JavaScript 核心第七天

常用方法&递归&异常处理

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

学习目标

Learning Objectives

1. 掌握对象、数组、字符串、数字、日期类型的常见属性和方法，便捷完成功能
2. 理解递归，可以使用递归来处理数据
3. 知道在JS中如何处理异常

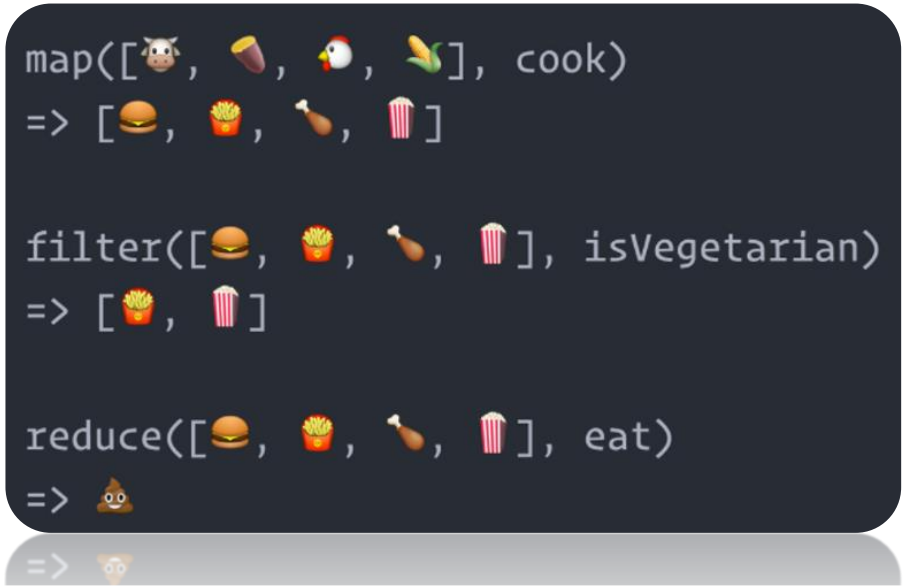
目录

Contents

常用方法

- 数组其余常用方法
- 对象
- 字符串
- 数值
- 日期

数组常用方法



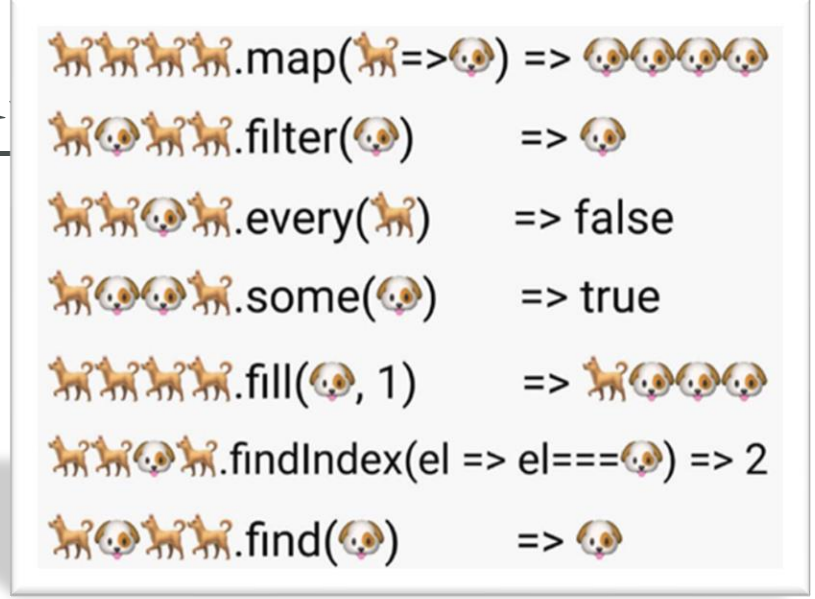
方法	作用	说明
filter	过滤数组	返回新数组，返回的是筛选满足条件的数组元素
map	迭代数组	返回新数组，返回的是处理之后的数组元素，想要使用返回的新数组
reduce	累计器	返回累计处理的结果，经常用于求和等
forEach	遍历数组	不返回数组，经常用于查找遍历数组元素

练习

filter 练习

需求：查找到所有的 vue 视频并展示到列表中

数组常见方法-汇总



5. 实例方法 `join` 数组元素拼接为字符串，返回字符串(重点)
6. 实例方法 `find` 查找元素， 返回符合测试条件的第一个数组元素值，如果没有符合条件的则返回 undefined(重点)
7. 实例方法 `every` 检测数组所有元素是否都符合指定条件，如果**所有元素**都通过检测返回 true，否则返回 false(重点)
8. 实例方法 `some` 检测数组中的元素是否满足指定条件 **如果数组中有**元素满足条件返回 true，否则返回 false
9. 实例方法 `concat` 合并两个数组，返回生成新数组
10. 实例方法 `sort` 对原数组单元值排序
11. 实例方法 `splice` 删除或替换原数组单元
12. 实例方法 `reverse` 反转数组
13. 实例方法 `findIndex` 查找元素的索引值

练习

find、some、every 练习

需求：

①：查找最先超过5K学习人数的课程

②：判断 是否都是高级内容

③：是否有超过5万学习人数的内容

常用方法

- 数组其余常用方法
- 对象
- 字符串
- 数值
- 日期

对象常用方法

思考：

```
// 想要获得对象里面的属性和值怎么做的？
const o = { name: '佩奇', age: 6 }
```

```
for (let k in o) {
  console.log(k) // 属性  name  age
  console.log(o[k]) // 值  佩奇  6
}
```

对象常用方法

- **作用：** Object.keys() 方法获取对象中所有属性（键）
- **语法：**

```
const o = { name: '佩奇', age: 6 }
// 获得对象的所有键，并且返回是一个数组
const arr = Object.keys(o)
console.log(arr) // ['name', 'age']
```

- **注意：** 返回的是一个数组

对象常用方法

- **作用：** Object.values() 方法获取对象中所有属性值
- **语法：**

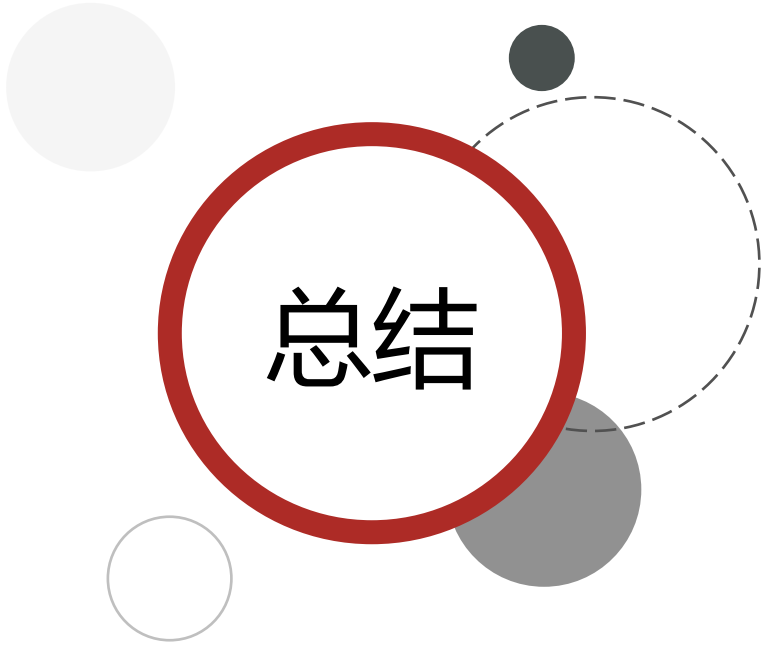
```
const o = { name: '佩奇', age: 6 }
// 获得对象的所有值，并且返回是一个数组
const arr = Object.values(o)
console.log(arr) // ['佩奇', 6]
```

- **注意：** 返回的是一个数组

对象常用方法

- **作用：** Object. assign() 方法常用于对象拷贝
- **语法：**

```
// 拷贝对象 把 o 拷贝给 obj
const o = { name: '佩奇', age: 6 }
const obj = {}
Object.assign(obj, o)
console.log(obj) // {name: '佩奇', age: 6}
```



1. Object.keys()方法的作用是什么？
   - 获取对象中所有属性（键）
   - 返回的是数组
2. Object.values()方法的作用是什么？
   - 获取对象中所有属性值（值）
   - 返回的是数组

练习 请完成以下需求

const spec = { size: '40cm*40cm' , color: '黑色'}

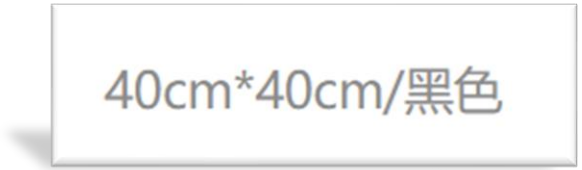
请将size和color里面的值拼接为字符串之后，写到div标签里面，展示如下：

40cm*40cm/黑色

练习 请完成以下需求

const spec = { size: '40cm*40cm' , color: '黑色'}

请将size和color里面的值拼接为字符串之后，写到div标签里面，展示如下：



思路： 获得所有的属性值，然后拼接字符串就可以了

①： 获得所有属性值是： Object.values() 返回的是数组

②： 拼接数组是 join('') 这样就可以转换为字符串了

```
const spec = { size: '40cm*40cm', color: '黑色' }
// console.log(Object.values(spec))
document.querySelector('div').innerHTML = Object.values(spec).join('/')
// 第一种方法更简单，万一对象里面有多个属性，第二种方法就不方便了
// document.querySelector('div').innerHTML = spec.size + '/' + spec.color
```

常用方法

- 数组其余常用方法
- 对象
- 字符串
- 数值
- 日期

字符串常用方法

1. 实例属性 `length` 用来获取字符串的度长(重点)
2. 实例方法 `split('分隔符')` 用来将字符串拆分成数组(重点)
3. 实例方法 `substring (需要截取的第一个字符的索引[,结束的索引号])` 用于字符串截取(重点)
4. 实例方法 `startsWith(检测字符串[, 检测位置索引号])` 检测是否以某字符开头(重点)
5. 实例方法 `includes(搜索的字符串[, 检测位置索引号])` 判断一个字符串是否包含在另一个字符串中，根据情况返回 true 或 false(重点)
6. 实例方法 `trim()` 字符串的两端清除空格，返回一个新的字符串，而不修改原始字符串
7. 实例方法 `toUpperCase` 用于将字母转换成大写
8. 实例方法 `toLowerCase` 用于将就转换成小写
9. 实例方法 `indexOf` 检测是否包含某字符
10. 实例方法 `endsWith` 检测是否以某字符结尾
11. 实例方法 `replace` 用于替换字符串，支持正则匹配
12. 实例方法 `match` 用于查找字符串，支持正则匹配

练习

- 字符串翻转

需求：请把’ 传智教育 ‘ 四个字翻转为 ‘育教智传'

实现步骤

1. 把字符串转换为数组　　split('分隔符')
2. 数组翻转　　reverse()
3. 把数组转换为字符串　　join('分隔符')

练习 显示赠品练习

请将下面字符串渲染到准备好的 p标签内部，结构必须如左下图所示，展示效果如右图所示：

const gift = '50g茶叶,清洗球'

```
<p class="name"> == $0
  <span class="tag">【赠品】50g茶叶</span>
  <span class="tag">【赠品】清洗球</span>
</p>
```

【赠品】50g茶叶
【赠品】清洗球

思路：

①：把字符串拆分为数组，这样两个赠品就拆分开了 用哪个方法？ split(',')

②：渲染页面： map + join 方法

练习 · 显示赠品练习

请将下面字符串渲染到准备好的 p标签内部，结构必须如左下图所示，展示效果如右图所示：

【赠品】50g茶叶
【赠品】清洗球

```
// 答案
const p = document.querySelector('.name')
const gift = '50g茶叶,清洗球'
p.innerHTML = gift.split(',').map(item => `<span class='tag'>【赠品】${item}</span> <br>`).join('')
```

字符串常用方法

1. 实例属性 `length` 用来获取字符串的度长(重点)
2. 实例方法 `split('分隔符')` 用来将字符串拆分成数组(重点)
3. 实例方法 `substring (需要截取的第一个字符的索引[,结束的索引号])` 用于字符串截取(重点)
4. 实例方法 `startsWith(检测字符串[, 检测位置索引号])` 检测是否以某字符开头(重点)
5. 实例方法 `includes(搜索的字符串[, 检测位置索引号])` 判断一个字符串是否包含在另一个字符串中，根据情况返回 true 或 false(重点)
6. 实例方法 `trim()` 字符串的两端清除空格，返回一个新的字符串，而不修改原始字符串
7. 实例方法 `toUpperCase` 用于将字母转换成大写
8. 实例方法 `toLowerCase` 用于将就转换成小写
9. 实例方法 `indexOf` 检测是否包含某字符
10. 实例方法 `endsWith` 检测是否以某字符结尾
11. 实例方法 `replace` 用于替换字符串，支持正则匹配
12. 实例方法 `match` 用于查找字符串，支持正则匹配

常用方法

- 数组其余常用方法
- 对象
- 字符串
- 数值
- 日期

数值

常用方法：

toFixed(保留位数长度) 设置保留小数位的长度

```
// 数字 toFixed 方法
const num = 12.345
console.log(num.toFixed(2))  // 12.35
console.log(num.toFixed(1))  // 12.3
const num1 = 12
console.log(num1.toFixed(2))  // 12.00
```

常用方法

- 数组其余常用方法
- 对象
- 字符串
- 数值
- 日期

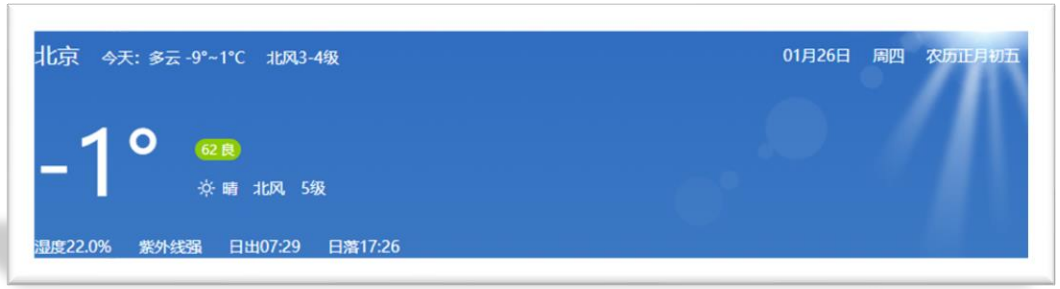
日期对象 Date

- 日期对象：用来表示日期和时间的对象
- 作用：可以得到当前系统日期和时间
- 获得当前日期

```
const date = new Date()
```

- 获得指定日期

```
const date = new Date('2099-9-9')
```



日期对象方法 - 格式化日期对象

使用场景：因为日期对象返回的数据我们不能直接使用，所以需要转换为实际开发中常用的格式（格式化日期对象）

方法	作用	说明
getFullYear()	获得年份	获取四位年份
getMonth()	获得月份	取值为 0 ~ 11
getDate()	获取月份中的每一天	不同月份取值也不相同
getDay()	获取星期	取值为 0 ~ 6
getHours()	获取小时	取值为 0 ~ 23
getMinutes()	获取分钟	取值为 0 ~ 59
getSeconds()	获取秒	取值为 0 ~ 59

总结

1. 如何获取日期对象
   - new Date()
2. 日期对象方法里面月份和星期有什么注意的?
   - 月份是0~11, 星期是 0~6

方法	作用	说明
getFullYear()	获得年份	获取四位年份
getMonth()	获得月份	取值为 0 ~ 11
getDate()	获取月份中的每一天	不同月份取值也不相同
getDay()	获取星期	取值为 0 ~ 6
getHours()	获取小时	取值为 0 ~ 23
getMinutes()	获取分钟	取值为 0 ~ 59
getSeconds()	获取秒	取值为 0 ~ 59

日期对象方法

格式化日期对象另外一种方法

方法	作用	说明
toLocaleString()	返回该日期对象的字符串（包含日期和时间）	2099/9/20 18:30:43
toLocaleDateString()	返回日期对象日期部分的字符串	2099/9/20
toLocaleTimeString()	返回日期对象时间部分的字符串	18:30:43

案例 日期对象案例

需求：把学成在线中每一门课程的发布时间 展示到页面上

核心：在渲染函数中，解构出 time 数据，处理成需要的年月日格式展示到页面上

目录

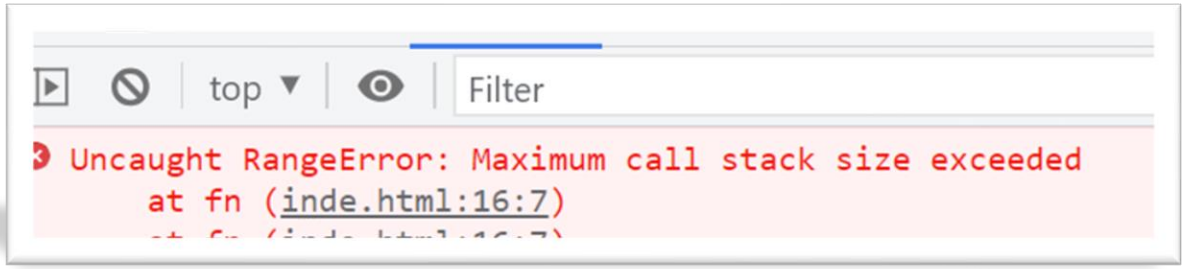
Contents

递归

所谓递归就是一种函数调用自身的操作

- 简单理解:函数内部自己调用自己, 就是递归，这个函数就是递归函数
- 递归函数的作用和循环效果类似
- 由于递归很容易发生“栈溢出”错误（stack overflow），所以记得添加退出条件

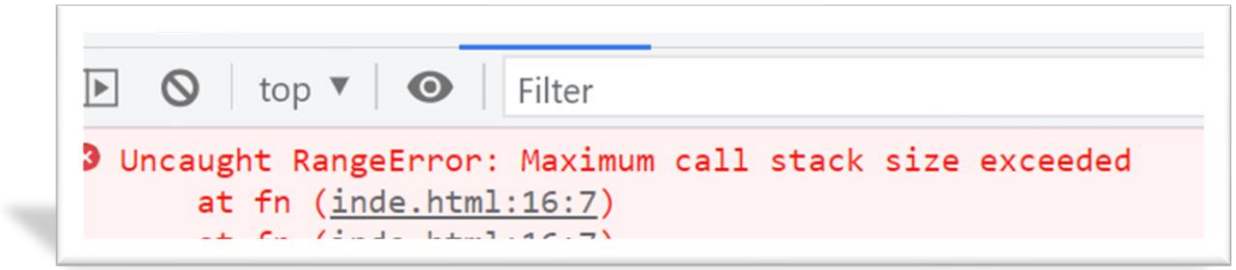
```
function fn() {
  // 业务逻辑
  fn()
}
```



总结

1. 什么是递归？有什么注意事项？

- 函数内部自己调用自己，就是递归，这个函数就是递归函数
- 递归很容易发生“栈溢出”错误（stack overflow），所以记得添加退出条件 return



案例

递归-扁平化处理数据

需求：通过递归将树形结构数据转换为扁平化的员工列表，并展示到页面中

目录

Contents

异常处理

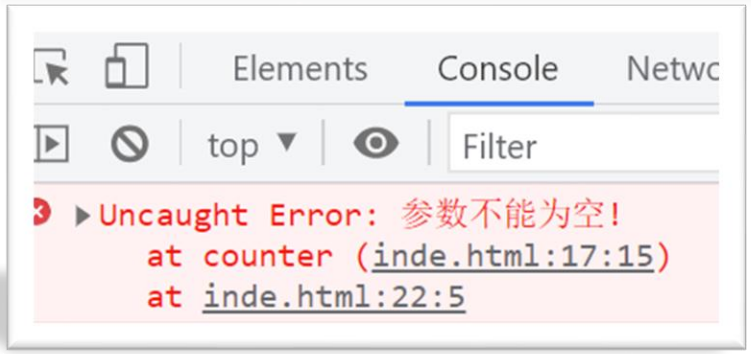
- throw 抛异常
- try /catch 捕获异常

了解 JavaScript 中程序异常处理的方法，提升代码运行的健壮性。

throw 抛异常

异常处理是指预估代码执行过程中可能发生的错误，然后最大程度的避免错误的发生导致整个程序无法继续运行

```
function counter(x, y) {
  if (!x || !y) {
    // throw '参数不能为空!';
    throw new Error('参数不能为空!')
  }
  return x + y
}
counter()
```



总结：

1. throw 抛出异常信息，程序也会终止执行
2. throw 后面跟的是错误提示信息
3. Error 对象配合 throw 使用，能够设置更详细的错误信息

总结

1. 抛出异常我们用哪个关键字？它会终止程序吗？
   - throw 关键字
   - 会中止程序
2. 抛出异常经常和谁配合使用？
   - Error 对象配合 throw 使用

```
function counter(x, y) {
  if (!x || !y) {
    // throw '参数不能为空!';
    throw new Error('参数不能为空!')
  }
  return x + y
}
counter()
```

异常处理

- throw 抛异常
- try /catch 捕获异常

了解 JavaScript 中程序异常处理的方法，提升代码运行的健壮性。

try/catch 捕获错误信息

我们想要测试某些代码是否有异常，可以通过try / catch 捕获错误信息（浏览器提供的错误信息）

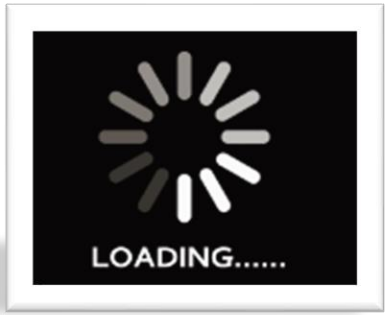
try 试试　catch 拦住　finally　最后

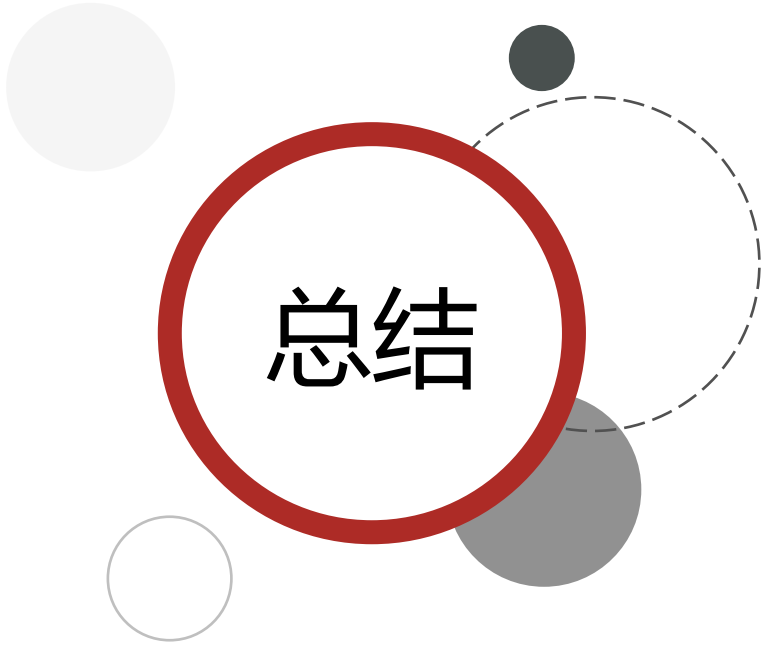
```
try {
  // 可能出现错误的代码
  const p = document.querySelector('.p')
  p.style.color = 'red'
} catch (error) {
  // try出现错误会进入 catch并捕获异常  error 错误参数
  console.log(error) // 错误信息

} finally {
  // 不管有么有错误都会进入 finally
  console.log('不管有没有错误都会执行')
}
```

说明：

1. 将预估可能发生错误的代码写在 try 代码段中
2. 如果 try 代码段中出现错误后，会执行 catch 代码段，并截获到错误信息
3. finally 不管是否有错误，都会执行





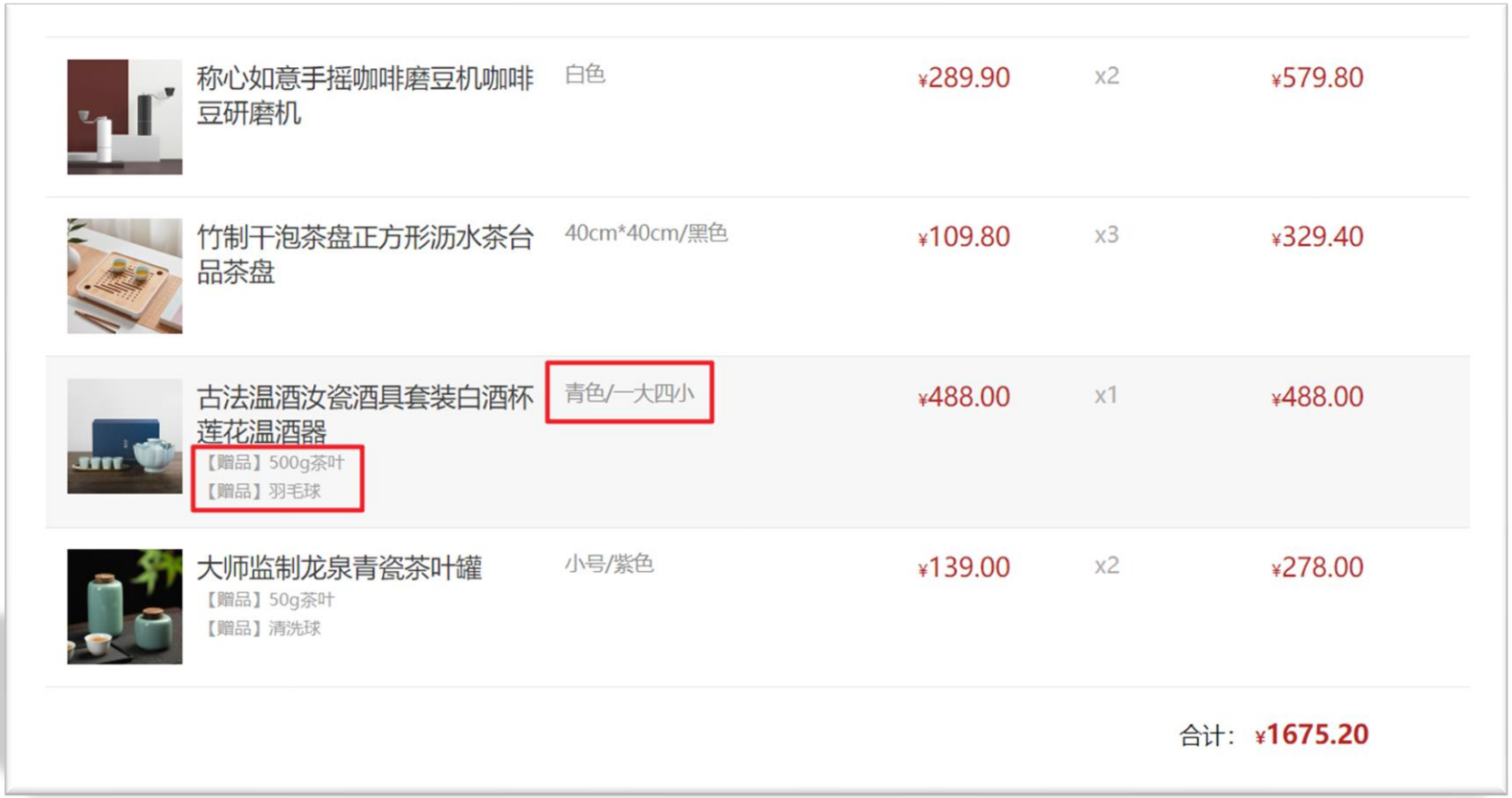
1. 捕获异常我们用那3个关键字？可能会出现的错误代码写到谁里面
   - try / catch / finally
   - try
2. 怎么调用错误信息？
   - 利用 catch 的参数， 比如 error
3. finally 什么时候执行呢？
   - 不管有没有错误都会执行
   - 比如页面加载的loading动画，不管页面加载有没有成功，时间到了都会消失

目录

Contents

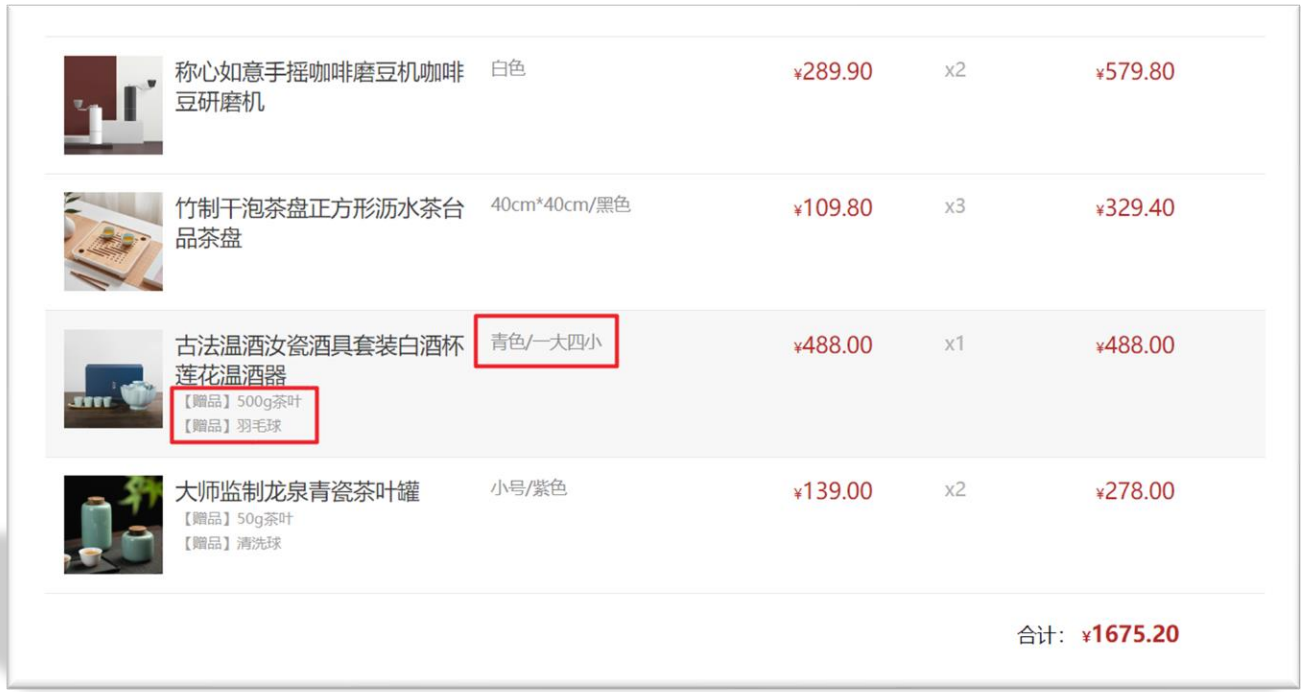
案例 购物车展示

需求：根据后台提供的数据，渲染购物车页面



案例 购物车展示

分析业务模块：



- 渲染业务（已完成）
- 数据处理业务
- 总价业务（已完成）

步骤 购物车展示

数据处理业务

②：数据处理业务

1. 处理 规格文字 模块

- 获取 每个对象里面的 spec，对象解构添加 spec
- 获得所有属性值是： Object.values() 返回的是数组
- 数组转换为字符串 join('/') 把字符串数据放入对应盒子里面即可


```
{
  id: '4001649',
  name: '大师监制龙泉青瓷茶叶罐',
  price: 139,
  picture: 'https://yanxuan-item.nosdn.127.net/4356c9
  count: 1,
  spec: { size: '小号', color: '紫色' },
  gift: '50g茶叶,清洗球'
}
```

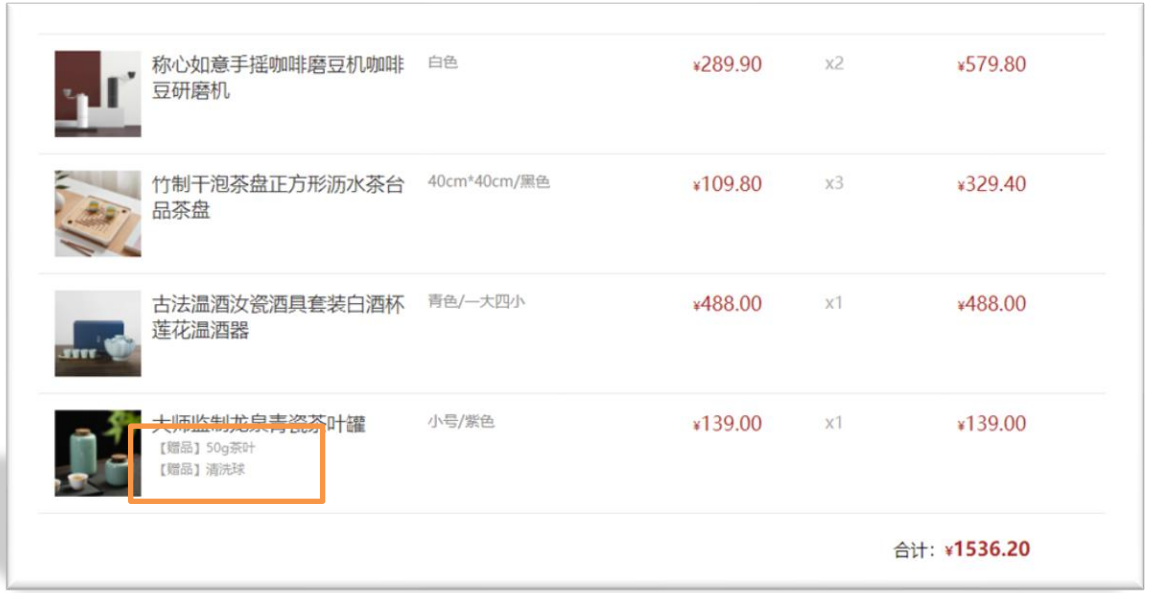
步骤 购物车展示

数据处理业务

②：数据处理业务

2. 处理 赠品 模块

- 获取 每个对象里面的 gift，对象解构添加 gift
- 把字符串转换为数组 split(',')，然后利用 map + join 生成对应标签， 放入对应盒子中
- 注意要判断是否有 gift 属性，没有的话不需要 map + join 生成标签




```
{
  id: '4001649',
  name: '大师监制龙泉青瓷茶叶罐',
  price: 139,
  picture: 'https://yanxuan-item.nosdn.127.net/435
  count: 1,
  spec: { size: '小号', color: '紫色' },
  gift: '50g茶叶,清洗球'
}
```

步骤 购物车展示

数据处理业务

②：数据处理业务

3. 处理 小计 模块

- 小计 = 单价 * 数量
- 小计名可以为: subTotal = price * count
- 注意保留2位小数

关于小数的计算精度问题:

0.1 + 0.2 = ?

解决方案： 我们经常转换为整数

(0.1*10 + 0.2*10) / 10 === 0.3

这里是给大家拓展思路和处理方案

传智教育旗下高端IT教育品牌