常用方法

ELEMENT_UI

Dropdown 下拉菜单

下拉事件绑定，@click 无法触发和获取参数

不采用原生指令事件，是因为要获取for循环中item，index；

```
<el-dropdown class="button">
  <span class="el-dropdown-link">
    操作按钮
  </span>
  <el-dropdown-menu slot="dropdown">
    <el-dropdown-item @click.native="view(o,index)"><div><i class="el-icon-view"></i>预览</div></el-dropdown-item>
    <el-dropdown-item @click.native="addImage(o,index)"><i class="el-icon-circle-plus"></i>新增</el-dropdown-item>
    <el-dropdown-item @click.native="updateImage(o,index)"><i class="el-icon-refresh"></i>更新</el-dropdown-item>
    <el-dropdown-item @click.native="deleteImg(o,index)"> <i class="el-icon-delete-solid"></i>删除</el-dropdown-item>
  </el-dropdown-menu>
</el-dropdown>
```

下拉框互斥事件

```
<fu-select v-model="scope.row[item.prop]" placeholder="请选择"
 filterable clearable @focus="mutex">
                    <fu-option
                      v-for="item in options[item.prop]"
                      :key="item.value"
                      :label="item.text"
                      :disabled="item.text !== scope.row[item.prop] && dsColumnName.includes(item.text)"
                      :value="item.value">
                    </fu-option>
</fu-select>

```

```
mutex(e){
      let now  = e.target.value;
      //表格数据判断 除当前数据外其他被西选中的值
      //符合条件的值 disabled = true
      this.dsColumnName.splice(0)
      this.dialogTable.data.rows.forEach((ele)=>{
        let name = ele.dsColumnName;
        if(name){
          this.dsColumnName.push(name)
        }
      })
    },
```

Table

自定义表头

```
<fu-table-column
        v-for="(item, colIndex) in column"
        :key="`col_${colIndex}`"
        :label="item.label"
        :prop="dropcolumn[colIndex].prop"
        :min-width="item.width"
        :render-header="resetHeader"
      >
      </fu-table-column>
/**
     * @description 自定义表头
     */
    resetHeader(h, { column, $index }) {
      return h("div", [
        h("span", column.label),
        h("i", {
          class: {
            "el-icon-sort-down": true,
            iconStyle: true,
            myStyle: this.insetStyle($index),
          },
          style: "color:#C0C4CC",

          on: {
            click: () => {
```

```
              //一定要用箭头函数
              this.switchClick($index, column, "desc");
            },
          },
        }),
        h("i", {
          class: {
            "el-icon-sort-up": true,
            iconStyle: true,
            myStyle: this.insetStyle2($index),
          },
          style: "color:#C0C4CC",
          on: {
            click: () => {
              //一定要用箭头函数
              this.switchClick($index, column, "asc");
            },
          },
        }),
      ]);
    },


// switchClick(index, column, type) {
    //   this.orderColumns = [
    //     {
    //       type: type,
    //       orderColumn: this.column[index].prop,
    //     },
    //   ];
    // },
    insetStyle(index) {
      return (
        this.orderColumns[0] &&
        this.column[index].prop === this.orderColumns[0].orderColumn &&
        this.orderColumns[0].type === "desc"
      );
    },
    insetStyle2(index) {
      return (
```

```
        this.orderColumns[0] &&
        this.column[index].prop === this.orderColumns[0].orderColumn &&
        this.orderColumns[0].type === "asc"
      );
    },
```

表格筛选功能

```
//弹框 表格的callback 用于实现表格的筛选功能
    callbackTable(data){
      if(this.filterValue){
        let New = {page: 1,
          pagerows: 0,
          rows: null,
          totalrows: 0,}
        New.rows = data.rows.filter(
          item => item.dsColumnLabel.toLowerCase().includes(this.filterValue.toLowerCase())
        )
        return New
      }else{
        return data
      }
    },
```

表格BUG

1.表格右侧固定列显示一条横线

```
/* 表格样式问题 */
.el-table__fixed-right{
    height: 100% !important;
   }
 //设置高优先，以覆盖内联样式
 .el-table__fixed {
height: 100% !important;
    }
```

判断文件是否为图片类型

```
isPicFile(fileName) {
    //lastIndexOf如果没有搜索到则返回为 -1
    if (fileName.lastIndexOf(".") != -1) {
```

```
        var fileType = (fileName.substring(fileName.lastIndexOf(".") + 1,
ileName.length)).toLowerCase();
        var suppotFile = new Array();
            suppotFile[0] = "jpg";
            suppotFile[1] = "gif";
            suppotFile[2] = "bmp";
            suppotFile[3] = "png";
             suppotFile[4] = "jpeg";
             suppotFile[5] = "svg";
        for ( var i = 0; i < suppotFile.length; i++) {
            if (suppotFile[i] == fileType) {
                return true;
            } else {
                continue;
            }
        }
        this.$message("文件类型不合法,只能是jpg、gif、bmp、png、jpeg、png、
svg类型！");
        return false;
    } else {
        this.$message("文件类型不合法,只能是 jpg、gif、bmp、png、jpeg、pn
g、svg类型！");
        return false;
    }
}
```

文件流导出

```
get(`/api/bi/v1/myData/exportData/download.do?fileName=${fileName}&dataSe
tName=${this.dataSetName}&templateLabel=${templateLabel}`).then((res)=>{
            Message({
                  type: "success",
                  message: `开始导出`,
            });
            const a = document.createElement('a');
            a.download = fileName;
            a.href = window.URL.createObjectURL(res);
            document.body.appendChild(a);
             a.click();
             document.body.removeChild(a);
          })
 function get  (url){
```

```
    // 返回一个 promise 对象.
    return new Promise(function(resolve, reject) {
    // 创建一个 XML 对象
    var req = new XMLHttpRequest();
    req.open('GET', url);

    req.responseType = "blob";
    req.onload = function() {

     if (req.status == 200) {
     // 请求 200的时候处理 response
     resolve(req.response);

     }
     else {
     // 处理请求错误信息
     reject(Error(req.statusText));
     }
    };

    // 处理网络错误信息
    req.onerror = function() {
     reject(Error("Network Error"));
    };

    // 发送请求
    req.send();
    });
   }
```

图片上传

js

```

/**
 * @description 触发input上传事件
 */
beforeUpload(inputclass) {
  document.getElementsByClassName(inputclass)[0].click();
},
/**
```

```
 * @description  上传图片预览
 */
previewFile(inputclass,imgContainerClass) {
  var preview = document.getElementsByClassName(imgContainerClass)[0];
  var file = document.getElementsByClassName(inputclass)[0].files[0];
  var reader = new FileReader();
  reader.addEventListener(
    "load",
    function () {
      preview.src = reader.result;
    },
    false
  );

  if (file) {
    reader.readAsDataURL(file);
  }
},
```

html

```
<input
  type="file"
  class="addFile"
  @change="previewFile('addFileDefault','addImgDefault')"
  name="sPicture"
/>
 <div
  class="uploadImage"
  @click="beforeUpload('addFileDefault')"
  >
<img src="" class="addImgDefault"/>
</div>
```

file 图片文件对象

```
1. lastModified: 1605153921048
2. lastModifiedDate: Thu Nov 12 2020 12:05:21 GMT+0800 (中国标准时间) {}
3. name: "pic_zdzb.png"
4. size: 1691
5. type: "image/png"
6. webkitRelativePath: ""
```

滚动条自动滚动

```
<script>
 roll: function () {
 // 获取父盒子（肯定有滚动条）
 var parent = document.getElementById('parent');
 // 获取子盒子（高度肯定比父盒子大）
 var child1 = document.getElementById('child1');
 var child2 = document.getElementById('child2');
 // 第一个子盒子内容复制一遍给第二个子盒子，产生循环视觉，辅助作用
 // 可以注释下这条代码，看会出现什么情况
  child2.innerHTML = child1.innerHTML;
  // 设置定时器，时间即为滚动速度
  setInterval(function () {
  if(parent.scrollTop >= child1.scrollHeight) {
  parent.scrollTop = 0;
  } else {
  // 如果存在网页缩放，很有可能没有效果，但是else部分的代码会执行
  // 原因：刚才讲到的scrollTop三个注意中标黄的一条
  // 设置scrollTop的值小于0，即scrollTop被设为0
  // 可以缩放跑一下，然后不刷新的状态下恢复百分之百跑一下，再缩放，打印scrollTop的值
  // 你会发现正常尺寸执行时打印的第一个值不是加法，而是减法，即scrollTop++增加负值
  // 这样的话就对应上了scrollTop的注意点了，增加的值小于0，就被设为0
  parent.scrollTop++;
  }
  }, 20);
  }
</script>
```

文件上传

```
export const  ajax = (url, data)=> {
   return new Promise((resolve, reject) => {
       var xhr = new XMLHttpRequest()
       xhr.open("POST", url, true)
       // xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");//上传文件
       xhr.onreadystatechange = function () {
           if (xhr.readyState != 4) return
           if (xhr.readyState === 4 && xhr.status === 200) {
```

```
                resolve(xhr.responseText)
            } else {
                reject(xhr.responseText)
            }
        }
        xhr.open("POST", url, !0);
        xhr.send(data);
    })
}

    /**
     * @description file类型文件导入
     */
    upload(param){
      let fileObj = param.file;
      let form = new FormData();
      form.append("FileData", fileObj);
      form.append("client", "formUpload");
      form.append("name", fileObj.name);
      form.append("size", fileObj.size);
      ajax('/api/core/v1/attach/upload.do',form).then((res)=>{
        this.dialogFile = false;
        let result = JSON.parse(res)
        let params = {
          data:[
            {
                        "name": "parameter",
                        "vtype": "formpanel",
                        "data": {
                            "dsID": this.fileData.dsId,
                            "sourceType": "FILE",
                            "fileID":result.fileId,
                            "tableName":this.fileData.tableName,
                            "isDirectory":"N"
                        }
                    }
          ]
        }
        postJSON('/api/directories/v1/directories/maintenance/import.do',
{
```

```
          postData: JSON.stringify(params),
        }).then((msg)=>{
          Message.success(msg.data[0].data)
        })

      }).catch(()=>{
        console.log('上传失败')
      })
    },
```

Fusion_UI

表格筛选功能

```
//弹框 表格的callback 用于实现表格的筛选功能
    callbackTable(data){
      if(this.filterValue){
        let New = {page: 1,
          pagerows: 0,
          rows: null,
          totalrows: 0,}
        New.rows = data.rows.filter(
          item => item.dsColumnLabel.toLowerCase().includes(this.filterValue.toLowerCase())
        )
        return New
      }else{
        return data
      }
    },
```

关于下拉框的使用

```
selectReq: {
        url: "/api/core/v1/dictionary/queryData.do?dicId=createType",
        params: [],
      },


 <fu-select
```

```
              v-model="queryForm.newPattern"
              :fu-request="selectReq"
               fu-id="fu-select-1"
               clearable
               filterable
             >
               <fu-option
                 v-for="item in fuSelect1"
                 :key="item.value"
                 :label="item.text"
                 :value="item.value"
               >
               </fu-option>
             </fu-select>


  computed: {
    fuSelect1() {
      return this.$store.state.reqData["fu-select-1"] || [];
    },
```

多选下拉框组件

```
<template>
  <fu-select
    clearable
    :placeholder="placeholder"
    multiple
    size="medium"
    :disabled="disabled"
    v-model="value"
    @change="change"
     ><fu-option
       v-for="item in options"
       :key="item.value"
       :label="item.text"
       :value="item.value"
       size="medium"
    >
    </fu-option
  ></fu-select>
</template>
```

```
<script>
import { Select, Option } from "@/FuComponents";
export default {
  components: {
    FuSelect: Select,
    FuOption: Option,
  },
  props: {
    modelValue: {
      type: String,
      default: "",
    },
    disabled: {
      type: Boolean,
      default: false,
    },
    options: {
      type: Array,
      default: () => {
        return [];
      },
    },
    placeholder: {
      type: String,
      default: "请选择",
    },
  },
  data() {
    return {
      value: [],
    };
  },
  created() {
    this.value = this.modelValue.split(",");
  },
  methods: {
    change() {
      this.$emit("selectMultileChange", this.value.join(","));
    },
```

```
  },
};
</script>

<style></style>

```

```
<select-multiple
            :placeholder="'请选择被审计人'"
            :disabled="disabled"
            :model-value="formData1.auditedUser"
            :options="userSJOptions"
            @selectMultileChange="selectMultileChange"
          >
          </select-multiple>
```

转码函数

```
changeCodeUser(prop, arr) {
      return function(prop, arr) {
        try {
          let tempArr = prop.split(",");
          const res = [];
          tempArr.forEach((val) => {
            res.push(arr.find((value) => value.value === val).text);
          });
          return res.join(",");
         } catch (err) {
           return prop;
         }
       };
    },
```

去重

```
//多属性去重
const dict = {};
          for (const item of this.shopDetails) {
            dict[item.item_id + "," + item.unit_id] = item;
          }
          this.shopDetails = Object.values(dict);
 //单属性去重
```

```
// const res = new Map();
   let newSet = arrA.filter(
    (a) => !res.has(a.item_id) && res.set(a.item_id, 1)
      );
```

打印

https://segmentfault.com/a/1190000021913990?utm_source=tag-newest

vue 使用ueditor插件

https://www.jianshu.com/p/fe1559a9223e

cookie读取设置

```
//读取cookie
function getCookie(name) {
  var arr,
    reg = new RegExp("(^| )" + name + "=([^;]*)(;|$)");

  if ((arr = document.cookie.match(reg))) {
    return unescape(arr[2]);
  } else {
    return null;
   }
}


// //写cookie
// function setCookie(name, value) {
//   var Days = 30;
//   var exp = new Date();
//   exp.setTime(exp.getTime() + Days * 24 * 60 * 60 * 1000);
//   document.cookie =
//     name + "=" + escape(value) + ";expires=" + exp.toGMTString();
// }

// //删除cookie
// function delCookie(name) {
```

```
//   var exp = new Date();
//   exp.setTime(exp.getTime() - 1);
//   var cval = getCookie(name);
//   if (cval != null) {
//     document.cookie = name + "=" + cval + ";expires=" + exp.toGMTStri
ng();
//   }
// }

设置httponly的cookie无法获取；

关于cookie：
    1.是什么？
*         本质就是一个字符串，里面包含着浏览器和服务器沟通的信息（交互时产生的信息）。
*         存储的形式以：key-value的形式存储。
*         浏览器会自动携带该网站的cookie，只要是该网站下的cookie，全部携带。
*     2.分类：
*           --会话cookie（关闭浏览器后，会话cookie会自动消失，会话cookie存储在浏览器运行的那块内存上）。
*           --持久化cookie：（看过期时间，一旦到了过期时间，自动销毁，存储在用户的硬盘上,备注：如果没有到过期时间，同时用户清理了缓存，持久化cookie也会消失）。
*
*     3.工作原理：
*           --当浏览器第一次请求服务器的时候，服务器可能返回一个或多个cookie给浏览器
*           --浏览器判断cookie种类
*               --会话cookie：存储在浏览器运行的那块内存上
*               --持久化cookie：存储在用户的硬盘上
*           --以后请求该网站的时候，自动携带上该网站的所有cookie（无法进行干预）
*           --服务器拿到之前自己“种”下cookie，分析里面的内容，校验cookie的合法性，根据cookie里保存的内容，进行具体的业务逻辑。
*
*     4.应用：
*           解决http无状态的问题（例子：7天免登录，一般来说不会单独使用cookie，一般配合后台的session存储使用）
*
*     5.不同的语言、不同的后端架构cookie的具体语法是不一样的，但是cookie原理和工作过程是不变的。
*         备注：cookie不一定只由服务器生成，前端同样可以生成cookie，但是前端生成的cookie几乎没有意义。
```

git命令

git查看分支来源	git reflog show branchName
git删除本地分支	git branch -d 20201120_kang_css
git查看分支	git branch -a
git删除远程分支	git push origin --delete 20201218_wyk_CPZ-12

背景

- 项目需要删除不用的远程分支
- 项目需要在原来某一次 commitid 的基础上修改一些东西
- 简单记录分享，方便日后查找

删除远程分支

- 分支切换到 master 使用：

```
git checkout master
```

- 删除远程不再使用的分支，使用：

```
git push origin --delete [废弃分支名称]
```

有的还需要再次验证 git 密码，输入密码确认后即可成功删除远程分支

拉取指定 commitid 代码

- 从远程分支 clone 到本地新建文件夹。使用：

```
git clone [git-url] -b [branch-name]
```

- 指定 commit 版本。使用：

```
git reset --hard [commit-number]
```

- 推送到远程（新建）分支

```
git push origin [本地分支]:[远程分支名称]
```

如果远程分支名称不存在则会新建该分支