目录

学习目标

- 能够使用call方法继承父构造函数中的属性
- 能够借用原型对象继承方法
- 能够获取指定对象的属性名
- 能够通过Object.defineProperty方法定义新属性或修改原有的属性

案例-别踩白块

```
思路:
1.创建页面文件,写静态游戏可视区域和游戏的4x4 div的游戏区域
2.让div游戏区 动起来 向下运动(不断增加游戏区域的top值,让游戏动起来)
3.动态添加一行
3.1 什么时候添加一行? 当游戏区的top大于等于0的时候
3.2 如何动态创建一行 document.createElement
3.3 每行中有4个div格子
3.4 把动态创建好的行,添加到游戏区的最前面
3.5 重新将游戏区的top值设置-100px
```

```
10
11  4.随机创建黑块
12  4.1 随机得到 1 , 2, 3, 4之间的一个数
13  4.2 如果当前的i等于我们的随机整数,我们就添加.black类,那么就是黑块
14
15
16  5.删除底部多余的行  如果游戏区中行数大于5,我们就删除最后一行
17
18
19  6.绑定点击事件  在事件中进行判断  点击的是黑块还是白块;如果是黑块就加分变白,白块就游戏结束;
20  6.1 如果点击的是黑块就加1分,黑块变白
21  6.2 获取分数对象文本框
22  6.3 定义一个保存分数的全局变量
23  6.4 如果点击的白块就弹窗提示游戏结束  清除定时器
24
25  7.游戏结束后的判断和处理
26  7.1 游戏结束了,此时黑白块都应该是不允许点击的了,而且点击黑块以后也不能加分
27  7.2 定义一个全局变量保存游戏是否结束
28  7.3 在判断点击黑白块之前,需要先判断游戏是否已经结束,如果结束了我们就弹窗提示用户游戏已经结束
29  7.4 如果点击的白块,除了清除定时器,还需要修改gameover的值为true
30
31  8.根据分数游戏加速(为了让游戏更有趣味性)
32  8.1 定义一个全局变量speed保存游戏区下落的速度
33  8.2 判断全局变量fenshu是否是5的倍数(我们假设每多5分,就增加一次难度)
34  8.3 在用户成功点击黑块以后,加完分之后,我们就判断当前分数是否为是5的倍数,如果是我们就增加下落的速度
35
36  9.运动到底部判断最后一行是否全部黑块被点击
37  9.1 如果当前行的黑块被成功点击,我们就给这行添加一个自定义属性pass,并且赋值为true
38  9.2 当game.offsetTop大于等于0的时候,代表移动到底部了
39  9.3 判断游戏区最后一行上的自定义属性的值是否为null,
40  如果是则清除定时器,设置游戏状态为结束,并且弹窗提示用户,最后加上一个return终止函数体后续的代码执行
```

1.创建页面文件,写静态游戏可视区域和游戏的4x4 div的游戏区域

```
<!DOCTYPE html>
<html lang="en">
```

```html
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        .score{
            margin-left: 50px;
            margin-top: 44px;
        }
        .box{
            width: 400px;
            height: 400px;
            margin-left: 50px;
            margin-top: 50px;
            border:1px solid blue;
            /* 父相 */
            position: relative;
            /* 溢出隐藏 */
            overflow: hidden;
        }
        .box .game {
            /* 因为后期我们需要让游戏区下落运动起来,所以需要给游戏区设置绝对定位 */
            /* 子绝 */
            position: absolute;
            top: -50px;
            left: 0;
        }
        .box .game .row div{
            width: 98px;
            height: 98px;
            border:1px solid gray;
            float: left;
        }
        .box .game .row .black{
            background: black;
        }
    </style>
</head>
<body>
    <!-- 统计分数的文本框  disabled代表禁用属性,不能可以编辑-->
    <input type="text" class="score" value="0" disabled="disabled">

    <!-- 外层大盒子  可视区 -->
    <div class="box">
        <!-- 内层盒子 游戏区域 也就是需要一直向下运动并且可以点击的区域  -->
        <div class="game">
            <div class="row">
                <div class="black"></div>
                <div></div>
                <div></div>
                <div></div>
            </div>
            <div class="row">
```

```
                <div></div>
                <div class="black"></div>
                <div></div>
                <div></div>
            </div>
            <div class="row">
                <div></div>
                <div></div>
                <div class="black"></div>
                <div></div>
            </div>
            <div class="row">
                <div></div>
                <div></div>
                <div></div>
                <div class="black"></div>
            </div>
        </div>
    </div>
</body>
</html>
```

2.让game游戏区 动起来 向下运动(不断增加游戏区域的top值,让游戏动起来)

```
    <script>
        // 2.让game游戏区 动起来 向下运动(不断增加游戏区域的top值,让游戏动起来)
        // 获取game对象
        var game = document.querySelector(".game");
        // 向下运动就是使用定时器不断增加game游戏区的top值
        setInterval(function(){
            // 怎么获取当前game游戏区的相对父元素box的上外边距?  怎么设置偏移量top值变化?
            // offsetTop是一个只读属性,得到的是一个数值
            // style属性可以读写,但是读的时候,需要是行内式才可以读取到,得到的时候一个字符串的值,带单位
            game.style.top = game.offsetTop + 1 + "px";
        },15)
    </script>
```

3.动态添加一行

```
3.1 什么时候添加一行? 当游戏区的top大于等于0的时候
3.2 如何动态创建一行 document.createElement
3.3 每行中有4个div格子
3.4 把动态创建好的行,添加到游戏区的最前面
3.5 重新将游戏区的top值设置-100px
```

```
<script>
    // 2.让game游戏区 动起来 向下运动(不断增加游戏区域的top值,让游戏动起来)
    // 获取game对象
    var game = document.querySelector(".game");
    // 向下运动就是使用定时器不断增加game游戏区的top值
    setInterval(function(){
        // 怎么获取当前game游戏区的相对父元素box的上外边距?  怎么设置偏移量top值变化?
        // offsetTop是一个只读属性,得到的是一个数值
        // style属性可以读写,但是读的时候,需要是行内式才可以读取到,得到的时候一个字符串的值,带单位
        game.style.top = game.offsetTop + 1 + "px";


        // 3.动态添加一行
        // 3.1 什么时候添加一行? 当游戏区的top大于等于0的时候
        if( game.offsetTop >= 0){
            // 3.2 如何动态创建一行 document.createElement
            var row = document.createElement("div");
            row.className = "row";
            // 3.3 每行中有4个div格子
            for(var i = 1;i<=4;i++){
                // 创建div标签
                var div = document.createElement("div");
                // 把div添加到row行中
                row.appendChild(div);
            }
            // 3.4 把动态创建好的行,添加到游戏区的最前面,因为我们游戏区域是从上向下运动的,所以新添加的一行需要添加到游戏区子元素列表最前面
            // 语法:  父节点.insertbefore(子节点,指定位置)
            game.insertBefore(row, game.children[0] )
            // 3.5 重新将游戏区的top值设置-100px  这样看起来才像无缝滚动
            game.style.top = "-100px";
        }


    },50)
</script>
```

4.随机创建黑块

4.1 随机得到 1 , 2, 3, 4之间的一个数

4.2 如果当前的i等于我们的随机整数,我们就添加.black类,那么就是黑块

```
<script>
    // 2.让game游戏区 动起来 向下运动(不断增加游戏区域的top值,让游戏动起来)
    // 获取game对象
    var game = document.querySelector(".game");
    // 向下运动就是使用定时器不断增加game游戏区的top值
    setInterval(function(){
        // 怎么获取当前game游戏区的相对父元素box的上外边距?  怎么设置偏移量top值变化?
        // offsetTop是一个只读属性,得到的是一个数值
        // style属性可以读写,但是读的时候,需要是行内式才可以读取到,得到的时候一个字符串的值,带单位
        game.style.top = game.offsetTop + 1 + "px";


        // 3.动态添加一行
        // 3.1 什么时候添加一行? 当游戏区的top大于等于0的时候
        if( game.offsetTop >= 0){
            // 3.2 如何动态创建一行 document.createElement
            var row = document.createElement("div");
            row.className = "row";


            // 4.随机创建黑块  因为我们刚才第3步创建行是没有黑块,如何给row行随机某个div设置黑块呢?  其实就是随机给row行某个div设置类名black即可
            // 4.1 随机得到 1 , 2, 3, 4之间的一个数
            var random = Math.floor( Math.random()*4 + 1);

            // 3.3 每行中有4个div格子
            for(var i = 1;i<=4;i++){
                // 创建div标签
                var div = document.createElement("div");

                // 4.2 如果当前的i等于我们的随机整数,我们就添加.black类,那么就是黑块
                if(i == random ){
                    div.className = "black";
                }


                // 把div添加到row行中
                row.appendChild(div);
            }
            // 3.4 把动态创建好的行,添加到游戏区的最前面,因为我们游戏区域是从上向下运动的,所以新添加的一行需要添加到游戏区子元素列表最前面
            // 语法:  父节点.insertbefore(子节点,指定位置)
            game.insertBefore(row, game.children[0] )
            // 3.5 重新将游戏区的top值设置-100px  这样看起来才像无缝滚动  因为一个行的高度就是100高,这样才可以让新行完整显示并下落
            game.style.top = "-100px";
        }


    },30)
</script>
```

5.删除底部多余的行 如果游戏区中行数大于5,我们就删除最后一行

```
<script>
    // 2.让game游戏区 动起来 向下运动(不断增加游戏区域的top值,让游戏动起来)
    // 获取game对象
    var game = document.querySelector(".game");
    // 向下运动就是使用定时器不断增加game游戏区的top值
    setInterval(function(){
        // 怎么获取当前game游戏区的相对父元素box的上外边距?  怎么设置偏移量top值变化?
        // offsetTop是一个只读属性,得到的是一个数值
        // style属性可以读写,但是读的时候,需要是行内式才可以读取到,得到的时候一个字符串的值,带单位
        game.style.top = game.offsetTop + 1 + "px";


        // 3.动态添加一行
        // 3.1 什么时候添加一行? 当游戏区的top大于等于0的时候
        if( game.offsetTop >= 0){ // 代表到达底部
            // 5.删除底部多余的行 可视区中最多能够显示游戏区行黑白块,新行还没完全出现,最下面一行还没有完成消失,再加上中间3行; 如果游戏区中行数大于5,我们就删除最后一行
            // 如何得到game游戏层中的行数?
            // console.log( game.children.length );
            if( game.children.length >= 5 ){ // 到达底部以后,判断当前游戏区中的行数是不是大于等于5
                // 删除game游戏区中的最后一行
                // 父节点.removeChild(子节点)
                // game.removeChild( game.lastElementChild );
                game.removeChild( game.children[ game.children.length-1 ] );


                // 指定节点.remove
                // game.lastElementChild.remove();
            }



            // 3.2 如何动态创建一行 document.createElement
            var row = document.createElement("div");
            row.className = "row";


            // 4.随机创建黑块  因为我们刚才第3步创建行是没有黑块,如何给row行随机某个div设置黑块呢?  其实就是随机给row行某个div设置类名black即可
            // 4.1 随机得到 1 , 2, 3, 4之间的一个数
            var random = Math.floor( Math.random()*4 + 1);

            // 3.3 每行中有4个div格子
            for(var i = 1;i<=4;i++){
                // 创建div标签
                var div = document.createElement("div");

                // 4.2 如果当前的i等于我们的随机整数,我们就添加.black类,那么就是黑块
                if(i == random ){
                    div.className = "black";
                }


                // 把div添加到row行中
                row.appendChild(div);
            }
            // 3.4 把动态创建好的行,添加到游戏区的最前面,因为我们游戏区域是从上向下运动的,所以新添加的一行需要添加到游戏区子元素列表最前面
            // 语法:  父节点.insertbefore(子节点,指定位置)
            game.insertBefore(row, game.children[0] )
            // 3.5 重新将游戏区的top值设置-100px  这样看起来才像无缝滚动  因为一个行的高度就是100高,这样才可以让新行完整显示并下落
            game.style.top = "-100px";


        }


    },30)
</script>
```

6.绑定点击事件 在事件中进行判断 点击的是黑块还是白块;如果是黑块就加分变白,白块就游戏结束;

```
6.1 如果点击的是黑块就加1分,黑块变白
6.2 获取分数对象文本框
6.3 定义一个保存分数的全局变量
6.4 如果点击的白块就弹窗提示游戏结束  清除定时器
```

```
// 6.绑定点击事件 在事件中进行判断 点击的是黑块还是白块;如果是黑块就加分变白,白块就游戏结束;
// 6.2 获取分数对象文本框
var score = document.querySelector(".score");
// 6.3 定义一个保存分数的全局变量
var fenshu = 0;

// 使用事件委托绑定点击事件,这样就不需要给game中的row行每个div绑定单击事件
game.onclick = function( e ){
    // 如果给了游戏区域绑定单击事件,如何得到我点击的那个小方块? 通过事件对象e.target属性得到触发事件目标对象
    e = e || window.event;

    // 6.1 如果点击的是黑块就加1分,黑块变白
    // console.log( e.target );
    // console.log( e.target.className );
    if( e.target.className == "black"){
        // 只需要移动div上面的.black类名即可
        e.target.removeAttribute("class");
        // 分数加1
        score.value = ++fenshu;
    }else{
        // 6.4 如果点击的白块就弹窗提示游戏结束 清除定时器
        alert("游戏结束");
        // 清除向下运动的定时器
        window.clearInterval( timer );
    }
}
```

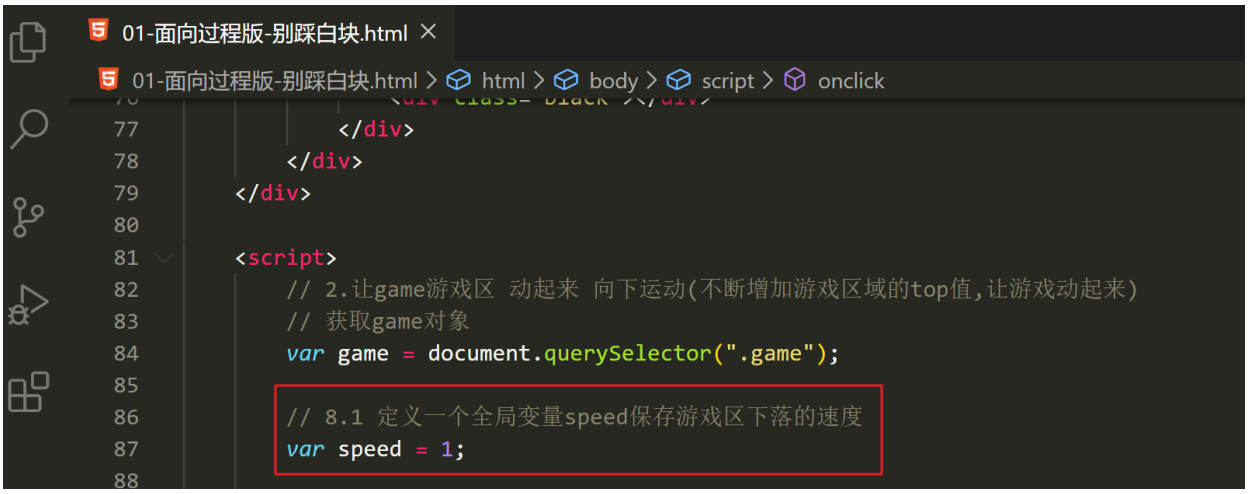
7.游戏结束后的判断和处理

1. 7.1 游戏结束了,此时黑白块都应该是不允许点击的了,而且点击黑块以后也不能加分
2. 7.2 定义一个全局变量保存游戏是否结束
3. 7.3 在判断点击黑白块之前,需要先判断游戏是否已经结束,如果结束了我们就弹窗提示用户游戏已经结束
4. 7.4 如果点击的白块,除了清除定时器,还需要修改gameover的值为true

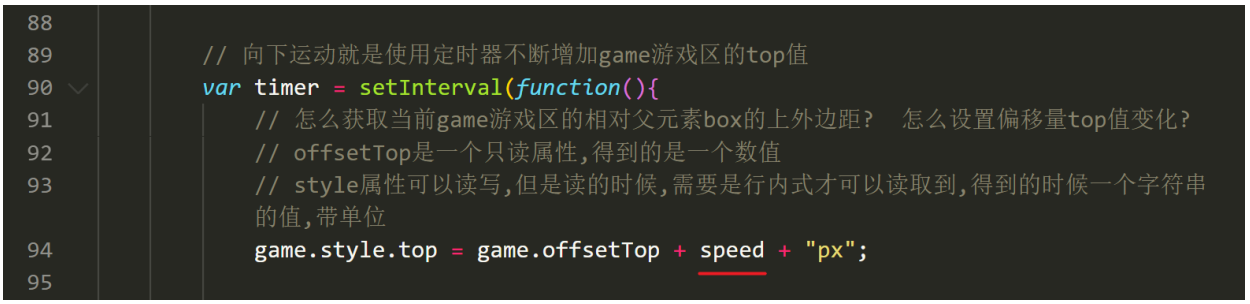
```
148         // 7.游戏结束后的判断和处理
149         // 7.1 游戏结束了,此时黑白块都应该是不允许点击的了,而且点击黑块以后也不能加分
150         // 7.2 定义一个全局变量保存游戏是否结束
151         var gameover = false;// false代表游戏没有结束,true代表游戏结束
152         // 使用事件委托绑定点击事件,这样就不需要给game中的row行每个div绑定单击事件
153         game.onclick = function( e ){
154             // 如果给了游戏区域绑定单击事件,如何得到我点击的那个小方块? 通过事件对象e.target属性得到触发事件目标对象
155             e = e || window.event;
156             // 6.1 如果点击的是黑块就加1分,黑块变白
157             // console.log( e.target );
158             // console.log( e.target.className );
159             // 7.3 在判断点击黑白块之前,需要先判断游戏是否已经结束,如果结束了我们就弹窗提示用户游戏已经结束
160             if( gameover ){
161                 alert("游戏结束,请重新开始");
162             }else if( e.target.className == "black"){
163                 // 只需要移动div上面的.black类名即可
164                 e.target.removeAttribute("class");
165                 // 分数加1
166                 score.value = ++fenshu;
167             }else{
168                 // 6.4 如果点击的白块就弹窗提示游戏结束 清除定时器
169                 alert("游戏结束");
170                 // 清除向下运动的定时器
171                 window.clearInterval( timer );
172                 // 7.4 如果点击的白块,除了清除定时器,还需要修改gameover的值为true
173                 gameover = true;
174             }
175         }
```

8.根据分数游戏加速(为了让游戏更有趣味性)

```
8.1 定义一个全局变量speed保存游戏区下落的速度
8.2 判断全局变量fenshu是否是5的倍数(我们假设每多5分,就增加一次难度)
8.3 在用户成功点击黑块以后,加完分之后,我们就判断当前分数是否为是5的倍数,
如果是我们就增加下落的速度
```



```
01-面向过程版-别踩白块.html
01-面向过程版-别踩白块.html > html > body > script > onclick
            </div>
        </div>
    </div>

    <script>
        // 2.让game游戏区 动起来 向下运动(不断增加游戏区域的top值,让游戏动起来)
        // 获取game对象
        var game = document.querySelector(".game");

        // 8.1 定义一个全局变量speed保存游戏区下落的速度
        var speed = 1;
```

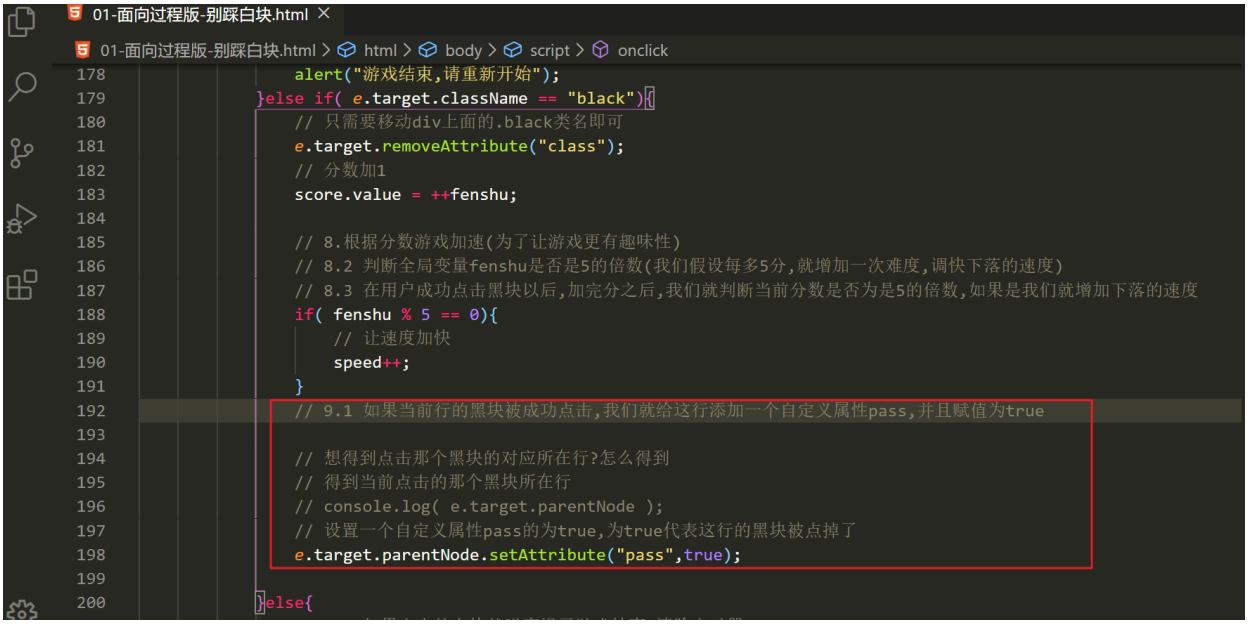


```
    // 向下运动就是使用定时器不断增加game游戏区的top值
    var timer = setInterval(function(){
        // 怎么获取当前game游戏区的相对父元素box的上外边距?  怎么设置偏移量top值变化?
        // offsetTop是一个只读属性,得到的是一个数值
        // style属性可以读写,但是读的时候,需要是行内式才可以读取到,得到的时候一个字符串
        的值,带单位
        game.style.top = game.offsetTop + speed + "px";
```

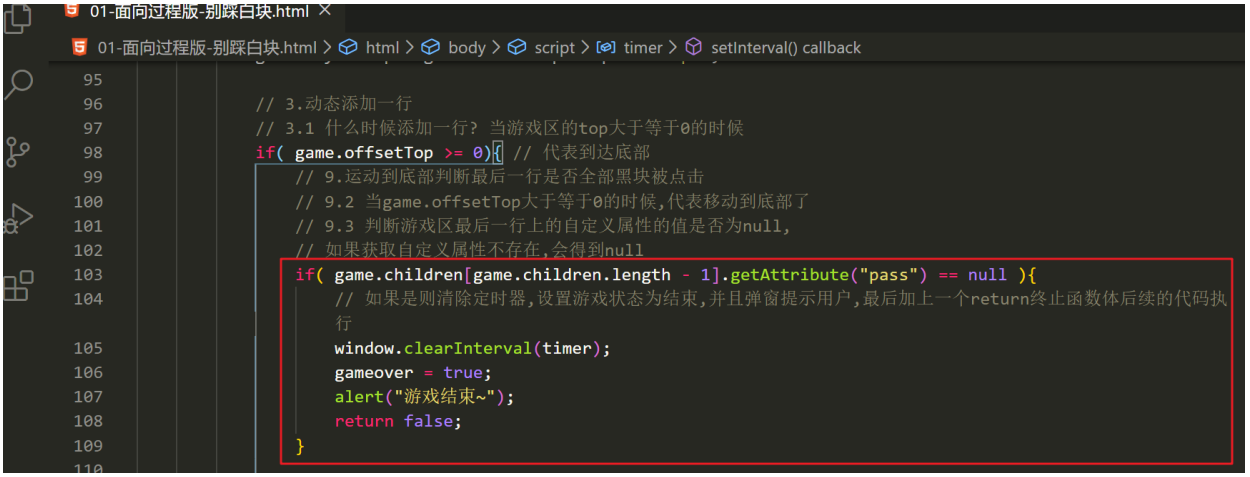
```
01-面向过程版-别踩白块.html
01-面向过程版-别踩白块.html > html > body > script > onclick
    game.onclick = function( e ){
        // 如果给了游戏区域绑定单击事件,如何得到我点击的那个小方块? 通过事件对象e.target属性得到触发事件目标对象
        e = e || window.event;
        // 6.1 如果点击的是黑块就加1分,黑块变白
        // console.log( e.target );
        // console.log( e.target.className );
        // 7.3 在判断点击黑白块之前,需要先判断游戏是否已经结束,如果结束了我们就弹窗提示用户游戏已经结束
        if( gameover ){
            alert("游戏结束,请重新开始");
        }else if( e.target.className == "black"){
            // 只需要移动div上面的.black类名即可
            e.target.removeAttribute("class");
            // 分数加1
            score.value = ++fenshu;

            // 8.根据分数游戏加速(为了让游戏更有趣味性)
            // 8.2 判断全局变量fenshu是否是5的倍数(我们假设每多5分,就增加一次难度,调快下落的速度)
            // 8.3 在用户成功点击黑块以后,加完分之后,我们就判断当前分数是否为是5的倍数,如果是我们就增加下落的速度
            if( fenshu % 5 == 0){
                // 让速度加快
                speed++;
            }
```

9.运动到底部判断最后一行是否全部黑块被点击

```
9.1 如果当前行的黑块被成功点击,我们就给这行添加一个自定义属性pass,并且赋值为true
9.2 当game.offsetTop大于等于0的时候,代表移动到底部了
9.3 判断游戏区最后一行上的自定义属性的值是否为null,
如果是则清除定时器,设置游戏状态为结束,并且弹窗提示用户,
最后加上一个return终止函数体后续的代码执行
```



```
01-面向过程版-别踩白块.html > html > body > script > onclick
178                 alert("游戏结束,请重新开始");
179             }else if( e.target.className == "black"){
180                 // 只需要移动div上面的.black类名即可
181                 e.target.removeAttribute("class");
182                 // 分数加1
183                 score.value = ++fenshu;
184
185                 // 8.根据分数游戏加速(为了让游戏更有趣味性)
186                 // 8.2 判断全局变量fenshu是否是5的倍数(我们假设每多5分,就增加一次难度,调快下落的速度)
187                 // 8.3 在用户成功点击黑块以后,加完分之后,我们就判断当前分数是否为是5的倍数,如果是我们就增加下落的速度
188                 if( fenshu % 5 == 0){
189                     // 让速度加快
190                     speed++;
191                 }
192                 // 9.1 如果当前行的黑块被成功点击,我们就给这行添加一个自定义属性pass,并且赋值为true
193
194                 // 想得到点击那个黑块的对应所在行?怎么得到
195                 // 得到当前点击的那个黑块所在行
196                 // console.log( e.target.parentNode );
197                 // 设置一个自定义属性pass的为true,为true代表这行的黑块被点掉了
198                 e.target.parentNode.setAttribute("pass",true);
199
200             }else{
```



```
01-面向过程版-别踩白块.html > html > body > script > timer > setInterval() callback
95
96              // 3.动态添加一行
97              // 3.1 什么时候添加一行? 当游戏区的top大于等于0的时候
98              if( game.offsetTop >= 0){ // 代表到达底部
99                  // 9.运动到底部判断最后一行是否全部黑块被点击
100                 // 9.2 当game.offsetTop大于等于0的时候,代表移动到底部了
101                 // 9.3 判断游戏区最后一行上的自定义属性的值是否为null,
102                 // 如果获取自定义属性不存在,会得到null
103                 if( game.children[game.children.length - 1].getAttribute("pass") == null ){
104                     // 如果是则清除定时器,设置游戏状态为结束,并且弹窗提示用户,最后加上一个return终止函数体后续的代码执行
105                     window.clearInterval(timer);
106                     gameover = true;
107                     alert("游戏结束~");
108                     return false;
109                 }
110
```

继承

ES6之前并没有给我们提供 extends 继承。我们可以通过构造函数+原型对象模拟实现继承，被称为组合继承。

es6中的继承是用extends关键字实现的

```
class Father{

}
class Son extends Father{
 // 子类就可以使用父类的属性和方法
}
```

call()

调用这个函数,并且修改函数运行时this指向

语法

```
函数名.call(thisArg,arg1,arg2,...)

thisArg      : 当前调用函数this的指向对象
arg1,arg2... : 传递的其他参数,使用逗号隔开
```

举例:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // call()可以调用函数,还可以改变函数this的指向
        // 语法
        // 函数名.call( [this指向谁 ,参数1,参数2...] )


        function fn(x,y){
            // 因为普通函数是由window对象调用的
            // 所以普通函数里面的this代表window对象
            console.log( this );
            console.log( x );
            console.log( y );
            console.log( x + y);
        }


        // fn(10,20);
        // 之前我们也说过,定义在全局的变量和函数,都会放在window对象下
        // window.fn(10,20);


        // call()的第一个功能就是调用函数,执行函数体中的代码
        fn.call();
        console.log("");


        // call()的第二个功能是改变this的指向,并且可以给函数传实参
        var obj = {
            uname : "zhangsan",
            age   : 23
        };
        // 以下代码意思就是调用fn函数,并且把函数里面的this指向obj对象,再传两个实参进去
        fn.call( obj, 50, 30 );



        console.log("====================我是华丽的分割线====================");


        var sex = "男";
        function fn2(){
            console.log( this );
            console.log( this.sex );
            console.log("");
        }


        var obj2 = {
            uname : "李四",
            sex   : "女"
        };
```

```
        var obj3 = {
            uname : "王五",
            sex   : "保密"
        }


        fn2.call();
        fn2.call( obj2 );
        fn2.call( obj3 );
    </script>
</body>
</html>
```

子构造函数继承父构造函数中的属性

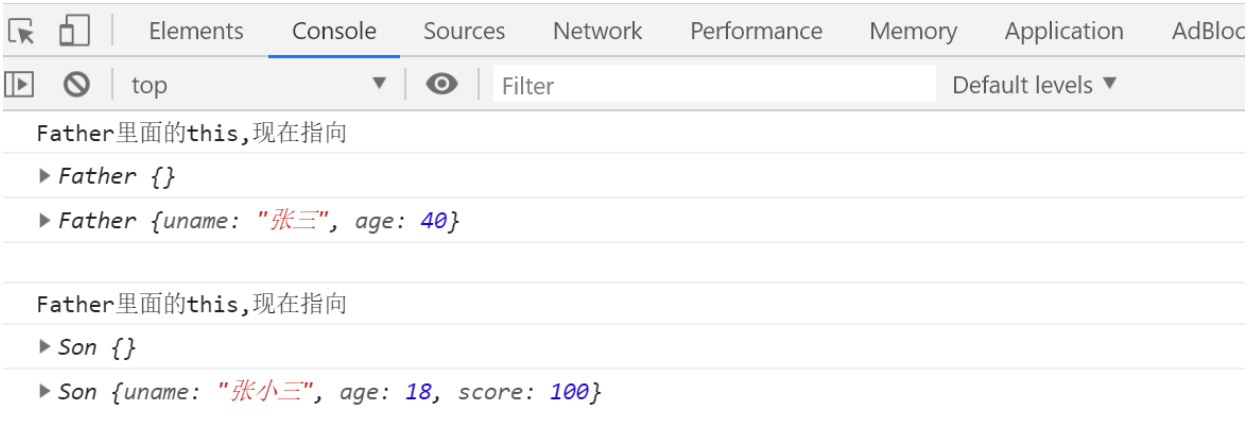
ES6之前并没有给我们提供 extends 继承。我们可以通过构造函数+原型对象模拟实现继承，被称为组合继承。

核心原理： 通过 call() 把父类型的 this 指向子类型的 this ，这样就可以实现子类型继承父类型的属性。

实现步骤:

```
先定义一个父构造函数

再定义一个子构造函数

子构造函数继承父构造函数的属性(使用call方法)
也通过 call() 把父类型的 this 指向子类型的 this
Father.call(this,uname,age);
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // 子构造函数继承父构造函数中的属性
        // ES6之前并没有给我们提供 extends 继承。我们可以通过构造函数+原型对象模拟实现继承，被称为组合继承。
        // 核心原理： 通过 call() 把父类型的 this 指向子类型的 this ，这样就可以实现子类型继承父类型的属性。


        // 父构造器中有两个属性
        function Father(uname,age){
            console.log( "Father里面的this,现在指向" );
            console.log( this );
            this.uname = uname;
            this.age = age;
        }


        // 子构造器
        function Son(uname,age,score){
            // 函数名.call( this指向谁 ,参数1,参数2 );
            // 构造函数也是函数,所以也可以使用call来调用


            // 在Son构造函数里面this代表Son实例对象
            // console.log( this );



            // 调用通过call方法调用Father构造函数,并且把Father里面的this改为Son的实例对象
            Father.call(this , uname , age);

            // 子构造器还可以有的属性
            this.score = score;
        }


        var father = new Father("张三",40);
        console.log( father );


        console.log("");


        var son = new Son("张小三",18, 100);
        console.log( son );
    </script>
</body>
</html>
```



借用原型对象继承方法

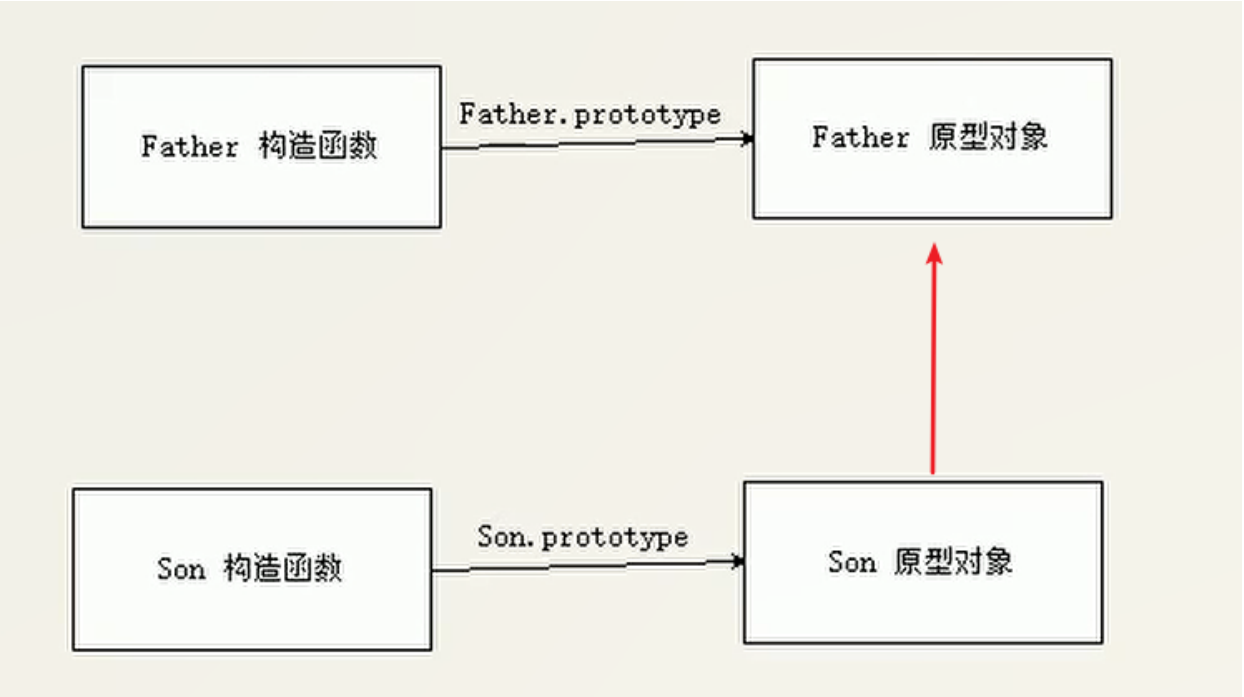
Son.prototype = Father.prototype; 如果直接这样赋值会有问题,如果修改了子原型对象,父原型对象也会跟着一起变化

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // 父构造函数
        function Father(uname,age){
            this.uname = uname;
            this.age = age;


            /* this.money = function(){
                console.log( "可以赚好多好多的软妹子" );
            } */
        }


        // 如果把方法加到Father的原型对象上
        Father.prototype.money = function(){
            console.log( "可以赚好多好多的软妹子" );
        }

        // 子构造函数
        function Son(uname,age,score){
            Father.call(this , uname , age);
            // score是子构造函数独有的属性
            this.score = score;
        }


        // 借用原型对象继承方法
        // Son.prototype = Father.prototype; "如果直接这样赋值会有问题,如果修改了子原型对象,父原型对象也会跟着一起变化"  因为对象是复合类型 复合类型之前的赋值是引用传递,传地址的


        // 让Son的原型对象等于Father的原型对象
        Son.prototype = Father.prototype;


        // 给Son原型对象设置一个考试的方法
        Son.prototype.exam = function(){
            console.log( "这次考了" + this.score + "分" );
        }


        var father = new Father("张三",40);
        console.log( father );
        father.money();
        father.exam();


        console.log("========================================");

        var son = new Son("张小三",18, 100);
        console.log( son );
        // 如果Father的money方法是写在原型对象上的话,son就无法使用了,因为Son构造函数中和Son原型对象上都没有money方法
        son.money();
        son.exam();
    </script>
</body>
</html>
```

以上代码执行效果:

```
▼Father {uname: "张三", age: 40}
    age: 40
    uname: "张三"
  ▼__proto__:
    ▶exam: ƒ ()
    ▶money: ƒ ()
    ▶constructor: ƒ Father(uname,age)
    ▶__proto__: Object
可以赚好多好多的软妹子
这次考了undefined分
==========================================
▼Son {uname: "张小三", age: 18, score: 100}
    age: 18
    score: 100
    uname: "张小三"
  ▼__proto__:
    ▶exam: ƒ ()
    ▶money: ƒ ()
    ▶constructor: ƒ Father(uname,age)
    ▶__proto__: Object
可以赚好多好多的软妹子
这次考了100分
```

分析原因,因为对象是复合类型,所以对象之间的值传递是引用传递,如果直接写Son.prototype = Father.prototype ,如下图



那么子构造函数应该如何继承父构造函数的方法呢?

让Son的原型对象为Father的实例,因为Father的实例有一个对象原型__proto__可以找到Father的原型对象

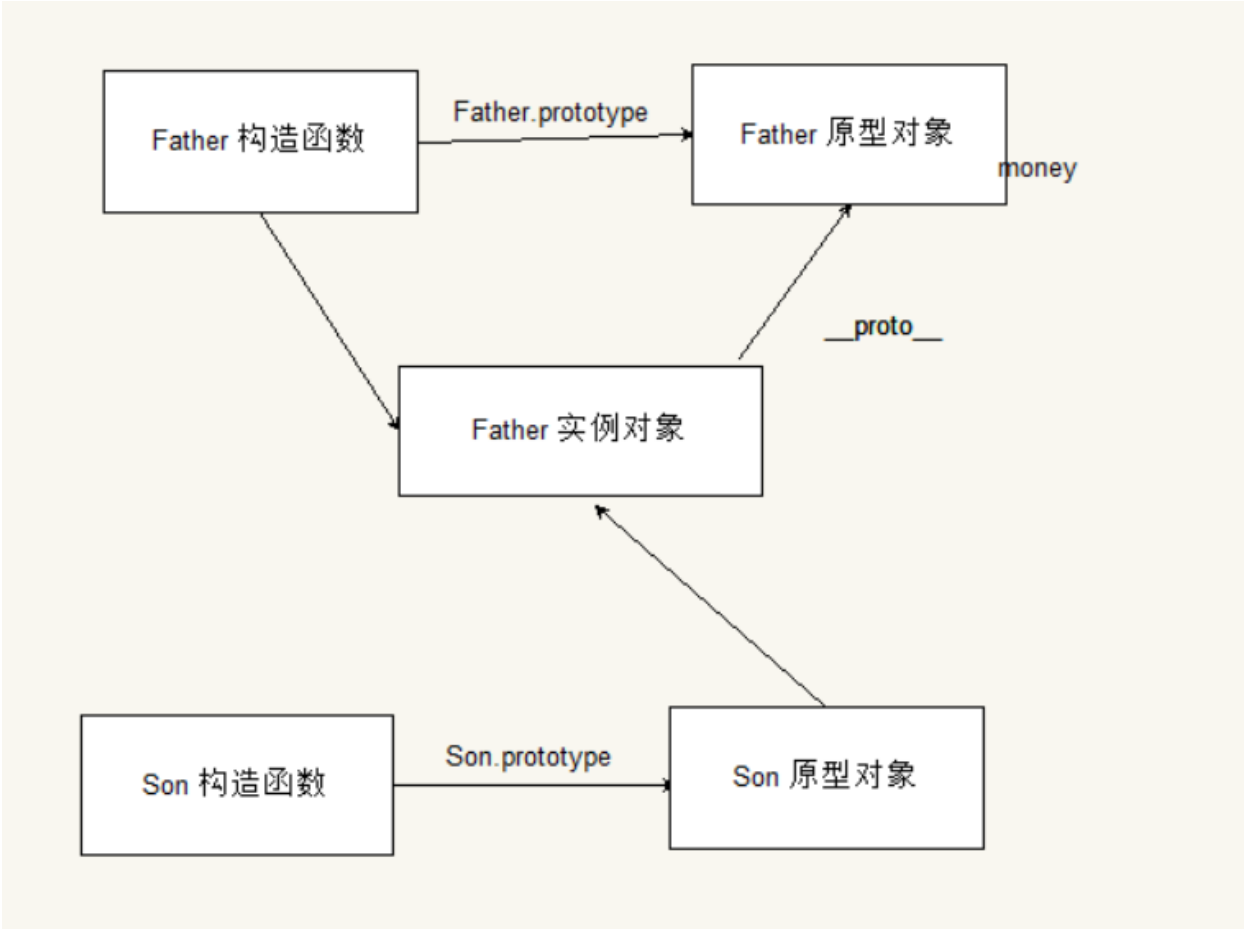
```
Son.prototype = new Father();
```

如果利用对象的形式修改了原型对象,别忘了利用constructor指回原来的构造函数

```
Son.prototype.constructor = Son;
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // 父构造函数
        function Father(uname,age){
            this.uname = uname;
            this.age = age;
        }


        // 如果把方法加到Father的原型对象上
        Father.prototype.money = function(){
            console.log( "可以赚好多好多的软妹子" );
        }

        // 子构造函数
        function Son(uname,age,score){
            Father.call(this , uname , age);
            // score是子构造函数独有的属性
            this.score = score;
        }


        // 借用原型对象继承方法
        // Son.prototype = Father.prototype; "如果直接这样赋值会有问题,如果修改了子原型对象,父原型对象也会跟着一起变化"  因为对象是复合类型 复合类型之前的赋值是引用传递,传地址的


        // 让Son的原型对象等于Father的原型对象
        // Son.prototype = Father.prototype;


        // 那么子构造函数应该如何继承父构造函数的方法呢?


        // 让Son的原型对象为"Father的实例",因为Father的实例有一个对象原型__proto__可以找到Father的原型对象
        Son.prototype = new Father();


        // 如果利用"对象的形式"修改了原型对象,别忘了利用constructor指回原来的构造函数
        // console.log( Son.prototype );
        Son.prototype.constructor = Son;



        // 给Son原型对象设置一个考试的方法
        Son.prototype.exam = function(){
            console.log( "这次考了" + this.score + "分" );
        }


        var father = new Father("张三",40);
        console.log( father );
        father.money();


        console.log("========================================");

        var son = new Son("张小三",18, 100);
        console.log( son );
        // 如果Father的money方法是写在原型对象上的话,son就无法使用了,因为Son构造函数中和Son原型对象上都没有money方法
        son.money();
        son.exam();
    </script>
</body>
</html>
```

图解:


Father 构造函数
Father.prototype
Father 原型对象
money
__proto__
Father 实例对象
Son 构造函数
Son.prototype
Son 原型对象

以上代码执行效果:

```
▼Father {uname: "张三", age: 40}
    age: 40
    uname: "张三"
  ▼__proto__:
    ▶money: ƒ ()
    ▶constructor: ƒ Father(uname,age)
    ▶__proto__: Object
可以赚好多好多的软妹子
==========================================
▼Son {uname: "张小三", age: 18, score: 100}
    age: 18
    score: 100
    uname: "张小三"
  ▼__proto__: Father
      age: undefined
    ▶constructor: ƒ Son(uname,age,score)
    ▶exam: ƒ ()
      uname: undefined
    ▼__proto__:
      ▶money: ƒ ()
      ▶constructor: ƒ Father(uname,age)
      ▶__proto__: Object
可以赚好多好多的软妹子
这次考了100分
```

今日总结

xmind总结知识点

今日作业

请查作业文件夹