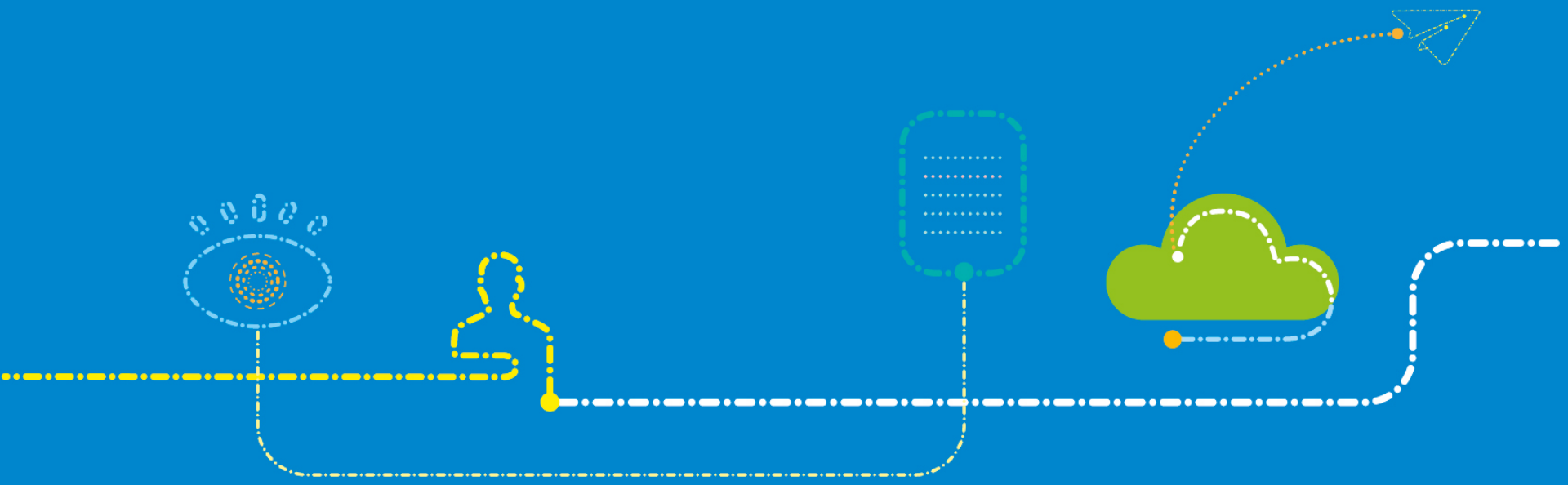
深度玩转 typescript + angular 编译器

ZTE中兴

通过对编译器的深度定制以节约90%的编译时间

中兴大数据 陈旭
https://github.com/rdkmaster



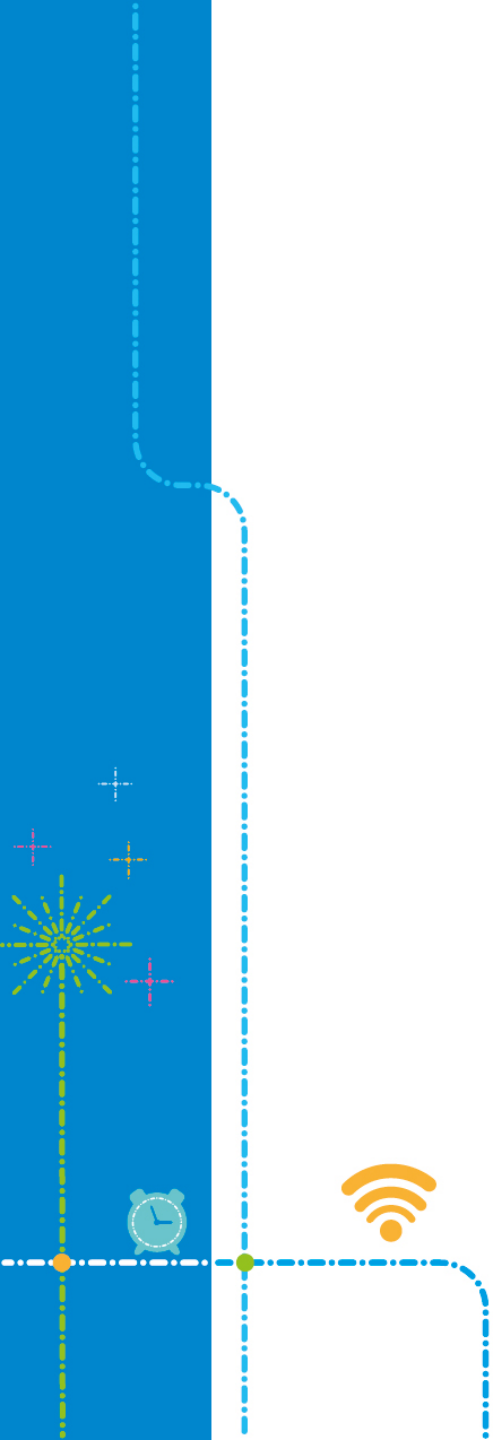
目 录

关于我

关于我

我个人微信 欢迎交流
备注NodeParty

https://github.com/rdkmaster/jigsaw
帮忙点个星！



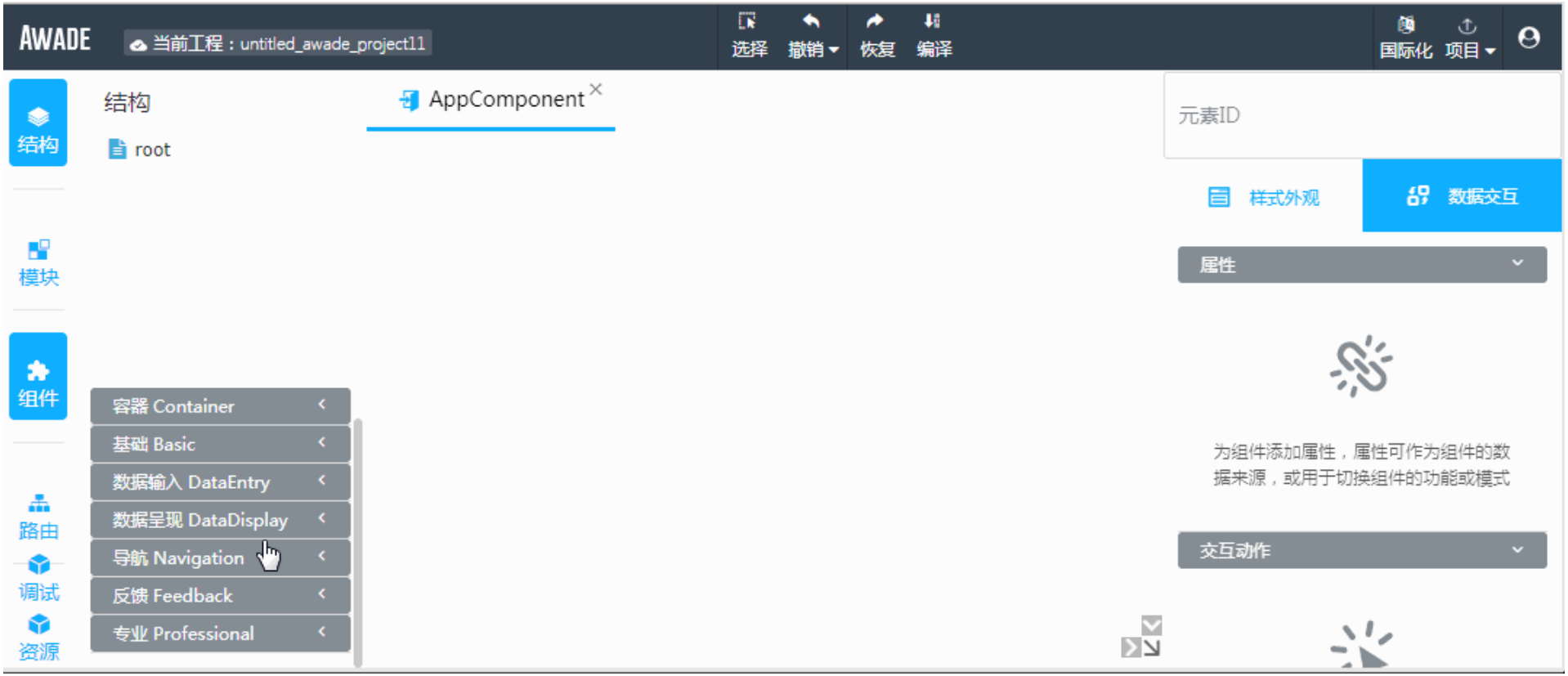
背景

关于Awade

背景 - 关于Awade

- Awade是一个前后端一体化的在线可视化Web应用开发IDE
- 适用于开发PC端Web应用，这类Web应用具备如下特点：
  - 绝大多数带有复杂的交互过程，PC端的交互过程比移动端要复杂许多
  - 对页面的视觉要求很高，典型如监控类/Dashboard类应用
- 目前已经实现了少量代码开发，而无代码开发是Awade终极目标
- 目前已经实现了零Web技能，会编码就能开发Web应用的目标，而让人人都是Web应用开发专家是Awade的终极目标

背景 - 关于Awade - 布局

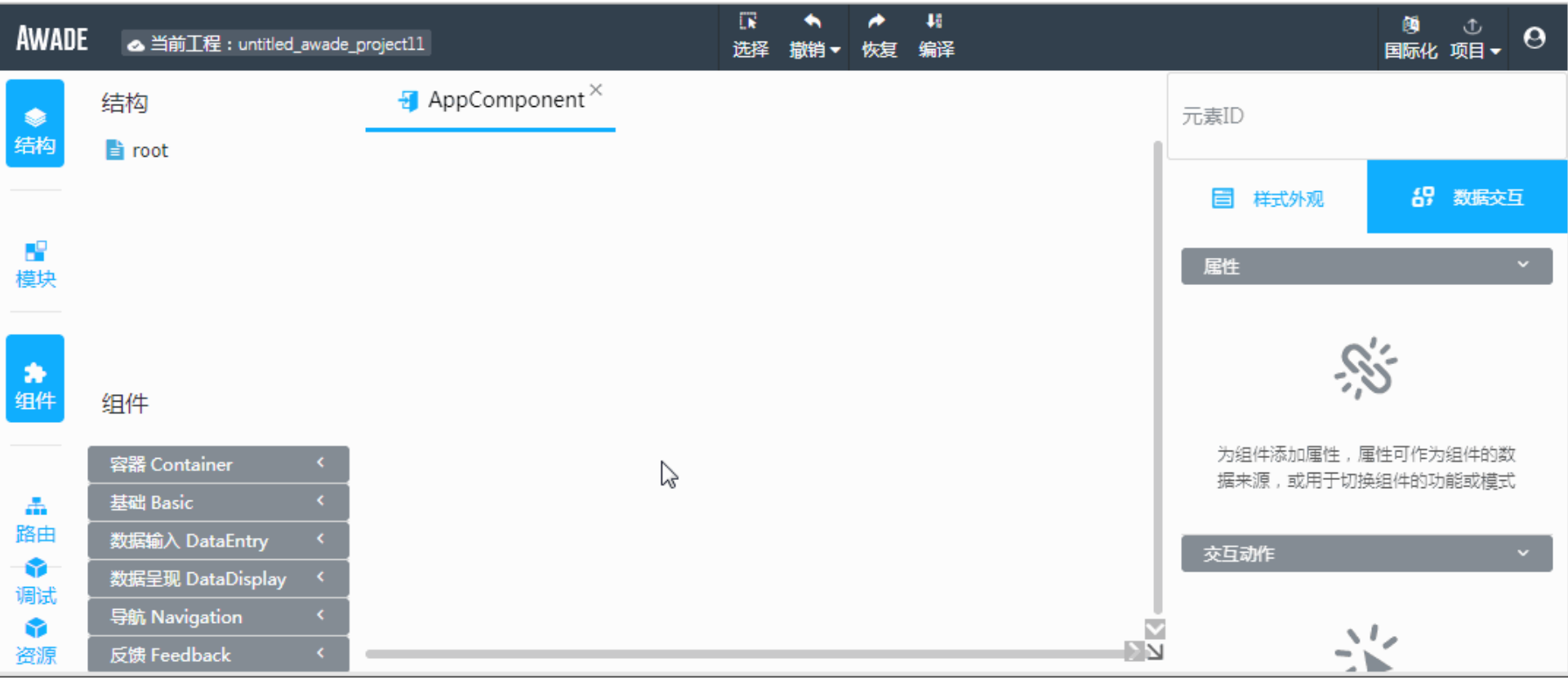


背景 - 关于Awade - 交互

背景 - 关于Awade - 同步编程

背景 - 关于Awade - 撤回/重做修改

背景 - 关于Awade - 容器/嵌套容器



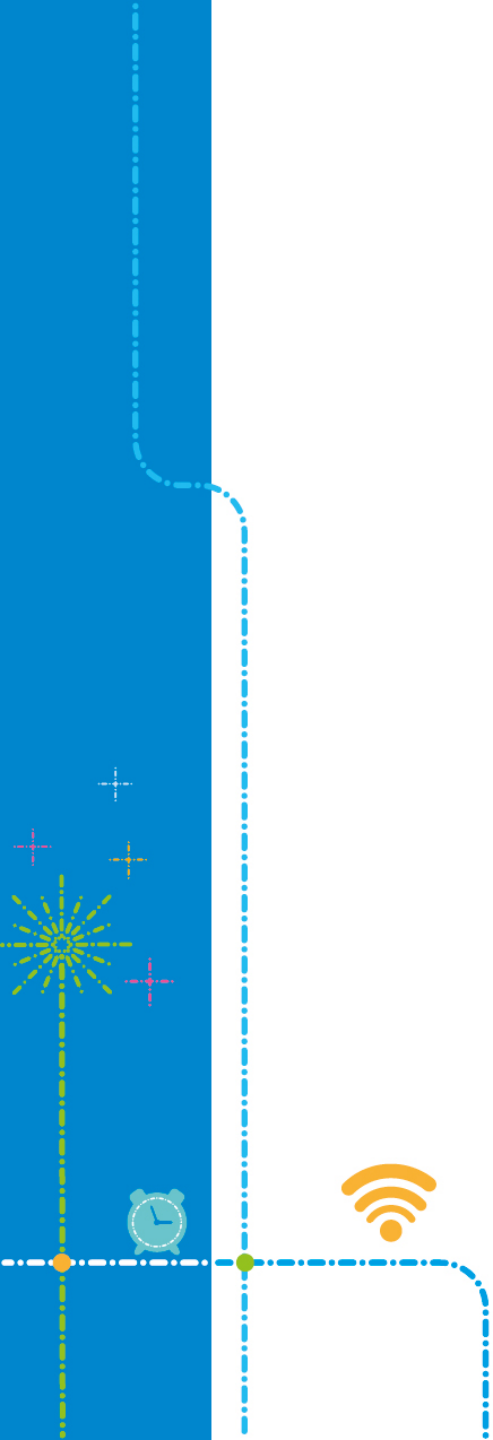
背景 - 实时编译

- 为什么要实时编译?
  - 在可视化开发过程中，有部分变更可以直接对组件实例操作来实现，无需实时编译
  - 但，更多的变更是需要实时编译的，主要是涉及到交互逻辑的变更、服务端Rest服务的逻辑变更
  - 而且，只有实时编译才能尽可能的实现所见即所得的目标
  - 既然实时编译必不可少，因而为了简化实现，我们选择了全部变更都执行实时编译

背景 - 实时编译

- 实时编译的困境：

效率！效率！效率！



解决方案

增量编译

解决方案 - 增量编译

- AngularCli 提供了非常方便的增量编译能力，任意变更的增量编译时间小于2s
- 由于增量编译必须在服务端进行，因此需要消耗时间处理并传输编译后的数据
- webpack写文件稳定阈值，约1-2s
- 在最优情况下，每个变更也需要 2-4s 的时间
- 开发者无法接受!

解决方案 - 定制编译过程

- 所有编译过程，必须在浏览器中完成！
- 这是一个终极挑战，达成这个目标意味着只能放弃Angular官方脚手架
- 另起炉灶，一切从零开始：
  - SVD → Typescript
  - Typescript → Javascript
  - Javascript → Angular组件
  - Angular组件 → DOM

- SVD → Typescript
- Typescript → Javascript
- Javascript → Angular组件
- Angular组件 → DOM

解决方案 - SVD → Typescript

- 这是Awade自研编译器
- 基于Typescript实现，并且系统设计之初就做足了在浏览器中运行的准备
- 所以这个编译过程的移植是非常简单的

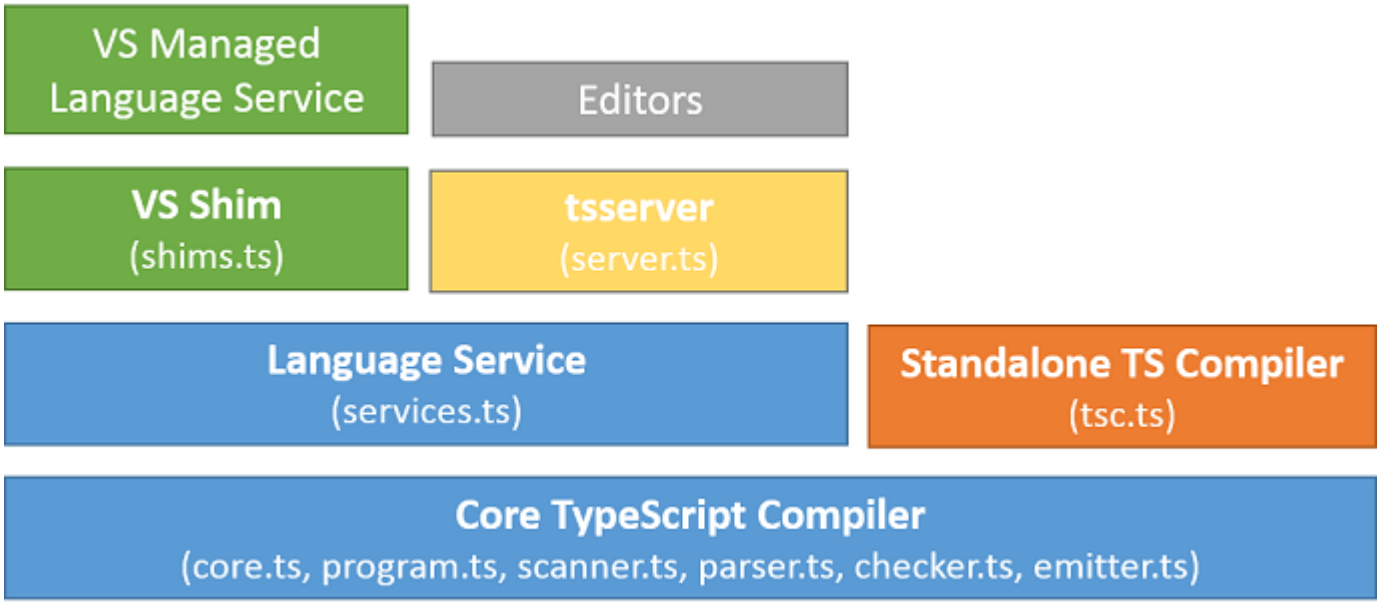
- SVD → Typescript
- Typescript → Javascript
- Javascript → Angular组件
- Angular组件 → DOM

解决方案 - Typescript → Javascript

- 强大、易用的Typescript编译器API
- 学习资源
  - Wiki: Using-the-Compiler-API
  - typescript.d.ts

解决方案 - Typescript编译器必要的概念

- Typescript编译器层级关系



引用自 https://github.com/Microsoft/TypeScript/wiki/Architectural-Overview

解决方案 - Typescript编译器必要的概念

- transpileMod
  - 最简易版
  - 可接受tra
  - 不会报错

- transpile()
  - 已废弃，

```
var ts = require("typescript");
var content = 'import {f} from "foo"\n' + "export var x = f()";

var compilerOptions = { module: ts.ModuleKind.System };

var res1 = ts.transpileModule(content, {
  compilerOptions: compilerOptions,
  moduleName: "myModule2"
});
console.log(res1.outputText);

console.log("============");

var res2 = ts.transpile(
  content,
  compilerOptions,
  /*fileName*/ undefined,
  /*diagnostics*/ undefined,
  /*moduleName*/ "myModule1"
);
console.log(res2);
```

分析

解决方案 - Typescript编译器必要的概念

- Typescript Language Services
  - 用于处理复杂的编译场景，比如类似Awade这样需要深度定制的场景
  - 主要提供了快速编译响应、解耦各个编译环节的能力
  - 使用 Language Service Host 对象来描述被编译对象（Program）
  - tsserver：IDE们能够做到编码过程中的准实时输入提示和错误提示的秘密武器

解决方案 - Typescript编译器必要的概念

一个基于磁盘IO的经典 Language Service Host

```
const servicesHost: ts.LanguageServiceHost = {
    getScriptFileNames: () => scriptFileNames,
    getScriptVersion: fileName => String(scriptVersions.get(fileName)),
    getScriptSnapshot: getScriptSnapshot,
    getCurrentDirectory: () => awadeRoot,
    getCompilationSettings: () => ({
        module: ts.ModuleKind.CommonJS,
        target: ts.ScriptTarget.ES5,
        experimentalDecorators: true
    }),
    getDefaultLibFileName: options => ts.getDefaultLibFilePath(options),
    fileExists: ts.sys.fileExists,
    readFile: ts.sys.readFile,
    readDirectory: ts.sys.readDirectory
};
```

解决方案 - Typescript编译器必要的概念

- 必须创建一个基于内存的 Language Service Host
- 实现一个虚拟文件系统，在内存中模拟磁盘IO

```
public fileExists(filename: string, suppressLog: boolean = false): boolean { lse): void {
    /*const ret = filename in this.files;
    if (!suppressLog && filename.indexOf("uid-sdk") != -1) {
        console.info("vfs =====> ", `fileExists("${filename}") => ${ret}`);
    }*/
    return filename in this.files;
}

public deleteFile(filename: string): void {
    if (this.fileExists(filename, suppressLog: true)) {
        delete this.files[filename];
    }
}

public readFile(filename: string): string {
    if (!this.fileExists(filename, suppressLog: true)) {
        throw new Error( message: `The file ${filename} doesn't exist`);
    }
    return this.files[filename].content;
}
```

解决方案 - 定制 Language Service Host

```
const servicesHost: ts.LanguageServiceHost = {
    getScriptFileNames: () => scriptFileNames,
    getScriptVersion: fileName => String(scriptVersions.get(fileName)),
    getScriptSnapshot: getScriptSnapshot,
    getCurrentDirectory: () => awadeRoot,
    getCompilationSettings: () => ({
        module: ts.ModuleKind.CommonJS,
        target: ts.ScriptTarget.ES5,
        experimentalDecorators: true
    }),
    getDefaultLibFileName: options => ts.getDefaultLibFilePath(options),
    fileExists: virtualFileSystem.fileExists,
    readFile: virtualFileSystem.readFile,
    readDirectory: virtualFileSystem.readDirectory
};
```

一个基于内存的定制版 Language Service Host

解决方案 - 实例化 Typescript 核心编译器

- Typescript核心编译器：node_modules/typescript/lib/typescript.js
- 这是一个零依赖，纯js实现的核心编译实现
- 通常是用在node运行时（tsc/tsserver为入口）
- 需要一些操作才能搬到浏览器里

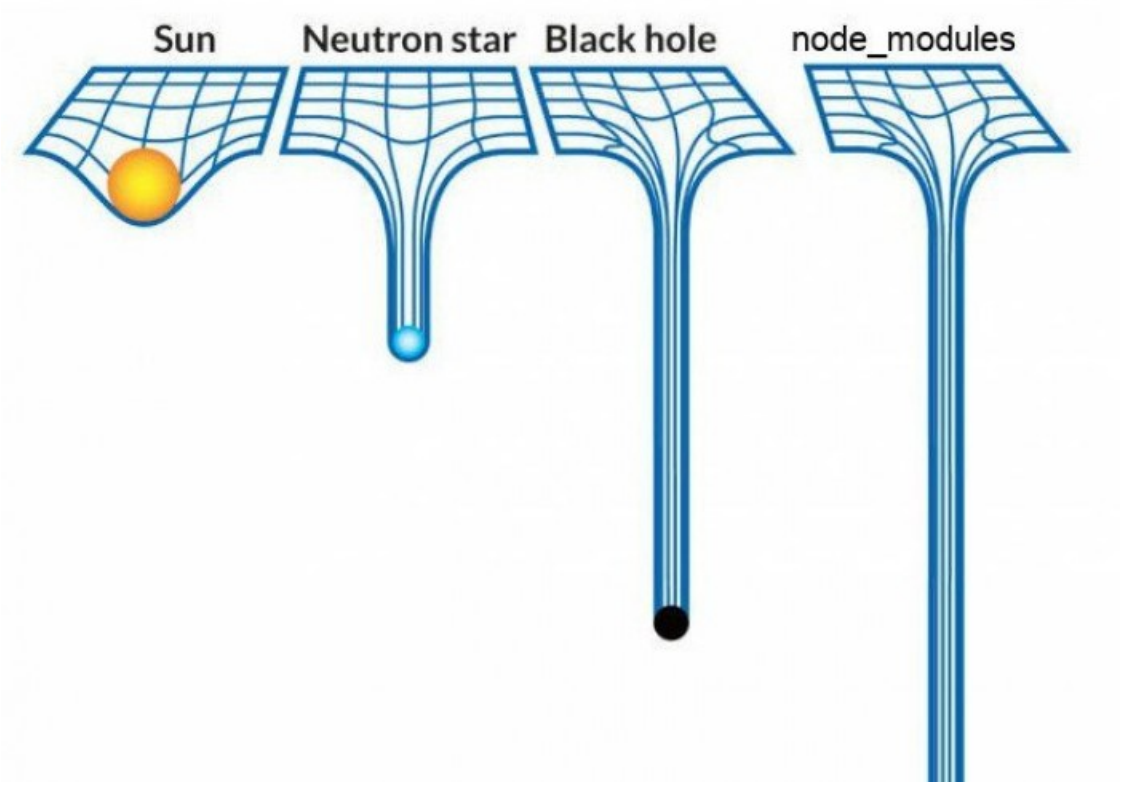
```
private _initTypescriptCompiler(source: string): boolean {
    try {
        this.typescript = eval( x: `(function() { ${source}; return ts; })();`);
    } catch (e) {
        console.error('unable to init typescript compiler! detail:', e);
        return false;
    }
    return true;
}
```

也可以直接import到源码中

```
import * as ts from "typescript";
```

解决方案 - 管理依赖

- 经典的编译是通过磁盘来IO源码的
- 依赖项也直接通过磁盘IO来完成



解决方案 - 管理依赖

- 内存虚拟文件系统中，只能通过HTTP来获取依赖项，不能无视IO的时延
- 编译器是在编译过程中才逐一索引出所有的依赖项
- 而网络IO依赖项的过程只能是异步的，采用编译器默认方式处理依赖，会让编译过程变的极度复杂
- 同时如果处理不好，网络时延会让编译效率会大打折扣
- 这里也需要定制

解决方案 - 管理依赖

- 采用
- 提供
- 这样
- 并且
- 缺

为啥不用SystemJS?

Typescript编译时，只需要依赖项的定义文件，而不需要依赖项的实现。

依赖项的实现在Awade主工程编译时，已经打包好了，并且浏览器打开Awade时，这些实现就已经下载到浏览器内存中了，因此，这里不需要使用SystemJS来管理依赖项。
一方面是节约浏览器内存，另一方面是减少首次编译时间。

- SVD → Typescript
- Typescript → Javascript
- Javascript → Angular组件
- Angular组件 → DOM

解决方案 - Javascript → Angular组件

```
import { Compiler } from “@angular/core”;
private compiler: Compiler;
this.compiler.compileModuleSync(
    this.createComponentModule(tempComponents));
```

- SVD → Typescript
- Typescript → Javascript
- Javascript → Angular组件
- Angular组件 → DOM

解决方案 - Angular组件 → DOM

- 这里采用Angular的解决方案即可，无需定制

```
<ng-container *ngComponentOutlet="
                dynamicComponent; ngModuleFactory: dynamicModule;">
</ng-container>
```

- 也可以使用 ViewContainerRef.createComponent()，两者差不多

效果与建议

效果与建议

- 效果
  - 采用脚手架的增量编译，每次变更大约需要 2-4s 的编译时间
  - 采用定制化编译过程，每次变更需要100-300ms的编译时间
  - 基本上达到准实时的目标

- 建议
  - 需要吃透编译器，深入到编译器内部，才能玩的转
  - 本文是一个不错的入坑引子
  - 强烈鼓励在编译时间敏感的Typescript应用中，果敢定制适合自己的编译流程

遗留问题 及
后续计划

遗留问题

- Angular组件 → DOM

```
import { Compiler } from “@angular/core”;
private compiler: Compiler;
this.compiler.compileModuleSync(
    this.createComponentModule(tempComponents));
```

- 这一步将消耗80%以上的编译时间
- 并且其时间复杂度是O(n)

后续计划

- Angular组件在内存中的增量编译，暂不对其进行定制了
- 主要原因是：Angular 8 会引入一个新的编译器 Ivy，我们与Angular团队联系得知，Ivy编译器可以支持在 JiT 上下文中实现增量编译
- 次要原因是：实在太复杂，而且零文档，只能啃源码

- 所以，遗留问题的后续计划是：等。。。

我个人微信 欢迎交流

备注NodeParty

https://github.com/rdkmaster/jigsaw

帮忙点个星！

问与答

谢谢！

5G 先锋