



黑马程序员™
www.itheima.com

传智播客旗下
高端IT教育品牌

下棋游戏



目录

Contents

- ◆ 游戏准备
- ◆ 点击下棋
- ◆ 游戏判赢
- ◆ 重新游戏

■ 1. 游戏准备

1.1 游戏演示

玩法：两个玩家，一个玩家使用（X），一个玩家使用（O），轮流在棋盘上下棋（点击单元格）。

获胜条件：横、竖、斜（对角线）三个棋子相同。

平局：棋盘满子，但是，不满足任何一种获胜条件。



1. 游戏准备

1.2 游戏模板说明

重点：运用学到的 TS 知识，来开发下棋游戏。

游戏的模板（HTML、CSS），已准备好，直接使用即可。

模板（HTML、CSS）的说明：

1. 下一步提示：给游戏面板（`#bord`）标签，添加 `x` 或 `o` 类名。
2. 下棋（点击单元格）：给单元格（`.cell`）标签，添加 `x` 或 `o` 类名。
3. 展示和隐藏获胜信息：设置获胜信息面板（`#message`）标签的样式属性 `display`。

目录

Contents

- ◆ 游戏准备
- ◆ 点击下棋
- ◆ 游戏判赢
- ◆ 重新游戏

■ 2. 点击下棋

2.1 单元格点击

效果：点击棋盘的任意单元格，单元格显示 x（默认）。

1. 获取到所有的单元格列表。
2. 遍历单元格列表，给每一个单元格添加点击事件。
3. 给当前被点击的单元格添加类名 x。

优化（1）：防止单元格重复点击，在添加事件时，使用 `once` 属性，让单元格只能被点击一次。

优化（2）：使用函数声明形式的事件处理程序（代码多了后，代码结构会更清晰）。

■ 2. 点击下棋

2.2 切换玩家

效果：玩家（x）和玩家（o）轮流交替下棋。

1. 创建一个存储当前玩家的变量（`currentPlayer`），默认值为：x。
2. 将添加给单元格时写死的类名 x，替换为变量（`currentPlayer`）的值。
3. 切换到另一个玩家：在添加类名（下棋完成一步）后，根据当前玩家，得到另外一个玩家。
4. 处理下一步提示：移除游戏面板中的 x 和 o 类名，添加下一个玩家对应的类名。

■ 2. 点击下棋

枚举

使用变量 (`currentPlayer`) 处理当前玩家, 存在的问题:

变量的类型是 `string`, 它的值可以是任意字符串。

如果不小心写错了 (`o` \rightarrow `0`), 代码不会报错, 但功能就无法实现了, 并且很难找错。

也就是: `string` 类型的变量, 取值太宽泛, 无法很好的限制值为 `x` 和 `o`。

枚举是组织有关联数据的一种方式 (比如, `x` 和 `o` 就是有关联的数据)。

使用场景: 当变量的值, **只能是几个固定值中的一个**, 应该使用**枚举**来实现。

注意: **JS 中没有枚举**, 这是 TS 为了弥补 JS 自身不足而新增的。

枚举

创建枚举的语法：

```
enum 枚举名称 { 成员1, 成员2, ... }
```

示例：

```
enum Gender { Female, Male }  
enum Player { X, O }
```

约定枚举名称、成员名称以大写字母开头。

多个成员之间使用逗号(,)分隔。

注意：枚举中的成员，根据功能自己指定！

注意：枚举中的成员不是键值对！

枚举

使用枚举：

枚举是一种类型，因此，可以其作为变量的类型注解。

```
enum Gender { Female, Male }  
let userGender: Gender
```

访问枚举（Gender）中的成员，作为变量（userGender）的值：

```
userGender = Gender.Female  
userGender = Gender.Male
```

注意：枚举成员是只读的，也就是说枚举中的成员可以访问，但是不能赋值！

```
Gender.Female = '男'           // 错误!
```

枚举

枚举的基本使用总结：

枚举是组织有关联数据的一种方式。

使用场景：当变量的值，只能是几个固定值中的一个，应该使用**枚举**来实现。

```
enum Gender { Female, Male }  
  
let userGender: Gender = Gender.Male
```

注意点：**枚举中的成员是只读的**，因此，只能访问不能赋值！

枚举

问题：将枚举成员赋值给变量，变量的值是什么呢？

```
enum Gender { Female, Male }  
  
let userGender: Gender = Gender.Female  
  
console.log(userGender) // ? 0
```

枚举成员是有值的，默认为：从 0 开始自增的数值。

我们把，枚举成员的值为数字的枚举，称为：**数字枚举**。

当然，也可以给枚举中的成员初始化值。

```
enum Gender { Female = 1, Male }           // Female => 1  Male => 2  
  
enum Gender { Female = 1, Male = 100 }     // Female => 1  Male => 100
```

枚举

字符串枚举：枚举成员的值是字符串。

```
enum Gender { Female = '女', Male = '男' }
```

注意：字符串枚举没有自增长行为，因此，**每个成员必须有初始值**。

```
console.log(Gender.Female) // 女  
console.log(Gender.Male)   // 男
```

枚举

两种常用的枚举总结：

1. 数字枚举：枚举成员的值是数字，默认情况下就是数字枚举。

```
enum Gender { Female, Male }  
enum Gender { Female = 100, Male } // 初始化成员的值
```

特点：成员的值是从 0 开始自增的数值。

2. 字符串枚举：枚举成员的值是字符串。

```
enum Gender { Female = '女', Male = '男' }
```

特点：没有自增行为，需要为每一个成员赋值！

枚举是一组有名字的常量（只读）的集合。

■ 2. 点击下棋

2.3 使用枚举修改当前玩家

效果：使用枚举代替原来的字符串类名（`x` 和 `o`）。

1. 创建字符串枚举（`Player`），提供 `X` 和 `O` 两个成员。
2. 将成员 `X` 的值设置为：'`x`'（类名）；将成员 `O` 的值设置为：'`o`'（类名）。
3. 将变量（`currentPlayer`）的类型设置为 `Player` 枚举类型，默认值为 `Player.X`。
4. 将所有用到 `x` 和 `o` 的地方全部使用枚举成员代替。

目录

Contents

- ◆ 游戏准备
- ◆ 点击下棋
- ◆ 游戏判赢
- ◆ 重新游戏

■ 3. 游戏判赢

3.1 判赢的思路

思路：判断棋盘上，横、竖、斜（对角线）是否存在三个相同的 x 或 o。

只要有一个满足条件，就说明 x 或 o 获胜了。

如果所有单元格都有内容，但没有获胜的情况，就说明是平局。

如何判断？

3. 游戏判赢

3.1 判赢的思路

单元格元素列表 (cells) 中，每个单元格元素都有自己的索引，如下图所示：

0	1	2
3	4	5
6	7	8



// 获胜的所有情况 (8 种) :

```
[0, 1, 2]  [3, 4, 5]  [6, 7, 8]  // 横
[0, 3, 6]  [1, 4, 7]  [2, 5, 8]  // 竖
[0, 4, 8]  [2, 4, 6]                // 斜
```

使用**单元格索引**，来表示每种获胜情况（使用**数组**来存储，比如：[0, 1, 2]）。

3. 游戏判赢

3.1 判赢的思路

单元格元素列表 (cells) 中，每个单元格元素都有自己的索引，如下图所示：

0	1	2
3	4	5
6	7	8



// 获胜的所有情况 (8 种) :

```
[  
    [0, 1, 2], [3, 4, 5], [6, 7, 8], // 横  
    [0, 3, 6], [1, 4, 7], [2, 5, 8], // 竖  
    [0, 4, 8], [2, 4, 6]           // 斜  
]
```

使用**单元格索引**，来表示每种获胜情况（使用**数组**来存储，比如：[0, 1, 2]）。

然后，使用一个“大”数组（外层），来存储这 8 种情况（因为每次判赢都要判断所有情况）。

判断过程：遍历这个大数组，分别判断每一种情况对应的 3 个单元格元素，是否都是相同的 x 或 o 类名。

只要有一种情况满足，就说明获胜了。

3. 游戏判赢

分析判赢数组

数组的基本结构：

```
[ 元素1, 元素2, ... ]
```

判赢数组：每个元素又是数组（二维数组，概念知道即可）。

```
let winsArr = [  
  [0, 1, 2], [3, 4, 5], ...  
]
```

只要是数组用法都一样，比如：

```
// 访问数组元素：  
winsArr[0]      // [0, 1, 2]  
winsArr[0][1]   // 1
```

单元格元素列表说明

单元格元素列表（cells），实际上是一个伪数组。

伪数组的特征：具有长度（length）属性和索引。

伪数组的操作：1 通过索引获取元素 2 使用 for 循环遍历（推荐使用 forEach 方法）。

1. 通过索引获取元素

```
console.log(cells[0])  
console.log(cells[1])
```

2. 使用 for 循环遍历

```
for (let i = 0; i < cells.length; i++) {  
    console.log(cells[i])  
}
```

3. 游戏判赢

3.2 封装判赢函数

封装函数，主要考虑：参数和返回值。

该函数的返回值是什么？ 布尔值（判断是否获胜）

该函数的有参数吗？是什么？ 当前玩家

说明：判赢，就是在判断当前玩家下棋后是否获胜（玩家没下棋，不可能获胜，不需要判断）。

```
// 声明函数：  
function checkWin(player: Player): boolean {}  
// 调用函数：  
let isWin = checkWin(currentPlayer)
```

技巧：如果想不到返回值和参数，可以反推，也就是从如何调用函数的角度来分析。

问题：什么时候判赢？ 玩家点击单元格下棋后

■ 3. 游戏判赢

3.2 封装判赢函数

1. 声明函数 (`checkWin`)，指定参数 (`player`)，类型注解为：`Player` 枚举。
2. 指定返回值：现在函数中写死返回 `true` 或 `false`。
3. 在给单元格添加类名后（下棋后），调用函数 `checkWin`，拿到函数返回值。
4. 判断函数返回值是否为 `true`，如果是，说明当前玩家获胜了。

■ 3. 游戏判赢

3.3 实现判赢函数

思路：遍历判赢数组，分别判断每种情况对应的 3 个单元格元素，是否同时包含当前玩家的类名。

问题：使用哪种方式遍历数组呢？

只要有一种情况满足，就表示玩家获胜，后续的情况就没有必要再遍历，因此，数组遍历时可以终止。

判赢函数的返回值是布尔类型，如果玩家获胜（有一种情况满足），就返回 `true`；否则，返回 `false`。

数组的 `some` 方法：1 遍历数组时可终止 2 方法返回值为 `true` 或 `false`。

■ 3. 游戏判赢

3.3 实现判赢函数

思路：遍历判赢数组，分别判断每种情况对应的 3 个单元格元素，是否同时包含当前玩家的类名。

1. 使用 `some` 方法遍历数组，并将 `some` 方法的返回值作为判赢函数的返回结果。
2. 在 `some` 方法的回调函数中，获取到每种获胜情况对应的 3 个单元格元素。
3. 判断这 3 个单元格元素是否同时包含当前玩家的类名。
4. 如果包含（玩家获胜），就在回调函数中返回 `true` 停止循环；否则，返回 `false`，继续下一次循环。

■ 3. 游戏判赢

3.4 优化判赢函数

1. 去掉判赢函数的中间变量（isWin、cell11、cell12、cell13）。
2. 封装函数（hasClass）：判断 DOM 元素是否包含某个类名。

■ 3. 游戏判赢

3.5 判断平局

思路：创建变量（`steps`），记录已下棋的次数，判断 `steps` 是否等于 9，如果等于说明平局。

注意：先判赢，再判断平局！

1. 创建变量（`steps`），默认值为 0。
2. 在玩家下棋后，让 `steps` 加 1。
3. 在判赢的代码后面，判断 `steps` 是否等于 9。
4. 如果等于 9 说明是平局，游戏结束，就直接 `return`，不再执行后续代码。

■ 3. 游戏判赢

3.6 展示获胜信息

效果：在获胜或平局时，展示相应信息。

1. 获取到与获胜信息相关的两个 DOM 元素：1 #message 2 #winner。
2. 显示获胜信息面板（通过 style 属性实现）。
3. 展示获胜信息：如果获胜，展示“x 赢了！”或“o 赢了！”；如果是平局，展示“平局”。

目录

Contents

- ◆ 游戏准备
- ◆ 点击下棋
- ◆ 游戏判赢
- ◆ 重新游戏

■ 4. 重新游戏

效果：点击重新开始按钮，重新开始下棋游戏。

说明：重新开始游戏，实际上就是要重置游戏中的所有数据，恢复到初始状态。

比如：隐藏获胜信息、重置下棋次数、清空棋盘等等。

1. 获取到重新开始按钮（`#restart`），并绑定点击事件。
2. 在点击事件中，重置游戏数据。
3. 隐藏获胜信息、清空棋盘、移除单元格点击事件、重新给单元格绑定点击事件。
4. 重置下棋次数、重置默认玩家为 `x`、重置下棋提示为 `x`。

■ 4. 重新游戏

优化重新游戏功能：

原来，下棋分为：1 第一次游戏 2 重新开始游戏。

现在，将第一次游戏，也看做是“重新开始游戏”，就可以去掉第一次游戏时重复的初始化操作了。

1. 将重新开始按钮的事件处理程序修改为：函数声明形式（`startGame`）。
2. 直接调用函数（`startGame`），来开始游戏。
3. 移除变量 `steps`、`currentPlayer` 的默认值，并添加明确的类型注解。
4. 移除给单元格绑定事件的代码。



总结

下棋游戏 (XXOO)

1. 使用学到的 **TS**、Web 开发知识，从零开始完成了下棋游戏。
2. TS 知识：变量声明、枚举、类型断言、函数（参数、返回值）等。
3. 枚举：是一组有名字的常量的集合，用来组织有关联的数据。
4. 类型断言：在我们比 TS 更明确变量的类型时，来指定具体类型。
5. 函数封装：主要考虑参数和返回值，也就是接收要处理的数据，返回处理后的结果。
6. 实现功能：分步骤实现，完成一步，验证一步（先实现，再优化）。
7. DOM 操作：获取元素、添加移除事件、事件对象、样式操作、文本内容。
8. 伪数组：具有长度（length）属性和索引（长得像、操作也像数组）。



传智播客旗下高端IT教育品牌