Typescript

For the win!

Why Typescript

1) Its framework agnostic *use it with whatever framework you want — Typescript don't care its a Honey Badger*

2) Start using the next version of javascript—>ES6/ES7/ES.next, today in a mature 2.x+ version language. The ES6 language adds features such as modules and classes. Typescript will track the emerging ES.next standard and add those features as they become solid.

3) Typescript is a *superscript of javascript* so you can begin to use it **incrementally**. The first step can be to change *.js to *.ts and immediately see the benefits of some internal type checking.

4) Typescript produces idiomatic javascript without any need for runtime libs. So it you want to ditch typescript you can and just start using the javascript files it produces. It does the type checking on the compile step so no extra code is added. [Some ES6 transpilers like traceur require a runtime lib as well]

5) Type safety. Typescript has an optional type safety feature so that if you only want type safe signatures on the public apis then you only pay that price on the public API.

Why Typescript

6) Get the benefits of a nice/powerful syntax like Coffeescript, but with type checking and the emerging standard ES.next syntax as well

7) Typescript works well with all of your existing javascript libraries. Want to do just a few modules in Typescript and have it integrate with your existing codebase — no problem.

8) If you are coming from languages like java or C#, then familiar design patterns (like GoF) fit right in instead of having to learn the all the javascript design patterns (many of which rely upon closures for modularity).

9) Typescript is opensource. Yes, Microsoft created it, but you can fork the source on https://github.com/Microsoft/TypeScript

10) Use whatever tooling you want. It is just a command-line compiler. Want full ide support use WebStorm or VS code (they have intelligent support for Typescript). Want something lightweight use Sublime Text (there are Typescript plugins available as well).

What is Typescript

- Superset of Javascript (compatible with JS ES5)
- Superset of Javascript ES6/ES7/ES.next (so you can use these features before they are supported by the browsers)
- Optional Typing (maybe you only want to use on public APIs)
- Typing is only used for compile-time checking. It compiles out to plain javascript with no overhead.
    - No Runtime (no plugins needed; produces plain javascript)

Starting Out

- Easy to start
- Copy your *.js files to *.ts files
- Use ‘tsc’ to compile *.ts to *.js
- Can even use ‘tsc —watch’ to have it watch for changes

tsconfig

- tsc —init : produces a sample tsconfig.json file

- https://www.typescriptlang.org/docs/handbook/compiler-options.html

Type Definition Files

- Brings typing of untyped/plain javascript files/ libraries

```
declare var myGlobalVar: string;
declare function log(message: string);
```

- *.d.ts (or *.tsd -old) files
- Ideally, the types would already be built into the library (but that means that its a typescript library)

No more ‘var’

- ‘Var’ is now an antipattern with typescript
- No hoisting issues
- Instead use
  - ‘let’ for mutable values
  - ‘const’ for immutable values

Classes

Interfaces

- Define the shape of the data

Type Aliases

- Aliases the type to a more meaningful name
- Id’s or values are very useful to track but they are just strings
- Aliases allow us to give more meaningful context to the name
- Most IDE’s allow us to find by type (i.e., Show me where ‘AlertId’ type is used)
- For instance:

```
type int = number;
type float = number;

// Or

type TenantId = string;
type AlertId = string;
type AlertType = string;
```

Strict Null Checking

- Compiler Option ‘—strictNullCheck’

```
// Compiled with --strictNullChecks
let x: number;
let y: number | undefined;
let z: number | null | undefined;
x = 1;  // Ok
y = 1;  // Ok
z = 1;  // Ok
x = undefined;  // Error
y = undefined;  // Ok
z = undefined;  // Ok
x = null;  // Error
y = null;  // Error
z = null;  // Ok
x = y;  // Error
x = z;  // Error
y = x;  // Ok
y = z;  // Error
z = x;  // Ok
z = y;  // Ok
```

Enums

Rest/Spread Operator

- Shallow Copy

```
let copy = {...original};
```

- Merge Multiple Objects together

```
let merged = {...defaults, ...overrides};
```

- Can be used instead of ‘Object.assign'

Union and Type Guards

- Combines multiple disparate types together

Tuples

- Typed ‘Columns’ of data

```
let user: [number, string] = [2, 'mtho11'];
let userList: [number, string][] = [[2, 'mtho11'], [3, 'joed']];
```

Modules

- Namespacing
- Control visibility of individual classes and variables via ‘import/export’
- Can be nested
- Module Merging

Where is it Used Now

Currently, typescript is being used with Angular 1 components in http://github.com/manageiq/ui-components/

Although it should be used more widely in the codebase soon with Angular 2+

Modules

alalal

lslsals

lsls

When can I use it?

So when can we use this in ManageIQ?

Dunno? Probably 2-4 months from now. We are currently doing PoC with various technologies and winners will be announced.

More…

- Asynch/await
- Generics

Resources

- Links to this presentation: https://github.com/mtho11/presentations-2017/blob/master/Typescript.key