链滴

React+.NetCore API 实现动态列导出

作者：ambres
原文链接：https://ld246.com/article/1565687545118
来源网站：链滴

所用技术栈

1. 前端：React,Antd
2. 后端：ASP.NET CORE ， System.Linq.Dynamic.Core,EPPlus.Core

基本思路

1. NetCore解决生成Excel文件（流），使用了EPPlus.Core。
2. EF（LINQ）解决动态列的问题，，System.Linq.Dynamic.Core
3. 解决文件下载的问题，最开始的思路是，直接传流到前端，前端读response.blob()，然后生成文，结果.....失败，因为所有请求要先过Node层，流经过一次转发后，死活没办法变成excel文件。前能力不强，所以放弃这个思路了。 按照后端的解决方式，就是直接把文件生成到服务端，然后返回路给前端，前端跳转一次，就可以实现下载，尝试，成功！

核心代码

后端代码模块

```
public class ExportRequestModel<T>
  {

      public List<ExportRowModel> RowRequest { get; set; }
      /// <summary>

      /// 导出人ID

      /// </summary>
```

```
        public string ExportUserID { get; set; }

        /// <summary>

        /// 导出模块名称

        /// </summary>

        public string ModuleName { get; set; }

        public T Where { get; set; }

    }
    public class ExportRowModel
    {

        public string RowName { get; set; }

        public string RowKey { get; set; }

        public string TableName { get; set; }

    }
```

ExcelHelper.cs:

```
public static string Export(ExportRequestModel<TWhere> THead, List<TBody> TBody, string
ocalPath)
        {
            try
            {
                string sFileName = $"{THead.ExportUserID}-{Guid.NewGuid()}.xlsx";
                var finallypath = LocalPath + "\\excel\\" + THead.ModuleName;
                DirectoryInfo di = new DirectoryInfo(finallypath);
                if (!di.Exists) { di.Create(); }
                FileInfo file = new FileInfo(Path.Combine(finallypath, sFileName));
                var pack = GetExcelPackage(THead.RowRequest, TBody);
                pack.SaveAs(file);
                return ("\\excel\\" + THead.ModuleName + "\\" + sFileName).Replace("\\", "/");
            }
            catch (Exception)
            {
                throw;
            }
        }
        private static ExcelPackage GetExcelPackage(List<ExportRowModel> THead, List<TBody
 TBody)
        {
            ExcelPackage package = new ExcelPackage();
            var worksheet = package.Workbook.Worksheets.Add("sheet1");
            //循环生成表头
            for (int i = 1; i <= THead.Count; i++)
            {
                worksheet.Cells[1, i].Value = THead[i - 1].RowName;
```

```
            }
            //循环值
            for (int i = 1; i <= TBody.Count; i++)
            {
                var body = TBody[i - 1];
                var bodyPro = body.GetType().GetProperties();
                for (int j = 1; j <= bodyPro.Length; j++)
                {
                    if (bodyPro[j - 1].Name == "Item")
                    {

                        break;
                    }
                    worksheet.Cells[i + 1, j].Value = bodyPro[j - 1].GetValue(body, null);
                }
            }
            return package;

        }
    }
```

示例代码

```
 ApiResult<string> apiResult = new ApiResult<string>();

            List<string> selectKey = new List<string>();

            foreach (var item in request.RowRequest)

            {

                var tabOtherName = string.IsNullOrEmpty(item.TableName) ? item.TableName : ite
.TableName + ".";

                selectKey.Add($"{tabOtherName}{item.RowKey} as {item.RowName}");

            }

            var ExportList = _studentStatuService.Query()

                .Include(n => n.Payment)

                .Include(n => n.Student)

                .Include(n => n.Student.Department)

                .Where(n => (int)n.StudentType > 1)

                .HasWhere(request.Where.DepartMentID, n => n.Student.DepartmentID == request
Where.DepartMentID)

                .HasWhere(request.Where.EduMajorID, n => n.Payment.EduMajorId == request.Wh
re.EduMajorID)
```

```
            .HasWhere(request.Where.EduSchoolID, n => n.Payment.EduSchoolId == request.
here.EduSchoolID)
            .HasWhere(request.Where.Name, n => n.StudentNo.Contains(request.Where.Name)

            .HasWhere(request.Where.IdCard, n => n.Student.IDCard.Contains(request.Where.I
Card))
            .HasWhere(request.Where.StartTime, n => n.CreateTime>= request.Where.StartTim
)
            .HasWhere(request.Where.EndTime, n => n.CreateTime < request.Where.EndTime)
            .Select(n => new StudentLessResponse
            {
                PhoneNum = n.Student.PhoneNum,
                School = n.Payment.EduSchoolName,
                Major = n.Payment.EduMajorName,
                CreateTime = n.CreateTime.ToString("yyyy-MM-dd HH:mm:ss"),
                Guid = n.Student.Guid.ToString(),
                Name = n.Student.Name,
                DeptName = n.Student.Department.Name,
                StudentType = (int)n.StudentType,
                DiplomaType = string.Join("/", EnumHelper.GetDescription(n.DiplomaType)),
                GetSeason = EnumHelper.GetDescription(n.GetSeason),
                GetYear = n.GetYear,
                LevelSeason = EnumHelper.GetDescription(n.LevelSeason),
                LevelYear = n.LevelYear,
                StudentNo = n.StudentNo,
                TemporaryNo = n.TemporaryNo
            })
            .Select($"new ({string.Join(',', selectKey.ToArray())})")
            .ToDynamicList();
```

```
            var pack = ExcelHelper<dynamic, StudentPagesRequest>.Export(request, ExportList, L
calPath);
```

前端代码

Export.js组件：

```
import React from 'react';
import { Popover, Button, Checkbox, Row, Col, DatePicker } from 'antd';
import PropTypes from 'prop-types';
import { setLocalStorage, getLocalStorage } from '../../utils/cookieHelper';
const { RangePicker } = DatePicker;

class Export extends React.Component {

    state = {
        visible: false,
        checkedValues: getLocalStorage(this.props.moduleName),
        StartTime: null,
        EndTime: null
    }
    onChange = (checkedValues) => {
        const { exportList } = this.props;

        this.setState({ checkedValues });
    }
    export = () => {
        const { checkedValues, EndTime, StartTime } = this.state;
        const { moduleName, onExport, exportList } = this.props;

        setLocalStorage(moduleName, checkedValues);
```

```
        const finallyValues = exportList.filter((item, index) => {
            return checkedValues.indexOf(item.RowKey) > -1;
        });
        const data = {
            ModuleName: moduleName,
            RowRequest: finallyValues,
            where: {
                StartTime,
                EndTime
            }
        }
        console.log(data);
        if (onExport) {
            onExport(data);
        }

    }

    onTimeChange = (date, dateString) => {
        this.setState({ StartTime: dateString[0], EndTime: dateString[1] })

    }
    render() {
        const { exportList, moduleName } = this.props;
        const ccontent = (<div>

            <Checkbox.Group onChange={this.onChange} style={{ width: '220px' }} value={this.sta
e.checkedValues}>
```

```
            <Row>
                <div style={{ marginBottom: '12px' }}>
                    <div style={{ marginBottom: '5px' }}> 请选择数据产生的时间： </div>
                    <RangePicker onChange={this.onTimeChange} />
                </div>

                {exportList.map((item, index) => {
                    return (<Col span={12}><Checkbox key={item.RowKey} value={item.RowKey}>
item.RowName}</Checkbox></Col>)
                })}
            </Row>
        </Checkbox.Group>
        <Row style={{ textAlign: 'center', marginTop: '10px' }}>
            <Col span={2}></Col>
            <Col span={10}> <a onClick={this.export}>确定</a></Col>
            <Col span={10}> <a onClick={() => {
                let visb = !this.state.visible;
                this.setState({ visible: visb })
            }}>取消</a></Col>
            <Col span={2}></Col>

        </Row></div>);
    return (
        <Popover
            content={ccontent}
            title="请选择导出列"
            trigger="click"
```

```
                visible={this.state.visible}
                placement="bottom"

            >
                <Button type="primary" onClick={() => {
                    let visb = !this.state.visible;
                    this.setState({ visible: visb })
                }} icon={'export'} style={{ marginRight: '10px' }}>导出</Button>
            </Popover >
        );
    }
}

Export.propTypes = {
    onExport: PropTypes.func,
    moduleName: PropTypes.string,
    exportList: PropTypes.any
};

export default Export;
import  Export  from  '../../components/common/Export';

exportList  = [
{ RowKey:  "materialName", RowName:  "教材名称", TableName:  "" },
{ RowKey:  "followLevel", RowName:  "所跟级别", TableName:  "" },
{ RowKey:  "studentName", RowName:  "学生姓名", TableName:  "" },
{ RowKey:  "schoolName", RowName:  "学校名称", TableName:  "" },
{ RowKey:  "classAndMajorName", RowName:  "班级和专业名称", TableName:  "" },
{ RowKey:  "diplomaType", RowName:  "专业层次", TableName:  "" },
{ RowKey:  "courseName", RowName:  "课程名称", TableName:  "" },
{ RowKey:  "department", RowName:  "校区名称", TableName:  "" },
{ RowKey:  "teacherName", RowName:  "老师名称", TableName:  "" },
{ RowKey:  "phoneNum", RowName:  "老师电话", TableName:  "" },
```

```
{ RowKey: "hour", RowName: "时间安排", TableName: "" },
{ RowKey: "time", RowName: "上课日期", TableName: "" },
{ RowKey: "roomNo", RowName: "上课课室", TableName: "" }]

export = (data) => {
  const {routeParams: {guid}} = this.props;
  data.where = {
     ...data.where,
     guid,
     StartTime: this.state.startTime,
     EndTime: this.state.endTime
}
   this.props.dispatch({ type: this.baseConfig.baseName + '/export', payload: data });
}


<Export exportList={this.exportList} moduleName={this.baseConfig.baseName} onExport={
his.export} />;
```