Stack

一个简单的、实现了简单动画的栈的可视化界面

实现

```
let num = 0;
```

声明一个全局变量，用于标识栈中数字

```
function createBlock(className: string, value: number,): HTMLDivElement {
    let div = document.createElement('div');
    div.classList.add(className)
    div.innerHTML = `
        <span>${value}</span>
    `
    return div;
}
```

创建`Html`元素，并为其添加类名

```
const StackHTML = document.querySelector<HTMLDivElement>('#stack')!
```

选择DOM树中用于显示栈的`div`

```
let stack: HTMLDivElement[] = [];
```

利用`Typescript`中的`Array`模拟栈的实现

```
const BtnStackPush = document.querySelector<HTMLButtonElement>('#stack-push')!
BtnStackPush.addEventListener('click', () => {

    stack.push(createBlock('stack-elem', num++)) // 创建新的栈元素
    // 遍历栈中元素
    stack.forEach(value => {
        let flag = true
        // 对其中元素和DOM中元素进行比对，若相同则不需要重新渲染，减小GPU压力，优化性能
        StackHTML.childNodes.forEach(v => {
            if (v === value) {
                flag = false
            }
        })

        if (flag) {
            // 入场动画
            value.style.animation = "0.5s ease slideInLeft-enter"
            StackHTML.appendChild(value)
        }
    })
})
```

获取用于压栈的按钮，并添加监听器

```
const BtnStackPoP = document.querySelector<HTMLButtonElement>('#stack-pop')!
BtnStackPoP.addEventListener('click', () => {
    let val = stack.pop()
    if (val) {
        // 离场动画
        val.style.animation = "0.5s ease slideInLeft-leave"
        // 动画展示完全后再删除元素
        setTimeout(() => {
            StackHTML.removeChild<HTMLDivElement>(val!)
        }, 500)
    } else {
        // 栈为空则提示
        alert('Stack is EMPTY!')
    }
})
```

获取用于出栈的按钮，并添加监听器

*css文件见源码