Mixins

介绍

除了传统的面向对象继承方式，还流行一种通过可重用组件创建类的方式，就是联合另一个简单类的代码。 你可能在Scala等语言里对mixins及其特性已经很熟悉了，但它在JavaScript中也是很流行的。

混入示例

下面的代码演示了如何在TypeScript里使用混入。 后面我们还会解释这段代码是怎么工作的。

```
// Disposable Mixin
class Disposable {
    isDisposed: boolean;
    dispose() {
        this.isDisposed = true;
    }

}

// Activatable Mixin
class Activatable {
    isActive: boolean;
    activate() {
        this.isActive = true;
    }
    deactivate() {
        this.isActive = false;
    }
}

class SmartObject implements Disposable, Activatable {
    constructor() {
        setInterval(() => console.log(this.isActive + " : " + t
his.isDisposed), 500);
    }

    interact() {
        this.activate();
    }

    // Disposable
    isDisposed: boolean = false;
    dispose: () => void;
    // Activatable
    isActive: boolean = false;
    activate: () => void;
    deactivate: () => void;
}
applyMixins(SmartObject, [Disposable, Activatable]);

let smartObj = new SmartObject();
setTimeout(() => smartObj.interact(), 1000);

////////////////////////////////////////
// In your runtime library somewhere
////////////////////////////////////////

function applyMixins(derivedCtor: any, baseCtors: any[]) {
    baseCtors.forEach(baseCtor => {
        Object.getOwnPropertyNames(baseCtor.prototype).forEach(
name => {
            derivedCtor.prototype[name] = baseCtor.prototype[na
me];
        });
    });
}
```

理解这个例子

代码里首先定义了两个类，它们将做为mixins。 可以看到每个类都只定义了一个特定的行为或功能。 稍后我们使用它们来创建一个新类，同时具有这两种功能。

```
// Disposable Mixin
class Disposable {
    isDisposed: boolean;
    dispose() {
        this.isDisposed = true;
    }

}

// Activatable Mixin
class Activatable {
    isActive: boolean;
    activate() {
        this.isActive = true;
    }
    deactivate() {
        this.isActive = false;
    }
}
```

下面创建一个类，结合了这两个mixins。 下面来看一下具体是怎么操作的：

```
class SmartObject implements Disposable, Activatable {
```

首先应该注意到的是，没使用 `extends` 而是使用 `implements` 。 把类当成了接口，仅使用Disposable和Activatable的类型而非其实现。 这意味着我们需要在类里面实现接口。 但是这是我们在用mixin时想避免的。

我们可以这么做来达到目的，为将要mixin进来的属性方法创建出占位属性。 这告诉编译器这些成员在运行时是可用的。 这样就能使用mixin带来的便利，虽说需要提前定义一些占位属性。

```
// Disposable
isDisposed: boolean = false;
dispose: () => void;
// Activatable
isActive: boolean = false;
activate: () => void;
deactivate: () => void;
```

最后，把mixins混入定义的类，完成全部实现部分。

```
applyMixins(SmartObject, [Disposable, Activatable]);
```

最后，创建这个帮助函数，帮我们做混入操作。 它会遍历mixins上的所有属性，并复制到目标上去，把之前的占位属性替换成真正的实现代码。

```
function applyMixins(derivedCtor: any, baseCtors: any[]) {
    baseCtors.forEach(baseCtor => {
        Object.getOwnPropertyNames(baseCtor.prototype).forEach(
name => {
            derivedCtor.prototype[name] = baseCtor.prototype[na
me];
        })
    });
}
```