20.回调函数与图形用户界面

回调函数

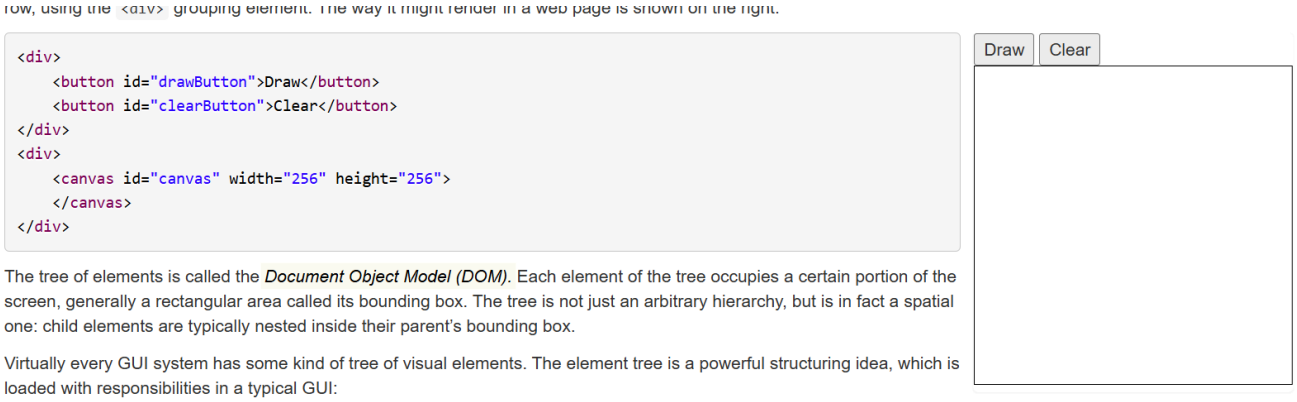
正常应该是：模块提供函数，然后客户端调用函数

回调函数：客户端提供函数，模块调用（比如16.Map,Filter,Reduce这三个东西就是回调函数）

事件循环

JavaScript/TypeScript运行的核心是先进先出的队列，会把事件逐个放进去，然后逐个处理，以setTimeout()(一个异步回调函数)为例，事件的一个来源是由 `setTimeout` 创建的定时器。定时器到期时，会将一个 “定时器完成” 事件及其回调函数放置在队列的末尾。当事件循环最终从队列顶部取出这个事件时，它会调用回调函数，从而完成 `setTimeout` 的任务。

HTML与文档对象模型



输入左边HTML那段代码，会出现右边的图形化界面。

其中在每个模块中都要用到回调函数来“监听”键盘和鼠标行为

```
drawButton.addEventListener('click', (event:MouseEvent) => {
    drawRandomShape();
});
```

该方法的第二个参数是一个使用箭头函数语法的函数表达式，我们在上一节课中已经学过这种语法。

图形用户界面的事件处理是监听器模式的一个实例。在监听器模式中：

- 一个事件源会生成一系列离散的事件。
- 一个或多个其他模块订阅（或监听）这些事件流，并提供一个在新事件发生时会被调用的函数。

在这个例子中：

- 按钮是事件源；
- 它产生的事件是按钮被按下；
- 监听器是由客户端提供的函数。

一个事件通常会包含额外的信息，这些信息可能会被封装在一个事件对象中（就像这里的 `MouseEvent` ），或者作为参数传递给监听器函数。

当一个事件发生时，事件源会通过调用已订阅的监听器回调函数，将事件分发给所有订阅了该事件的监听器。

这种设计模式叫作“监听器模式”。包括事件源和监听器，程序员写的叫做事件源，客户端通过调用事件源相关的监听器来实现对鼠标，键盘内容的及时响应。
例子：

Counter spec

`Counter` has several operations:

```
class Counter {

    /** Make a counter initially set to zero. */
    public constructor()  { ... }

    /** @returns the value of this counter. */
    public get value():bigint  { ... }

    /** Increment this counter. */
    public increment():void  { ... }

    /** Modifies this counter by adding a listener.
      * @param listener called by this counter when it changes. */
    public addEventListener(listener: (numberReached:bigint) => void):void  { ... }

    /** Modifies this counter by removing a listener.
      * @param listener will no longer be called by this counter. */
    public removeEventListener(listener: (numberReached:bigint) => void):void  { ... }
}
```

From a client's point of view, we can make a counter and attach listeners to it:

```
const counter = new Counter();
counter.addEventListener((value:bigint) => console.log(value));
```

这里的Counter是程序员写的计算器模块，客户创建一个Counter后调用addEventListener监听器，每当counter.increment时都会打印内容。