink);
    }

    void toString (scope void delegate(in char[]) sink) const
        @safe(sink) pure(sink) nothrow(sink) @nogc(sink)
    {
        sink("Hello World");
    }
}
```

Bonus goodies

- Make `opApply` usable
- Mitigates `Object`'s issues (`toString`, `toHash`)
- Backwards compatible (`Throwable.toString`)

`-preview=in`

Or: How I Learned to Stop Worrying about my parameters and Love the Compiler.

Reality sets in

```
ubyte[] serializeFull (T) (scope const auto ref T record);

void serializePart (T) (scope const auto ref T record,
    scope SerializeDg dg);
```

What ?

- New preview switch to give a new meaning to `in`
- Available since DMD v2.094.0 / LDC v1.24.0
- Almost equivalent to `const scope auto ref`
- `in ref` is now an error
- dmd -preview=in [-preview=dip1000] -run test.d

Why ?

- Modern code is litered with `const scope auto ref`
- `auto ref` forces you to use templates
- `ref` doesn't accept rvalues (literals)

Rule book

- Assume pass-by- `ref` , prepare for pass-by-value
- Optimized if value is small
- No side effect
- Parameter aliasing
- Good replacement for `auto ref`

Say what, not how

- Input parameters: `in`
- Ouput parameters: `out` (+ return value)
- Input/Output parameters: `ref` (formerly `inout`)

```
bool readPatientData (in ubyte[] serialized,
    ref size_t offset, out PatientData result)
```

I hear you say `-preview`...

- Works with Phobos & Vibe.d
- Also 47/62 Buildkite packages (dlang/dmd#11632)
- Safeguard against different qualifiers (linker)
- Blocking DUB bug (working on it)

Thanks 👋