TypeScript：从零开始的RPG——序章

钟MOS
杂食动物，不是前端开发工程师

+ 关注他

创世界：准备工作

上路之前，你要准备好一些东西，比如Node.js和npm。安装Node.js的传送门在这里。然后还需要一个世界转换器TypeScript：

```
npm install -g typescript
```

TypeScript现在是这个世界的规则。它跟JavaScript世界很像，嗯，甚至没什么差别，比如：

> ts-the-rpg.ts（v1）

```
function hello(hero) {
    console.log("Hello , " + hero);
}
var hero = "勇者";

hello(hero);
```

我们需要将ts-the-rpg.ts变成JavaScript世界的才能玩。

```
tsc ts-the-rpg.ts
```

这个时候我们发现生成了ts-the-rpg.js。我们打开看看，发现，怎么这两个世界一模一样?

没错，这两个世界几乎一样。但是接下来不一样的来了：

> ts-the-rpg.ts（v2）

```
function hello(hero: string) {
    console.log("Hello , " + hero);
}
var hero1 = "勇者";
var hero2 = 2;

hello(hero1);
hello(hero2);
```

你在转换世界的时候发现，怎么出了一个问题?

```
ts-the-rpg.ts(8,7): error TS2345: Argument of type 'number' is not assignable to parameter of type 'string'.
```

看起来有一个怪物混入了勇者的队伍之中啊。勇者都有自己的名字，你却是一个数字？你想蒙混过关？世界转换器就会告诉你，你很危险了勇者。

但是即使这样，世界转换器很公正，它还是把TypeScript世界的一切带到了JavaScript世界。因为JavaScript世界中，勇者、怪物，傻傻分不清楚。转换后的世界：

> ts-the-rpg.js

```
function hello(hero) {
    console.log("Hello , " + hero);
}
var hero1 = "勇者";
var hero2 = 2;
hello(hero1);
hello(hero2);
```

你是一个创世者，也是一个游戏玩家，世界转换器告诉了你这个世界有问题，但是你的世界你来决定。我建议，为了这个世界不至于在最后分崩离析，还是好好处理完问题再上路吧。

数据类型，是构建这个世界的基础。这个世界本身由这么几个基本数据类型组成：string、number、array、enum、any……等等。我们怎么判断它是不是这种类型的怎么做？只要把类型带在你的声明之中。比如function hello(hero: string)就表示，你必须是一个string类型的数据，才能通过这里。

新手村：欢迎你，勇者

“你好，勇者！”突然你看到了一阵光，看板娘站在光中，笑靥如画。

你想起来了，你是一个勇者。等等，什么是勇者？你作为一个创世者，深深的陷入了思考，最后得出一个结论：

ts-the-rpg.ts（v3）

```
var hero = {
    name: "勇者",
    hp: 10
}
function hello(hero) {
    console.log("Hello , " + hero.name);
}
hello(hero);
```

不对，这样做不是依然什么样的牛鬼蛇神都能以勇者的身份进入这个世界吗？嗯，作为一个先知，我告诉你怎么做：

ts-the-rpg.ts（v4）

```
// 定义什么是勇者
class Hero {
    name: string;   // 每个勇者都有一个名字
    hp: number;     // 每个勇者有自己的HP值
    // 召唤一个勇者的规则
    constructor(name: string, hp: number) {
        this.name = name;
        this.hp = hp;
    }
}

// 召唤一个勇者
var hero = new Hero("勇者", 10);

// 只能由勇者通过的路
function hello(hero: Hero) {
```

```
    console.log("Hello , " + hero.name);
}

hello(hero);
hello("我也是一个勇者啊！");
```

我们先来试试看转换这个世界，之后再来解释一下为什么这么做。开始转换：

```
ts-the-rpg.ts(21,7): error TS2345: Argument of type 'string' is not assignable to parameter of type 'Hero'.
```

最后一个假装是勇者的字符串想要蒙混过关，被转换器拦住了。等等，转换器怎么知道勇者长什么样？

没错，我们这时候祭出了勇者召唤的特殊形式：class。我们通过定义一个叫Hero的数据类型来告诉世界，这个世界开始有勇者了。那么以后，我们就可以在入口处判断你是不是一个Hero。

class由这么几个部分组成：它是什么（定义的数据类型名）、它由什么构成（类的成员数据）、它能做什么（定义数据类型的行为）、它需要哪些素材才能被召唤出来（构造函数）。

从ts-the-rpg.ts（v4）上看，我们定义了它是勇者（class Hero），它有名字（name: string;）和血量（hp: number;），召唤他的规则就是必须要给他起个名字并提供血量（constructor(name: string, hp: number){ this.name = name; this.hp = hp; }）,然后通过特殊仪式召唤它（var hero = new Hero("勇者", 10);）。

那你会问，如果有一个史莱姆，伪装的特别像一个勇者，就像下面这样，世界转换器会怎么做？

> ts-the-rpg.ts（v5）

```
// 定义什么是勇者
class Hero {
    name: string;    // 每个勇者都有一个名字
    hp: number;      // 每个勇者有自己的HP值
```

```
    // 召唤一个勇者的规则
    constructor(name: string, hp: number) {
        this.name = name;
        this.hp = hp;
    }
}

// 召唤一个勇者
var hero = new Hero("勇者", 10);

// 只能由勇者通过的路
function hello(hero: Hero) {
    console.log("Hello , " + hero.name);
}

hello(hero);
// 伪装成勇者的史莱姆
hello({ name: "我不是史莱姆", hp: 1 });
```

我们试着转换到普通世界后，怎么世界转换器什么都没做？于是你陷入了深深的恐惧。没错，这样召唤一个勇者肯定一瞬间就被危险的史莱姆识破并且伪装，这个时候我们需要把勇者不为人知的一面隐藏起来，比如勇者有hp这件事。

> ts-the-rpg.ts（v6）

```
// 定义什么是勇者
class Hero {
    name: string;           // 每个勇者都有一个名字
    private hp: number;     // 每个勇者有自己的HP值，但是受保护
    // 召唤一个勇者的规则
    constructor(name: string, hp: number) {
        this.name = name;
```

```
        this.hp = hp;
    }
}

// 召唤一个勇者
var hero = new Hero("勇者", 10);

// 只能由勇者通过的路
function hello(hero: Hero) {
    console.log("Hello , " + hero.name);
}

hello(hero);
hello({ name: "我不是史莱姆", hp: 1 });
```

这时候我们转换这个世界，你看史莱姆被拦在了外面：

```
ts-the-rpg.ts(21,7): error TS2345: Argument of type '{ name: string; hp: number; }' is not assignable to parameter of type 'Hero'.

Property 'hp' is private in type 'Hero' but not in type '{ name: string; hp: number; }'.
```

没错，你一个堂堂史莱姆，把hp值这种对于勇者如此重要的属性暴露在外面，这种作风肯定不是勇者所为，你出去。

这时候史莱姆又来搞事情，它想，如果这样，我也隐藏我的hp，除了定义不一样，其他的都一摸一样，那样我一定能进去。

> ts-the-rpg.ts（v7）

```
// 定义什么是勇者
class Hero {
    name: string;           // 每个勇者都有一个名字
```

```
    private hp: number;     // 每个勇者有自己的HP值，但是受保护
    // 召唤一个勇者的规则
    constructor(name: string, hp: number) {
        this.name = name;
        this.hp = hp;
    }
}

// 定义什么是史莱姆
class Slime {
    name: string;           // 每个史莱姆都有一个名字
    private hp: number;     // 每个史莱姆有自己的HP值，但是受保护
    // 召唤一个史莱姆的规则
    constructor(name: string, hp: number) {
        this.name = name;
        this.hp = hp;
    }
}

// 召唤一个勇者
var hero = new Hero("勇者", 10);
var slime = new Slime("勇者", 10);

// 只能由勇者通过的路
function hello(hero: Hero) {
    console.log("Hello , " + hero.name);
}

hello(hero);
hello(slime);
```

这时候世界转换器非常聪明：

```
ts-the-rpg.ts(33,7): error TS2345: Argument of type 'Slime' is not assignable to parameter of type 'Hero'.

Types have separate declarations of a private property 'hp'.
```

你作为一只史莱姆，身上流着史莱姆的血，你的血的味道，我一闻就能闻出来不一样。史莱姆直接被推了出去。

那你肯定这时候肯定觉得很奇怪ts-the-rpg.ts（v5）和ts-the-rpg.ts（v7）同样是将史莱姆伪装成了勇者，为什么v5成功了，v7却失败了？

世界转换器在这里是这么做处理的：

1. v5部分因为所有的部分都是公开的，那么他只会判断是否存在，这种原理别处称作“鸭子模型”，就是说“呱呱叫又会游泳的鸟那就肯定是鸭子”，在这里就是“有名字有hp的肯定是勇者”，所以就放行了，这是一种弱的类型检查机制，可以抵挡住大部分的伪装。
2. 在v7中，有部分是隐藏的，那么不只会判断隐藏的部分是不是存在的，还会判断它是否来自不同的定义。

好了，一切都安全了。你到这里应该明白了TypeScript世界的一部分，类型检查。这个特性能够在某种程度上保护你程序的安全，不至于让你在每个通道内设置关卡，判断进来的东西是勇者还是史莱姆，或者是伪装成勇者的史莱姆。在JavaScript的世界里，你需要处处小心，勇者即使进了城也要被处处浪费时间去盘问，而在TypeScript中，勇者只需要在城门口被盘问一边，确定你是勇者后，你在城里能得到所有你能得到的东西，而不用再一遍一遍的被盘问：“你是不是勇者？”

转职：你依然是个勇者

你站在新手村中心，不知所措的时候，边上有三个导师，分别在招揽着自己的学徒，分别是战士、魔法师和弓箭手：

> ts-the-rpg.ts（v8）

```
class Hero {
    name: string;
    private hp: number;
    // 勇者的召唤方式
```

```
    constructor(name: string, hp: number) {
        this.name = name;
        this.hp = hp;
    }
}

class Warrior extends Hero {
    weapon: string;
    // 战士的召唤方式
    constructor(name: string, hp: number , weapon: string) {
        // 你的名字和你的血液是勇者的名字和勇者的血液，这是你的内心
        super(name, hp);
        this.weapon = weapon;
    }
    swing() {
        console.log("swing");
    }
}

class Magician extends Hero {
    weapon: string;
    // 魔法师的召唤方式
    constructor(name: string, hp: number , weapon: string) {
        // 你的名字和你的血液是勇者的名字和勇者的血液，这是你的内心
        super(name, hp);
        this.weapon = weapon;
    }
    fireball() {
        console.log("fireball");
    }
}
```

```
class Archer extends Hero {
    weapon: string;
    // 弓箭手的召唤方式
    constructor(name: string, hp: number , weapon: string) {
        // 你的名字和你的血液是勇者的名字和勇者的血液，这是你的内心
        super(name, hp);
        this.weapon = weapon;
    }
    shoot() {
        console.log("shoot");
    }
}
```

没错，三种职业有着自己的攻击方式。勇者你要学什么呢?

> ts-the-rpg.ts（v9）

```
class Hero {
    name: string;
    private hp: number;
    // 勇者的召唤方式
    constructor(name: string, hp: number) {
        this.name = name;
        this.hp = hp;
    }
}

// 通过extends继承了勇者之力
class Warrior extends Hero {
    weapon: string;
    // 战士的召唤方式
```

```
    constructor(name: string, hp: number , weapon: string) {
        // 你的名字和你的血液是勇者的名字和勇者的血液，这是你的内心
        super(name, hp);
        this.weapon = weapon;
    }
    swing() {
        console.log("swing");
    }
}

// 通过extends继承了勇者之力
class Magician extends Hero {
    weapon: string;
    // 魔法师的召唤方式
    constructor(name: string, hp: number , weapon: string) {
        // 你的名字和你的血液是勇者的名字和勇者的血液，这是你的内心
        super(name, hp);
        this.weapon = weapon;
    }
    fireball() {
        console.log("fireball");
    }
}

// 通过extends继承了勇者之力
class Archer extends Hero {
    weapon: string;
    // 弓箭手的召唤方式
    constructor(name: string, hp: number , weapon: string) {
        // 你的名字和你的血液是勇者的名字和勇者的血液，这是你的内心
        super(name, hp);
        this.weapon = weapon;
```

```
    }
    shoot() {
        console.log("shoot");
    }
}

function forest(hero: Hero) {
    console.log("Enter Forest !!");
}

var hero1 = new Warrior("warrior", 10, "sword");
var hero2 = new Magician("magician", 10, "wand");
var hero3 = new Archer("archer", 10, "bow");

forest(hero1);
forest(hero2);
forest(hero3);
```

世界转换器转换后发现，竟然战士、法师和弓箭手都能进入森林！

你要知道，即使你选择了职业，但是你体内的勇者之名和勇者之血都通过extends Hero方式继承了下来。你的召唤方式变了，但是你的召唤规则里还通过super(name, hp);这个方式保留着你的内心，这个就像是当初召唤你的方式，new Hero(name , hp)。所以即使这时候你们用着不同的武器，有着不同的攻击方式，却依然内心都还是个勇者。

技能训练：技能虽好，可不要偷师哦！

这时候你犹豫了，你想去三个练功房都看看，再来考虑转职的事情：

> ts-the-rpg.ts（v10）

```
class Hero {
    name: string;
    private hp: number;
    // 勇者的召唤方式
    constructor(name: string, hp: number) {
        this.name = name;
        this.hp = hp;
    }
}

// 通过extends继承了勇者之力
class Warrior extends Hero {
    weapon: string;
    // 战士的召唤方式
    constructor(name: string, hp: number , weapon: string) {
        // 你的名字和你的血液是勇者的名字和勇者的血液，这是你的内心
        super(name, hp);
        this.weapon = weapon;
    }
    swing() {
        console.log("swing");
    }
}

// 通过extends继承了勇者之力
class Magician extends Hero {
    weapon: string;
    // 魔法师的召唤方式
    constructor(name: string, hp: number , weapon: string) {
        // 你的名字和你的血液是勇者的名字和勇者的血液，这是你的内心
        super(name, hp);
        this.weapon = weapon;
```

```
    }
    fireball() {
        console.log("fireball");
    }
}

// 通过extends继承了勇者之力
class Archer extends Hero {
    weapon: string;
    // 弓箭手的召唤方式
    constructor(name: string, hp: number , weapon: string) {
        // 你的名字和你的血液是勇者的名字和勇者的血液，这是你的内心
        super(name, hp);
        this.weapon = weapon;
    }
    shoot() {
        console.log("shoot");
    }
}

function trainWarrior(hero: Warrior) {}
function trainMagician(hero: Magician) {}
function trainArcher(hero: Archer) {}

var hero = new Hero("普通勇者", 10);

trainWarrior(hero);
trainMagician(hero);
trainArcher(hero);
```

毫不意外，你被三个练功房都踢了出来。

```
ts-the-rpg.ts(60,14): error TS2345: Argument of type 'Hero' is not assignable to parameter of type 'Warrior'.

Property 'weapon' is missing in type 'Hero'.

ts-the-rpg.ts(61,15): error TS2345: Argument of type 'Hero' is not assignable to parameter of type 'Magician'.

Property 'weapon' is missing in type 'Hero'.

ts-the-rpg.ts(62,13): error TS2345: Argument of type 'Hero' is not assignable to parameter of type 'Archer'.

Property 'weapon' is missing in type 'Hero'.
```

三个房间的导师都说，你没有武器，学不了技能。村长这个时候走过来告诉你真相：即使你有了武器，你也学不了。对，职人是继承了勇者的内心，职人永远都是勇者，而勇者没有转职，没有职人才有的能力，你永远只是个勇者，不是一个职人。这就是继承的真相。

> 职人是勇者，勇者不是职人。
>
> 职人是勇者，勇者不是职人。
>
> 职人是勇者，勇者不是职人。

你默默念了三遍。铭记在心。

各式各样的武器：选一件吧少年

实际上专职没有那么简单，你的武器不仅仅是个名字而已，这时候你的老师让你去挑一把武器带过来。你去了武器铺。武器铺的铁匠大叔一看到你是一个勇者，非常热心的跟你介绍了不同的武器。

> ts-the-rpg.ts（v11）

```
class Weapon {
    name: string;
    private atk: number;
    constructor(name: string, atk: number) {
        this.name = name;
        this.atk = atk;
    }
}

class Sword extends Weapon {
    constructor(name: string, atk: number) {
        super(name, atk);
    }
    swing() {
        console.log("swing");
    }
}

class Wand extends Weapon {
    constructor(name: string, atk: number) {
        super(name, atk);
    }
    fireball() {
        console.log("fireball");
    }
}

class Bow extends Weapon {
    constructor(name: string, atk: number) {
        super(name, atk);
    }
    shoot() {
```

```
        console.log("shoot");
    }
}
```

你说你要转职成战士，你只看剑。他很热情，问问你是不是要附魔。有火焰效果和寒冰效果，然后可以给你打造一把独一无二的剑：

> ts-the-rpg.ts（v12）

```
class Weapon {
    name: string;
    private: atk;
    constructor(name: string, atk: number) {
        this.name = name;
        this.atk = atk;
    }
}

class Sword extends Weapon {
    constructor(name: string, atk: number) {
        super(name, atk);
    }
    swing() {
        console.log("swing");
    }
}

interface Fire {
    fire();
}

interface Ice {
```

```
    ice();
}
```

你想了想说，都要。铁匠一愣，笑了笑，你小子为难老夫！好，难不倒老夫，于是你的剑做好了：

> ts-the-rpg.ts（v12）

```
class Weapon {
    name: string;
    private: atk;
    constructor(name: string, atk: number) {
        this.name = name;
        this.atk = atk;
    }
}

class Sword extends Weapon {
    constructor(name: string, atk: number) {
        super(name, atk);
    }
    swing() {
        console.log("swing");
    }
}

interface Fire {
    fire();
}

interface Ice {
    ice();
```

```
}

class SwordOfIceAndFire extends Sword implements Ice, Fire {
    constructor(name: string, atk: number) {
        super(name, atk);
    }
    swing() {
        console.log("swing");
        this.ice();
        this.fire();
    }
    ice() {
        console.log("ice");
    }
    fire() {
        console.log("fire");
    }
}
```

没错，interface在面向对象中就好像是附魔属性，只要你愿意，可以通过implements无限的往一把剑上叠加能力。但是，即使你无限叠加了能力，它还是一把剑，而不能成为魔杖或者弓箭。

冒险才刚刚开始，而我们的序章到了尾声。我们再来回顾一下有哪些概念：

1. 类和类型检查：一个史莱姆伪装的再好，还是史莱姆。
2. 继承是什么：勇者转职之后还是勇者，不管他变成了剑士、魔法师还是弓箭手。
3. 子类和父类的继承关系：一个职人是勇者，但是一个勇者不是职人。
4. 接口是什么：附魔属性。
5. 通过接口扩展类：只要你愿意，可以无限的往一把剑上叠加能力，让它成为一把新的剑。但是，即使你无限叠加了能力，它还是一把剑，而不能成为魔杖或者弓箭。

编辑于 2019-12-05

TypeScript 面向对象编程 前端开发入门