第 1 章

电影产业市场数据分析和可视化系统 (Flask+FastAPI+Vue+Echarts)

在当前的市场环境下，去影院看电影仍是消费者休闲娱乐的主要方式之一，这一点可以从近些年电影市场的高速发展和私人影院的迅速崛起得到佐证。大数据分析电影票房并提取出有关资料，对电影行业从业者们尤为重要。本章通过一个综合实例，详细讲解使用 Python 开发一个电影产业市场数据分析和可视化系统的方法。

1.1 电影产业介绍

扫码看视频

电影产业是指以电影的制作、发行和放映三个行业为主，同时包括电影的后产品开发(如音像制品、电影频道、相关图书、玩具等)以及与电影相关的市场活动的总称，属第三产业中娱乐业的一部分。其主要功能是通过视听技术传递艺术形象信息，为人们提供审美、娱乐、宣教服务。

从电影行业整体来看，其产业链包括电影制作、出品、发行、放映环节。电影制作作为整个产业的最前端，决定了行业的影片供给数量、质量等情况，具有一定的议价权。影片制作完成后，通过出品及发行方使得影片得以面世，向下游院线企业进行宣发。电影产业的终端是院线市场，其基本职能是提供放映服务获取票务收入，一般占据 45%的票房分账比例，同时还为合作商提供广告服务、卖品等衍生品服务以获得非票务收入。

在电影产业链中，发行上承制片方，下连院线播映方，是将影片全国推广的渠道。电影制作方主要包括国内外文学与剧本等原始作品方，如国外的漫威、迪士尼，国内的华策影视、腾讯文学等；内容出品方包括海外的华纳兄弟、环球影业，国内的万达影视、华谊兄弟等专业影视公司；宣传发行方包括华纳兄弟、环球影业等传统影视公司发行方，以及淘票票、爱奇艺与猫眼电影等网络发行方；电影产业链终端的院线平台代表公司有海外的AMC 与国内的万达影院、大地影院、横店影视等。

1.2 电影市场的需求分析

扫码看视频

在中国影视行业政策注重内容端的输出、市场竞争加大的背景下，影视行业的作品将迎来规范化发展，影视 IP 产业链的变现能力有待充分挖掘。在过去的 3 年中，由于受到疫情影响，中国线下影视行业受到较大的冲击。随着疫情逐渐可控化，以及中国人均文化娱乐支出的提升，中国影视行业市场规模预计将在 2024 年达到 3618 亿元。在如此大的市场下，精细化分析电影产业市场的数据势在必行。

1.2.1 市场需要高质量作品

在过去十年里，国产电影规模飞速增长，虽然期间有波动，但整体趋势向好。伴随着

人口红利的逐渐减弱、互联网票补减少、政策监管趋严、进口片票房下滑、影院扩张边际递减等因素的叠加影响，国内电影行业增速放缓，这就意味着，上一波靠城镇院线扩张和互联网票补红利驱动的粗放式票房增长带来的电影牛市已告一段落。在下一个十年，国产电影的辉煌需要靠提高优质影片的供给推动。得益于国民经济的快速发展，持续增加的居民精神消费构成了电影产业蓬勃发展的重要支撑。

1.2.2 国内电影市场的变化

中国电影市场在制作端持续沉淀、劣后产能出清的过程中保持蓬勃发展态势，中国电影票房已经由数量驱动转为质量驱动，工业化体系逐渐形成。国内市场的突出变化如下。

(1) 明星效应淡化，口碑效应提升，优质内容和口碑的传递已经成为国内电影市场的核心驱动力。一方面，评分 8～9 分区间的电影票房占比不断扩大，口碑效应对电影的影响力传播效果更显著；另一方面，很多头部影片即使没有流量明星的加盟，一样可以凭借内容本身赢得市场的认可。区别于以往，流量明星的票房带动效能减弱，内容制作的精良更能驱动票房的增长。

(2) 电影题材更丰富，小众电影逆袭。近两年，电影类型丰富多元，小众电影也能突出重围，多部现实主义题材的影片成绩斐然，收获了经济效益与社会效益的双赢，引起了观众的强烈反响。

(3) 从 2019 年开始，流量明星尽数退场，实力派演员成为各大出品方青睐的阵容主体，中生代与新生代实力派也成为更加普遍的新型组合。一方面，电影公司对艺人偏向的集体转化直接反映出当下市场的理性回归，口碑的“自来水效应”成为传播的最大利器；另一方面，电影的类型更加多元化，不再仅仅局限于传统电影公司青睐的喜剧、悬疑、爱情等题材，互联网公司的入局，更加重视小众电影的试水，科幻、动画、现实主义题材影片也逐渐得到大众的认可，精良的制作、不受限制的题材都将赢得市场；同时，行业的“二八效应”仍在深化，各大电影公司的主控影片相对较少，更多地以参投的形式布局，以弱化投资风险，将有限的资金分布于多部影片，加大压中爆款的概率，不难看出抱团取暖不失为行业寒冬中的明智之举。

1.3 系统架构

扫码看视频

在开发一个大型应用程序时，系统架构是一项非常重要的前期准备工作，

是整个项目的实现流程能否顺利完成的关键。根据严格的市场需求分析，得出本项目的系统架构，如图 1-1 所示。

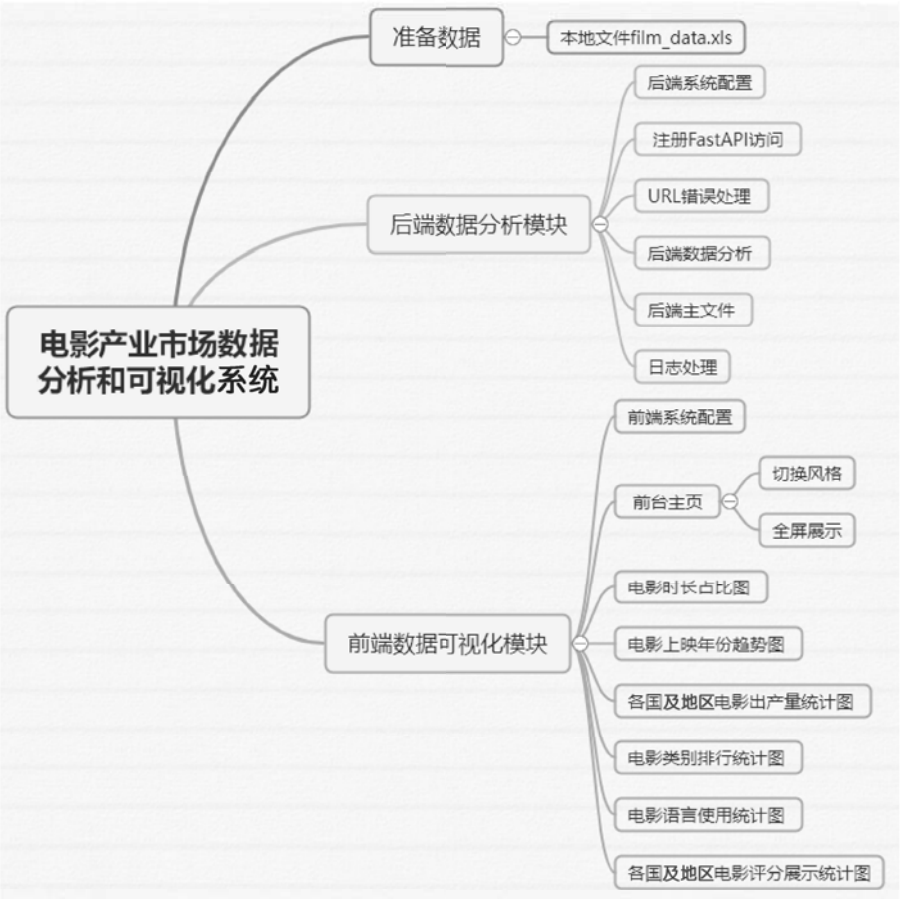

图 1-1　系统架构

1.4　准备数据

在本地文件 film_data.xls 中保存了 5804 部在国内上映的电影信息，上映时间从 2009 年到 2022 年，保存的数据包括：译名、片名、年代、产地、类别、语言、字幕、上映日期、豆瓣评分、片长、导演、编剧、主演、简介，如图 1-2 所示。

扫码看视频

film_data.xls [受保护的视图] - Excel

文件 开始 模板 插入 页面布局 公式 数据 审阅 视图 PDF工具 百度网盘 登录

A1 译名

	A	B	C	D	E	F	G	H
5776	无	无	无	无	无	无	无	无
5777	妙狗拯救圣诞节	The Dog Who Saved Ch	2009	无	喜剧	英语	中英双字	无
5778	旧情不散	My EX	2009	无	恐怖	泰语	中文	无
5779	听说	Hear Me	2009	无	剧情	普通话	中文	无
5780	僵尸之地	Zombieland	2009	无	喜剧/恐怖	英语	中英双字	无
5781	死神傻了	Split Second Murderrs	2009	无	喜剧	普通话	中文	无
5782	关人7事/关人七事/玩大吐	Seven 2 One	2009	无	剧情/犯罪	国语	中文	无
5783	国家代表	Take Off	2009	无	动作	韩语	中文	无
5784	谎言的诞生/事实的这一面	The Invention of Lying	2009	无	喜剧	英语	中英双字	无
5785	午夜出租车	Midnight Taxi	2009	无	爱情/恐怖	普通话	中文	无
5786	轮滑女郎/滚逐青春	Whip It	2009	无	喜剧/剧情	英语	中英双字	无
5787	道林·格雷	Dorian Gray	2009	无	剧情	英语	中英双字	无
5788	兄弟	Brothers	2009	无	剧情/惊悚	英语	中文	无
5789	僵尸类门人	Vampire Super	2009	无	剧情/喜剧	粤语	中文	无
5790	圣诞故事/圣诞节的传说/圣诞童话/圣诞物语	A Christmas Tale	2008	无	喜剧/剧情	法语	中英双字	无
5791	海军陆战队员2/怒火街头2	The Marine 2	2009	无	动作	英语	中英双字	无
5792	一次完美的逃亡/蜜月变奏曲	A Perfect Getaway	2009	无	剧情/恐怖	英语	中字	无
5793	截稿日期	Deadline	2009	无	惊悚/剧情	英语	中英双字	无
5794	愚昧年代	The Age of Stupid	2009	无	纪录片	英语	中英双字	无
5795	海军陆战队员2/怒火街头2	The Marine 2	2009	无	动作	英语	中英双字	无
5796	扑克王	Poker King	2009	无	喜剧	普通话	中文	无
5797	灵动：鬼影实录/午夜灵异录象	Paranormal Activity	2009	无	恐怖/神秘	英语	中英双字	无
5798	扑克王	Poker King	2009	无	喜剧	粤语	中文	无
5799	迈克尔·杰克逊：就是这样/天王终点/这就是了	This Is It	2009	无	记录/音乐	英语	中字	无
5800	一席之地	A Place of Ones Own	2009	无	喜剧/音乐	普通话	中文	无
5801	灵动：鬼影实录/午夜灵异录象	Paranormal Activity	2009	无	恐怖/神秘	英语	中英双字	无
5802	万圣节10/万圣节9续集/月光光心慌慌10/新万	Halloween II	2009	无	惊悚/恐怖	英语	中英双字	无
5803	无	无	无	无	无	无	无	无
5804	搞怪一家人/盖酷家庭：星际大战大反击/恶搞	"Family Guy" Something	2009	无	动画/喜剧	英语	中英双字	无
5805								
5806								

图 1-2　文件 film_data.xls 中的电影数据

本项目将分析文件 film_data.xls 中的数据，以可视化方式展示电影产业市场的数据。

1.5　后端数据分析模块

本项目由前端和后端构成，其中后端主要负责数据分析，保存在 data_analysis 目录中。后端模块由 Python 和相关的库实现，各个库的版本信息保存在文件 requirements.txt 中。

扫码看视频

1.5.1　后端系统配置

本项目的后端系统配置功能由文件 config.py 实现，实现了系统跨域访问参数、API 接口和访问日志等功能，主要实现代码如下所示。

```
from typing import Union, List
from pydantic import BaseSettings, AnyHttpUrl

class Settings(BaseSettings):
    PROJECT_VERSION: Union[int, str] = 5.0                  # 版本
    BASE_URL: AnyHttpUrl = "http://127.0.0.1:8000"          # 开发环境
```

```
    API_PREFIX: str = "/api"                # 接口前缀
    TEMPLATES_DIR: str = 'templates'        # 静态文件目录
    GLOBAL_ENCODING: str = 'utf-8'          # 全局编码
    CORS_ORIGINS: List[AnyHttpUrl] = ["http://101.43.79.137:8999",
                                      "http://localhost:8999"]  # 跨域请求
    JWT_SECRET_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"
    JWT_ALGORITHM = "HS256"
    JWT_ACCESS_TOKEN_EXPIRE_MINUTES = 24 * 60
    TOKEN_URL = "/111"

    # DATABASE_URI: str = "mysql+asyncmy://root:123456@localhost:3306/zyr_init?charset=utf8mb4"
    # MySQL(异步)
    # DATABASE_URI: str = "mysql+pymysql://root:123456@localhost:3306/zyr_init?charset=utf8"
    DATABASE_ECHO: bool = False  # 是否打印数据库日志 (可看到创建表、表数据增删改查的信息)
    # REDIS_URI: str = "redis://:@localhost:6379/1"  # redis

    LOGGER_DIR: str = "logs"  # 日志文件夹名
    LOGGER_NAME: str = '{time:YYYY-MM-DD_HH-mm-ss}.log'  # 日志文件名 (时间格式)
    LOGGER_LEVEL: str = 'DEBUG'  # 日志等级: ['DEBUG' | 'INFO']
    LOGGER_ROTATION: str = "12:00"  # 日志分片: 按时间段/文件大小切分日志。 例如 ["500 MB" |
        "12:00" | "1 week"]
    LOGGER_RETENTION: str = "7 days"  # 日志保留的时间: 超出将删除最早的日志。例如 ["1 days"]
```

1.5.2 注册 FastAPI 访问

FastAPI 是用于构建 Web API 的现代、开源、快速、高性能的 Web 框架。正如它的名字那样，FastAPI 就是为构建快速的 API 而生。FastAPI 适用于构建高性能的 API，本身支持异步。如果要构建异步 API，可以优先选择 FastAPI，如 Netflix 将其用于内部危机管理。它还可以在部署准备就绪的机器学习模型时完美缩放，因为当 ML(机器学习)模型封装在 REST API 并部署在微服务中时，它在生产中会发挥最佳作用。

在本项目后端，通过 FastAPI 构建了不同功能的 API 接口，这些接口将和前端 URL 建立映射，将后端实现的数据分析结果在前端用图形化的方式(Echarts)展示出来。

(1) 编写文件 router.py，注册路径导航路由，具体实现代码如下所示。

```
from fastapi import FastAPI
from api import api_router
from core import settings

def register_router(app: FastAPI):
    """ 注册路由 """
    app.include_router(api_router, prefix=settings.API_PREFIX)
```

(2) 编写文件 middleware.py，实现请求拦截与响应拦截功能，将访问用户的 IP 信息添加到日志文件中，具体实现代码如下所示。

```
def register_middleware(app: FastAPI):

    @app.middleware("http")
    async def intercept(request: Request, call_next):
        logger.info(f"访问记录:IP:{request.client.host}-method:{request.method}-url:
                    {request.url}")
        return await call_next(request)  # 返回请求(跳过token)
```

(3) 编写文件 exception.py，创建错误信息处理函数，用于处理不同的错误类型，例如 HTTP 异常、存储失败、权限不足、请求参数丢失等，具体实现代码如下所示。

```
from fastapi import FastAPI
from fastapi.exceptions import RequestValidationError, HTTPException
from pydantic import ValidationError
from pymysql import ProgrammingError
from requests import Request
from sqlalchemy.orm.exc import UnmappedInstanceError
from starlette.responses import JSONResponse
from utils import IpError, ErrorUser, UserNotExist, IdNotExist, SetRedis, AccessTokenFail,
    PermissionNotEnough, resp_400, resp_401, resp_403, resp_500, resp_422, resp_404
from core import logger

def register_exception(app: FastAPI):
    """ 全局异常捕获 """

    @app.exception_handler(HTTPException)
    async def http_error_handler(request: Request, exc: HTTPException):
        """
        http 异常处理
        :param _:
        :param exc:
        :return:
        """
        if exc.status_code == 401:
            logger.warning(f"{exc.detail}\nURL:{request.method}-{request.url}\nHeaders:
                           {request.headers}")
            return resp_401()
        if exc.status_code == 403:
            logger.warning(f"{exc.detail}\nURL:{request.method}-{request.url}\nHeaders:
                           {request.headers}")
            return resp_403()
        if exc.status_code == 404:
            logger.warning(f"{exc.detail}\nURL:{request.method}-{request.url}\nHeaders:
                           {request.headers}")
```

```
        return resp_404()
    return JSONResponse({
        "code": exc.status_code,
        "message": exc.detail,
        "back": exc.detail
    }, status_code=exc.status_code, headers=exc.headers)

@app.exception_handler(IpError)
async def ip_error_handler(request: Request, exc: IpError):
    """ ip 错误(自定义异常) """
    logger.warning(f"{exc.err_desc}\nURL:{request.method}-{request.url}\nHeaders:
                   {request.headers}")
    return resp_400(msg=exc.err_desc)

@app.exception_handler(IdNotExist)
async def id_not_exist_handler(request: Request, exc: IdNotExist):
    """ 查询 id 不存在(自定义异常) """
    logger.warning(f"{exc.err_desc}\nURL:{request.method}-{request.url}\nHeaders:
                   {request.headers}")
    return resp_400(msg=exc.err_desc)

@app.exception_handler(SetRedis)
async def set_redis_handler(request: Request, exc: SetRedis):
    """ Redis 存储失败(自定义异常) """
    logger.warning(f"{exc.err_desc}\nURL:{request.method}-{request.url}\nHeaders:
                   {request.headers}")
    return resp_400(msg=exc.err_desc)

@app.exception_handler(AccessTokenFail)
async def access_token_fail_handler(request: Request, exc: AccessTokenFail):
    """ 访问令牌失败(自定义异常) """
    logger.warning(f"{exc.err_desc}\nURL:{request.method}-{request.url}\nHeaders:
                   {request.headers}")
    return resp_401(msg=exc.err_desc)

@app.exception_handler(PermissionNotEnough)
async def permission_not_enough_handler(request: Request, exc: AccessTokenFail):
    """ 权限不足，拒绝访问(自定义异常) """
    logger.warning(f"{exc.err_desc}\nURL:{request.method}-{request.url}\nHeaders:
                   {request.headers}")
    return resp_403(msg=exc.err_desc)

@app.exception_handler(ProgrammingError)
async def programming_error_handle(request: Request, exc: ProgrammingError):
    """ 请求参数丢失 """
    logger.error(f"请求参数丢失\nURL:{request.method}-{request.url}\nHeaders:
                 {request.headers}\nerror:{exc}")
```

```
    return resp_400(msg='请求参数丢失!(实际请求参数错误)')

@app.exception_handler(RequestValidationError)
async def request_validation_exception_handler(request: Request, exc: RequestValidationError):
    """ 请求参数验证异常 """
    logger.error(f"请求参数格式错误\nURL:{request.method}-{request.url}\nHeaders:
                 {request.headers}\nerror:{exc.errors()}")
    return resp_422(msg=exc.errors())

@app.exception_handler(ValidationError)
async def inner_validation_exception_handler(request: Request, exc: ValidationError):
    """ 内部参数验证异常 """
    logger.error(f"内部参数验证错误\nURL:{request.method}-{request.url}\nHeaders:
                 {request.headers}\nerror:{exc.errors()}")
    return resp_500(msg=exc.errors())

@app.exception_handler(Exception)
async def all_exception_handler(request: Request, exc: Exception):
    """ 捕获全局异常 """
    logger.error(f"全局异常\n{request.method}URL:{request.url}\nHeaders:
                 {request.headers}\n{traceback.format_exc()}")
    return resp_500(msg="服务器内部错误")
```

1.5.3 URL 错误处理

在本项目中提供了各种数据分析的可视化结果页面，为了提高用户体验，将在访问 URL 的过程中对所遇见的错误进行集中处理。上述功能由文件 resp_code.py 实现，具体实现代码如下所示。

```
from typing import Union, Any, Optional
from starlette import status
from starlette.responses import Response
from fastapi.responses import ORJSONResponse
from core.logger import logger

def resp_200(*, data: Any = '', msg: str = "Success") -> dict:
    logger.info(msg)
    return {'code': 200, 'back': data, 'msg': msg}

def resp_400(code: int = 400, data: str = None, msg: str = "请求错误(400)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_400_BAD_REQUEST, content={'code': code,
'msg': msg, 'back': data})

def resp_401(*, data: str = None, msg: str = "未授权，请重新登录(401)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_401_UNAUTHORIZED, content={'code': 401,
'msg': msg, 'back': data})
```

```
def resp_403(*, data: str = None, msg: str = "拒绝访问(403)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_403_FORBIDDEN, content={'code': 403,
        'msg': msg, 'back': data})

def resp_404(*, data: str = None, msg: str = "请求出错(404)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_404_NOT_FOUND, content={'code': 404,
        'msg': msg, 'back': data})

def resp_422(*, data: str = None, msg: Union[list, dict, str] = "不可处理的实体") -> Response:
    return ORJSONResponse(status_code=status.HTTP_422_UNPROCESSABLE_ENTITY,
                          content={'code': 422, 'msg': msg, 'back': data})

def resp_500(*, data: str = None, msg: Union[list, dict, str] = "服务器错误(500)") ->
Response:
    return ORJSONResponse(headers={'Access-Control-Allow-Origin': '*'},
                          status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                          content={'code': 500, 'msg': msg, 'back': data})

def resp_502(*, data: str = None, msg: str = "网络错误(502)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_502_BAD_GATEWAY, content={'code': 502,
        'msg': msg, 'back': data})

def resp_406(*, data: str = None, msg: str = "请求的格式不可得(406)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_406_NOT_ACCEPTABLE, content={'code':
        406, 'msg': msg, 'back': data})

def resp_408(*, data: str = None, msg: str = "请求超时(408)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_408_REQUEST_TIMEOUT, content={'code':
        408, 'msg': msg, 'back': data})

def resp_410(*, data: str = None, msg: str = "请求的资源被永久删除，且不会再得到的(410)") ->
Response:
    return ORJSONResponse(status_code=status.HTTP_410_GONE, content={'code': 410, 'msg':
msg, 'back': data})

def resp_501(*, data: str = None, msg: str = "服务未实现(501)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_501_NOT_IMPLEMENTED, content={'code':
        501, 'msg': msg, 'back': data})

def resp_503(*, data: str = None, msg: str = "服务不可用(503)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_503_SERVICE_UNAVAILABLE,
                          content={'code': 503, 'msg': msg, 'back': data})

def resp_504(*, data: str = None, msg: str = "网络超时(504)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_504_GATEWAY_TIMEOUT, content={'code':
        504, 'msg': msg, 'back': data})
```

```
def resp_505(*, data: str = None, msg: str = "HTTP版本不受支持(505)") -> Response:
    return ORJSONResponse(status_code=status.HTTP_505_HTTP_VERSION_NOT_SUPPORTED,
                          content={'code': 505, 'msg': msg, 'back': data})
```

1.5.4 后端数据分析

本项目的后端数据分析功能由文件 film.py 实现，在此文件中定义了多个映射能函数，用于实现不同的数据分析功能，每个功能函数和一个 URL 相对应。文件 film.py 的具体实现流程如下所示。

(1) 编写映射参数 release_year，数据分析 2008 年到 2022 年的电影上映情况，对应代码如下所示。

```
@router.get('/release_year', summary="年份上映情况")
async def excel():
    # filename 是文件的路径名称
    workbook = xlrd2.open_workbook('./models/film_data.xls')
    # 通过 sheet 名称获取
    table = workbook.sheet_by_name(sheet_name='Sheet1')
    table_list = table.col_values(colx=2, start_rowx=1, end_rowx=None)

    # year_list={''+str(i)+':'+str(1) for i in range(2008,2023)}
    year_list = {'2008': 1, '2009': 1, '2010': 1, '2011': 1, '2012': 1, '2013': 1, '2014':
1, '2015': 1, '2016': 1, '2017': 1, '2018': 1, '2019': 1, '2020': 1, '2021': 1, '2022': 1}

    for i in range(0, len(table_list)):
        try:
            if year_list.get('' + str(int(table_list[i]))):
                year_list['' + str(int(table_list[i]))] = year_list['' +
str(int(table_list[i]))] + 1
        except ValueError:
            pass

    data_list = [year_list[i] for i in year_list]
    return resp_200(msg="查询成功！", data={"data": data_list,
                                "year": [2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015,
                                         2016, 2017, 2018, 2019, 2020, 2021, 2022]})
```

(2) 编写映射参数 language_use，分析 top10 上映电影中所使用的语言，对应代码如下所示。

```
@router.get('/language_use', summary="语言使用统计(top10)")
async def excel():
    # filename 是文件的路径名称
```

```
    workbook = xlrd2.open_workbook('./models/film_data.xls')
    # 通过 sheet 名称获取
    table = workbook.sheet_by_name(sheet_name='Sheet1')
    table_list = table.col_values(colx=5, start_rowx=1, end_rowx=None)
    result = {}
    # 数据统计
    for i in table_list:
        data = re.split("[ /无,;]", i)
        for j in data:
            if j == '':
                break
            if j not in result:
                result[j] = 1
            else:
                result[j] = result[j] + 1

    result["普通话"] = result["普通话"] + result["国语"]
    result["普通话"] = result["普通话"] + result["汉语普通话"]
    del result["国语"]
    del result["汉语普通话"]
    # 数据排序
    result = sorted(result.items(), key=lambda x: x[1], reverse=True)
    return resp_200(msg="查询成功！", data=result[0:10])
```

(3) 编写映射参数 film_type，分析 top20 电影类型，对应代码如下所示。

```
@router.get('/film_type', summary="电影类型统计(top20)")
async def excel():
    # filename 是文件的路径名称
    workbook = xlrd2.open_workbook('./models/film_data.xls')
    # 通过 sheet 名称获取
    table = workbook.sheet_by_name(sheet_name='Sheet1')
    table_list = table.col_values(colx=4, start_rowx=1, end_rowx=None)
    result = {}
    # 数据统计
    for i in table_list:
        data = re.split("[ /无,;]", i)
        for j in data:
            if j == '':
                break
            if j not in result:
                result[j] = 1
            else:
                result[j] = result[j] + 1
    # 数据排序
    result = sorted(result.items(), key=lambda x: x[1], reverse=True)
    return resp_200(msg="查询成功！", data=result[0:20])
```

(4) 编写映射参数 country_film_score，分析各国及地区 top10 电影的评分信息，要求电影总数不少于 30 部，对应代码如下所示。

```
@router.get('/country_film_score', summary="各国及地区电影评分(top10)电影数不少于 30 部")
async def excel():
    # filename 是文件的路径名称
    workbook = xlrd2.open_workbook('./models/film_data.xls')
    # 通过 sheet 名称获取
    table = workbook.sheet_by_name(sheet_name='Sheet1')
    country_list = table.col_values(colx=3, start_rowx=1, end_rowx=None)
    score_list = table.col_values(colx=8, start_rowx=1, end_rowx=None)
    del_list = ["中国大陆", "中国香港", "中国台湾", '']
    result = {}
    score_result = []
    result_list = {}

    for i in score_list:
        # 非数字无法转成 float，直接变成 0
        try:
            data = re.split("[ /,无]", i)
            score_result.append(float(data[0]))
        except ValueError:
            score_result.append(0)

    for i in range(len(country_list)):
        i_data = re.split("[ /,无N]", country_list[i])
        for data in i_data:
            if data in del_list:
                break
            if data not in result:
                result[data] = {"count": 0, "score": 0}
            else:
                result[data]["count"] = result[data]["count"] + 1
                result[data]["score"] = result[data]["score"] + score_result[i]

    # 求平均数
    for i in result:
        if result[i]["score"] < 0 or result[i]["count"] < 30:
            pass
        else:
            result_list[i] = round(result[i]["score"] / result[i]["count"], 1)
    # 数据排序
    result_list = sorted(result_list.items(), key=lambda x: x[1], reverse=True)

    return resp_200(msg="查询成功！", data=result_list[0:12])
```

(5) 编写映射参数 film_time_play，分析各电影时长的占比情况，对应代码如下所示。

```
@router.get('/film_time_play', summary="电影时长占比")
async def excel():
    # filename 是文件的路径名称
    workbook = xlrd2.open_workbook('./models/film_data.xls')
    # 通过 sheet 名称获取
    table = workbook.sheet_by_name(sheet_name='Sheet1')
    time_list = table.col_values(colx=9, start_rowx=2, end_rowx=None)
    result = {"30": 0, "60": 0, "90": 0, "120": 0, "150": 0, "180": 0, "210": 0, "更多": 0}
    result_list = []
    tihuan = ''
    for i in time_list:
        data = re.split("[分钟 h:m:s Min Mins 无]", str(i))
        try:
            if int(data[0]):
                for j in result:
                    if int(data[0]) > 210:
                        result["更多"] = result["更多"] + 1
                        break
                    if int(data[0]) < 30:
                        result["30"] = result["30"] + 1
                        break
                    if int(data[0]) > int(j):
                        tihuan = j
                        continue
                    else:
                        result[tihuan] = result[tihuan] + 1
                        tihuan = ''
                        break
        except Exception:
            pass
    for i in result:
        result_list.append({"value": result[i], "name": i+'分钟左右'})
    return resp_200(msg="查询成功！", data=result_list)
```

(6) 编写映射参数 film_from_country，分析各国及地区电影出产量(不少于 10 部)的信息，对应代码如下所示。

```
@router.get('/film_from_country', summary="各国及地区电影出产量(不少于 10 部)")
async def excel():
    # filename 是文件的路径名称
    workbook = xlrd2.open_workbook('./models/film_data.xls')
    # 通过 sheet 名称获取
    table = workbook.sheet_by_name(sheet_name='Sheet1')
    table_list = table.col_values(colx=3, start_rowx=1, end_rowx=None)
    result = {"中国":0}
    result_list=[]
    agin_list = ["中国大陆", "中国香港", "中国台湾"]
```

```
    del_list=[]
    # 数据统计
    for i in range(len(table_list)):
        i_data = re.split("[ /,无N]", table_list[i])
        for data in i_data:
            if data=='':
                break
            if data in agin_list:
                result["中国"] = result["中国"] + 1
                break
            if data not in result:
                result[data] = 0
            else:
                result[data] = result[data] + 1
    # 数据筛选(不少于10部)
    for i in result:
        if result[i]<10:
            del_list.append(i)
    for i in del_list:
        del result[i]
    # 数据重构
    for i in result:
        result_list.append({"name":i,"value":result[i]})
    return resp_200(msg="查询成功！", data=result_list)
```

1.5.5 后端主文件

本项目的后端主文件是main.py，功能是调用前面的功能函数，实现后端API和前端的跨域访问，具体实现代码如下所示。

```
import uvicorn
from fastapi import FastAPI
from core import logger
from register import register_exception, register_router, register_middleware,
register_cors

app = FastAPI()

def create_app():
    """ 注册中心 """
    register_exception(app)             # 注册捕获全局异常

    register_router(app)                # 注册路由

    register_middleware(app)            # 注册请求响应拦截
```

```
    register_cors(app)                    # 注册跨域请求

    logger.info("日志初始化成功！！！")    # 初始化日志

@app.on_event("startup")
async def startup():
    create_app()                          # 加载注册中心

@app.on_event("shutdown")
async def shutdown():
    pass

if __name__ == '__main__':
    uvicorn.run(app='main:app', host="localhost", port=38000, log_level="info")
```

1.5.6 日志处理

为了提高易维护性，本项目提供了日志处理模块，只要有用户访问本项目，就会记录此用户的详细访问信息。日志处理模块由文件 logger.py 实现，它不仅能在控制台显示日志信息，而且还会在本地目录中生成.log 格式的日志文件。文件 logger.py 的主要实现代码如下所示。

```
import os
from loguru import logger
from core import settings
from utils import create_dir

def logger_file() -> str:
    """ 创建日志文件名 """
    log_path = create_dir(settings.LOGGER_DIR)

    """ 保留日志文件夹下最大个数(本地调试用)
    本地调试需要多次重启，日志轮转片不会生效 """
    file_list = os.listdir(log_path)
    if len(file_list) > 3:
        os.remove(os.path.join(log_path, file_list[0]))
    # 日志输出路径
    return os.path.join(log_path, settings.LOGGER_NAME)

logger.add(
    logger_file(),
    encoding=settings.GLOBAL_ENCODING,
    level=settings.LOGGER_LEVEL,
```

```
    rotation=settings.LOGGER_ROTATION,
    retention=settings.LOGGER_RETENTION,
    enqueue=True
)
```

在控制台中显示的日志信息如图 1-3 所示。

```
INFO:     ::1:53684 - "GET /api/country_film_score HTTP/1.1" 200 OK
INFO:     ::1:53679 - "GET /api/film_time_play HTTP/1.1" 200 OK
2023-02-06 19:11:44.038 | INFO     | register.middleware:intercept:22 - 访问记录:IP:::1-method:GET-url:http://localhost:38000/router
INFO:     ::1:53944 - "GET /router HTTP/1.1" 404 Not Found
2023-02-06 19:11:51.816 | INFO     | register.middleware:intercept:22 - 访问记录:IP:::1-method:GET-url:http://localhost:38000/api/film_from_country
2023-02-06 19:11:54.929 | INFO     | utils.resp_code:resp_200:17 - 查询成功!
2023-02-06 19:11:54.934 | INFO     | register.middleware:intercept:22 - 访问记录:IP:::1-method:GET-url:http://localhost:38000/api/film_time_play
INFO:     ::1:53957 - "GET /api/film_from_country HTTP/1.1" 200 OK
2023-02-06 19:11:59.168 | INFO     | utils.resp_code:resp_200:17 - 查询成功!
2023-02-06 19:11:59.171 | INFO     | register.middleware:intercept:22 - 访问记录:IP:::1-method:GET-url:http://localhost:38000/api/release_year
2023-02-06 19:12:01.365 | INFO     | utils.resp_code:resp_200:17 - 查询成功!
2023-02-06 19:12:01.366 | INFO     | register.middleware:intercept:22 - 访问记录:IP:::1-method:GET-url:http://localhost:38000/api/language_use
2023-02-06 19:12:04.361 | INFO     | utils.resp_code:resp_200:17 - 查询成功!
2023-02-06 19:12:04.364 | INFO     | register.middleware:intercept:22 - 访问记录:IP:::1-method:GET-url:http://localhost:38000/api/film_type
2023-02-06 19:12:06.539 | INFO     | utils.resp_code:resp_200:17 - 查询成功!
2023-02-06 19:12:06.541 | INFO     | register.middleware:intercept:22 - 访问记录:IP:::1-method:GET-url:http://localhost:38000/api/country_film_score
INFO:     ::1:53958 - "GET /api/film_time_play HTTP/1.1" 200 OK
2023-02-06 19:12:08.688 | INFO     | utils.resp_code:resp_200:17 - 查询成功!
INFO:     ::1:53959 - "GET /api/release_year HTTP/1.1" 200 OK
INFO:     ::1:53960 - "GET /api/language_use HTTP/1.1" 200 OK
INFO:     ::1:53961 - "GET /api/film_type HTTP/1.1" 200 OK
2023-02-06 19:12:08.696 | INFO     | register.middleware:intercept:22 - 访问记录:IP:::1-method:GET-url:http://localhost:38000/api/film_time_play
INFO:     ::1:53962 - "GET /api/country_film_score HTTP/1.1" 200 OK
INFO:     ::1:53957 - "GET /api/film_time_play HTTP/1.1" 200 OK
2023-02-06 19:12:10.584 | INFO     | utils.resp_code:resp_200:17 - 查询成功!
```

图 1-3　在控制台中显示的日志信息

在本地目录 logs 中保存日志文件，例如文件 2023-02-06_18-17-51.log 的日志信息如图 1-4 所示。

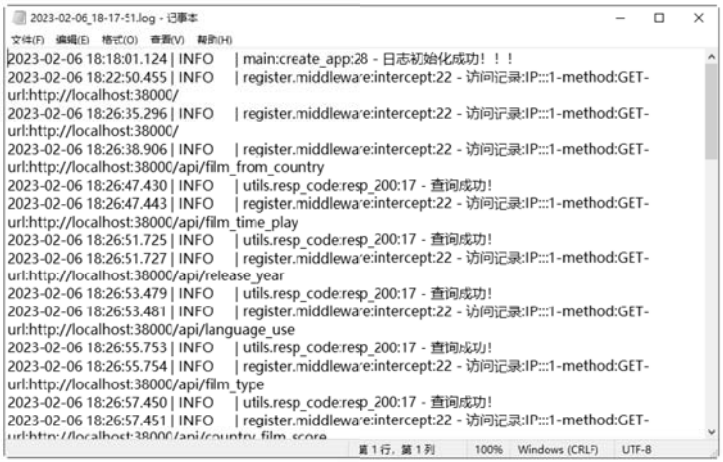

图 1-4　本地文件中的日志信息

1.6 前端数据可视化模块

本项目的前端主要负责数据可视化，源码保存在 data-view-master 目录中。本项目前端借助于 Vue 和 Echarts 技术，将后端的数据分析结果在前端以图表的方式展示出来。

扫码看视频

1.6.1 前端系统配置

本项目的前端系统配置功能由文件 vue.config.js 实现，设置了后端服务器、内容展示映射、网络映射、入口文件等内容。文件 vue.config.js 的主要实现代码如下所示。

```
module.exports = {
  // publicPath: '/data-view', // 根据情况自行修改
  publicPath: './',
  devServer: {
    port: 8999, //端口号
    open: true, //自动打开浏览器
  },
  configureWebpack: {
    resolve: {
      alias: {
        components: '@/components',
        content: '@/components/content',
        common: '@/components/common',
        assets: '@/assets',
        network: '@/network',
        views: '@/views',
        utils: '@/utils',
      },
    },
  },
  chainWebpack: config => {
    // 发布模式
    config.when(process.env.NODE_ENV === 'production', cofnig => {
      // 根据当前模式来决定使用哪个入口文件
      config.entry('app').clear().add('./src/main-prod.js')

      //  打包时排除指定包，手动添加 CDN
      config.set('externals', {
        vue: 'Vue',
        'vue-router': 'VueRouter',
        axios: 'axios',
        lodash: '_',
```

```
      echarts: 'echarts',
    })
    // 在 public 下的 index.html 中，可以通过以下命令拿到当前设置的值：
    // <%= htmlWebpackPlugin.options.isProd ? '' : 'dev-'%>
    config.plugin('html').tap(args => {
      args[0].isProd = true
      return args
    })
  })

  // 开发模式
  config.when(process.env.NODE_ENV === 'development', cofnig => {
    config.plugin('html').tap(args => {
      args[0].isProd = false
      return args
    })
    config.entry('app').clear().add('./src/main-dev.js')
  })
 },
}
```

1.6.2 前台主页

前台主页的实现文件是 Home.vue，具体实现流程如下所示。

(1) 展示 6 项数据分析的可视化结果：

- 电影上映年份趋势；
- 各国及地区电影出产量；
- 电影时长占比；
- 电影语言使用统计；
- 电影类别排行；
- 各国及地区电影评分展示。

对应的实现代码如下所示。

```
<template>
 <div class="screen-container" :style="containerStyle">
  <header class="screen-header">
   <div>
    <!-- <img :src="headerSrc" alt=""> -->
    <img v-show="theme == 'chalk'" src="~@/assets/images/header_border_dark.png" alt="" />
    <img v-show="theme != 'chalk'" src="~@/assets/images/header_border_light.png" alt="" />
   </div>
   <!-- <span class="logo"> <a :style="titleColor" href="https://www.bookbook.cc"
            title="去bookbook.cc主站" target="_blank">bookbook.cc</a> </span> -->
```

```
    <span class="title">电影市场数据分析展示</span>
    <div class="title-right">
      <!-- <img :src="themeSrc" class="qiehuan" @click="handleChangeTheme"
                    alt="切换主题" title="切换主题"> -->
      <img v-show="theme == 'chalk'" src="~@/assets/images/qiehuan_dark.png"
           class="qiehuan" @click="handleChangeTheme" alt="切换主题" title="切换主题" />
      <img v-show="theme != 'chalk'" src="~@/assets/images/qiehuan_light.png"
           class="qiehuan" @click="handleChangeTheme" alt="切换主题" title="切换主题" />
      <div class="datetime">{{ systemDateTime }}</div>
    </div>
  </header>
  <div class="screen-body">
    <section class="screen-left">
      <div id="left-top" :class="{ fullscreen: fullScreenStatus.trend }">

        <Trend ref="trend"></Trend>
        <div class="resize">
          <span @click="changeSize('trend')" :class="['iconfont', fullScreenStatus.trend ?
                    'icon-compress-alt' : 'icon-expand-alt']"></span>
        </div>
      </div>
      <div id="left-bottom" :class="{ fullscreen: fullScreenStatus.seller }">

        <Seller ref="seller"></Seller>
        <div class="resize">
          <span @click="changeSize('seller')" :class="['iconfont', fullScreenStatus.seller ?
                   'icon-compress-alt' : 'icon-expand-alt']"></span>
        </div>
      </div>
    </section>
    <section class="screen-middle">
      <div id="middle-top" :class="{ fullscreen: fullScreenStatus.map }">

        <single-map ref="map"></single-map>
        <div class="resize">
          <span @click="changeSize('map')" :class="['iconfont', fullScreenStatus.map ?
                    'icon-compress-alt' : 'icon-expand-alt']"></span>
        </div>
      </div>
      <div id="middle-bottom" :class="{ fullscreen: fullScreenStatus.rank }">

        <Rank ref="rank"></Rank>
        <div class="resize">
          <span @click="changeSize('rank')" :class="['iconfont', fullScreenStatus.rank ?
                    'icon-compress-alt' : 'icon-expand-alt']"></span>
        </div>
      </div>
    </section>
    <section class="screen-right">
      <div id="right-top" :class="{ fullscreen: fullScreenStatus.hot }">
```

```
        <Hot ref="hot"></Hot>
        <div class="resize">
          <span @click="changeSize('hot')" :class="['iconfont', fullScreenStatus.hot ?
                      'icon-compress-alt' : 'icon-expand-alt']"></span>
        </div>
      </div>
      <div id="right-bottom" :class="{ fullscreen: fullScreenStatus.stock }">

        <Stock ref="stock"></Stock>
        <div class="resize">
          <span @click="changeSize('stock')" :class="['iconfont', fullScreenStatus.stock ?
                     'icon-compress-alt' : 'icon-expand-alt']"></span>
        </div>
      </div>
    </section>
  </div>
 </div>
</template>
```

(2) 设置 6 项数据分析可视化结果对应的链接，代码如下。

```
<script>
import Hot from '@/components/report/Hot.vue'
import Map from '@/components/report/Map.vue'
import Rank from '@/components/report/Rank.vue'
import Seller from '@/components/report/Seller.vue'
import Stock from '@/components/report/Stock.vue'
import Trend from '@/components/report/Trend.vue'
```

通过 import 命令调用上述 6 个链接，在主页的 6 个选项卡区域中显示对应的数据分析结果。执行结果如图 1-5 所示。

图 1-5 前台主页执行结果

(3) 设置两种网页主题，单击右上角的图标，可以将页面切换为另外一种风格，代码如下。

```
import { mapState } from 'vuex'
// 导入自定义的主题工具函数，用于返回不同主题下的配置对象
import { getThemeValue } from 'utils/theme_utils'

export default {
  name: 'ScreenPage',
  components: {
    Hot,
    'single-map': Map,
    Rank,
    Seller,
    Stock,
    Trend,
  },
  created() {
    // 注册服务端广播的全屏事件
    // this.$socket.registerCallBack('fullScreen', this.recvData)
    // // 注册服务器广播的主题切换事件
    // this.$socket.registerCallBack('themeChange', this.recvThemeChange)
    this.currentTime(),
    this.handleChangeTheme()
  },
  computed: {
    ...mapState(['theme']),
    // 头部的边框路径
    headerSrc() {
      return '/static/img/' + getThemeValue(this.theme).headerBorderSrc
    },
    // 主题图片的路径
    themeSrc() {
      return '/static/img/' + getThemeValue(this.theme).themeSrc
    },
    containerStyle() {
      return {
        backgroundColor: getThemeValue(this.theme).backgroundColor,
        color: getThemeValue(this.theme).titleColor,
      }
    },
    titleColor() {
      return {
```

```
      color: getThemeValue(this.theme).titleColor,
    }
  },
},
destroyed() {
  // 组件销毁时，销毁事件
  // this.$socket.unRegisterCallBack('fullScreen')
  // this.$socket.unRegisterCallBack('themeChange')
  clearInterval(this.timerID)
},

  // 主题切换事件
  handleChangeTheme() {
    this.$store.commit('changeTheme')

    // this.$socket.send({
    //   action: 'themeChange',
    //   socketType: 'themeChange',
    //   chartName: '',
    //   value: '',
    // })
  },
  // 接收到服务器切换主题事件
  // recvThemeChange() {
  //   this.$store.commit('changeTheme')
  // },
  currentTime() {
    this.systemDateTime = new Date().toLocaleString()

    this.timerID && clearInterval(this.timerID)

    this.timerID = setInterval(() => {
      this.systemDateTime = new Date().toLocaleString()
    }, 1000)
  },
},
}
</script>
```

执行结果如图 1-6 所示。

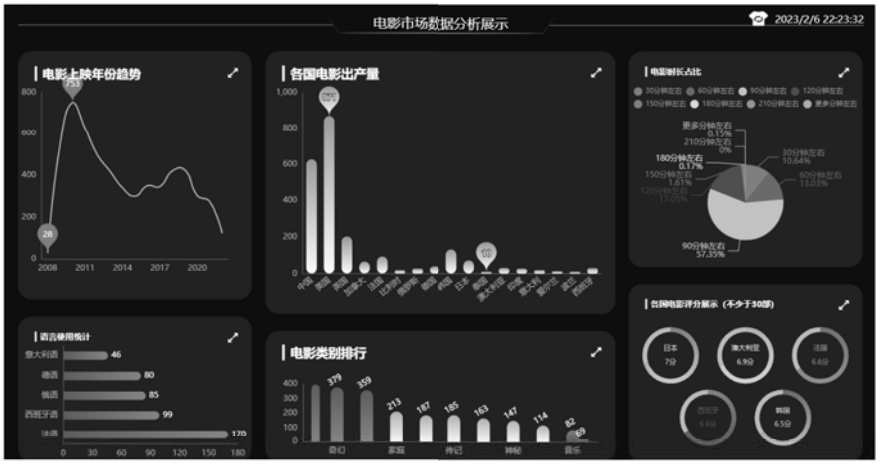

图 1-6　另外一种网页主题的执行结果

(4) 单击 6 个选项卡区域中的图标⤢，可以完成 6 个选项卡区域的全屏数据分析，对应的实现代码如下所示。

```
data() {
  return {
    // 各组件是否为全屏状态
    fullScreenStatus: {
      trend: false,
      seller: false,
      map: false,
      rank: false,
      hot: false,
      stock: false,
    },
    // 当前的系统时间
    systemDateTime: null,
    // 用于保存当前系统日期的定时器 id
    timerID: null,
  }
},
created() {
  // 注册服务端广播的全屏事件
  // this.$socket.registerCallBack('fullScreen', this.recvData)
  // // 注册服务器广播的主题切换事件
  // this.$socket.registerCallBack('themeChange', this.recvThemeChange)
  this.currentTime(),
  this.handleChangeTheme()
},
methods: {
```

```
    // 监听全屏事件
    changeSize(chartName) {
      // 1.改变 fullScreenStatus
      this.fullScreenStatus[chartName] = !this.fullScreenStatus[chartName]
      // 2.手动调用每个图表中的 screenAdapter 触发响应式
      this.$nextTick(() => {
        this.$refs[chartName].screenAdapter()
      })

      // 一端操作多端同步效果
      // 将事件发送给服务端，服务端广播事件为 true 则显示全屏，为 false 则取消全屏
      // const targetValue = !this.fullScreenStatus[chartName]
      // this.$socket.send({
      //   action: 'fullScreen',
      //   socketType: 'fullScreen',
      //   chartName: chartName,
      //   value: targetValue,
      // })
    },
    // 服务端广播全屏事件的客户端响应
    recvData(data) {
      // 取出一个图表进行切换
      const chartName = data.chartName
      // 判断切换成什么类型[true 表示全屏，false 表示取消全屏]
      const targetValue = data.value

      this.fullScreenStatus[chartName] = targetValue
      this.$nextTick(() => {
        this.$refs[chartName].screenAdapter()
      })
    },
<style lang="less" scoped>
// 全屏样式的定义
.fullscreen {
  position: fixed !important;
  top: 0 !important;
  left: 0 !important;
  width: 100% !important;
  height: 100% !important;
  margin: 0 !important;
  z-index: 9999;
}

.screen-container {
  width: 100%;
  height: 100%;
  padding: 0 20px;
```

```
  background-color: #161522;
  color: #fff;
  box-sizing: border-box;
}
.screen-header {
  width: 100%;
  height: 64px;
  font-size: 20px;
  position: relative;
  > div {
    img {
      width: 100%;
    }
  }
  .title {
    position: absolute;
    left: 50%;
    top: 50%;
    font-size: 20px;
    transform: translate(-50%, -50%);
  }
  .title-right {
    display: flex;
    align-items: center;
    position: absolute;
    right: 0px;
    top: 50%;
    transform: translateY(-80%);
  }
  .qiehuan {
    width: 28px;
    height: 21px;
    cursor: pointer;
  }
  .datetime {
    font-size: 15px;
    margin-left: 10px;
  }
  .logo {
    position: absolute;
    left: 0px;
    top: 50%;
    transform: translateY(-80%);
    a {
      text-decoration: none;
    }
  }
```

```
}
.screen-body {
  width: 100%;
  height: 100%;
  display: flex;
  margin-top: 10px;
  .screen-left {
    height: 100%;
    width: 27.6%;
    #left-top {
      height: 53%;
      position: relative;
    }
    #left-bottom {
      height: 31%;
      margin-top: 25px;
      position: relative;
    }
  }
  .screen-middle {
    height: 100%;
    width: 41.5%;
    margin-left: 1.6%;
    margin-right: 1.6%;
    #middle-top {
      width: 100%;
      height: 56%;
      position: relative;
    }
    #middle-bottom {
      margin-top: 25px;
      width: 100%;
      height: 28%;
      position: relative;
    }
  }
  .screen-right {
    height: 100%;
    width: 27.6%;
    #right-top {
      height: 46%;
      position: relative;
    }
    #right-bottom {
      height: 38%;
      margin-top: 25px;
      position: relative;
```

```
    }
  }
}
.resize {
  position: absolute;
  right: 20px;
  top: 20px;
  cursor: pointer;
}
</style>
```

1.6.3 电影时长占比图

前台文件 Hot.vue 用于可视化展示电影时长占比图，主要实现代码如下所示。

```
export default {
  //电影时长占比(右上)
  name: 'Hot',
  data() {
    return {
      // 图表的实例对象
      chartInstance: null,
      // 从服务器中获取的所有数据
      allData: null,
      // 当前显示的一级分类数据类型
      currentIndex: 0,
      // 字体响应式大小
      titleFontSize: null,
    }
  },
  created() {
    // this.$socket.registerCallBack('hotData', this.getData)
    this.getData()
  },
  computed: {
    ...mapState(['theme']),
    cateName() {
      if (!this.allData) return ''
      return this.allData[this.currentIndex].name
    },
    themeStyle() {
      if (!this.titleFontSize) {
        return { color: getThemeValue(this.theme).titleColor }
      }
      return {
        fontSize: this.titleFontSize + 'px',
```

```
      color: getThemeValue(this.theme).titleColor,
    }
  },
},
watch: {
  theme() {
    // 销毁当前的图表
    this.chartInstance.dispose()
    // 以最新主题初始化图表对象
    this.initChart()
    // 屏幕适配
    this.screenAdapter()
    // 渲染数据
    this.updateChart()
  },
},
mounted() {
  this.initChart()
  this.getData()
  window.addEventListener('resize', this.screenAdapter)
  // 主动触发响应式配置
  this.screenAdapter()
},
destroyed() {
  window.removeEventListener('resize', this.screenAdapter)
},
methods: {
  // 初始化图表的方法
  initChart() {
    this.chartInstance = this.$echarts.init(this.$refs.hotRef, this.theme)
    const initOption = {
      title: {
        text: '▌电影时长占比',
        left: 20,
        top: 20,
      },
      legend: {
        top: '15%',
        // 图标类型：圆形
        icon: 'circle',
      },
      tooltip: {
        show: true,
        // formatter:'hhh'
        formatter: arg => {
          // 拿到三级分类的数据
          const thirdCategory = arg.data.children
```

```
        // 计算所有三级分类的数值总和，才能计算出百分比
        let total = 0
        thirdCategory.forEach(item => {
          total += item.value
        })
        // 显示的文本
        let showStr = ''
        thirdCategory.forEach(item => {
          showStr += '${item.name}: ${_.round((item.value / total) * 100, 2)}% <br/>'
        })
        return showStr
      },
    },
    series: [
      {
        type: 'pie',
        label: {
          show: true,
          formatter:'{b}\n{d}%'
        },
        // 高亮状态下的样式
        emphasis: {
          labelLine: {
            // 连接文字的线条
            show: true,
          },
        },
      },
    ],
  }
  this.chartInstance.setOption(initOption)
},
// 发送请求，获取数据
async getData() {
  const { data: res } = await this.$http.get('/film_time_play')
  this.allData = res.back
  console.log(this.allData)
  this.updateChart()
},
// 更新图表配置项
updateChart() {
  const dataOption = {
    legend: {
      data: '',
    },
    series: [
      {
        data: this.allData,
```

```
      },
    ],
  }
  this.chartInstance.setOption(dataOption)
},
// 不同分辨率的响应式
screenAdapter() {
  this.titleFontSize = (this.$refs.hotRef.offsetWidth / 100) * 3.6

  const adapterOption = {
    title: {
      textStyle: {
        fontSize: this.titleFontSize,
      },
    },
    legend: {
      itemWidth: this.titleFontSize,
      itemHeight: this.titleFontSize,
      // 图例的间隔
      itemGap: this.titleFontSize / 2,
      textStyle: {
        fontSize: this.titleFontSize / 1.2,
      },
    },
    series: [
      {
        // 饼图的半径
        radius: this.titleFontSize * 4.5,
        // 控制饼图的位置 x,y
        center: ['50%', '70%'],
      },
    ],
  }
  this.chartInstance.setOption(adapterOption)
  this.chartInstance.resize()
},
// 单击左侧按钮
toLeft() {
  this.currentIndex--
  // 已到达最左边
  if (this.currentIndex < 0) this.currentIndex = this.allData.length - 1
  this.updateChart()
},
// 单击右侧按钮
toRight() {
  this.currentIndex++
  // 已到达最右边
  if (this.currentIndex > this.allData.length - 1) this.currentIndex = 0
  this.updateChart()
```

```
    },
  },
}
</script>

<style lang="less" scoped>
.com-container {
  i {
    z-index: 999;
    position: absolute;
    transform: translateY(-50%);
    top: 50%;
    cursor: pointer;
  }
  i.icon-left {
    left: 10%;
  }
  i.icon-right {
    right: 10%;
  }
  .cate-name {
    position: absolute;
    right: 10%;
    bottom: 20px;
    z-index: 999;
  }
}
</style>
```

执行结果如图 1-7 所示。

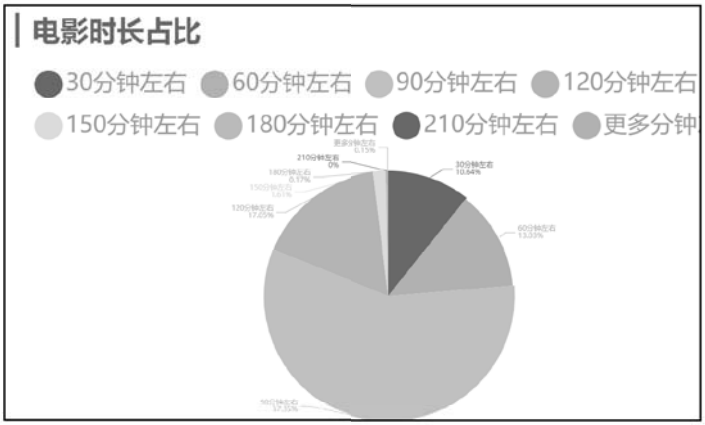

图 1-7　电影时长占比图

1.6.4 电影上映年份趋势图

前台文件 Trend.vue 用于可视化展示电影上映年份趋势图，主要实现代码如下所示。

```
<template>
  <div class="com-container">
    <div class="com-chart" ref="trendRef"></div>
  </div>
</template>

<script>
import { mapState } from 'vuex'
import { getThemeValue } from 'utils/theme_utils'

export default {
  // 电影上映年份趋势(左上)
  name: 'Trend',
  data() {
    return {
      // 图表的实例对象
      chartInstance: null,
      // 从服务器中获取的所有数据
      allData: null,
      // 是否显示可选项
      showMenu: false,
      // 默认显示的数据类型
      activeName: 'map',
      // 指明标题的字体大小
      titleFontSize: 0,
      value: ''
    }
  },
  created() {
    // 在组件创建完成之后，进行回调函数的注册
    // this.$socket.registerCallBack('trendData', this.getData)
  },
  computed: {
    ...mapState(['theme']),
    // 单击过后需要显示的数组
    selectTypes() {
      if (!this.allData) return []
      // 过滤掉当前选中的类别
      return this.allData.type.filter(item => item.key !== this.activeName)
    },
    // 显示的标题
```

```
  showTitle() {
    if (!this.allData) return ''
    return this.allData[this.activeName].title
  },
  // 设置标题的样式
  comStyle() {
    return {
      fontSize: this.titleFontSize + 'px',
      color: getThemeValue(this.theme).titleColor
    }
  }
},
watch: {
  theme() {
    // 销毁当前的图表
    this.chartInstance.dispose()
    // 以最新主题初始化图表对象
    this.initChart()
    // 屏幕适配
    this.screenAdapter()
    // 渲染数据
    this.updateChart()
  }
},
mounted() {
  this.initChart()
  this.getData()
  window.addEventListener('resize', this.screenAdapter)
  // 主动触发响应式配置
  this.screenAdapter()
},
destroyed() {
  window.removeEventListener('resize', this.screenAdapter)
  // 销毁注册的事件
  this.$socket.unRegisterCallBack('trendData')
},
methods: {
  // 初始化图表的方法
  initChart() {
    this.chartInstance = this.$echarts.init(this.$refs.trendRef, this.theme)
    const initOption = {
      title: {
        text: '▌电影上映年份趋势',
        left: 20,
        top: 20,
      },
      // 工具提示
```

```
      tooltip: {
        // 当鼠标指针移入坐标轴的提示
        trigger: 'axis'
      },
      legend: {
        left: 'center',
        top: '50%',
        // 图例的 icon 类型
        icon: 'circle'
      },

    }
    this.chartInstance.setOption(initOption)
  },
  // 发送请求，获取数据  //websocket：  realData 服务端发送给客户端需要的数据
  async getData() {
    const { data: res } = await this.$http.get('/release_year')
    this.allData = res.back
    this.updateChart()
  },
  // 更新图表配置项
  updateChart() {
  const dataOption = {
  xAxis: {
  type: 'category',
  data: this.allData.year
},
yAxis: {
  type: 'value'
},
series: [
  {
    data: this.allData.data,
    type: 'line',
    smooth: true,
    markPoint:{
      data:[{
        type:'max'
      },{type:'min'}],
    },
    lineStyle: {
    color: "rgba(106, 202, 249, 1)"
  },
  }
]
    }
    this.chartInstance.setOption(dataOption)
```

```
    },
    // 不同分辨率的响应式
    screenAdapter() {
      // 测算出来的合适的字体大小
      this.titleFontSize = (this.$refs.trendRef.offsetWidth / 100) * 3.6

      const adapterOption = {
        legend: {
          itemWidth: this.titleFontSize,
          itemHeight: this.titleFontSize,
          // 间距
          itemGap: this.titleFontSize,
          textStyle: {
            fontSize: this.titleFontSize / 1.3
          }
        }
      }
      this.chartInstance.setOption(adapterOption)
      this.chartInstance.resize()
    },
    // 当前选中的类型
    handleSelect(currentType) {
      this.activeName = currentType
      this.updateChart()
    }
  }
}
</script>

<style lang="less" scoped>
.title {
  position: absolute;
  left: 50px;
  top: 20px;
  z-index: 999;
  color: white;
  cursor: pointer;

  .before-icon {
    position: absolute;
    left: -20px;
  }
  .title-icon {
    margin-left: 10px;
  }
}
</style>
```

执行结果如图 1-8 所示。

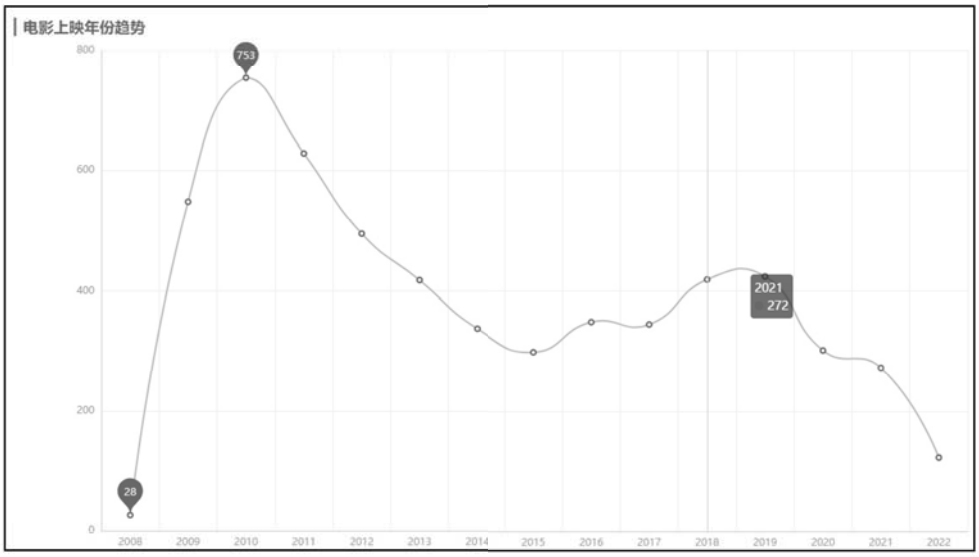

图 1-8 电影上映年份趋势图

1.6.5 各国及地区电影出产量统计图

前台文件 Map.vue 用于可视化展示各国及地区电影出产量统计图，主要实现代码如下所示。

```
<script>
import { mapState } from 'vuex'

export default {
  //各国及地区电影出产量(中上)
  name: 'Map',
  data() {
    return {
      // axios 实例对象
      axiosInstance: null,
      // 图表的实例对象
      chartInstance: null,
      // 从服务器中获取的所有数据
      allData: null,
      // 获取各国及地区矢量地图数据缓存
      cityMapData: {},
    }
```

```
},
computed: {
  ...mapState(['theme']),
},
watch: {
  theme() {
    // 销毁当前的图表
    this.chartInstance.dispose()
    // 以最新主题初始化图表对象
    this.initChart()
    // 屏幕适配
    this.screenAdapter()
    // 渲染数据
    this.updateChart()
  },
},
created() {
  this.getData()
},
mounted() {
  this.initChart()
  window.addEventListener('resize', this.screenAdapter)
  // 主动触发响应式配置
  this.screenAdapter()
},
destroyed() {
  window.removeEventListener('resize', this.screenAdapter)
},
methods: {
  // 初始化图表的方法
  async initChart() {
    this.chartInstance = this.$echarts.init(this.$refs.mapRef, this.theme)
    const initOption = {
      title: {
        text: '▌各国及地区电影出产量',
        left: 20,
        top: 20,
      },

    }
    this.chartInstance.setOption(initOption)
  },
  // 发送请求，获取数据
  async getData() {
    // http://101.34.160.195:8888/api/map
    const { data: res } = await this.$http.get('/film_from_country')
    this.allData = res.back
```

```
    console.log(this.allData)
    this.updateChart()
  },
  // 更新图表配置项
  updateChart() {
    const radiusdata= this.allData.map(item => item.name)
    const serdata= this.allData.map(item => item.value)
    console.log("123456",radiusdata,serdata)
    // 数据配置项
    const dataOption = {
      tooltip: {
                trigger: 'axis',
                axisPointer: {
                   type: 'shadow'
  }
 },
 grid: {
  left: '3%',
  right: '4%',
  bottom: '3%',
  containLabel: true
 },
 xAxis: [
  {
    type: 'category',
    data: radiusdata,
    axisTick: {
      alignWithLabel: true
    },
    axisLabel: {
    rotate:40,
}
  }
 ],
 yAxis: [
  {
    type: 'value'
  }
 ],
 series: [
  {
    type: 'bar',
    barWidth: '60%',
    data: serdata,
    markPoint:{
      data:[{type:'max',name:'最大值'},{type:'min',name:'最小值'}]
    },
```

```
    itemStyle:{
      barBorderRadius: 10,
      color: new this.$echarts.graphic.LinearGradient(
                              0, 0, 0, 1,
                              [{
                                    offset: 0,
                                    color: '#00C78C'
                                },
                                {
                                    offset: 1,
                                    color: '#FFFFCD'

                                }
                              ]
                            )
    }
  }
]
    }
    this.chartInstance.setOption(dataOption)
  },
  // 不同分辨率的响应式
  screenAdapter() {
    // 当前比较合适的字体大小
    const titleFontSize = (this.$refs.mapRef.offsetWidth / 100) * 3.6

    // 响应式的配置项
    const adapterOption = {
      title: {
        textStyle: {
          fontSize: titleFontSize,
        },
      },
      legend: {
        // 图例宽度
        itemWidth: titleFontSize / 2,
        // 图例高度
        itemHeight: titleFontSize / 2,
        // 间隔
        itemGap: titleFontSize / 2,
        textStyle: {
          fontSize: titleFontSize / 2,
        },
      },
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
```

```
    },
  },
}
</script>

<style lang="less" scoped></style>
```

执行结果如图 1-9 所示。

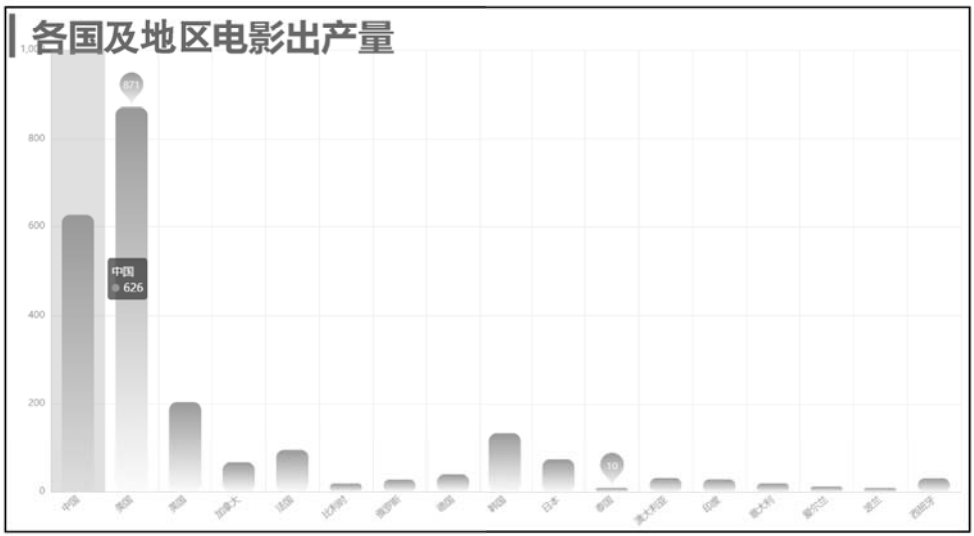

图 1-9　各国及地区电影出产量统计图

1.6.6　电影类别排行统计图

前台文件 Rank.vue 用于可视化展示电影类别排行统计图，主要实现代码如下所示。

```
<script>
import { mapState } from 'vuex'

export default {
  // 电影类别排行(中下)
  name: 'Rank',
  data() {
    return {
      // 图表的实例对象
      chartInstance: null,
      // 从服务器中获取的所有数据
      allData: null,
      // 柱形图：区域缩放起点值
```

```
    startValue: 0,
    // 柱形图：区域缩放终点值
    endValue: 9,
    // 定时器
    timerId: null
  }
},
created() {
},
computed: {
  ...mapState(['theme'])
},
watch: {
  theme() {
    // 销毁当前的图表
    this.chartInstance.dispose()
    // 以最新主题初始化图表对象
    this.initChart()
    // 屏幕适配
    this.screenAdapter()
    // 渲染数据
    this.updateChart()
  }
},
mounted() {
  this.initChart()
  this.getData()
  window.addEventListener('resize', this.screenAdapter)
  // 主动触发响应式配置
  this.screenAdapter()
},
destroyed() {
  window.removeEventListener('resize', this.screenAdapter)
  clearInterval(this.timerId)
},
methods: {
  // 初始化图表的方法
  initChart() {
    this.chartInstance = this.$echarts.init(this.$refs.rankRef, this.theme)

    const initOption = {
      title: {
        text: '▌电影类别排行',
        left: 20,
        top: 20
      },
      grid: {
```

```
      top: '40%',
      left: '5%',
      right: '5%',
      bottom: '5%',
      // 把 x 轴和 y 轴纳入 grid
      containLabel: true
    },
    tooltip: {
      show: true
    },
    xAxis: {
      type: 'category'
    },
    yAxis: {
      value: 'value'
    },
    series: [
      {
        type: 'bar',
        label: {
          show: true,
          position: 'top',
          color: 'white',
          rotate: 30
        }
      }
    ]
  }
  this.chartInstance.setOption(initOption)

  // 光标经过，关闭动画效果
  this.chartInstance.on('mouseover', () => {
    clearInterval(this.timerId)
  })
  // 光标离开，开启动画效果
  this.chartInstance.on('mouseout', () => {
    this.startInterval()
  })
},
// 发送请求，获取数据
async getData() {
  const { data: res } = await this.$http.get('/film_type')
  this.allData = res.back
  // 对数据进行排序(从大到小)
  // this.allData.sort((a, b) => b.value - a.value)

  this.updateChart()
```

```
    // 开始自动切换
    this.startInterval()
  },
  // 更新图表配置项
  updateChart() {
    // 渐变色数组
    const colorArr = [
      ['#0BA82C', '#4FF778'],
      ['#2E72BF', '#23E5E5'],
      ['#5052EE', '#AB6EE5'],
      ['#F4A460', '#FDF5E6'],
      ['#1E90FF', '#3D59AB']
    ]
    // const colorArr = [
    //   ['#b8e994', '#079992'],
    //   ['#82ccdd', '#0a3d62'],
    //   ['#f8c291', '#b71540'],
    // ]
    // 所有省份组成的数组
    const provinceInfo = this.allData.map(item => item[0])
    // 所有省份对应的销售金额
    const valueArr = this.allData.map(item => item[1])

    const dataOption = {
      xAxis: {
        data: provinceInfo
      },
      dataZoom: {
        // 区域缩放组件
        show: false,
        startValue: this.startValue,
        endValue: this.endValue
      },
      series: [
        {
          data: valueArr,
          itemStyle: {
            color: arg => {
              let targetColorArr = null

              if (arg.value > 1000) {
                targetColorArr = colorArr[0]
              } else if (arg.value > 500) {
                targetColorArr = colorArr[1]
              }else if (arg.value >300){
                targetColorArr = colorArr[2]
              }else if (arg.value >100){
```

```
            targetColorArr =colorArr[3]
          }
          else {
            targetColorArr = colorArr[4]
          }

          return new this.$echarts.graphic.LinearGradient(0, 0, 0, 1, [
            // 0%
            { offset: 0, color: targetColorArr[0] },
            // 100%
            { offset: 1, color: targetColorArr[1] }
          ])
        }
      }
    }
  ]
 }
 this.chartInstance.setOption(dataOption)
},
// 根据图表容器的宽度计算各属性、标签、元素的大小
screenAdapter() {
 const titleFontSzie = (this.$refs.rankRef.offsetWidth / 100) * 3.6

 const adapterOption = {
  title: {
   textStyle: {
    fontSize: titleFontSzie
   }
  },
  series: [
   {
    barWidth: titleFontSzie,
    itemStyle: {
     barBorderRadius: [titleFontSzie / 2, titleFontSzie / 2, 0, 0]
    }
   }
  ]
 }
 this.chartInstance.setOption(adapterOption)
 this.chartInstance.resize()
},
// 改变柱形图区域缩放起始点值与终点值的函数
startInterval() {
 // 如果存在则关闭
 this.timerId && clearInterval(this.timerId)

 this.timerId = setInterval(() => {
```

```
        this.startValue++
        this.endValue++
        if (this.endValue > this.allData.length - 1) {
          this.startValue = 0
          this.endValue = 9
        }
        this.updateChart()
      }, 2000)
    }
  }
}
</script>

<style lang="less" scoped></style>
```

执行结果如图 1-10 所示。

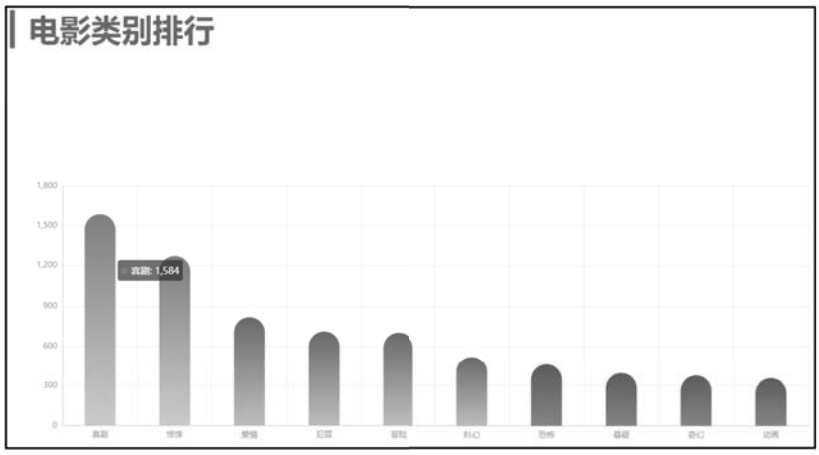

图 1-10　电影类别排行统计图

1.6.7　电影语言使用统计图

前台文件 Seller.vue 用于可视化展示电影语言使用统计图，主要实现代码如下所示。

```
<script>
import { mapState } from 'vuex'
import { getThemeValue } from 'utils/theme_utils'

export default {
  // 语言使用统计(左下)
  name: 'Seller',
  data() {
```

```
    return {
      // echarts 实例对象
      chartInstance: null,
      // 服务器返回的数据
      allData: null,
      // 当前显示的页数
      curretnPage: 1,
      // 总页数
      totalPage: 0,
      // 定时器标识
      timerId: null,
      // 当鼠标移入 axis(坐标轴)时展示底层的背景色
      PointerColor: this.axisPointerColor,
    }
  },
  created() {
    // this.$socket.registerCallBack('sellerData', this.getData)
  },
  computed: {
    ...mapState(['theme']),
    axisPointerColor() {
      return getThemeValue(this.theme).sellerAxisPointerColor
    },
  },
  watch: {
    theme() {
      // 销毁当前的图表
      this.chartInstance.dispose()
      // 以最新主题初始化图表对象
      this.initChart()
      // 屏幕适配
      this.screenAdapter()
      // 渲染数据
      this.updateChart()
    },
  },
  mounted() {
    // 由于初始化使用到了 DOM 元素，因此需要在 mounted 生命周期内调用
    this.initChart()
    this.getData()
    // 在界面加载完成时，主动对屏幕进行适配
    this.screenAdapter()
    window.addEventListener('resize', this.screenAdapter)
  },
  // 实例销毁后触发
  destroyed() {
    clearInterval(this.timeID)
```

```
  // 在组件销毁的时候，把监听器取消
  window.removeEventListener('resize', this.screenAdapter)
  // this.$socket.unRegisterCallBack('sellerData')
},
methods: {
  // 初始化 echartsInstance 对象
  initChart() {
    this.chartInstance = this.$echarts.init(this.$refs.sellerRef, this.theme)
    // 对图表初始化的配置
    const initOption = {
      title: {
        text: '▌语言使用统计',
        left: 20,
        top: 20,
      },
      grid: {
        top: '20%',
        left: '3%',
        right: '6%',
        bottom: '3%',
        // 默认grid不包含坐标轴文字，改为true
        containLabel: true,
      },
      xAxis: {
        type: 'value',
      },
      yAxis: {
        type: 'category',
      },
      tooltip: {
        // 当光标移入axis(坐标轴)时展示底层的背景色
        trigger: 'axis',
        axisPointer: {
          // 展示的类型是线条类型
          type: 'line',
          lineStyle: {
            color: this.axisPointerColor,
          },
          // 将tooltip的层级设置为最底层
          z: 0,
        },
      },
      series: [
        {
          type: 'bar',
          label: {
            show: true,
```

```
          position: 'right',
          textStyle: {
            color: 'white',
          },
        },
        // 每一个柱的样式
        itemStyle: {
          // 创建 echarts 全局对象的一个线性渐变方法
          // 指明方向(第四象限坐标轴)，以及不同百分比时的颜色值
          color: new this.$echarts.graphic.LinearGradient(0, 0, 1, 0, [
            // 0% 状态时的颜色
            { offset: 0, color: '#5052EE' },
            // 100% 状态时的颜色
            { offset: 1, color: '#AB6EE5' },
          ]),
        },
      },
    ],
  }

  this.chartInstance.setOption(initOption)

  // 在图表事件中进行鼠标事件的监听
  this.chartInstance.on('mouseover', () => {
    this.timerId && clearInterval(this.timerId)
  })
  this.chartInstance.on('mouseout', () => {
    this.startInterval()
  })
},
// 获取服务器数据
async getData() {
  // http://101.34.160.195:8888/api/seller
  const { data: res } = await this.$http.get('/language_use')

  this.allData = res.back
  // 对数组排序：从小到大
  // this.allData.sort((a, b) => b[1] - a[1])
  // 每五个元素显示一页，计算出总页数
  this.totalPage = Math.ceil(this.allData.length / 5)

  // 开始第一次渲染
  this.updateChart()
  // 开启定时器，开始动态渲染
  this.startInterval()
},
// 更新图表
```

```
updateChart() {
  // 从数组中动态取出 5 条数据
  const start = (this.curretnPage - 1) * 5
  const end = this.curretnPage * 5
  const showData = this.allData.slice(start, end)

  // y 轴上的数据
  const sellerNames = showData.map(item => item[0])
  // x 轴上的数据
  const sellerValues = showData.map(item => item[1])

  // 当拿到数据后，准备数据的配置项
  const dataOption = {
    yAxis: {
      data: sellerNames,
    },
    series: [
      {
        data: sellerValues,
      },
    ],
  }

  // 设置数据
  this.chartInstance.setOption(dataOption)
},
// 开启动态渲染的定时器
startInterval() {
  // 一般使用定时器都有一个保险操作，先关闭再开启
  this.timerId && clearInterval(this.timerId)

  this.timerId = setInterval(() => {
    this.curretnPage++
    // 当超出最大页数时，回滚到第一页
    if (this.curretnPage > this.totalPage) this.curretnPage = 1

    this.updateChart()
  }, 3000)
},
// 当浏览器窗口大小发生变化，完成屏幕适配
screenAdapter() {
  const titleFontSize = (this.$refs.sellerRef.offsetWidth / 100) * 3.6
  // 浏览器分辨率大小的相关配置项
  const adapterOption = {
    title: {
      textStyle: {
        fontSize: titleFontSize,
```

```
        },
      },
      tooltip: {
        axisPointer: {
          lineStyle: {
            width: titleFontSize,
          },
        },
      },
      series: [
        {
          barWidth: titleFontSize,
          itemStyle: {
            barBorderRadius: [0, titleFontSize / 2, titleFontSize / 2, 0],
          },
        },
      ],
    }
    this.chartInstance.setOption(adapterOption)
    // 手动调用图表的 resize 方法才能产生效果
    this.chartInstance.resize()
  },
 },
}
</script>

<style lang="less" scoped></style>
```

执行结果如图 1-11 所示。

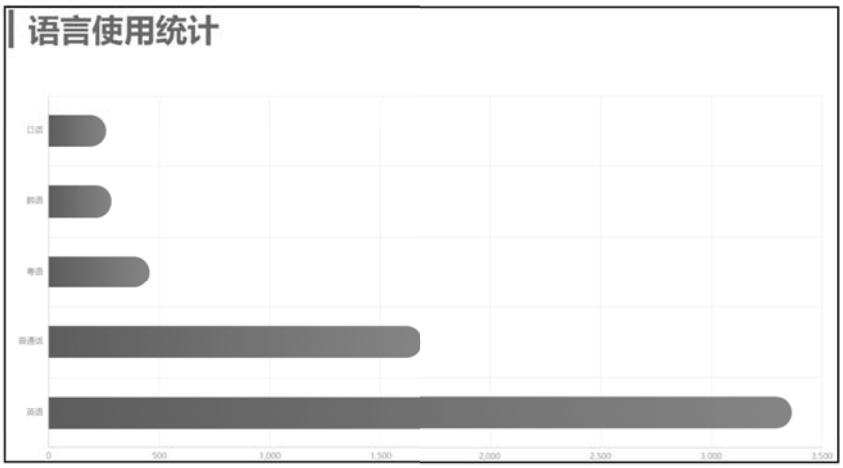

图 1-11　电影语言使用统计图

1.6.8 各国及地区电影评分展示统计图

前台文件 Stock.vue 用于可视化展示各国及地区电影评分展示统计图，主要实现代码如下所示。

```
<script>
import { mapState } from 'vuex'

export default {
  // 各国及地区电影评分展示(右下)
  name: 'Stock',
  data() {
    return {
      // 图表的实例对象
      chartInstance: null,
      // 从服务器中获取的所有数据
      allData: null,
      // 当前显示数据的页数
      currentIndex: 1,
      // 定时器标识
      timerId: null,
      // 圆环坐标
      centerArr: [
        ['18%', '40%'],
        ['50%', '40%'],
        ['82%', '40%'],
        ['34%', '75%'],
        ['66%', '75%'],
      ],
      // 圆环渐变色
      colorArr: [
        ['#4FF778', '#0BA82C'],
        ['#E5DD45', '#E8B11C'],
        ['#E8821C', '#E55445'],
        ['#5052EE', '#AB6EE5'],
        ['#23E5E5', '#2E72BF'],
      ],
    }
  },
  created() {
    // this.$socket.registerCallBack('stockData', this.getData)
  },
  computed: {
    ...mapState(['theme']),
  },
```

```
watch: {
  theme() {
    // 销毁当前的图表
    this.chartInstance.dispose()
    // 以最新主题初始化图表对象
    this.initChart()
    // 屏幕适配
    this.screenAdapter()
    // 渲染数据
    this.updateChart()
  },
},
mounted() {
  this.initChart()
  this.getData()
  // this.$socket.send({
  //   action: 'getData',
  //   socketType: 'stockData',
  //   chartName: 'stock',
  //   value: '',
  // })
  window.addEventListener('resize', this.screenAdapter)
  // 主动触发响应式配置
  this.screenAdapter()
},
destroyed() {
  window.removeEventListener('resize', this.screenAdapter)
  clearInterval(this.timerId)
  // this.$socket.unRegisterCallBack('stockData')
},
methods: {
  // 初始化图表的方法
  initChart() {
    this.chartInstance = this.$echarts.init(this.$refs.stockRef, this.theme)
    const initOption = {
      title: {
        text: '▌各国及地区电影评分展示(不少于30部)',
        left: 20,
        top: 20,
      },
    }
    this.chartInstance.setOption(initOption)

    this.chartInstance.on('mouseover', () => {
      clearInterval(this.timerId)
    })
    this.chartInstance.on('mouseout', this.startInterval)
```

```
},
// 发送请求，获取数据
async getData() {
  const { data: res } = await this.$http.get('/country_film_score')
  this.allData = res.back
  this.updateChart()
},
// 更新图表配置项
updateChart() {
  // 需要显示的原始数据，包含 0，不包含 5
  const start = (this.currentIndex - 1) * 5
  const end = start + 5
  const showData = this.allData.slice(start, end)
  // 真实显示的数据
  let seriesArr = showData.map((item, index) => {
    return {
      type: 'pie',
      // 设置成圆环图，外圆半径、内圆半径在响应式处指定
      // radius: [120, 100],

      // 饼图的位置
      center: this.centerArr[index],
      // 关闭光标移入到饼图的动画效果
      hoverAnimation: false,
      // 隐藏指示线条
      labelLine: {
        show: false,
      },
      label: {
        position: 'center',
        color: this.colorArr[index][0],
      },
      data: [
        // 销量
        {
          name: item[0]+'\n'+'\n'+item[1]+'分',
          value: item[1],
          itemStyle: {
            // 创建线性渐变的颜色：从下往上
            color: new this.$echarts.graphic.LinearGradient(0, 1, 0, 0, [
              // 0%
              { offset: 0, color: this.colorArr[index][0] },
              // 100%
              { offset: 1, color: this.colorArr[index][1] },
            ]),
          },
          // 内部的提示框，c 代表数值，d 代表百分比
```

```
        tooltip: {
          formatter: '${item[0]} <br/>评分：{c}',
        },
      },
      // 库存
      {
        value: 10-item[1],
        itemStyle: {
          color: '#bbb',
        },
        // 内部的提示框
        tooltip: {
          formatter: '${item[0]} <br/>距离满分还差：{c}分',
        },
      },
    ],
  }
})

const dataOption = {
  tooltip: {
    // 这里的 item 可以为内部的数据开启单独的 tooltip
    trigger: 'item',
  },
  series: seriesArr,
}
this.chartInstance.setOption(dataOption)

// 开启定时切换
this.startInterval()
},
// 不同分辨率的响应式
screenAdapter() {
const titleFontSize = (this.$refs.stockRef.offsetWidth / 100) * 3.6
// 圆的内圆半径和外圆半径
const innerRadius = titleFontSize * 2.8
const outerRadius = innerRadius * 1.2

const adapterOption = {
  title: {
    textStyle: {
      fontSize: titleFontSize,
    },
  },
  series: [
    {
      type: 'pie',
```

```
        radius: [outerRadius, innerRadius],
        label: {
          fontSize: titleFontSize / 1.2,
        },
      },
      {
        type: 'pie',
        radius: [outerRadius, innerRadius],
        label: {
          fontSize: titleFontSize / 1.2,
        },
      },
      {
        type: 'pie',
        radius: [outerRadius, innerRadius],
        label: {
          fontSize: titleFontSize / 1.2,
        },
      },
      {
        type: 'pie',
        radius: [outerRadius, innerRadius],
        label: {
          fontSize: titleFontSize / 1.2,
        },
      },
      {
        type: 'pie',
        radius: [outerRadius, innerRadius],
        label: {
          fontSize: titleFontSize / 1.2,
        },
      },
    ],
  }
  this.chartInstance.setOption(adapterOption)
  this.chartInstance.resize()
},
// 定时器不断切换当前页数
startInterval() {
  this.timerId && clearInterval(this.timerId)

  this.timerId = setInterval(() => {
    this.currentIndex++
    if (this.currentIndex > 2) this.currentIndex = 1
    // 在更新完数据后，需要更新页面
    this.updateChart()
```

```
      }, 5000)
    },
  },
}
</script>

<style lang="less" scoped>
</style>
```

执行结果如图 1-12 所示。

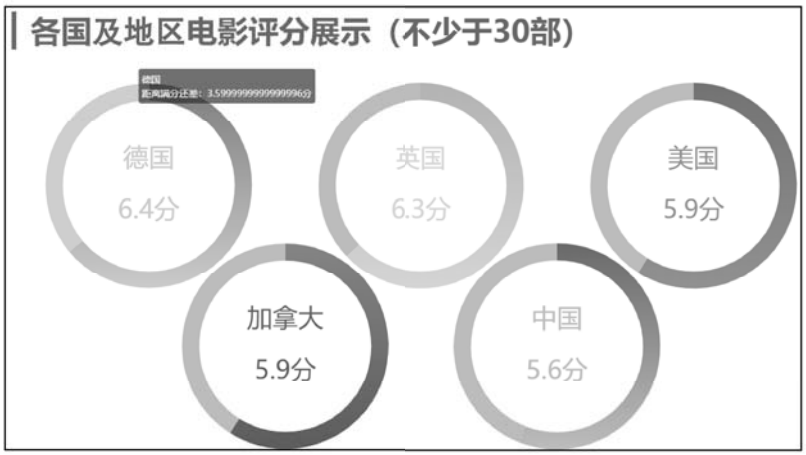

图 1-12　各国及地区电影评分展示统计图