TypeScript

Udaiappa Ramachandran (Udai)
//linkedin.com/in/udair

Who am I?

- Udaiappa Ramachandran (Udai)
- CTO, Akumina, Inc.,
- Consultant
- Azure Insider
- New Hampshire Cloud User Group (http://www.meetup.com/nashuaug)
- Focus on Cloud Computing (Microsoft Azure and AWS), IoT, SharePoint Online
- http://cloudycode.wordpress.com
- @nhcloud

TypeScript - Agenda

- Why & What
- Basic Types
- Interfaces
- Functions
- Classes
- Generics
- Modules
- Demo
- References
- Q & A

Typescript

- Why
  - Javascript is dynamic type
    - Pro– can hold any object, type on the fly
    - Con- Can get messy over the time
  - Migration from server-side to client-side will be hard
  - Hard to manage, difficult to ensure property types
- What
  - Any valid JavaScript is a typescript
  - Typescriptlang.org
    - Typescript lets you write JavaScript the way you really want to.
    - Typescript is a typed superset of JavaScript that compiles to plain JavaScript.
    - Any browser. Any host. Any OS. Open Source.

Typescript Alternatives

- Pure JavaScript
- Apply JavaScript patterns
  - Functions as abstractions
  - Functions to build modules
  - Functions to avoid global variables
- CoffeeScript
- Dart

Typescript Key Features

- Supports standard JavaScript code
- Provide static typing
- Encapsulation through classes and modules
- Support for constructors, properties, functions
- Define interfaces
- Lambda style function support
- Intellisense and syntax checking

Typescript tools

- Typescript playground
- Visual Studio
- sublime
- Node.js
- WebStorm
- Eclipse
- Vi
- IntelliJ
- Emacs

Typescript to JavaScript

TypeScript | Walkthrough: Classes | Share

```typescript
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

var greeter = new Greeter("world");

var button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function() {
    alert(greeter.greet());
}

document.body.appendChild(button);

```

Run | JavaScript

```javascript
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
})();
var greeter = new Greeter("world");
var button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function () {
    alert(greeter.greet());
};
document.body.appendChild(button);

```

Tyepscript BasicTypes

- Boolean
- Number
- String
- Array
- Enum
- Enum as Bit Flag 1,2,4,8,16,32,64,128 and so on
- Any
- Void

Tyepscript Annotation

- Type Annotation
  - var [identifier]:[type annotation]=value
  - var [identifier]:[type annotation];
  - var [identifier]=value;
- Example

```
var isValid: boolean = false;//Boolean
var latitude: number = 42.7575;//Number
var name: string = "TypeScript";//String
enum Color { Red, Blue, Green };//Enum
var color: Color = Color.Red;
enum ColorFlag { Red = 1, Blue = 2, Green = 4 };//Enum with Flag
var arrNum: number[] = [1, 2, 3];//Array of Number
var listNumArr: Array<number> = [1, 2, 3];//Array of Number using Array
var anyType: any = 4;//Number
anyType = "TypeScript";//assigning dynamic type
var listAny: any[] = [1, true, "TypeScript"];//Dynamic type array
listAny[1] = 100;//Changing boolean to number
```

Typescript Functions

- Optional Parameters using ?
  - function getAverage(a: number, b: number, c?: number): void { }
- Default parameters using =value
  - function concatenate(items: string[], separator = ",", beginAt = 0, endAt = items.length) :void{ }
- Rest parameter using …
  - Only one allowed, it must appear last in the parameter list and must be an array type
  - function getSum(...a: number[]): number {
  - var t = 0;a.forEach(p=>t=t+ p);
  - return t;
  - }
  - var result = getSum(1, 2, 3, 4);
- Overloads
  - Overloads in typescript cannot have own implementation but decorate a single implementation
  - function getSum(a: string, b: string, c: string): number;
  - function getSum(a: number, b: number, c: number): number;
  - function getSum(a: any, b: any, c: any): number {
  - // implementation signature
  - return parseInt(a, 10) + parseInt(b, 10) + parseInt(c, 10);
  - }
- Arrow function
  - var getSum: (a: number, b: number) => number =(x, y) => (x + y);

Typescript Interfaces

- Interfaces are used at design time to provide auto completion and at compile time to provide type checking
- Supported features
  - Optional properties
  - Function Types
  - Array Types
  - Class Types
  - Extending Interfaces
  - Hybrid Types

Typescript Interfaces

```typescript
//Interface as a class with optional property
interface UserResponse {
    firstName: string;
    lastName: string;
    middleName?: string;
}

//Interface
interface IAddress {
    getAddress(id: number): any;
}

//Extending Interface
interface IUser extends IAddress {
    getUser(id: number): UserResponse;
    getAllUsers(): UserResponse[];
}

//Interface function
interface ISearchFunc {
    (source: string, subString: string): boolean;
}

//Array Type
interface IStringArray {
    [index: number]: string; length: number;
}

//Hybrid Type
interface ICounter {
    (start: number): string; interval: number; reset(): void;
}
```

Typescript Classes

- Object-oriented class based approach
- Key features
  - Inheritance
  - Private/public modifiers
  - Accessors
  - Static properties
  - Constructor functions
  - Using class as an interface

Typescript Classes

```typescript
//User DTO
class UserResponse {
}

//User Inteface for operation contract
interface IUser {
    get():UserResponse;
}

//User implementation
class User implements IUser {
    static pwdsalt = "TypeScript";//static example
    constructor(firstName: string, lastName: string) { }//constructor example
    get(): UserResponse {
        return new UserResponse;
    }
}

//get or set
class Employee {
    private _fullName: string;
    get fullName(): string {return this._fullName;}
    set fullName(newName: string) {this._fullName = newName;}
}

//private, public
class UserFullNameProp {
    fullName: string;
    constructor(fullName: string) {
        this.fullName = fullName;
    }
}
class UserProp extends UserFullNameProp {
    fullName: string;
    constructor(fullName: string) {
        super(fullName);//caling base class
    }
}
class UserProp2 implements UserFullNameProp {
    constructor(public fullName: string) {//constructor defined as public to avoid private declaration of the property defined in base class
        this.fullName = fullName;
    }
}
```

Typescript Generics

- Supports generic type variables, types, interfaces, classes and constraints

```
function identity<T>(arg: T): T {
    return arg;
}
```

```
var output = identity<string>("Nashua");//identity("myString");
function loggingIdentity<T>(arg: T[]): T[] {//T[] can be written as Array<T>
    console.log(arg.length);  // Array has a .length, so no more error
    return arg;
}
```

Typescript Modules

- Encapsulate variables, interfaces, and classes
  - Define unique namespaces
  - Organize symbols and identifiers into a logical namespace hierarchy
  - Similar to namespaces/packages
- Splitting across files
  - Multiple files can use the same module name
- One file can contain multiple module
- Can define Alias to module
- Transpiles to IIFE
- Can define modules as internal or external
- External modules required only when used with node.js and require.js

Typescript Modules

```
module App.Demo {//module can be in separate file
    export interface IUser {
        getAll(): Array<UserResponse>;
        get(id: number): UserResponse;
    }
}

module App.Demo {//another module
    export class UserResponse {
        id: number;
        name: string;
    }
    export class User implements IUser {
        users: Array<UserResponse>;
        constructor() {
            this.users = [{ "id": 1, "name": "udai" }, { "id": 2, "name": "udai2" }];
        }
        getAll(): Array<UserResponse> {
            return this.users;
        }
        get(id: number): UserResponse {
            this.users.forEach(entry => {
                if (entry.id === id) return <UserResponse>entry;
                return <UserResponse>entry;
            });
            return null;
        }
    }
}

import ref = App.Demo;//Import
var user = new ref.User();
user.getAll().forEach(p=> console.log(p.name));
```

Typescript Declaration Merging

- Concept
- Merging Interfaces
- Merging Modules
- Merging Modules with classes, functions, and Enums
- Disallowed Merges

Typescript Type inference and Compatibility

- Type inference
  - Basics
  - Best common type
  - Contextual Type
- Type Compatibility
  - Starting out
  - Comparing two functions
  - Enums
  - Classes
  - Generics
  - Advanced Topics
- Common Errors
- Mixins

Typescript Definition Files

- Describes the types defined in external libraries
- .d.ts
- Not deployed
- Usually from DefinitelyTyped
- TypeScript Definition manager (tsd)
  - Specialized package manager
  - Locates and installs typescript definition files(d.ts)
  - From the definitelytyped repository

Demo

Reference

- //typescriptlang.org

Q & A