TypeScript 类型系统

分 享 人：陈文岗

学　　校：中国科学院大学

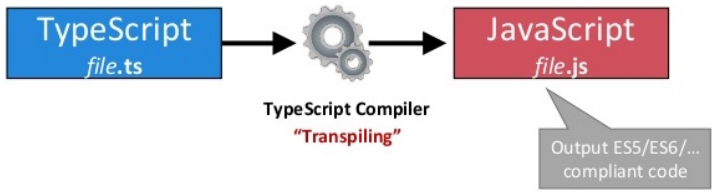
2021年1月21日

关于TypeScript

TypeScript是微软推出的JavaScript静态类型版本，它是

JavaScript的超集，可以编译为纯粹的JavaScript

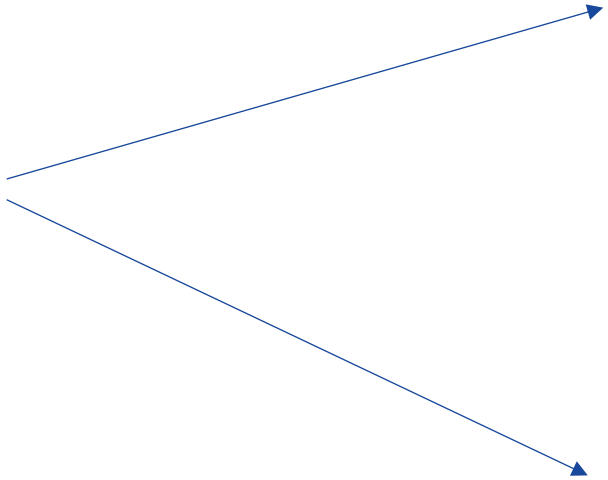
How Does TypeScript Work?



TypeScript 到 JavaScript

```ts
function add(a: number, b: number): number {
  return a + b;
}
```

add.ts



```js
function add(a, b) {
  return a + b;
}
```

add.js

```ts
function add(a: number, b: number): number;
```

add.d.ts

TypeScript 基础类型

```ts
// 数字类型
let num: number;
num = 123;

// 布尔类型
let flag: boolean;
flag = false;

// 字符串类型
let str: string;
str = 'Hello World';
```

```ts
// 类型别名
type Str = string;
type Num = number;
type Bool = boolean;
```

TypeScript 高级类型

```ts
// 数组类型
let arr: number[];
arr = [1, 2, 3, 4];
```

```ts
// 接口
interface Person {
  name: string;
  age: number;
}

let p1: Person = {
  name: 'chenwengang',
  age: 23
};
```

```ts
// 内联接口
let p2: {
  name: string;
  age: number;
} = {
  name: 'xxx',
  age: 25
};
```

```ts
// 函数类型/函数签名
type Callback = () => void;

let cb: Callback = () => {
  console.log('callback');
};
```

TypeScript 高级类型

```ts
// 联合类型
type keyType = string | number;

const key1: keyType = 'key';
const key2: keyType = 1;
```

```ts
// 交叉类型
type User1 = {
  name: string;
  age: number
};

type User2 = {
  sex: number
}

const user: User1 & User2 = {
  name: 'xxx',
  age: 23,
  sex: 0
};
```

TypeScript 高级类型

```ts
// 字面量类型
let number1: 1;

let oneChar = '1';

let falseLiteral: false;

type NumberChar = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
```

```ts
// 元组类型
let pair: [string, number] = ['xxxx', 2];
let triple: [string, number, boolean] = ['yyyy', 1, false];
```

TypeScript 高级类型

```ts
// 数字枚举
enum Color {
  Red,
  Green,
  Blue,
}

let col: Color = Color.Red;
```

```ts
// 字符串枚举
enum Tristate {
  False = 'False',
  True = 'True',
  Unknown = 'Unknown'
}
```

TypeScript 特殊类型

```typescript
// any类型兼容所有类型(关闭类型检查)
let x: any;

// null字面量类型(可以被赋给任意类型)
let y: number;
y = null;

// undefined字面量类型(可以被赋给任意类型)
let z: string;
z = undefined;

// void类型表示函数没有返回值
function func(): void {}

// unknown类型(Top Type)
let foo: unknown;

// never类型(Bottom Type)
let bar: never;
```

TypeScript 结构类型系统

```cpp
#include <iostream>
#include <string>

using namespace std;

struct Person1 {
    string name;
    int age;
};

struct Person2 {
    string name;
    int age;
};

int main() {
    Person1 p1;
    Person2 p2 = p1;
}
```

No viable conversion from 'Person1' to 'Person2'

Change type of local variable 'p1' to 'Person2' ⌥⇧⏎ More actions... ⌥⏎

Person2 p2 = p1

```ts
interface Person1 {
  name: string;
  age: number;
}

interface Person2 {
  name: string;
  age: number;
}

let p1: Person1 = {
  name: 'xxx',
  age: 23,
};

let p2: Person2 = p1;
```

标明类型系统

即使两个类的结构完全一致，也不能互相赋值

结构类型系统

类型形状一致即可互相赋值

TypeScript 类型声明空间

```ts
class Foo {}
interface Bar {}
type Bas = {};

let foo: Foo;
let bar: Bar;
let bas: Bas;
```

类型声明空间里包含用来当做类型注解的内容

```ts
interface Bar {}

let bar2 = Bar;
```

```
any
“Bar”仅表示类型，但在此处却作为值使用。 ts(2693)
速览问题 (⌥F8)    没有可用的快速修复
```

类型不能赋给一个变量，也不能作为值进行传递（class除外）

TypeScript 变量声明空间(值空间)

```ts
class Foo {}

const foo = Foo;
```

变量声明空间包含可用作变量的内容

class既属于类型声明空间，也属于变量声明空间

```ts
const foo = 123;

let bar: foo;
```

```
any
“foo”表示值，但在此处用作类型。是否指“类型 foo”? ts(2749)
速览问题 (⌥F8)    没有可用的快速修复
```

普通的变量/常量不能用作类型注解

TypeScript 函数重载

```ts
function add(a: number, b: number): number;
function add(a: number, b: number, c: number): number;
function add(a: number, b: number, c: number, d: number): number;
function add(a: number, b: number, c?: number, d?: number): number {
  if(c === undefined) {
    return a + b;
  }
  if(d === undefined) {
    return a + b + c;
  }
  return a + b + c;
}

add(1, 2);
add(1, 2, 3);
add(1, 2, 3, 4);
```

1. TypeScript类型信息只存在于编译期，不会带到运行期
2. TypeScript要与JavaScript兼容

TypeScript 接口

```ts
interface Person {
  name: string;
}

interface Person {
  age: number;
}
```

接口合并

```ts
interface Person {
  name: string;
  age: number;
}

interface Student extends Person {
  school: string;
}
```

接口继承

```ts
interface IPerson {
  name: string;
  age: number;
}

class Person implements IPerson {
  name: string;
  age: number;
  constructor(name: string, age: number) {
    this.name = name;
    this.age = age;
  }
}
```

类实现接口

TypeScript 接口

```ts
// 可调用
interface Animal {
  (content: string): string;
}

// 可索引
interface Good {
  [name: string]: string;
}

// 可实例化
interface TaskConstructor {
  new (name: string, taskStatus: number);
}

// 空接口(可接受任意类型的赋值)
interface All {}
```

TypeScript 类型断言

```ts
let a: any = 1;

let b1 = a as number;
let b2 = <number>a;
```

TypeScript 类型守卫/类型收敛

当遇到下面这些条件语句时，TypeScript会在语句作用域内部收敛变量的类型

类型断言：`as`

类型判断：`typeof`

实例判断：`instanceof`

属性判断：`in`

字面量相等判断：`==`, `===`, `!=`, `!==`

TypeScript 类型守卫/类型收敛

```ts
function func(a: string | number): string {
  if(typeof a === 'string') {
    return a.toLowerCase();
  } else {
    return a.toFixed(1);
  }
}
```

此处a为string类型

此处a为number类型

```ts
type Square = {
  kind: 'square';
  size: number;
};

type Rectangle = {
  kind: 'rectangle';
  width: number;
  height: number;
};

type Circle = {
  kind: 'circle';
  radius: number;
}

type Shape = Square | Rectangle | Circle;

function area(shape: Shape): number {
  switch (shape.kind) {
    case 'square':
      return shape.size * shape.size;
    case 'rectangle':
      return shape.width * shape.height;
    case 'circle':
      return Math.PI * shape.radius * shape.radius;
    default:
              (parameter) shape: never
      let c = shape;
  }
}
```

此处shape为Square类型

此处shape为Rectangle类型

此处shape为Circle类型

TypeScript 泛型编程

```ts
function reverse<T>(arr: T[]): T[] {
  let retArr: T[] = [];
  for(let i = arr.length - 1; i >= 0; --i) {
    retArr.push(arr[i]);
  }
  return retArr;
}
```

泛型函数

```ts
class Vector<T> {
  data: T[];
  push(item: T) {
    this.data.push(item);
  }
}
```

泛型类

TypeScript 泛型编程

```
T extends U ? X : Y
```

条件泛型

```
type TypeName<T> = T extends string
  ? 'string'
  : T extends number
  ? 'number'
  : T extends boolean
  ? 'boolean'
  : T extends symbol
  ? 'symbol'
  : T extends undefined
  ? 'undefined'
  : T extends Function
  ? 'function'
  : 'object';

type T = TypeName<string>;
// equivalent to
type T = 'string';
```

TypeScript 泛型编程

```ts
type ReturnType<T extends (...args: any) => any> = T extends (...args: any) => infer R ? R : any;

type ParamsType<T extends (...args: any) => any> = T extends (...args: infer R) => any ? R : any;
```

infer运算符在extends条件语句中表示待推测的类型量

TypeScript 泛型编程

```ts
// 把传入类型T中的各项属性转化为可选属性
type Partial<T> = {
  [P in keyof T]?: T[P];
};

// 把传入类型T中的各项属性转化为必选属性
type Required<T> = {
  [P in keyof T]-?: T[P];
};

// 把传入类型T中的所有属性转化为只读属性
type Readonly<T> = {
  readonly [P in keyof T]: T[P];
};

// 从类型T中挑出部分属性来构成一个新的类型
type Pick<T, K extends keyof T> = {
  [P in K]: T[P];
};

// 利用一个字符串字面量类型生成一个对象类型
type Record<K extends keyof any, T> = {
  [P in K]: T;
};
```

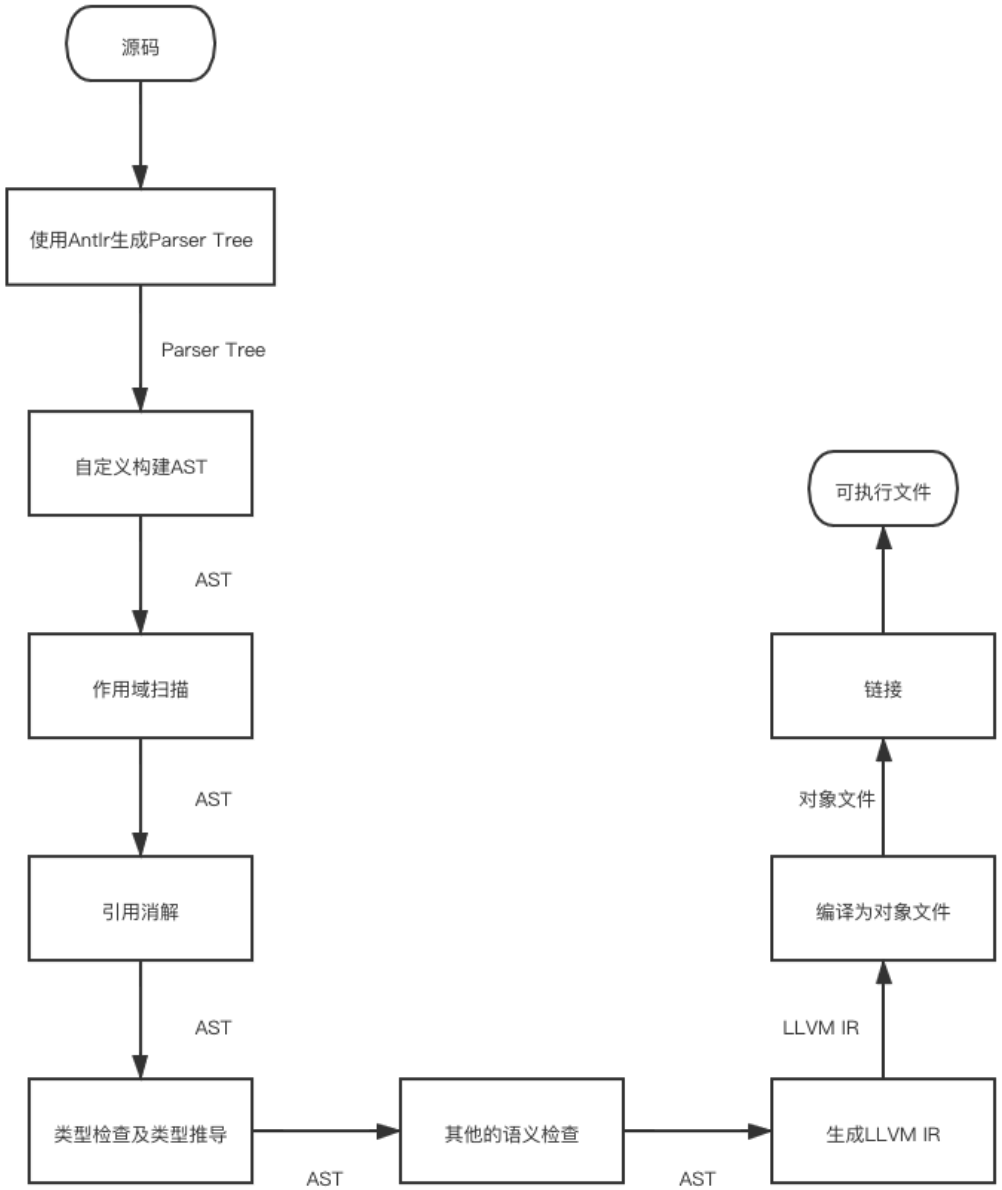
```ts
interface User {
  name: string;
  age: number;
}

type PartialUser = Partial<User>;
```

```ts
type PartialUser = {
  name?: string;
  age?: number;
}
```

其他的Utility Type工具类型……

StaticScript | 仿TypeScript静态类型编程语言



基础框架：Antlr + LLVM

特性：int / float / boolean / string / array+常见基础控制语句

状态：开发中

我目前的研究方向

目标：基于静态分析寻找脆弱的Node.js代码

需要做的：

1. 改造TypeScript编译器
2. 搭建TypeScript静态分析框架
3. 设计实现基于静态污点分析的漏洞检测技术
4. 设计实现面向Node.js应用的安全测试框架

Q & A