Notes - 10/9/2013

Wednesday, October 9, 2013 9:06 AM

- Engineering update
- Recent spec work discussion/overview
  - New overload resolution rules
  - New infinite expanding generics rules
  - Type assertions and contextual types
  - Contextual typing and ||
    - Interface Map<T> { [x:stirng]: T; };
    - Var d: Map<Customer> = myCusts || {};
  - Contextual typing and ?:
  - Handling of type of 'prototype' by instantiating type parameters with 'any'
- Resolution order of transitive dependencies (787647)

  //a.ts

  export var hello = "hello";

  //b.ts

  import a = require("a");
  module foo {
  import h = a.hello;
  console.log(h);
  }

  Seems like there is a bug where we're dropping a require because we need to transitively follow references to see if they are being used as a value

- Exact restrictions for the use of circular import declarations (774151)

  //a.ts

  Export var hello = "hello";

  //b.ts

  Import a = require("a");
  Module foo {
  Import x = a;
  Import y = x;
  }

  Not even clear this is still an issue because of the changes to import aliasing in the last few months.

- Should static indexers constrain the prototype? (777252)

  Wait for someone to ask for this. No static indexer for 1.0. Gives error.

- Instanceof implementation should not require a constructor (775169)

  In JavaScript, first it checks that the lhs is an object and the rhs is a function.

  Right now, users are having issues because they're trying to new up objects of HTML DOM elements that do not allow construction (like HTMLElement). This leads to confusion. The reason those constructors are even visible is to allow the instanceof functionality because of our current restrictions.

  One possible change is that we can check that the rhs has a prototype property. We would need to make sure we do have prototype properties on every built-in that would need it (those in lib.d.ts). This would also mean that any simple objects that people create would also have to have a prototype property, which seems to be cumbersome.

  Possible design:
  - First check for construct signature
  - If not a construct signature, check for prototype

  Yet, status quo is reasonable because JS does have the constructors, they just throw an error. It's possible this will change in the future.

  Design decision: leave it be. No change.

- Allowing 'declare' inside of another 'declare' (774069)

  Just a bug in the compiler

- What about declaring a quoted module in an external module?

  The key scenario here is actually that we want to be able to declare quoted modules from inside of an external module. Currently, we require you to put that quote declaration in a separate file and ///reference from the external module. Multiple people mentioned hitting this in practice, eg when prototyping Node modules.

To fix the issue with .d.ts generation, can we just raise a visibility error when the user tries to export a type that was declared inside of a quoted ambient module? This could be a possible restriction that prevents the issue yet largely allows the practice.

Design decision: go with the current PoR. Will be a breaking change against last released version

- Forbidding reserved names/built-in JS API names for codegen (725805)

  Design decision: go with the PoR and allow. We may revisit later.

- Inconsistent syntax for generic functions vs construct signatures (787488)

  By design.

- Current .d.ts generation is broken for some cases where shadowing will occur

  Example:

```
//Code:
module Outer.Inner {
    export var self = Outer.Inner;
}

module Doppleganger {
    var innerRef = Outer.Inner;

    export module Shadow.Outer {
        export var t = innerRef; // t: Outer.Inner
    }
}

//Current.d.ts emit:
declare module Outer.Inner {
    var self: {
        self: typeof Inner;
    };
}
declare module Doppleganger {
    module Shadow.Outer {
        var t: {
            self: typeof Outer.Inner; // Error: Inner is not a member of
Doppleganger.Shadow.Outer
        };
    }
}
```

Example #2:

```
interface Foo {
    a: Foo;
}

function x(…): Foo { … }

module M {
    interface Foo {
        z: Zoo;
    }

    export var y = x(…);  // y: { a: Foo }
}
```

Design decision: revisit on Monday, discussion incomplete