node.js 与 TypeScript

目标

1. 熟练node.js环境配置
2. 掌握常用的npm操作方法与技巧
3. 熟练TypeScript变量声明语法

大纲

# 全新的前端开发体系

从最初的“网页设计制作”进化到“前端开发”，再到如今“JavaScript天下一统”的“大前端”，不论是技术进步还是开发方式都在日新月异，飞速变化，应用项目体积也越来越庞大，而促使前端开发不断适应，而越来越趋于“模块化、工程化”模式开发。通俗的讲，就是前端已经不仅仅是做几个网页了，而是越来越接近“软件开发”。

前端开发体系主要包括息息相关的三个组成部分：

* nodejs
* npm
* github

## node.js

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。

Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效。

自此，JavaScript代码不仅可运行于浏览器的HTML中，并且可以借助node.js而脱离浏览器直接运行了。

## npm

https://www.npmjs.com/

Node.js 的包管理器 npm（node package manager），是全球最大的开源库生态系统。

其实现快速的开源代码或模块在node.js的安装、使用、管理。

npm负责代码的版本发布和使用。

## github

https://github.com/

GitHub 是一个面向开源及私有软件项目的托管平台（也称代码仓库），无数的开源项目可以让我们利用其更好的完成应用的开发。

可以将自己的项目在该平台进行托管（用于存储、备份、开源等），包括协作开发、代码管理等。

# 快速搭建node.js开发环境

## node.js版本管理工具nvm安装

nvm，即node.js version manager。

由于node.js版本变动过于频繁，也为了在必要的时候能快速切换版本，而不是使用删除再重新安装的笨拙办法，因此需要用到该工具。

nvm-setup在安装时，会检测已安装版本，询问是否对其纳入管理。

### nvm常用命令：

Usage:

nvm 查看帮助信息

nvm install [version] [arch] 下载指定版本x86或x64的node.js

nvm uninstall [version] [arch] 卸载指定版本x86或x64的node.js

nvm use [version] [arch] 修改node.js环境变量以必用指定版本

Example:

nvm install v0.12.0 x86 安装指定32位的v0.12.0版本node.js

nvm use v0.12.0 x86 启用32位的v0.12.0版本node.js

## 通过nvm安装node.js

安装命令如下：

nvm install v6.10.2 x64

nvm use v6.10.2 x64

检测安装成功与否：

node –v

npm -v

## npm

npm可以方便的管理一个应用项目中所依赖的多个独立模块，并且通过一个“package.json”文件来对所有的模块信息进行记录，在构建新的开发环境时，只需要借助这个“package.json”文件就可以快速的安装所有项目所依赖的模块和代码。

### 常用命令：

npm init 初始化创建package.json

npm install 按当前目录下的package.json配置安装所有模块

npm install –-global module 安装指定模块到全局环境

npm install module 安装指定模块到当前目录

npm install –-save module 安装模块并记录到dependencies列表中

npm install –-save-dev module 安装模块并记录到devDependencies测试模块列表中

npm uninstall module 卸载指定模块

npm update module 更新已经安装的模块

### 加速npm：

npm默认从国外的源获取和下载包信息，难免会存在被“墙”而安装失败的情况。

而解决这个问题办法可以使用“淘宝NPM镜像”，其完整的同步国外服务器的资源，具体使用方法请参考网址：npm.taobao.org

npm install -g cnpm --registry=https://registry.npm.taobao.org

而后，我们就可以利用cnpm来替代npm快速安装资源了。

## git工具安装

使用Git-2.12.0-64-bit来进行代码管理，该工具有窗口和CLI（即CMD命令行）两种使用方式。

### 常用git命令：

git clone <repository> [<directory>]

从远程github代码库中完整克隆（复制）一份到本地当前或指定目录中。

基本概念：

clone 从github复制已有库到本地

push 将本地库的修改和推送到github相对应的库中

fork 在github上把看中的别人的库拖到自己的帐号下，变成自己的库（成为一条分支，如此便可以对其进行自由的clone和push，就像自己的的库）

pull request 将fork别人的库，建立自己的分支修改后，想把这些修改共享提交给原作者，即可以请求推送，作者可以选择接受合并你的修改或者拒绝。

### 加速github访问：

使用VPN来加速国外资源的访问，推荐“greenvpn”，手机搜索“竹子VPN”。

每天免费使用1小时，对于安装资源已经足够了。

——————下午内容——————

# TypeScript概述

## 学习TypeScript的理由

1. TypeScript是基于ECMAScript6标准的，运行于Node.js环境的开源高级语言。
2. JavaScript 作为脚本语言，未提供类和模块的等高级语言概念，很难用于开发大型 Web 应用，而 TypeScript 扩展了 JavaScript 实现了这些特性。
3. 且TypeScript完全面向对象，类、接口在编译时会蒸发的一干二净从而生成纯净的Javascript脚本。
4. 最后，即使将来浏览器JavaScript升级到ECMAScript6标准，那么我们可以毫无成本的直接在TS和JS之中选择，因为我们早已经玩转了符合ES6标准特性的TypeScript。

### TypeScript 主要特点包括

TypeScript 是微软推出的开源语言，使用 Apache 授权协议

TypeScript 增加了可选数据类型、类和模块

TypeScript 可编译成可读的、标准的 JavaScript

TypeScript文件无法直接在浏览器环境运行，需要编译成JavaScript再使用

TypeScript 支持开发大规模 JavaScript 应用，并保证编译后的 JavaScript 代码兼容性

TypeScript 扩展了 JavaScript 的语法，已有JavaScript 代码可与 TypeScript 一起运行无需更改

TypeScript 文件扩展名是ts，而 TypeScript 编译器会编译成 js 文件

TypeScript 语法与 JScript .NET 相同

TypeScript 易学易于理解

# 配置TypeScript开发环境

## 安装TypeScript

直接通过npm或cnpm安装TypeScript到全局环境

cnpm install –g typescript

## 验证安装

通过TypeScript编译器来验证TypeScript安装是否正确

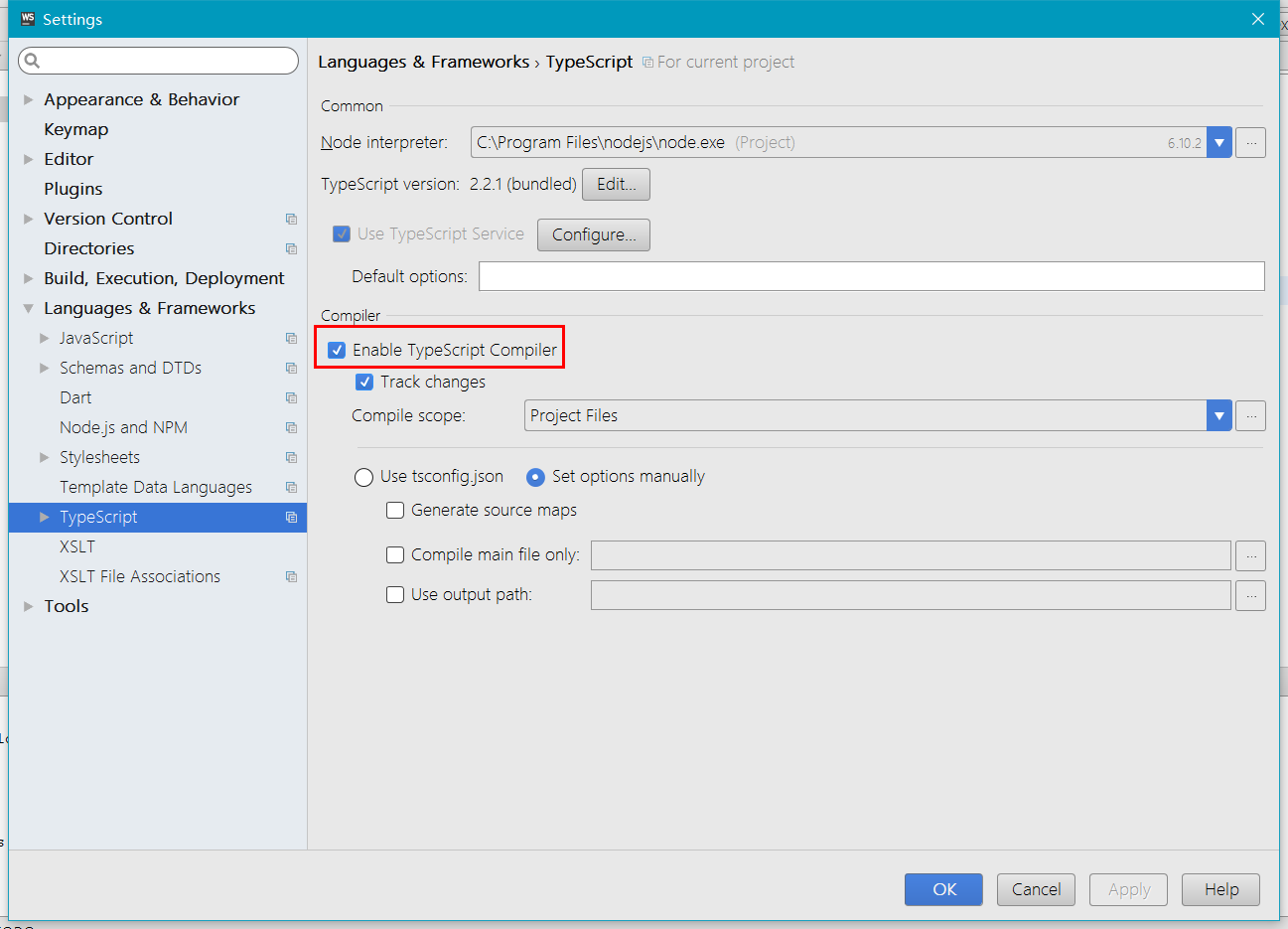
tsc -v

## 编译TypeScript

直接通过tsc编译器来编译TypeScript为JavaScript：

tsc example.ts

## 配置webstorm自动编译TypeScript



# TypeScript基础

## 变量声明

声明变量之let和var

var 延用于JavaScript。其声明的变量作用域不严谨，存在“变量提升”问题

let ES6标准，严格控制块级作用域，块以外无法访问。

## 常量声明

常量与变量不同之处在于，必须在定义时赋初始值，且在后续编码过程中再也不能更改其值。

TypeScript中使用const关键词定义常量数据。

# TypeScript数据类型声明

TypeScript支持静态数据类型声明，使用“标识符:类型”的方式进行声明。

## 常用类型

### 基本数据类型

**let *a***:**string** = "the content!";  
**let *b***:**number** = 100;  
**let *c***:**boolean** = **true**;

### 模板字符串

使用Tab键上面的``符号成对来包围需要包括的字符串。

可以是任何字符串内容，即使是HTML，不需要转义，并可以直接换行。

相对于JavaScript中以无数的+号来拼接字符串，并且需要进行相关的转义方便太多太多。

插入变量使用 ${变量名}来插入到合适的位置。

**let *nickname***:**string** = `Gene`;  
**let *age***:**number** = 37;  
**let *sentence***:**string** = `Hello, my name is ${ ***nickname*** }.  
I'll be ${ ***age*** + 1 } years old next month.`;

### 任意值类型

当我们编写应用时，我们可能会需要描述一些类型不明确的变量。

因为这些变量的值可能来源于一些动态的内容，如用户或第三方提供的库。

这种情况下，我们不希望类型检查器对这些值进行检查而是直接让它们通过编译阶段的检查。 那么我们可以使用 any类型来标记这些变量：

**let *notSure***:**any** = 4;  
***notSure*** = "maybe a string instead";  
***notSure*** = **false**; // okay, definitely a boolean

### 无值类型

某种程度上来说，void类型像是与any类型相反，它表示没有任何类型。

当一个函数没有返回值时，通常可以设置其返回值类型是 void：

**function** *warnUser*():**void**{  
 *alert*( "This is my warning message" );  
}

### 数组类型

指具有相同类型数据的集合。定义数组有两种方法：

**let *list***:**number**[] = [ 1, 2, 3 ];  
**let *list2***:**Array**<**number**> = [ 1, 2, 3 ];

### 元组类型（Tuple）

元组类型允许表示一个已知元素数量和类型的数组，各元素的类型不必相同。 比如，你可以定义一对值分别为 string和number类型的元组。

// 声明一个元组类型  
**let *x***:[ **string**, **number** ];  
// 初始化他  
***x*** = [ 'hello', 10 ]; // 准确  
// 错误的初始化  
***x*** = [ 10, 'hello' ]; // 错误

### 枚举类型（Enum）

枚举是一种给数字类型的值，设置易于辨别的名字的方法。

**enum** Color {*Red*, *Green*, *Blue*}  
**let *c***:Color = Color.*Green*;// c==1

在默认情况下，枚举类型会从数字0开始标记它的元素。

但我们也可以手动分别设置枚举元素的值：

**enum** Color {*Red* = 0xFF0000, *Green* = 0x00FF00, *Blue* = 0x0000FF}  
**let *c***:Color = Color.*Green*;

枚举类型有一个便捷特性，我们也可以直接用数值来查找其对应的枚举元素的名称。

举例来说，如果我们有一个值为2,但我们不确定这个数值对应枚举类型中的哪个元素，那我们可以直接查找这个数值对应的名称：

**enum** Color {*Red* = 1, *Green*, *Blue*}  
**let *colorName***:**string** = Color[ 0xFF0000 ];  
*alert*( ***colorName*** ); // Red

注意：枚举类型必须定义在全局环境下。

## 类型断言

通常这会发生在你清楚地知道一个实体具有比它现有类型更确切的类型。

通过类型断言这种方式可以告诉编译器，“相信我，我知道自己在干什么”。 类型断言好比其它语言里的类型转换，但是不进行特殊的数据检查和解构。 它没有运行时的影响，只是在编译阶段起作用。 TypeScript会假设你，程序员，已经进行了必须的检查。

// 类型断言有两种形式。 其一是“尖括号”语法：  
**let *someValue***:**any** = "this is a string";  
**let *strLength***:**number** = (<**string**>***someValue***).**length**;  
  
// 另一个为as语法：  
**let *someValue***:**any** = "this is a string";  
**let *strLength***:**number** = (***someValue* as string**).**length**;

然而，当你在TypeScript里使用React的JSX时，只有 as语法断言是被允许的。

# 快速上手

## 首先，创建一个hello.ts文件

用JavaScript语法来实现以下代码，并编译与查看结果，与ts代码进行对比

**function** *saywods*( person)  
{  
 **return** "Hello, " + person;  
}  
**var *user*** = "teacher";  
***document***.write( *saywods*( ***user*** ) );

在命令行中运行TypeScript编译器：

tsc hello.ts

编译完成，会发现编译结果基本上和源码没区别。

## 编译代码

为其增加类型声明：

**function** *saywods*( person:**string** )  
{  
 **return** "Hello, " + person;  
}  
**var *user***:**number** = 100;  
***document***.write( *saywods*( ***user*** ) );

再重新编译，会发现错误信息提示，因为传入函数的数据类型不匹配：

a.ts(15,26): error TS2345: Argument of type 'number' is not assignable to parameter of type 'string'.

# 练习题

* 熟练TypeScript基本语法和常用类型