TypeScript Fundamentals

Peter J. Jones
pjones@devalot.com
@devalot
http://devalot.com

DEVALOT

Introduction to TypeScript

What is TypeScript

- A language based on ESNEXT
- Compiles to ES5
- Contains the following additional features:
  - Types and type inference!
  - Generics (polymorphic types)
  - Interfaces and namespaces
  - Enums and union types

Type Annotations

```
function add(x: number, y: number): number {
  return x + y;
}
```

Type Checking

```
// Works!
const sum = add(1, 2);

// error: Argument of type '"1"' is not assignable
// to parameter of type 'number'.
add("1", "2");
```

Type Inference

```
// Works!
const sum = add(1, 2);

// error: Property 'length' does not exist
// on type 'number'.
console.log(sum.length);
```

Types

Built-in Types

JavaScript Types

- `Array` and other classes
- `boolean`
- `null`
- `number`
- `string`
- `undefined`

TypeScript Additions

- `any`
- `never`
- `object`
- `void`

Interfaces

Describing Types Using Interfaces

```
interface HasName {
  firstName: string;
  lastName: string;
}

function fullName(x: HasName): string {
  return `${x.firstName} ${x.lastName}`;
}
```

Classes

Declaring Properties

```
class Person {
  firstName: string;
  lastName: string;

  constructor(firstName: string, lastName: string) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
}
```

Parameter Properties

```
class Car {
  constructor(public make: string) {}
}

let c = new Car("Toyota");
console.log(c.make); // Toyota
```

Generics

Generic Classes

```
class Queue<T> {
  private entries: Array<T> = [];
  push(e: T) { this.entries.push(e);          }
  pop()      { return this.entries.shift(); }
}
```

Generic Functions (Part 1)

```
function log<T>(x: T): void {
  console.log(x);
}
```

Generic Functions (Part 2)

```
interface Ord {
  cmp(other: Ord): number;
}

function sort<T extends Ord>(xs: Array<T>): void {
  xs.sort((a, b) => a.cmp(b));
}
```

Enums

Enumerated Values

```
enum Compare {
  EQ,
  LT,
  GT,
}

function compare(a: number, b: number): Compare {
  if (a === b)     return Compare.EQ;
  else if (a < b) return Compare.LT;
  else             return Compare.GT;
}
```

Enums Can't Be Checked for Exhaustiveness

```
enum Compare {
  EQ,
  LT,
  GT,
}

function sortNums(ns: Array<number>): void {
  ns.sort((a: number, b: number): number => {
    switch (compare(a, b)) {
      case Compare.EQ: return  0;
      case Compare.LT: return -1;
      case Compare.GT: return  1;
      default:         return  0;
    }
  });
}
```

Enums Are Weakly Typed Like in C

```
enum Compare {
  EQ,
  LT,
  GT,
}

function logCompare(c: Compare): void {
  console.log(c);
}

// Works!  WTF?
logCompare(15);
```

Unions

Simple Union Types

```
function logAsString(msg: number | string): void {
  if (typeof msg === "number") {
    console.log(`msg is a number: ${msg}`);
  } else {
    console.log(`msg is a string: ${msg}`);
  }
}
```

Discriminated Unions

```
interface EQ {kind: "eq"}
interface LT {kind: "lt"}
interface GT {kind: "gt"}

type Compare2 = EQ | LT | GT;
```

Using Discriminated Unions

```
interface EQ {kind: "eq"}
interface LT {kind: "lt"}
interface GT {kind: "gt"}

type Compare2 = EQ | LT | GT;

function compare2(a: number, b: number): Compare2 {
  if (a === b)    return {kind: "eq"};
  else if (a < b) return {kind: "lt"};
  else            return {kind: "gt"};
}
```

Exhaustiveness Checking

```
interface EQ {kind: "eq"}
interface LT {kind: "lt"}
interface GT {kind: "gt"}

type Compare2 = EQ | LT | GT;

function sort2(ns: Array<number>): void {
  ns.sort((a: number, b: number): number => {
    switch (compare2(a, b).kind) {
      case "eq": return  0;
      case "lt": return -1;
      case "gt": return  1;
    }
  });
}
```