- typescript
  - 基础
    - 内置对象
      - JavaScript 中有很多内置对象，它们可以直接在 TypeScript 中当做定义好了的类型。
        - 定义在TS核心文件中https://github.com/Microsoft/TypeScript/tree/master/src/lib
        - ECMA内置对象：Boolean , Error Date RegExp
        - DOM和BOM内置对象 Document/ HTMLElement,Event,NodeList等
          - let body: HTMLElement = document.body;
let allDiv: NodeList = document.querySelectorAll('div');
document.addEventListener('click', function(e: MouseEvent) {
  // Do something
});
        - node不是内置对象一部分：用TS写node需要 npm install –D @types/node
    - 原始数据类型
      - 数字：let decLiteral: number=6
      - 字符串: let name: string=‘wm’
      - 布尔值：let isDone:boolean=false
      - 空值：let unusable: void = undefined;
      - Null/undefined: let n: null = null; let num: number = undefined;
      - never:永不存在的值的类型
    - 任意值any
      - let anyThing: any = 'hello';
      - 未声明类型的变量会被识别为任意值
    - 类型推导：如果没有明确的指定类型，那么 TypeScript 会依照类型推论（Type Inference）的规则推断出一个类型。
      - let myFavoriteNumber = 'seven';
myFavoriteNumber = 7;//报错
    - 联合类型：（Union Types）表示取值可以为多种类型中的一种。
      - let myFavoriteNumber: string | number;
let s:(number,string)[] = [1,'2']
      - 联合类型的变量在被赋值的时候，会根据类型推论的规则推断出一个类型
        - let myFavoriteNumber: string | number;
myFavoriteNumber = 'seven';
console.log(myFavoriteNumber.length); // 5
myFavoriteNumber = 7;
console.log(myFavoriteNumber.length); // 编译时报错
        - 假如你更清楚他 的类型可以用！后缀let s :string[] | null;s!.length
    - 对象的类型–接口（interface）
      - 首字符大写，有时要求以i为前缀
      - 概念：ts中我们用接口定义对象的类型，接口是对行为或状态的抽象，具体则是由类classes去实现implement
      - 可选属性
        - interface Person {
  name: string;
  age?: number;
}
      - 任意属性：
        - 注意：一旦定义了任意属性，那么确定属性和可选属性的类型都必须是它的类型的子集
        - interface Person {
  name: string;
  [propName: string]: any;
}
      - 只读属性：我们希望一些属性只能在创建的时候被赋值
        - 注意：只读的约束存在于第一次给对象赋值的时候，而不是第一次给只读属性赋值的时候
        - interface Person {
  readonly id: number;
  name: string;
  age?: number;
  [propName: string]: any;
}; let tom: Person = {
  id: 89757,
  name: 'Tom',
  gender: 'male'
};

tom.id = 9527;//errror TS25540
    - 数组的类型
      - 「类型 + 方括号」表示法
        - let fibonacci: number[] = [1, 1, 2, 3, 5]; let s:any[];
      - 数组泛型
        - let fibonacci: Array<number> = [1, 1, 2, 3, 5];
      - 用接口表示泛型
        - interface NumberArray {
  [index: number]: number;
} 表示 index是number,值也是number
let fibonacci: NumberArray = [1, 1, 2, 3, 5];
      - 类数组 非数组
        - 常见的类数组都有自己的接口定义
          - function sum() {
  let args: IArguments = arguments;
}
    - 函数的类型
      - 定义函数两种方式
        - 函数声明
          - function sum(x:number,y:number):number {
  return x + y
}
        - 函数表达式
          - let sum = function (x: number, y: number): number {
  return x + y;
};
          - question:上面其实只是对等号右侧的匿名函数进行了定义，等号左侧的sum其实是通过类型推论而推断出来的
          - let mySum: (x: number, y: number) => number = function (x: number, y: number): number {
  return x + y;
};
          - TypeScript 中的 => 和 ES6 中的 =>不同
            - TS的类型定义中 =>用来表示函数的定义，左侧是函数的输入需要用括号括起来，右侧是函数的输出
            - ES6中 则是箭头函数
      - 用接口定义函数的样子
        - interface SearchFunc {
  (source: string, subString: string): boolean;
}

let mySearch: SearchFunc;
mySearch = function(source: string, subString: string) {
  return source.search(subString) !== -1;
}
      - 可选参数：
        - 注意：可选参数后面不可再加必须参数
        - function buildName(firstName: string, lastName?: string) :string {
  if (lastName) {
  return firstName + ' ' + lastName;
  } else {
  return firstName;
  }
}
      - 默认参数：
        - ，TypeScript 会将添加了默认值的参数识别为可选参数，此时不受（可选参数后面不可加必须参数）的限制
        - function buildName(firstName='wang', lastName: string)
      - 剩余参数：
        - function push(array: any[], ...items: any[])
        - ...rest是能是最后一个
      - 重载：
        - 重载允许一个函数接受不同数量或类型的参数时，作出不同的处理。
        - 注意，TypeScript 会优先从最前面的函数定义开始匹配，所以多个函数定义如果有包含关系，需要优先把精确的定义写在前面。
        - function reverse(x: number): number;
function reverse(x: string): string;
function reverse(x: number | string): number | string {
  if (typeof x === 'number') {
  return Number(x.toString().split('').reverse().join(''));
  } else if (typeof x === 'string') {
  return x.split('').reverse().join('');
  }
}
        - 我们重复定义了多次函数 reverse，前几次都是函数定义，最后一次是函数实现。在编辑器的代码提示中，可以正确的看到前两个提示。
    - 类型断言 （Type Assertion）
      - 可以用来手动指定一个值的类型，类型断言不是类型转换，断言成一个联合类型中不存在的类型是不允许的
      - 语法
        - <类型>值
        - 值 as 类型 在 tsx 语法（React 的 jsx 语法的 ts 版）中必须用后一种。
      - demo: 将一个联合类型的变量指定为一个更加具体的类型
  - 声明文件
    - 书写声明文件
      - 书写声明文件的方式
        - 全局变量：通过<script>标签引入的第三方库 注入全局变量
          - declare var 声明全局变量
          - declare function 声明全局方法
          - declare class声明全局类
          - declare enum声明全局枚举
          - declare namespace声明(含有紫属性的)全局对象
          - interface 和type 声明全局类型
          - export导出变量
          - export namespace导出含有子属性的对象
          - export default ES6默认导出
          - export = commonjs导出
          - export as namespace UMD库声明全局变量
          - declare global 扩展全局变量
          - declare module 扩展模块
          - ///<reference> 三斜线指令
          - declare let/var/const jQuery: (selector: string) => any;
          - declare function jQuery(selector:string): any;
          - declare class Animal {
  name: string;
  constructor(name: string);
  sayHi(): string;
} 只能定义类型，不能用来定义具体实现
          - declare enum Directions {Up,Down}
          - declare namespace jQuery {
  function ajax(url: string, settings?: any): void;
}
          - jQuery 是一个全局变量，它是一个对象，提供了一个 jQuery.ajax 方法可以调用，那么我们就应该使用 declare namespace jQuery 来声明这个拥有多个子属性的全局变量
          - declare namespace 是ts早期为了解决模块化而创造的关键字， 目前用的很少，但是会用在声明文件
          - interface /type
        - npm包
          - 假如我们需要jquery,但是ts中编译器是不知道$和jQuery的，
            - declare var jQuery: (selector:string) => any
          - 声明语句：
          - 通常我们会把声明的语句放到一个单独文件 jQuery.d.ts中：*.d.ts
          - 声明文件：
          - 第三方声明文件：
            - 搜索：https://microsoft.github.io/TypeSearch/
            - 有些声明文件社区已经帮我们定义好，安装对应的生命模块即可
            - npm install @types/jquery –D
          - 当一个第三方库没有提供声明文件时，我们就需要自己书写声明文件了
            - 编写声明文件
              - 1.声明文件与npm包绑定在一起：依据：package.json中又types字段/或者有一个index.d.ts文件，这种不需要额外操作
              - 在我们尝试给一个npm包创建声明文件之前，先检测是否已存在声明文件
                - npm install –D @types/xxx
              - 1.创建node_modules/@types/foo/index.d.ts文件，不需要其他配置，但是 代码不会被保存
              - 2.新建types/index.d.ts。需要配置tsconfig.json的paths和baseUrl
                - {
  "compilerOptions": {
  "module": "commonjs",
  "baseUrl": "./",
  "paths": {
  "*": ["types/*"]
  }
  }
}
              - 方式：和全局变量的生命文件有很大区别 使用export
                - export 导出变量
                - export namespace 导出（含有子属性的）对象
                - export default ES6 默认导出（只有 function、class 和 interface 可以直接默认导出，其他的变量需要先定义出来，再默认导出）
                - export = commonjs 导出模块（对于commonjs规范的库，需要这种）
            - UMD库
              - 既可以通过 <script> 标签引入，又可以通过 import 导入的库，称为 UMD 库。相比于 npm 包的类型声明文件
              - 相对于npm.我们需要额外声明一个全局变量: export as namespace。
            - 在npm包或者UMD库中扩展全局变量
              - declare global
              - // types/foo/index.d.ts

declare global {
  interface String {
  prependHello(): string;
  }
}

export {}; // src/index.ts
'bar'.prependHello();此声明文件不需要导出任务东西，依然需要导出个空对象，告诉编译器这是个模块的声明文件，而不是全局变量的声明文件
            - 模块插件
              - declare module
            - 声明文件中的依赖
              - 1. 通过import引入另一个声明文件中的类型
              - 2.三斜线指令
                - 1.书写一个全局变量的声明文件 /// <reference types="jquery" />
declare function foo(options: JQuery.AjaxSettings): string;
                - 2.依赖一个全局变量的声明文件/// <reference types="node" />
export function foo(p: NodeJS.Process): string;
              - 自动生成声明文件
                - {
  "compilerOptions": {
  "module": "commonjs",
  "outDir": "lib",
  "declaration": true,
  }
}
        - 发布声明文件：
          - 1.和源码放一块
            - 1.package.json中types或者typings字段指定个声明文件地址
            - 2.项目根目录下编写index.d.ts
            - 针对入口文件(package.json中的main)编写一个同名不同后缀的.d.ts文件
          - 2.发布到@types下
            - 需要给 DefinitelyTyped 创建一个 pull-request，包含了类型声明文件，测试代码，以及 tsconfig.json 等。