# 2021.12.21(银行应用程序笔记)

## 功能

### 显示账户信息功能

/\*\* 显示账户信息

 \* 判断账户mov类型, 大于0代表存款, 小于0代表借款

 \*/

const displayMovements = function (movements) {

  containerMovements.innerHTML = '';

  movements.forEach((mov, i) => {

    const type = mov > 0 ? 'deposit' : 'withdrawal';

    const html = `

    <div class="movements\_\_row">

      <div class="movements\_\_type movements\_\_type--${type}">${

      i + 1

    } ${type}</div>

      <div class="movements\_\_value">${mov}€</div>

    </div>

  `;

    containerMovements.insertAdjacentHTML('afterbegin', html);

  });

};

displayMovements(account1.movements);

### 格式化账户名称功能

/\*\* 格式化账户名称

 \* accs === accounts (所有账户)

 \* 取姓名首字母小写 (jack chen => jc)

 \* 将格式后的姓名存到username新属性

 \*/

const createUsernames = function (accs) {

  accs.forEach(

    acc =>

      (acc.username = acc.owner

        .toLowerCase()

        .split(' ')

        .map(name => name[0])

        .join(''))

  );

};

createUsernames(accounts);

### 显示账户余额功能

/\*\* 显示账户余额

 \* 通过reduce方法计算账户余额

 \*/

const calcDisplayBalance = function (acc) {

  acc.balance = acc.movements.reduce((acc, mov) => acc + mov, 0);

  labelBalance.textContent = `${acc.balance}€`;

};

### 显示总汇率功能 --非常重要

/\*\* 显示总汇值

 \* 账户正数为收入, 显示总数

 \* 账户负数为支出, 显示总数

 \* 账户利息 = (单笔存款 \* 银行利率) / 100

 \* 过滤掉不足一元的利息

 \*/

  // 收入

  const incomes = acc.movements

    .filter(mov => mov > 0)

    .reduce((acc, mov) => acc + mov, 0);

  labelSumIn.textContent = `${incomes}€`;

  // 支出

  const out = acc.movements

    .filter(mov => mov < 0)

    .reduce((acc, mov) => acc + mov, 0);

  labelSumOut.textContent = `${Math.abs(out)}€`;

  // 利息

  const interest = acc.movements

    .filter(mov => mov > 0)

    .map(deposit => (deposit \* acc.interestRate) / 100)

    .filter((int, i, arr) => {

      // console.log(arr);

      return int >= 1;

    })

    .reduce((acc, int) => acc + int, 0);

  labelSumInterest.textContent = `${interest}€`;

};

#### Reduce()方法可以同时获取收入和支出总额 --非常重要

/\*\*

 \* { incomes, out } 结构出对象属性

 \* nums => {incomes: 0, out: 0} | cur => 当前movments元素

 \* 当nums是单个值时, 我们只需进行相应的逻辑计算, 当nums是对象时, 每次执行计算结果便返回对象, 这样不同属性的值便存储到对象中, += 运算符执行属性内值的累加, 最终我们得到收入和支出的总额 ((重点))

 \*/

const { incomes, out } = account1.movements.reduce(

  (nums, cur) => {

    nums[cur > 0 ? 'incomes' : 'out'] += cur;

    console.log(nums);

    return nums;

  },

  {

    incomes: 0,

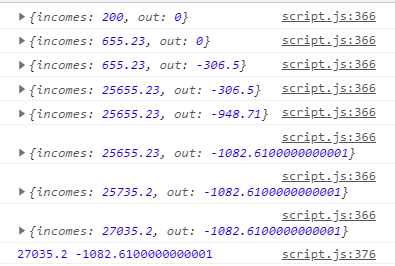
    out: 0,

  }

);

console.log(incomes, out);

#### nums对象与最终结果图



### 格式化日期功能 -- 重点

/\*\* 格式化国际日期

 \* date => 存贷款日期 | locale => 语种

 \* 日期进行计算时返回时间戳, 时间戳精确到毫秒 (重点)

 \* date2 - date1 => 时间戳 / 1000(毫秒) 60(秒) 60(分钟) 24(小时) => 已过的天数 (重点)

 \* 判断已过天数返回指定日期显示类型, 添加超过七天的返回国际日期

 \* Intl(国际).DateTimeFormat(语种).format(需要格式的日期)

 \*/

const formatMovementDate = function (date, locale) {

  const calcDaysPassed = (date1, date2) =>

    Math.round(Math.abs(date2 - date1) / (1000 \* 60 \* 60 \* 24));

  const dayPassed = calcDaysPassed(new Date(), date);

  if (dayPassed === 0) return 'Today';

  if (dayPassed === 1) return 'Yesterday';

  if (dayPassed < 7) return `${dayPassed} days ago`;

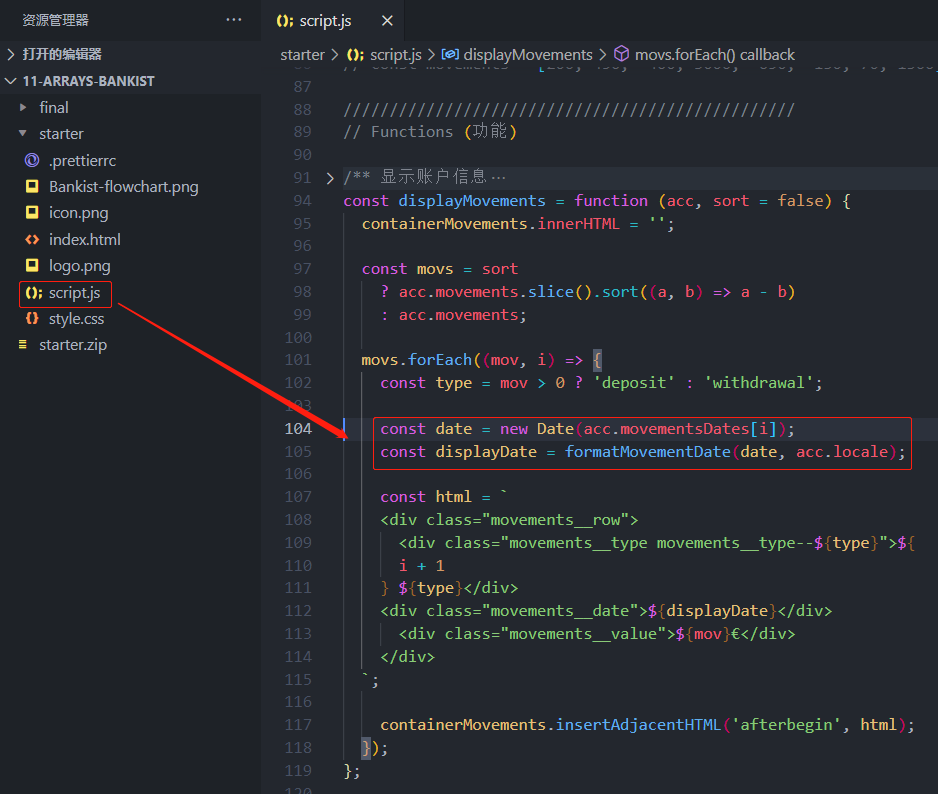
  // 返回国际化日期格式

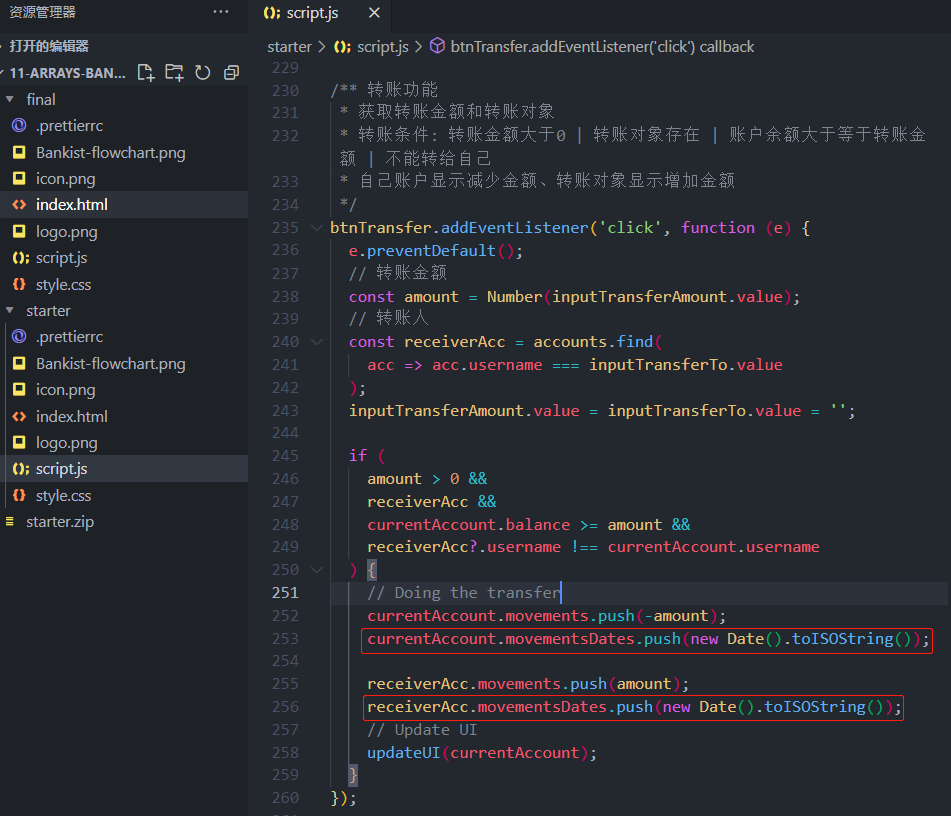
  return new Intl.DateTimeFormat(locale).format(date);

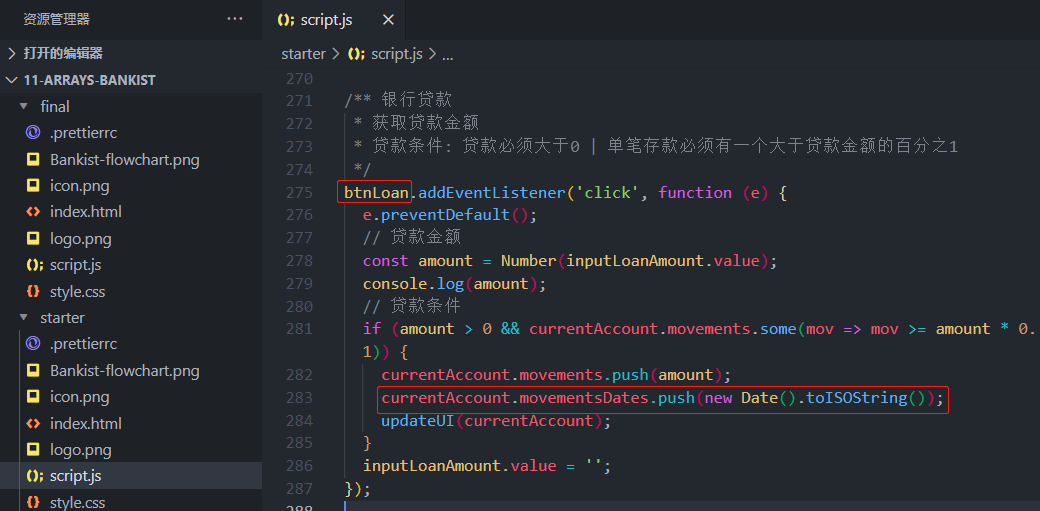
};

#### 添加日期到显示账户信息、账户总汇率、账户余额功能中

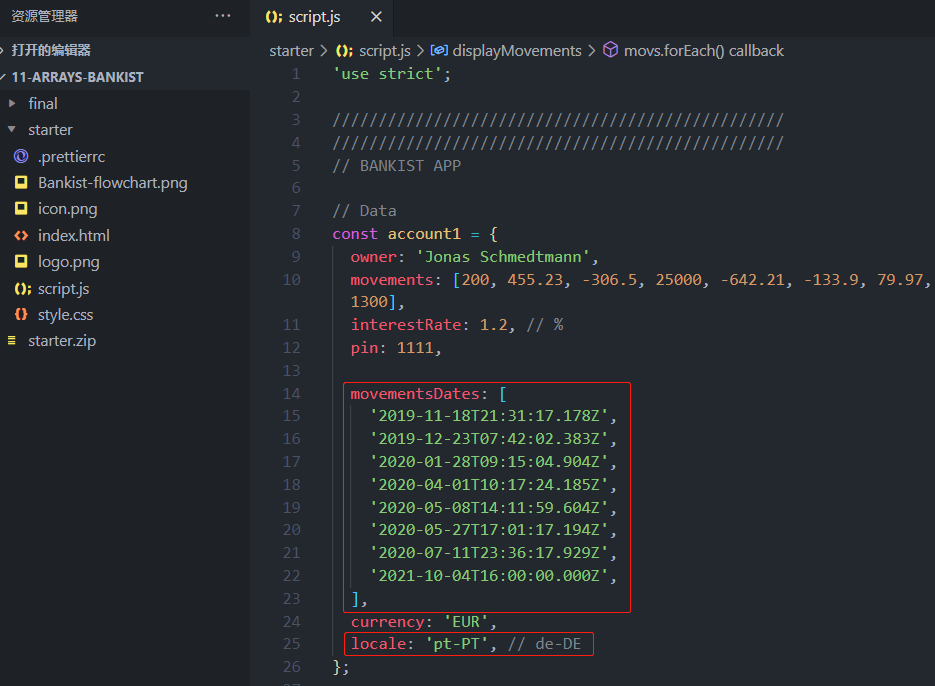
注意new Date() 是object, 我们将时间push到本地账户数组中需要转换为字符串使用toISOString()方法 (重点)







#### 本地账户数据添加相关日期



### 格式化数字功能 -- 重点

/\*\* 格式化数字

 \* value => 格式化数字值 | locale => 地点 | currency 货币

 \*/

const formatCur = function (value, locale, currency) {

  return new Intl.NumberFormat(locale, {

    style: 'currency',

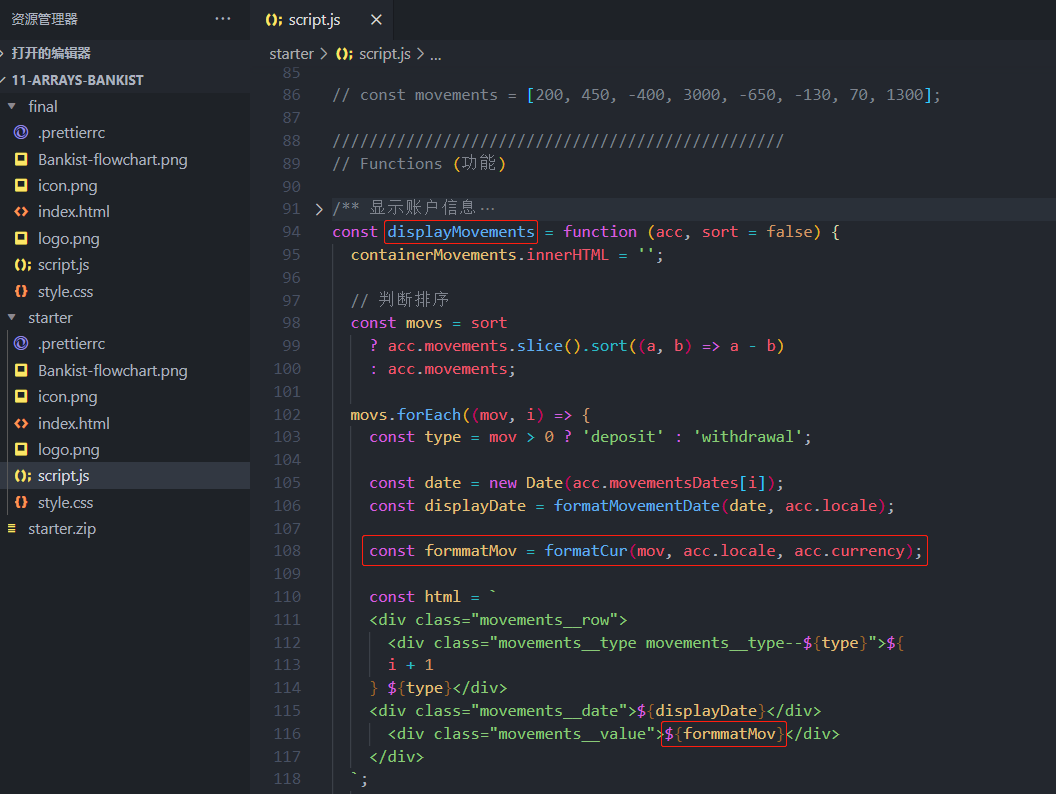
    currency: currency,

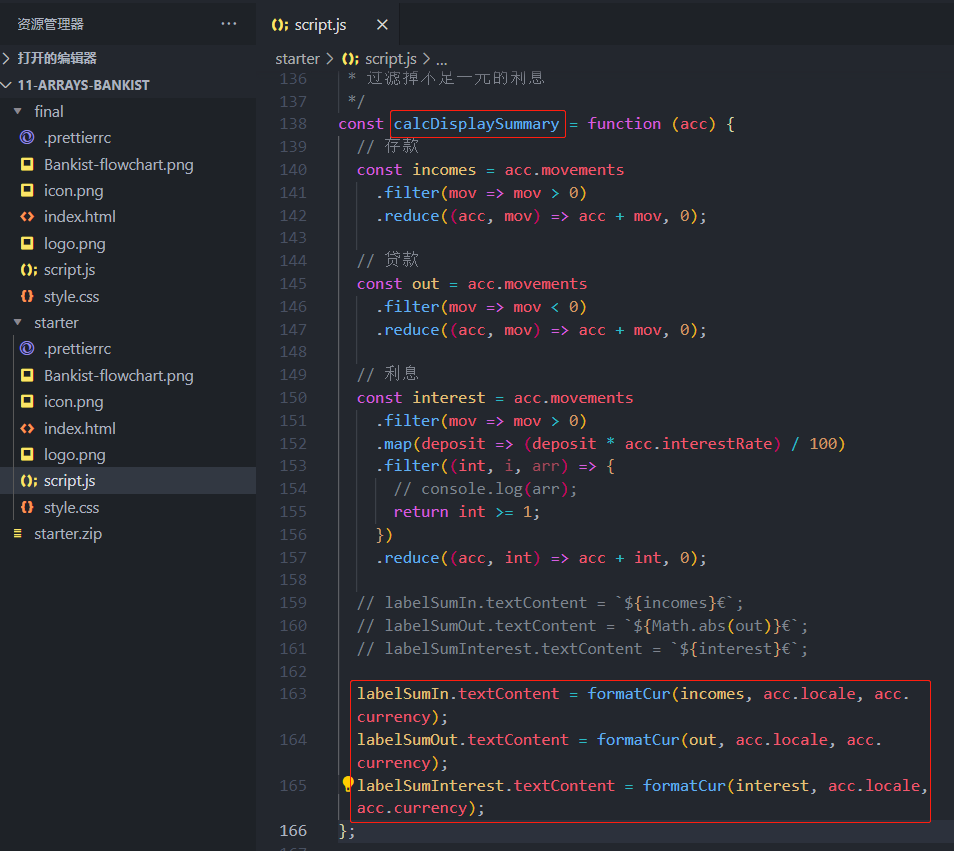
  }).format(value);

};

#### 添加格式化数字到显示账户信息、余额、总汇率、贷款、转账

注意国际化数字会自动保存小数点后两位执行四舍五入, 相当于使用toFixed()方法







### 注销计时器功能 --重点

/\*\* 启动注销计时器

 \* tick => 计时器标记 | time => 倒计时时间 | timer => 返回计时器(添加保护协议, 如果存在先删除再执行)

 \* Math.trunc()删除任何小数位返回数字整数部分

 \* 倒计时形式以 分钟 : 秒显示

 \* 倒计时为0时, 退出登录(隐藏界面)

 \*/

let timer;

const startLogOutTimer = function () {

  const tick = function () {

    const min = String(Math.trunc(time / 60)).padStart(2, 0);

    const sec = String(time % 60).padStart(2, 0);

    labelTimer.textContent = `${min}:${sec}`;

    if (time === 0) {

      clearTimeout(tick);

      containerApp.style.opacity = 0;

      labelWelcome.textContent = 'Log in to get started';

    }

    time--;

  };

  let time = 60; // 倒计时间

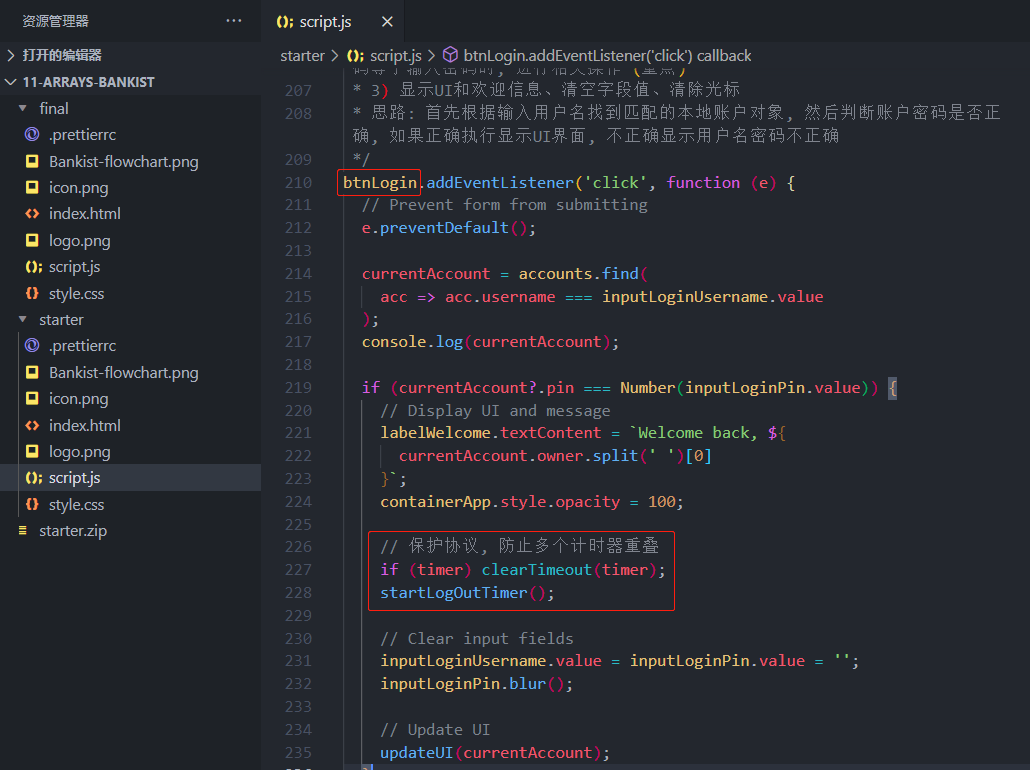
  tick();

  timer = setInterval(tick, 1000);

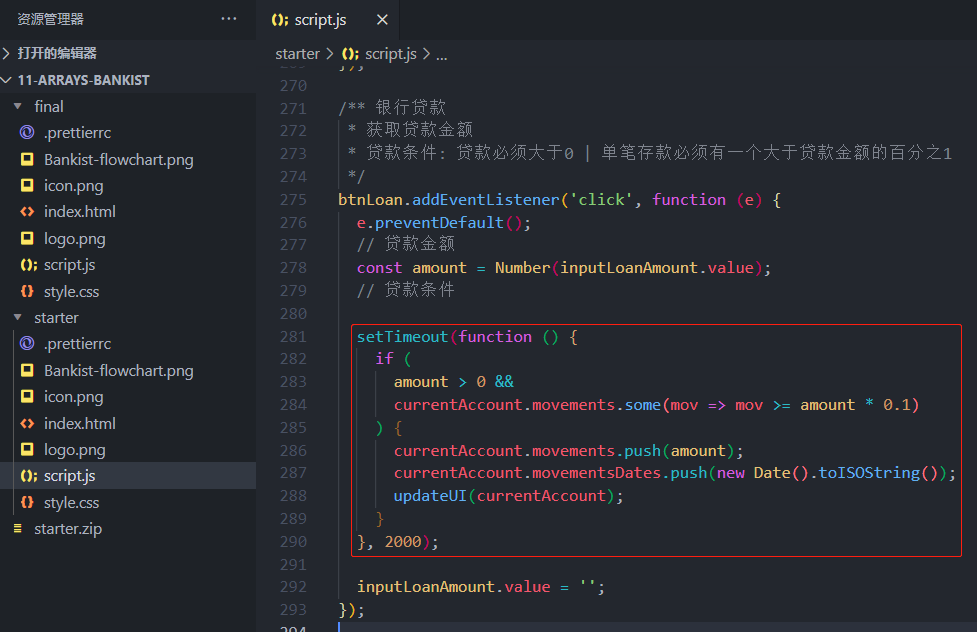
  return timer;

};

#### 添加计时器到btnLogin



#### 贷款功能可以添加计时器达到微延迟效果



### 显示UI界面

const updateUI = function (acc) {

  // Display movements

  displayMovements(acc.movements);

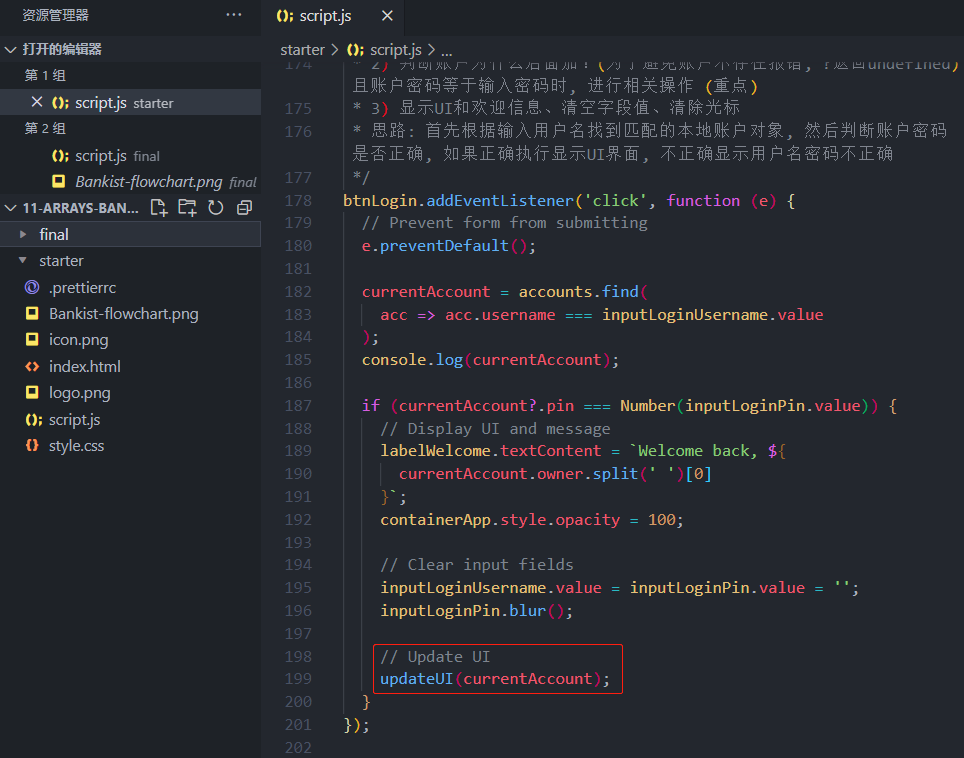
  // Display balance

  calcDisplayBalance(acc);

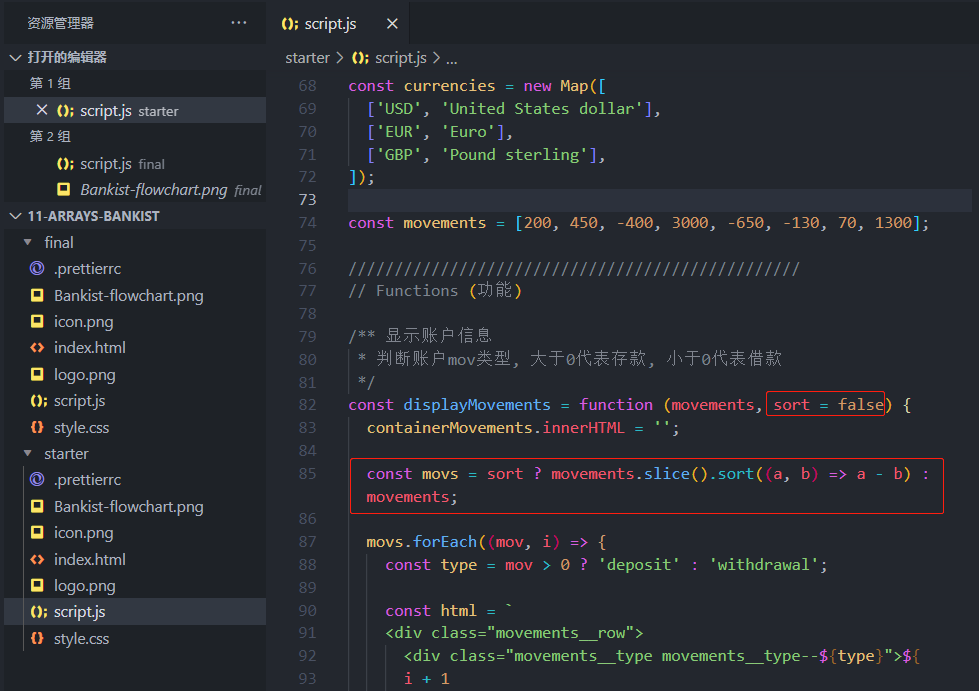
  // Display summary

  calcDisplaySummary(acc);

};



### 显示排序事件结果 (对应事件中的排序事件)



## 事件处理

### 登录账户

let currentAccount; // 当前账户 accounts 所有账户

/\*\* 用户登录

 \* 1) 首先根据表单输入用户名对比accounts全部账户用户名, 如果匹配返回匹配账户对象

 \* 2) 判断账户为什么后面加？(为了避免账户不存在报错, ?返回undefined)且账户密码等于输入密码时, 进行相关操作 (重点)

 \* 3) 显示UI和欢迎信息、显示账户信息、清空字段值、清除光标

 \* 思路: 首先根据输入用户名找到匹配的本地账户对象, 然后判断账户密码是否正确, 如果正确执行显示UI界面, 不正确显示用户名密码不正确

 \*/

btnLogin.addEventListener('click', function (e) {

  // Prevent form from submitting

  e.preventDefault();

  currentAccount = accounts.find(

    acc => acc.username === inputLoginUsername.value

  );

  console.log(currentAccount);

  if (currentAccount?.pin === Number(inputLoginPin.value)) {

    // Display UI and message

    labelWelcome.textContent = `Welcome back, ${

      currentAccount.owner.split(' ')[0]

    }`;

    containerApp.style.opacity = 100;

    // Clear input fields

    inputLoginUsername.value = inputLoginPin.value = '';

    inputLoginPin.blur();

    // Update UI

    displayMovements(currentAccount.movements);

  }

});

### 用户转账事件

/\*\* 转账功能

 \* 获取转账金额和转账对象

 \* 逻辑判断: 转账金额大于0 | 转账对象存在 | 账户余额大于等于转账金额 | 不能转给自己

 \* 自己账户显示减少金额、转账对象显示增加金额

 \*/

btnTransfer.addEventListener('click', function (e) {

  e.preventDefault();

  // 转账金额

  const amount = Number(inputTransferAmount.value);

  // 转账对象

  const receiverAcc = accounts.find(

    acc => acc.username === inputTransferTo.value

  );

  inputTransferAmount.value = inputTransferTo.value = '';

  if (

    amount > 0 &&

    receiverAcc &&

    currentAccount.balance >= amount &&

    receiverAcc?.username !== currentAccount.username

  ) {

    // Doing the transfer

    currentAccount.movements.push(-amount);

    receiverAcc.movements.push(amount);

    // Update UI

    updateUI(currentAccount);

  }

});

### 银行贷款事件

/\*\* 银行贷款

 \* 获取贷款金额

 \* 贷款条件: 贷款必须大于0 | 单笔存款必须有一个大于贷款金额的百分之1

 \*/

btnLoan.addEventListener('click', function (e) {

  e.preventDefault();

  // 贷款金额

  const amount = Number(inputLoanAmount.value);

  // 贷款条件

  if (amount > 0 && currentAccount.movements.some(mov => mov >= amount \* 0.1)) {

    // Add movement

    currentAccount.movements.push(amount);

    // Update UI

    updateUI(currentAccount);

  }

  inputLoanAmount.value = '';

});

### 注销用户事件

/\*\* 注销用户

 \* 获取用户名和密码

 \* 注销条件: 必须是当前用户名及密码

 \*/

 btnClose.addEventListener('click', function (e) {

  e.preventDefault();

  if (

    inputCloseUsername.value === currentAccount.username &&

    Number(inputClosePin.value) === currentAccount.pin

  ) {

    const index = accounts.findIndex(

      acc => acc.username === currentAccount.username

    );

    console.log(index);

    // .indexOf(23)

    // Delete account

    accounts.splice(index, 1);

    // Hide UI

    containerApp.style.opacity = 0;

  }

  inputCloseUsername.value = inputClosePin.value = '';

});

### 账户排序点击事件

/\*\* 账户信息排序

 \* 我们喜欢点击排序按钮显示排序, 再次点击显示之前的

 \* 设置一个布尔值变量sorted, 每次点击时将布尔值传递给变量sorted完成切换效果 (重点)

 \*/

let sorted = false;

btnSort.addEventListener('click', function (e) {

  e.preventDefault();

  displayMovements(currentAccount.movements, !sorted);

  sorted = !sorted;

});

### 计算所有账户余额

/\*\* 计算所有账户余额

 \* flat(1) 方法会按照一个可指定的深度递归遍历数组，并将所有元素与遍历到的子数组中的元素合并为一个新数组返回

 \* flatMap() 该flatMap()方法返回一个新数组，该数组通过将给定的回调函数应用于数组的每个元素，然后将结果展平一级而形成。它与深度为 1 的amap() 后跟 a相同flat()，但比分别调用这两个方法效率稍高

 \*/

const allMovements = accounts

  .flatMap(acc => acc.movements)

  .reduce((acc, mov) => acc + mov, 0);

console.log(allMovements); // 17840

## 格式化日期与数字 --非常重要

### 格式化日期知识点

#### 如何自定义时间日期

/\*\* 自定义当前时间日期(精确到秒)

 \* date.getMonth()得到的是数字类型, 因此我们使用padStart()方法前需要将其转换为字符串类型 (重点)

 \* `${date.getMonth() + 1}` 通过模板文字方式转换为字符串在

 \* 月份是从数字0开始计算

 \* new Date() => Tue Dec 21 2021 17:25:59 GMT+0800 (中国标准时间)

 \*/

const date = new Date();

const year = date.getFullYear();

const month = `${date.getMonth() + 1}`.padStart(2, 0); // 月份是从0开始计算

const day = `${date.getDate()}`.padStart(2, 0);

const hour = `${date.getHours()}`.padStart(2, 0);

const min = `${date.getMinutes()}`.padStart(2, 0);

console.log(`${year}/${month}/${day}, ${hour}:${min}`);



#### 如何格式化国际日期

const now = new Date();

const options = {

  hour: 'numeric', // 数字类型

  minute: 'numeric',

  day: '2-digit',

  month: '2-digit', // 两位数字

  year: 'numeric',

  // weekday: 'long'

};

// 语言环境 = 航海家.语言

const locale = navigator.language;

// 国际.日期时间格式('国家', 选项).格式(日期)

const now = new Intl.DateTimeFormat(locale, options).format(now);

console.log(now); // 2021/12/22上午10:37

#### JS如何将日期转换为时间戳



#### toString()与toISOString()

  const event = new Date("05 October 2011 0:00 UTC"); // 必须采用此格式

  console.log(event.toString()); // Wed Oct 05 2011 08:00:00 GMT+0800 (中国标准时间) 默认从八点开始计算

  console.log(event.toISOString()); // 2011-10-05T00:00:00.000Z

### 格式化数字知识点

#### 如何格式化国际数字

const num = 3884764.23;

const options = {

  style: 'currency', // 货币

  unit: 'celsius', // 摄氏

  currency: 'RMB', // 货币单位

  useGrouping: true, // 是否使用分组分隔符，例如千位分隔符或千/十万/千万分隔符 可能的值为真和假；默认为真

};

console.log(

  'US:           ',

  new Intl.NumberFormat('en-US', options).format(num)

); // US:        RMB 3884764.23

console.log(

  'de-DE:       ',

  new Intl.NumberFormat('de-DE', options).format(num)

); //de-DE:        3884764,23 RMB

console.log(

  'ar-SY:       ',

  new Intl.NumberFormat('ar-SY', options).format(num)

); // ar-SY:        ٣٨٨٤٧٦٤٫٢٣ RMB

