Notes - 2/7: SENT

Saturday, March 12, 2011 5:42 PM

IMPORTANT

Topics

- Namespaces (and imports)
  - Packaging:
    - Closure=assembly
    - Intra-type references are fast
    - Don't want to cross outer-closure boundary
    - Practice is to incrementally download
    -
  - Just like C#, with $ instead of .
    - Why not be objects
  - No-mangling output mode:
    - Namespaces are objects
    - Dynamic resolution of overloading
  - Mangling vs. speed?
    - One end of the spectrum…
  - Can you straddle – offer both or the benefits of both?
    -
  - Public fields
- Overloading
  - 'var' instead?
  - Need typeswitch?

    typeswitch(expr) {
     case int x:
       break;
     case double d:
     break;
    }

    Could later do structural pattern matching.
- Generics:
  - List<T> is such a huge part of the scenario
  - Maybe not needed if we have a solution for that
- Optional params
  - function f(int x)
  - x ? x|| 0 : 5
  - Named – no.
- Typed functions
  - Performance efficiency
    - Local variables
    - Closed
    - Shines when there is a mutually referential set of classes
- In/out of the box?
  - …
- Mangling
  -
- Classes
  - What do they erase to?

```
class Point(int x, int y) {

  void translate(Point p) {
   x+= p.x;
   y+= p.y;
  }

}

function assembly() {
```

```
  PointPrototype = {translate : function(p) { return this.dispatch(0,p);
}}
  function Point(x,y) {  // also have a static entry point
    x = x|0;
    y = y|0;
  }
  _this = {dispatch: PointDispatch }; // attach a prototype with
translate
 function PointDispatch(action, arg0) {
    Case 0: {
      checkpoint(arg0);
      x += arg0(1);
      x += arg0(2);
      break;
    }
    Case 1: {
      //get x
    }
    Case 2: {
      //get y
    }
  }
}

We generate:
  __tanslate:
   eax
   ecx
   …
```

function assembly() {
 functio
}

We can make this blazingly fast on IE10
Liability is that chrome could be slow – but we can generally make it faster than the

Action: Need a write up of the pattern.
What we have: Some hand built examples of this pattern.

Action:
- Interfaces
- Types
- Casts/is/as

Points in design space;
- We could dial back goals to be fast?
  - Just fast primitive types
- Enforced encapsulation
  - Do you care that private means private?
  - Most JS doesn't care
  - Not really the key value
  - This implementation (above) gives enforced encapsulation
- Loader of the strada code knows all the code that can touch state
  - Really important for perf
  - Can know there is a consistent type assignment, and represent as real int
- Third position
  - Choose on a per-type basis which model to emit

Readability:
- Not too much of an issue
- But experience of using it from outside matters a *lot*

If you call another Strada thing:
- Expensive
- 3x slower on chrome to have all calls go across boundaries

Are we clinging too hard to the perf thing?
- Strada has value for productivity/scalability/toolability

- IE9 demos get to stay in the box a *lot*
- Events seem compelling too – you always have 50ms user response, and only lose once
- Script# apps – one giant assembly

What principles and data needed to address these questions?

- Pick key apps and focus?
- Get data?

Action: Write up of the code gen?

Until we know the IL level, hard to discuss language level

Perf note:

- We currently run faster than object/prototype code in Chrome, but slower than optimized

Here we have a processor not designed for this