Notes - 1/23

Wednesday, January 23, 2013 2:04 PM

[In 41/1523 at 2:00 today.]

Agenda:

- Spaces in names (See attached)
- Close details of “readonly” in type system (see attached)
  - Off the shelf for now.
- Close any open issues with “export =”
- ‘this’-types: Off the table for now for v1?
  - Off the shelf for now.
- Concerns with ///<reference> (see attached)
- Other persistent feedback areas:
  - Union types
  - Generators
  - Mixin models
- Others?
  - Modules

Spaces in names

We plan to support this syntax:

```
interface Options {
    ["view engine"]?: ViewEngine;
    ['strict routing']?: bool;
    [x: string]: number;
    ["do something fun!"](x: string): void;
}
```

And this syntax;

```
class Options {
    ["view engine"] = 42;
    ['strict routing']: bool;
    ["do something fun!"](x: string) {
    }
    constructor() {
        this['strict routing'] = true;
    }
}
```

And an expression of the form:

```
x["foo"]
```

Is type checked by doing:

1. Check if the type of ‘o’ has a member named “foo”. If so, the type of this expression is the type of the member “foo”
2. Else, the type of ‘x’ has a string indexer (either from Object or overriden). The type of this expression is the return type of the string indexer.

On the shelf for now.

1. Readonly is part of the type system.
   a. Any given property of a TypeScript type can be readonly.
   b. Signatures cannot be ‘readonly’
2. Readonly is not itself runtime enforced in anyway, it is just statically checked.
   a. Readonly properties cannot be the final part of dotted name on LHS of assignment operation.
      a. This should actually just say "readonly properties are not a reference"
3. In the surface syntax, readonly can be used explicitly on the following:
   a. Property declarations in interfaces “readonly foo: string”
   b. Property declarations in classes “readonly foo: string = ‘hello’”
   c. Also in consturctor params
      a. constructor(private foo: string) // this lifts foo to being a proeprty
      b. constructor(readonly foo: string) // should this? No - because we want this syntax to be available in the future
   d. Order must be "public readonly"
4. Method declarations are implicitly ‘readonly’
   a. “foo(x: string): string” is now implicitly the same as “readonly foo: (x: string) => string”, ***not*** “foo: (x: string) => string”
   b. The equivalent of the above is true for both classes and interface methods
5. Get-only accessor properties in classes are implicitly readonly
   a. A declaration ‘get foo() { return 3; }’ in a class implies that the interface type associated with the member is ‘readonly foo: number’
6. All exported properties of modules are implicitly readonly
7. There is no ‘writeonly’.

```
import Boo = module('foo')

=>

class Boo {

}



// Yes
export = class Foo {}
export = <expression>

// Not
```

   a. In addition, we statically reject write-only accessors??
8. For assignment compatibility, { x: string } is assignable to { readonly x: string}, but { readonly x: string } is *not* a assignable to { x: string}.
   a. var z: { readonly x: string} = {x: “hello”}
   b. var y: { x: string } = z; // error
   c. Not clear if more is needed to characterize subtyping as well?
9. This proposal does not yet introduce a notion of ‘readonly’ for lexical names.
   a. It could in principal be extended to also describe how these work, both to formalize our current treatment of function declarations as being non-writable, and to model a future ‘const’ local variable.
10. Readonly does not apply in the constructor!

```
export = module Foo {

}
```

Concerns:
export = class Foo {} // This exports the value and type
export = Foo // This doesn't export the type name - confusing that this isn't the same
What goes in the declare file?
export: {x: string}; // The type of me is this