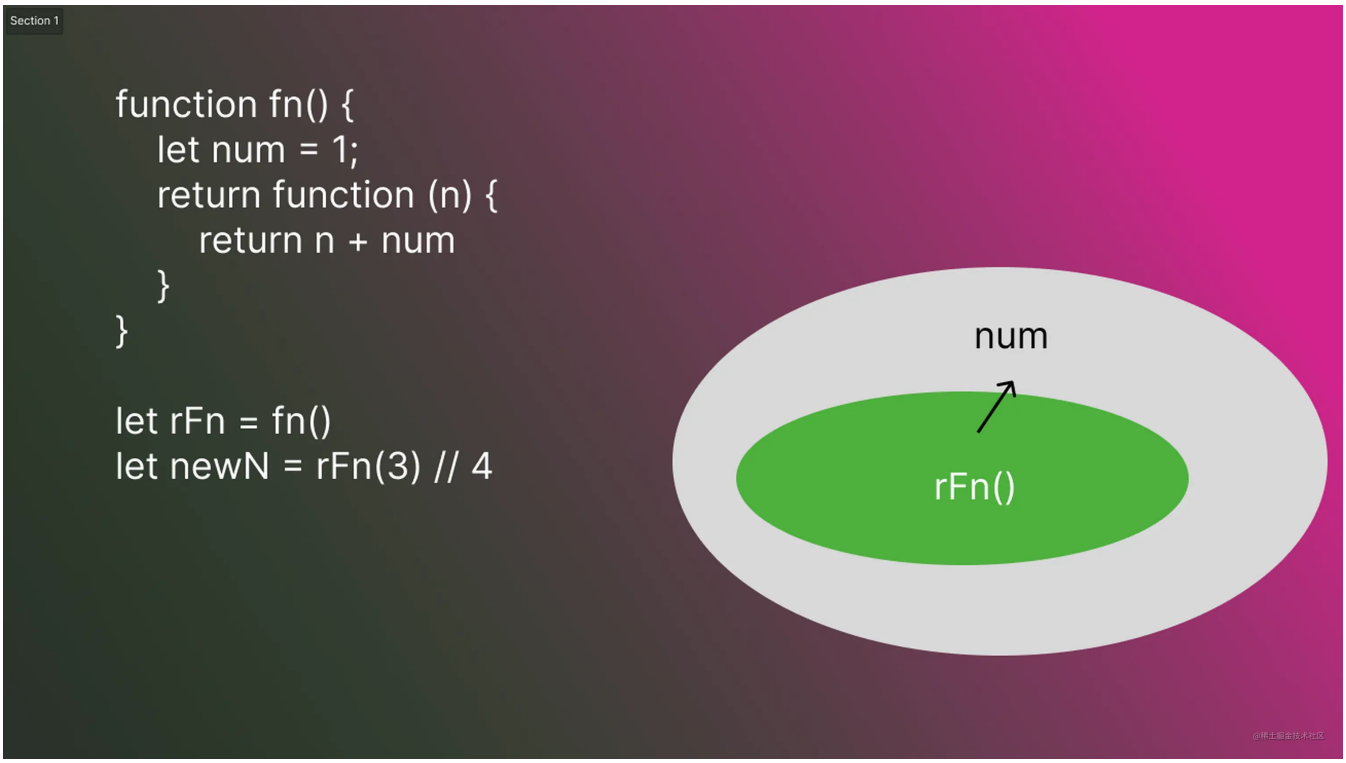
JavaScript 闭包



> 微信公众号搜索并关注： 进二开物 ， 更多技术周刊，React 技术栈、JavaScript/TypeScript/Rust 等等编程语言慢慢等你发现...

什么是闭包?

闭包的概念是有很多版本，不同的地方对闭包的说法不一

> 维基百科：在计算机科学中，闭包（英语：Closure），又称词法闭包（Lexical Closure）或函数闭包（function closures），是在支持头等函数的编程语言中实现词法绑定的一种技术。

> MDN: **闭包**（closure）是一个函数以及其捆绑的周边环境状态（**lexical environment**，**词法环境**）的引用的组合。

个人理解：

- 闭包是一个函数（返回一个函数）

- 返回的函数保存了对外变量引用

一个简单的示例

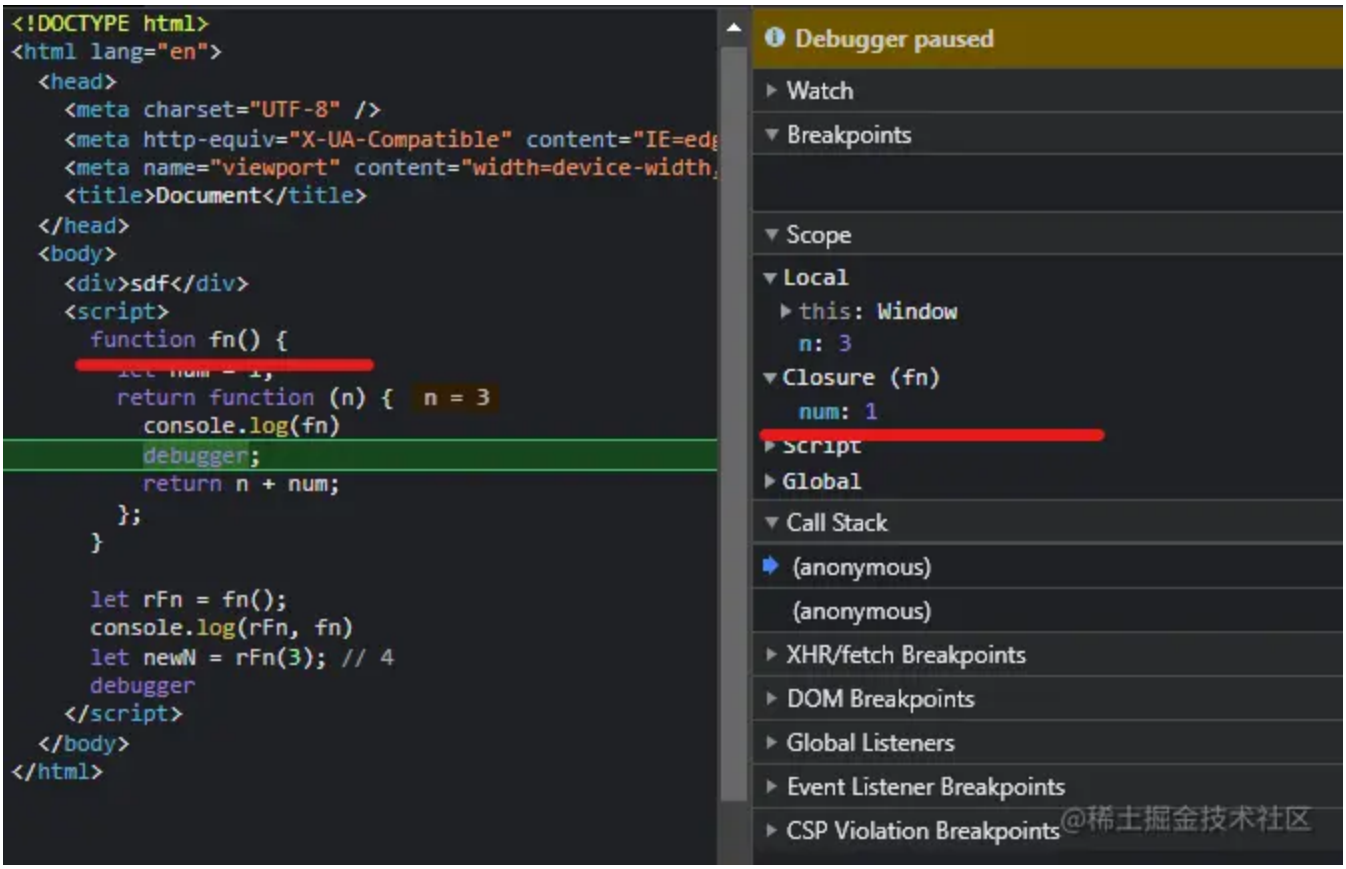
```js 复制代码
function fn() {
    let num = 1;
    return function (n) {
        return n + num
    }
}

let rFn = fn()
let newN = rFn(3) // 4
```

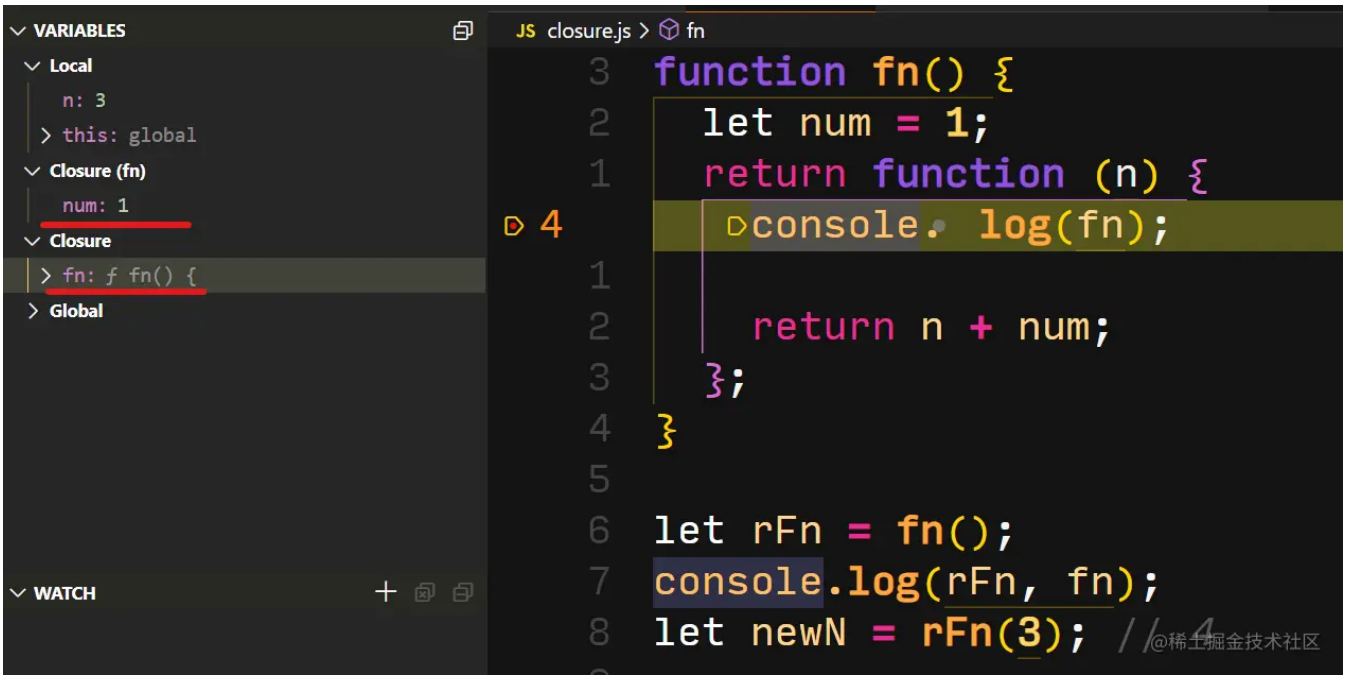
num 变量作用域在 fn 函数中, rFn 函数却能访问 num 变量，这就是闭包函数能访问外部函数变量。

从浏览器调试和 VSCode Nodejs 调试看闭包

- 浏览器



- VS Code 配合 Node.js



看到 Closure 中 fn 是闭包函数，其中保存 num 变量。

一个经典的闭包：单线程事件机制+循环问题，以及解决办法

```js
for (var i = 1; i <= 5; i++) {
  setTimeout(() => {
    console.log(i);
  }, i * 1000);
}
```

输出的结果都是 6，为什么?

- for 循环是同步任务
- setTimeout 异步任务

for 循环一次，就会将 setTimeout 异步任务加入到浏览器的异步任务队列中，同步任务完成之后，再从异步任务中拿新任务在线程中执行。由于 setTimeout 能够访问外部变量 i, 当同步任务完成之后，i 已经变成了6， setTimeout 中能够访问变量 i 都是 6。

解决办法1：使用 let 声明

```ts 复制代码
for (var i = 1; i <= 5; i++) {
  setTimeout(() => {
    console.log(i);
  }, i * 1000);
}
```

解决办法2：自执行函数 + 闭包

```ts 复制代码
for (var i = 1; i <= 5; i++) {
  (function(i){
      setTimeout(() => {
    console.log(i);
  }, i * 1000)
  })(i)
}
```

解决办法3：setTimeout 传递第三参数

> 第三个参数意思：附加参数，一旦定时器到期，它们会作为参数传递给要执行的函数

```ts 复制代码
for (var i = 1; i <= 5; i++) {
  setTimeout((j) => {
    console.log(j);
  }, 1000 * i, i);
}
```

闭包与函数科里化

```ts 复制代码
function add(num) {
  return function (y) {
    return num + y;
  };
};
let incOneFn = add(1); let n = incOneFn(1);  // 2
let decOneFn = add(-1); let m = decOneFn(1); // 0
```

add 函数的 参数 保存了闭包函数变量。

实际作用

在函数式编程闭包有非常重要的作用，lodash 等早期工具函数弥补 javascript 缺陷的工具函数，有大量的闭包的使用场景。

使用场景

- 创建私有变量
- 延长变量生命周期

节流函数

防止滚动行为，过度执行函数，必须要节流, 节流函数接受 `函数` + `时间` 作为参数，都是闭包中变量，以下是一个简单 setTimeout 版本:

```ts 复制代码
function throttle(fn, time=300){
    var t = null;
    return function(){
        if(t) return;
        t = setTimeout(() => {
            fn.call(this);
            t = null;
        }, time);
    }
}
```

防抖函数

一个简单的基于 setTimeout 防抖的函数的实现

```ts 复制代码
function debounce(fn,wait){
    var timer = null;
    return function(){
        if(timer !== null){
            clearTimeout(timer);
        }
        timer = setTimeout(fn,wait);
    }
}
```

React.useCallback 闭包陷阱问题

问题说明：`父/子` 组件关系, 父子组件都能使用 click 事件同时修改 state 数据, 并且子组件拿到传递下的 props 事件属性，是经过 `useCallback` 优化过的。也就是这个被优化过的函数，存在闭包陷阱，（保存一直是初始 state 值）

```tsx
import { useState, useCallback, memo } from "react";

const ChildWithMemo = memo((props: any) => {
  return (
    <div>
      <button onClick={props.handleClick}>Child click</button>
    </div>
  );
});

const Parent = () => {
  const [count, setCount] = useState(1);

  const handleClickWithUseCallback = useCallback(() => {
    console.log(count);
  }, []); // 注意这里是不能监听 count， 因为每次变化都会重新绑定，造成造成子组件重新渲染

  return (
    <div>
      <div>parent count : {count}</div>
      <button onClick={() => setCount(count + 1)}>click</button>
      <ChildWithMemo handleClick={handleClickWithUseCallback} />
    </div>
  );
};

export default Parent
```

- ChildWithMemo 使用 memo 进行优化，
- handleClickWithUseCallback 使用 useCallback 优化

> 问题是点击子组件时候，输出的 count 是初始值（被闭包了）。

解决办法就是使用 useRef 保存操作变量函数：

```tsx
import { useState, useCallback, memo, useRef } from "react";
```

```
const ChildWithMemo = memo((props: any) => {
  console.log("rendered children")
  return (
    <div>
      <button onClick={() => props.countRef.current()}>Child click</button>
    </div>
  );
});

const Parent = () => {
  const [count, setCount] = useState(1);
  const countRef = useRef<any>(null)

  countRef.current = () => {
    console.log(count);
  }
  return (
    <div>
      <div>parent count : {count}</div>
      <button onClick={() => setCount(count + 1)}>click</button>
      <ChildWithMemo countRef={countRef} />
    </div>
  );
};
export default Parent
```

针对这个问题，React 曾经认可过社区提出的增加 useEvent 方案，但是后面 useEvent 语义问题被废弃了，对于渲染优化 React 采用了编译优化的方案。其实类似的问题也会发生在 useEffect 中，使用时要注意闭包陷阱。

性能问题

- 闭包不要随意定义，定义了一定找到合适的位置进行销毁。因为闭包的变量保存在内存中，不会被销毁，占用较高的内存。

使用 chrome 面板功能 timeline + profiles 面板

> 1. 打开开发者工具，选择 Timeline 面板
> 2. 在顶部的 `Capture` 字段里面勾选 Memory
> 3. 点击左上角的录制按钮。
> 4. 在页面上进行各种操作，模拟用户的使用情况。

> 5. 一段时间后，点击对话框的 stop 按钮，面板上就会显示这段时间的内存占用情况。