上帝视角看 TypeScript

TypeScript 的学习资料非常多，其中也不乏很多优秀的文章和教程。但是目前为止没有一个我特别满意的。原因有：

* 它们大多数没有一个清晰的主线，而是按照 API 组织章节的，内容在**逻辑上**比较零散。
* 大多是“讲是什么，怎么用“，而不是”讲为什么，讲原理“。
* 大多数内容比较枯燥，趣味性比较低。都是干巴巴的文字，没有图片，缺乏能够引起强烈共鸣的例子。

因此我的想法是做一套不同市面上大多数的 TypeScript 学习教程。以人类认知的角度思考问题，学习 TypeScript，通过通俗易懂的例子和图片来帮助大家建立 TypeScript 世界观。 而本篇文章则是这个系列的开篇。

系列安排：

* 上帝视角看 TypeScript（就是本文）
* TypeScript 类型系统
* 什么是 types？什么是 @types？
* 类型推导， 类型断言与类型保护
* [你不知道的 TypeScript 泛型（万字长文，建议收藏）](https://lucifer.ren/blog/2020/06/16/ts-generics/)（已发布）
* TypeScript 练习题
* TypeScript 配置文件该怎么写？
* TypeScript 是如何与 React，Vue，Webpack 集成的？

目录将来可能会有所调整。

注意，我的系列文章基本不会讲 API，因此需要你有一定的 TypeScript 使用基础，推荐两个学习资料。

* [深入理解 TypeScript](https://jkchao.github.io/typescript-book-chinese/)
* [官方文档](https://www.typescriptlang.org/docs/home)

结合这两个资料和我的系列教程，掌握 TypeScript 指日可待。

接下来，我们通过几个方面来从宏观的角度来看一下 TypeScript。

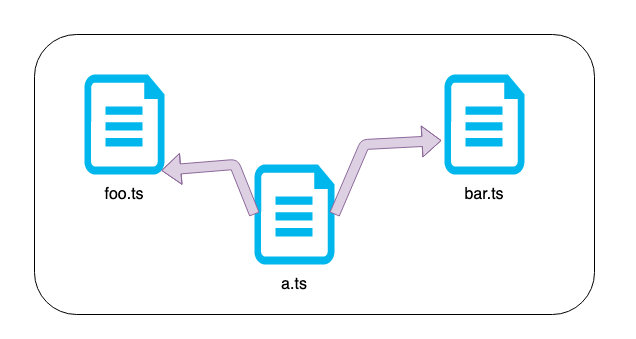
## 从输入输出上来看

如果我们把 Typescript 编译器看成一个黑盒的话。其**输入则是使用 TypeScript 语法书写的文本或者文本集合**。



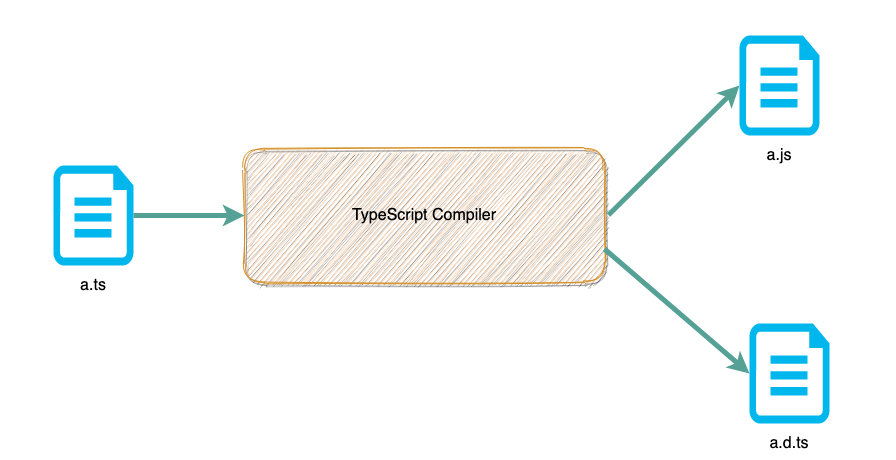
（文本）

如果几个文本有引用关系，比如 a.ts 依赖 foo.ts 和 bar.ts，其就是一个文本集合。



（文本集合）

**输出是编译之后的 JS 文件 和 .d.ts 的声明文件**。



其中 JS 是将来需要运行的文件，而 .d.ts 声明文件则是 ts 文件中的类型声明，**这个类型声明就是你在 ts 文件中声明的类型和 TypeScript 类型推导系统推导的类型**。当然你也可以自己写 .d.ts 声明文件。

## 从功能上来看

从宏观的视角来看，TypeScript 的功能就是：

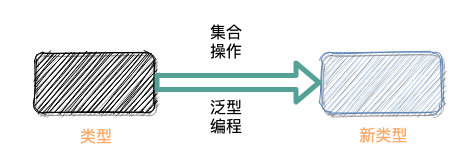
* 提供了丰富的类型系统。

最简单的就是 变量名:类型 = 值

const a: Number = 1;

除了这些基本类型，还提供了函数类型，复合类型等。

* 提供了类型操作 API。TypeScript 不但提供内置类型，用户也可以利用集合操作和泛型对类型操作从而生成新的类型。



* 对每一种类型的属性和方法都进行了定义。

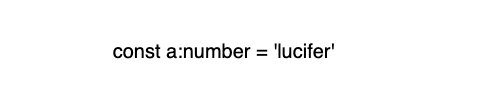
比如 String 类型有 toString 方法，但是没有 toFixed 方法，这就是 lib.d.ts 定义的。这样我在 String 类型的变量上使用 toFixed 方法就会报错，达到了“类型检查”的作用。

小提示：lib.d.ts 的内容主要是一些变量声明（如：window、document、math）和一些类似的接口声明（如：Window、Document、Math）。 你可以通过 --noLib 来关闭这一功能

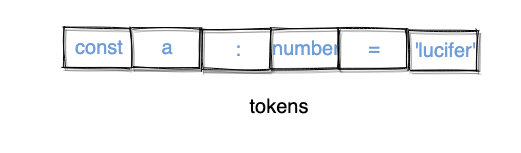
* 提供了模块系统（module，namespace）。
* 提供了更加方面的 API，比如 class（这在 ES6 class 出来之前尤其好用），装饰器等。
* 。。。

## TypeScript 编译器是如何工作的？

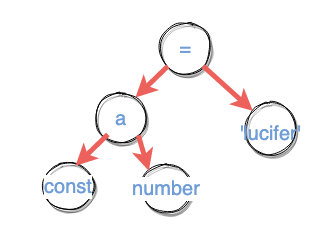
上面已经讨论了 TypeScript 编译器的输入和输出。那黑盒内部是怎么工作呢？这里我简单介绍一下：



* TypeScript 文本首先会被解析为 **token 流**。这个过程比较简单，就是单纯地按照分隔符去分割文本即可。

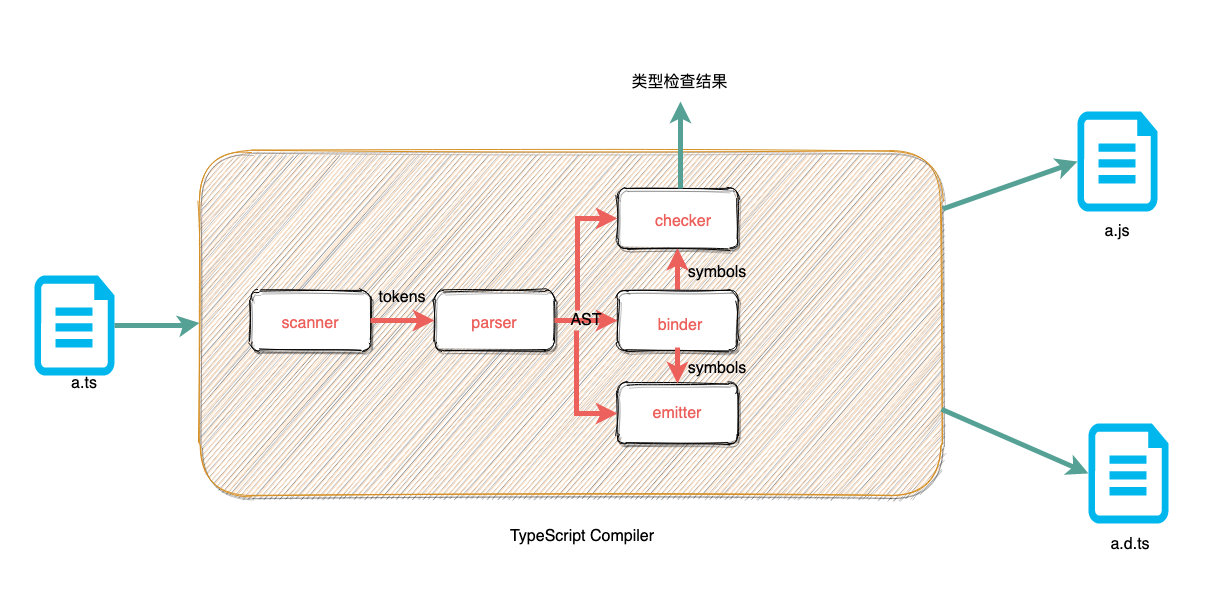


* 接着 token 流会被转换为 AST，也就是**抽象语法树**。



* binder 则根据 AST 信息生成 **Symbol**（TypeScript 中的一个数据结构）。拿上面的图来说，就是 number 节点。
* 当我们需要类型检查的时候， checker 会根据**前面生成的 AST 和 symbols 生成类型检查结果**。
* 当我们需要生成 JS 文件的时候，emitter 同样会根据**前面生成的 AST 和 symbols 生成 JS 文件**。

完整图：



## 总结

总的来说，TypeScript 就是一门语言，和 Java，Python，C++ 等类似。只不过这门语言主要目标就是为了弥补 JavaScript 弱类型带来的问题的。因此设计语言的出发点就是：

* 静态类型系统
* 可以编译成 JavaScript

因此 TypeScript 是一门最终编译为 JavaScript 的语言（当然还有类型文件）。既然是一门语言，就涉及词法分析，语法分析等流程。由于相对 JavaScript 增加了很多功能， 其中最主要的就是类型系统。因此 TypeScript 的分析工作要比 JavaScript 更加复杂， 集中体现在 binder 和 checker 部分。

由于提供了静态类型， 因此就需要提供一些内置类型给我们用，比如 number，string，Array 等。但是这并不能满足我们的所有需求，我们需要自定义类型，因此有了 type，有了 interface 等。后来我们又发现自定义的类型重复代码太多， 要是类型也可以通过编程生成新的类型就好了，于是有了集合运算和泛型。

代码都放到一起不方便维护，要是可以放到不同文件，需要用的时候组装起来就好了，于是有了模块化。我用了别人的用 TypeScript 开发的库，如果也能有类型校验就好了，于是有了 types。

。。。

其实这些都是有因果关系的，如果你可以牢牢地掌握这些因果关系，那么学起来还不是易如反掌？

## 相关阅读

* [TypeScript 编译原理](https://jkchao.github.io/typescript-book-chinese/compiler/overview.html)
* [Bring your own TypeScript with more internal definitions](https://github.com/basarat/byots)
* [Compiler Internals](https://github.com/microsoft/TypeScript/wiki/Compiler-Internals)
* [TypeScript 编译器是用 TypeScript 写的，那是先有编译器还是 TS？](https://github.com/azl397985856/fe-interview/issues/135)

## 点关注，不迷路

大家也可以关注我的公众号《脑洞前端》获取更多更新鲜的前端硬核文章，带你认识你不知道的前端。



公众号【 [力扣加加](https://tva1.sinaimg.cn/large/007S8ZIlly1gfcuzagjalj30p00dwabs.jpg)】 知乎专栏【 [Lucifer - 知乎](https://www.zhihu.com/people/lu-xiao-13-70)】