Notes - 1/31: SENT

Saturday, March 12, 2011 5:45 PM

IE9

- Performance is 100x ovetr 5 years – moore's law, scale
  - DOM 40x
- Web platform richer

Apps vs. sites

What are opportunities?

- New class of killer apps (MS has platform / tools)
- Scale (maintain, extensible, versioning, etc.)
- Tooling/productivity (intellisense, rename, unit test, type check, diagnostics)
- Programmability (workers, typed arrays)
- Performance

Strada

- Once sentence value statement
- Goalss
- Principles

- Private names proposal

-

From: Luke Hoban
Sent: Tuesday, February 01, 2011 1:54 PM
To: Strada Team; Anders Hejlsberg; Mads Torgersen
Subject: Strada Design Notes - 1/31/2011

1. Notes for 1/31/2011

Agenda

- Review of goals and principles
- Autos
- Typed arrays
- Expression typing
- Explicit coercions
- Object literal typing
- Fields/properties in interfaces

Attendees: SteveLuc, AndersH, MadsT, ChuckJ, MShields, LukeH

1. Review of Goals and Principles

With the larger design group we reviewed goals and principles.

Goals:

- Be the most productive language for developing medium to large scale standards-based browser applications
- Be faster than common hand-written JS on IE, as fast as common hand-written JS on other browsers
- Support excellent tooling and frameworks
- All JavaScript code is legal Strada code (almost)

Principles:

- Combine the expression-level semantics and runtime libraries of JavaScript with the known scalable typed OO of C#/Java/C++.
- All JavaScript code is legal Strada code (modulo minor corner-case exceptions) – but not all JavaScript code can be statically typed in Strada

- Strada can call any existing JavaScript library without wrapping
- Type system should be as strict as C#/Java in the fully statically typed cases – errors are a key to scalability goals
- Do language innovation only where critical to the Strada goals
- Optimistic class prediction in Chakra to ensure Strada code runs fast in IE – but don't sacrifice perf on other browsers.
- Generated code should be readable and debuggable - but not necessarily code a user would write by hand
- Strada users can easily develop a strong intuition and trust about JS execution semantics and performance
    - "Thin layer"
    - "Idiom capture" - reduce common idioms to first class concepts

Key Ideas:

- "var" is the static type for dynamic – implicit coercions to/from any type
- Enforce type assignments in generated code "result|0"
- Transmit strada types to runtime to achieve excellent and predictable performance
- Separable runtime and design-time investments – runtime performance investments also benefits vanilla JS (helpful community-facing story)
- Predictable performance can be targeted by other languages compiling to vanilla JS
- Can use interfaces to overlay types on existing libraries without runtime wrapping
- Interfaces are duck typed

Tool Vision:

- Source level debugging
- Reliable intellisense and language services with Strada semantics
- Background error checking
- Support for Corsica and WWA
- Further static analysis tools (JsCop)

During the discussion of goals and principles, a few new observations were raised, including "auto" and typed arrays.

2. Auto

There is a possibility of wanting an "auto"-like type, which behaves as C# var. Currently the type system is simple enough that this is likely not needed in v1, but generics or anonymous interface types would potentially argue for a need for this.

3. Typed arrays

Typed arrays at the Strada language level are going to be important for performance. We are working on adding TypedArrays support in IE10 timeframe matching an intersection of W3C and TC39 proposals. Strada could emit code which leverages these where available, and does explicit checking where not available. We will need to go into more detail on Strada-level arrays in a future meeting.

4. Expression Typing

We took another look through some expression typing examples.

```
//***** BASIC TYPES *****//
double d1 = 3.14; // ok
double d2 = 4; // ok - implicit int -> double coercion
double d3 = "hello"; // error - string not implicitly coercible to
double
double d3b = null; // error - null not implicitly coercible to double
double d3c = getIFoo(); // error - interface type not implicitly
coercible
            // to double
// NOTE : INumber -> double coercion?
double d4 = (double)("3"); // error - string not explicitly coercible to
             // double
// NEW ISSUE : This should probably be allowed,
        look at VB conversion semantics as a possibly better fit
var v1 = "hello"; // okay - implicit anything -> var coercion
double d5 = (var)("hello");  //ok - explicit string->var followed by
              //implicit var->double
              // NOTE: This is a runtime check, which results in NaN
```

```
double d6 = {}; // ok as in d5, also succeeds at runtime with NaN
// NOTE: Implicit coercions do not chain - only one is applied between
// explicit coercions.
//    C# spec has a good explanation of this mechanims I believe.
//    This explains why d3 above is an error, but d5 and v1 are not.
//    The implicit coercions var->* and *->var do not allow implicitly
//    coercing *->*
```

Some new ideas were raised in looking at these, below.

5. Explicit Coercions

Example d4 above matches C#'s explicit coercion rules, there is no legal explicit cast from string to double. It is probably more in line with the rest of Strada's coercion system to allow explicit coercion here. We should look at VB rules, which may be a good starting point. As a general rule, wherever there is a coercion that would succeed at runtime, we ought to allow at least an explicit cast.

6. Object Literal Typing

Example d6 above assume that object literals are typed as var. This loses a lot of potentially useful type information. Instead, there is a desire to have some sort of anonymous type or anonymous interface scheme.

It was also noted that introducing anonymous types may push down a road to needing "auto".

There is a separate topic of potentially adding an expression for typed object literals which directly construct instances of a particular class type.

7. Fields/Properties in Interfaces

We haven't yet nailed down the syntax and matching rules for fields/properties in interfaces. We would like to use field syntax in interfaces, and have this match both data properties and accessor properties at runtime. There are likely more details we need to flesh out here.

This will also require readonly and possibly writeonly modifiers on fields in interfaces.