泛型

学习目标

- 理解泛型概念与使用场景
- 在函数、类、接口中使用泛型

为什么要使用泛型

许多时候，标注的具体类型并不能确定，比如一个函数的参数类型

```
function getVal(obj, k) {
    return obj[k];
}
```

上面的函数，我们想实现的是获取一个对象指定的 k 所对应的值，那么实际使用的时候，obj 的类型是不确定的，自然 k 的取值范围也是不确定的，它需要我们在具体调用的时候才能确定，这个时候这种定义过程不确定类型的需求就可以通过泛型来解决

泛型的使用 - 函数

```
function getVal<T>(obj: T, k: keyof T) {
    return obj[k];
}
```

所谓的泛型，就是给可变（不定）的类型定义变量（参数），`<>` 类似 `()`

泛型类

在面向对象章节中，我们曾经给大家讲过一个基于泛型使用的例子：模拟组件

```
abstract class Component<T1, T2> {

    props: T1;
    state: T2;

    constructor(props: T1) {
        this.props = props;
    }

    abstract render(): string;
```

```
}



interface IMyComponentProps {
    val: number;
}
interface IMyComponentState {
    x: number;
}
class MyComponent extends Component<IMyComponentProps, IMyComponentState> {

    constructor(props: IMyComponentProps) {
        super(props);

        this.state = {
            x: 1
        }
    }

    render() {
        this.props.val;
        this.state.x;
        return '<myComponent />';
    }

}


let myComponent = new MyComponent({val: 1});
myComponent.render();
```

泛型接口

我们还可以在接口中使用泛型

后端提供了一些接口，用以返回一些数据，依据返回的数据格式定义如下接口：

```
interface IResponseData {
    code: number;
    message?: string;
    data: any;
}
```

根据接口，我们封装对应的一些方法

```
function getData(url: string) {
    return fetch(url).then(res => {
        return res.json();
    }).then( (data: IResponseData) => {
        return data;
    });
}
```

但是，我们会发现该接口的 `data` 项的具体格式不确定，不同的接口会返回的数据是不一样的，当我们想根据具体当前请求的接口返回具体 `data` 格式的时候，就比较麻烦了，因为 `getData` 并不清楚你调用的具体接口是什么，对应的数据又会是什么样的

这个时候我们可以对 `IResponseData` 使用泛型

```
interface IResponseData<T> {
    code: number;
    message?: string;
    data: T;
}
```

```
function getData<U>(url: string) {
    return fetch(url).then(res => {
        return res.json();
    }).then( (data: IResponseData<U>) => {
        return data;
    });
}
```

定义不同的数据接口

```
// 用户接口
interface IResponseUserData {
    id: number;
    username: string;
    email: string;
}
// 文章接口
interface IResponseArticleData {
    id: number;
    title: string;
    author: IResponseUserData;
}
```

调用具体代码

```
~(async function(){
    let user = await getData<IResponseUserData>('');
    if (user.code === 1) {
        console.log(user.message);
    } else {
        console.log(user.data.username);
    }

    let articles = await getData<IResponseArticleData>('');
    if (articles.code === 1) {
        console.log(articles.message);
    } else {
        console.log(articles.data.id);
        console.log(articles.data.author.username);
    }
});
```