实战 2.2 豆瓣电影客户端

一、实验目的

1. 了解 DevEco Studio 的使用
2. 学习 ArkTS 语言及 ArkUI 组件
3. 编写代码
4. 编译运行
5. 在模拟器上运行

二、实验原理

1. 鸿蒙开发原理
2. ArkTS，ArkUI 开发原理
3. 鸿蒙应用运行原理

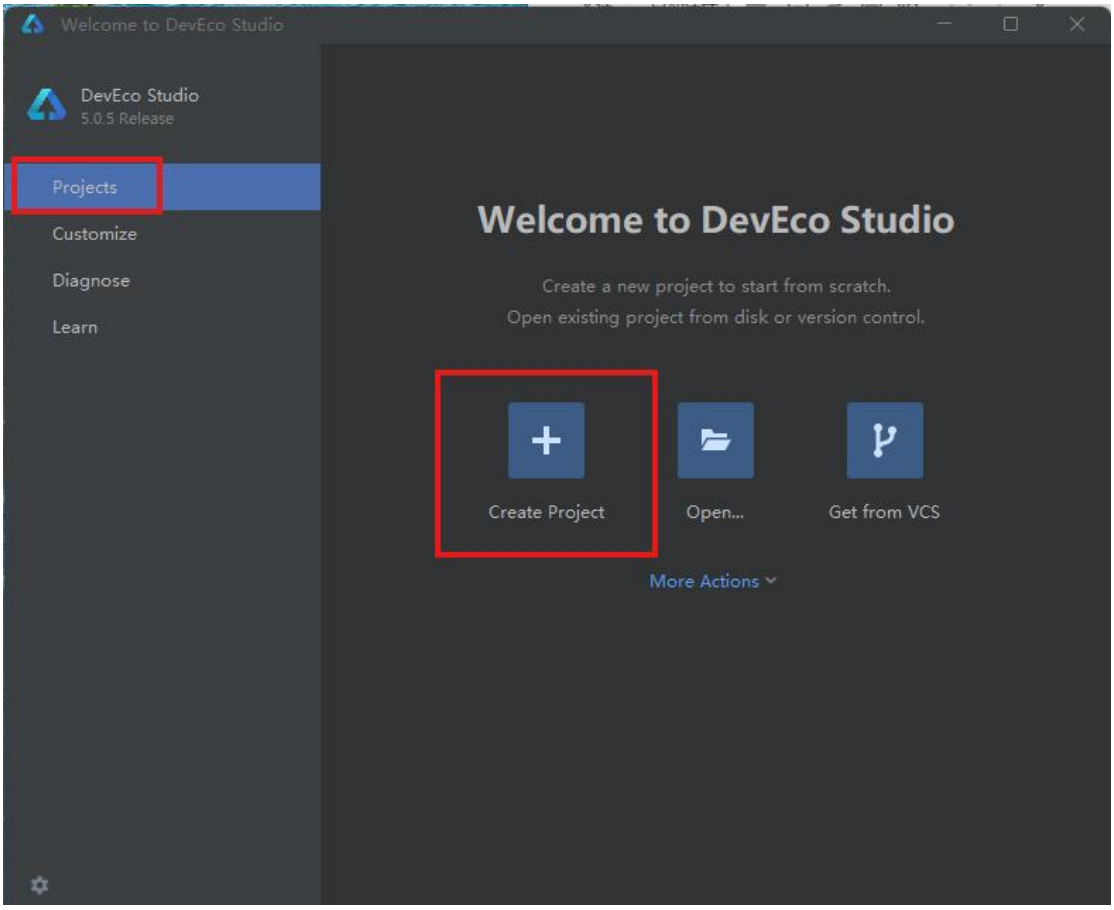
三、实验仪器材料

1. 计算机实训室电脑一台
2. DevEco Studio 开发环境及鸿蒙手机模拟器

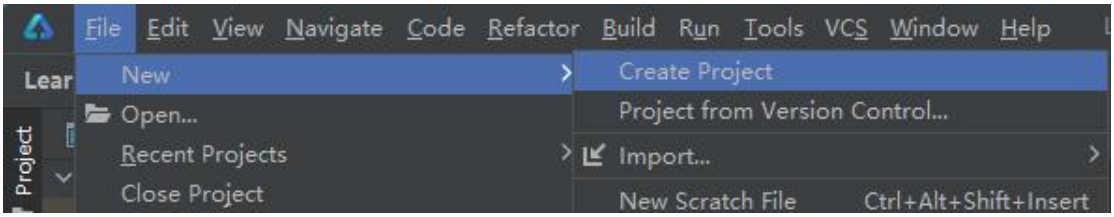
四、实验步骤

1. 创建项目

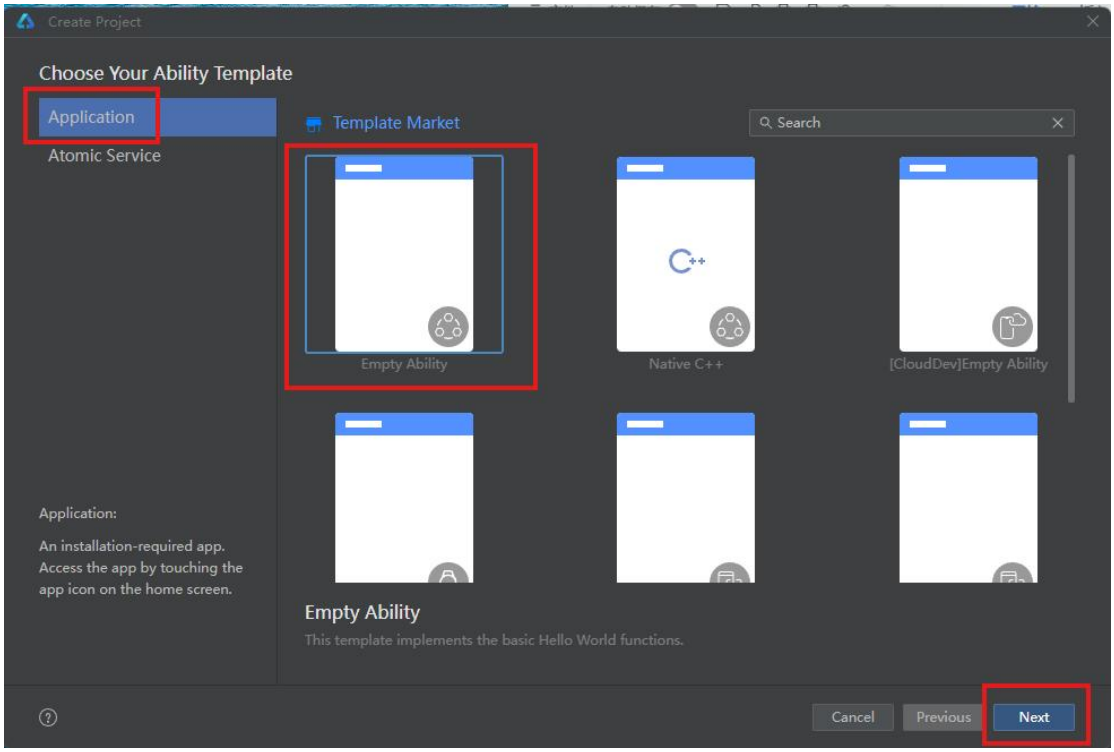
打开 DevEco Studio，点击 Create Project 创建工程。



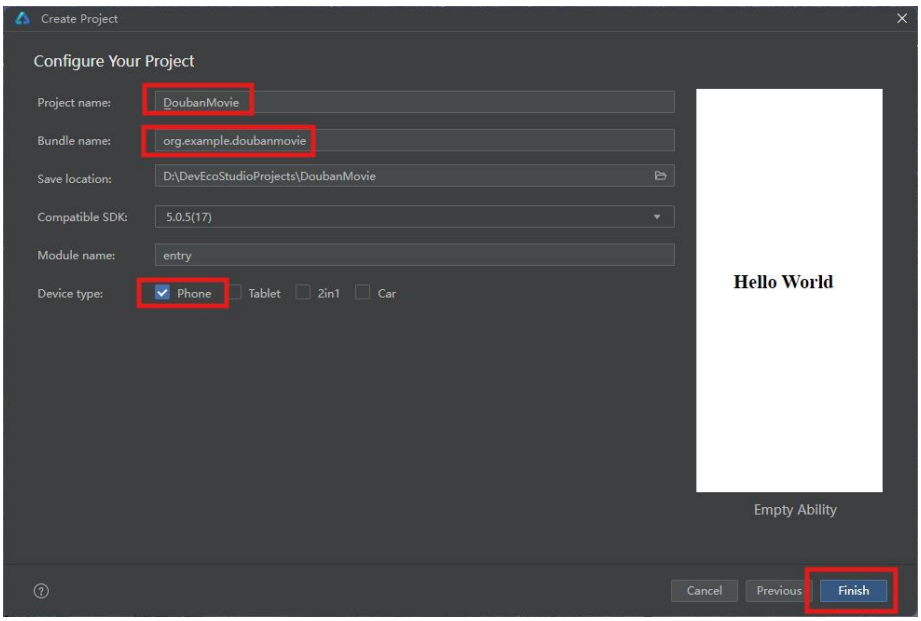
如果已经打开了一个工程，请在菜单栏选择 File > New > Create Project 来创建一个新工程。



点击“Create Project”，进入：



选择 Application，然后选择“Empty Ability”，点击 Next 按钮，进入：



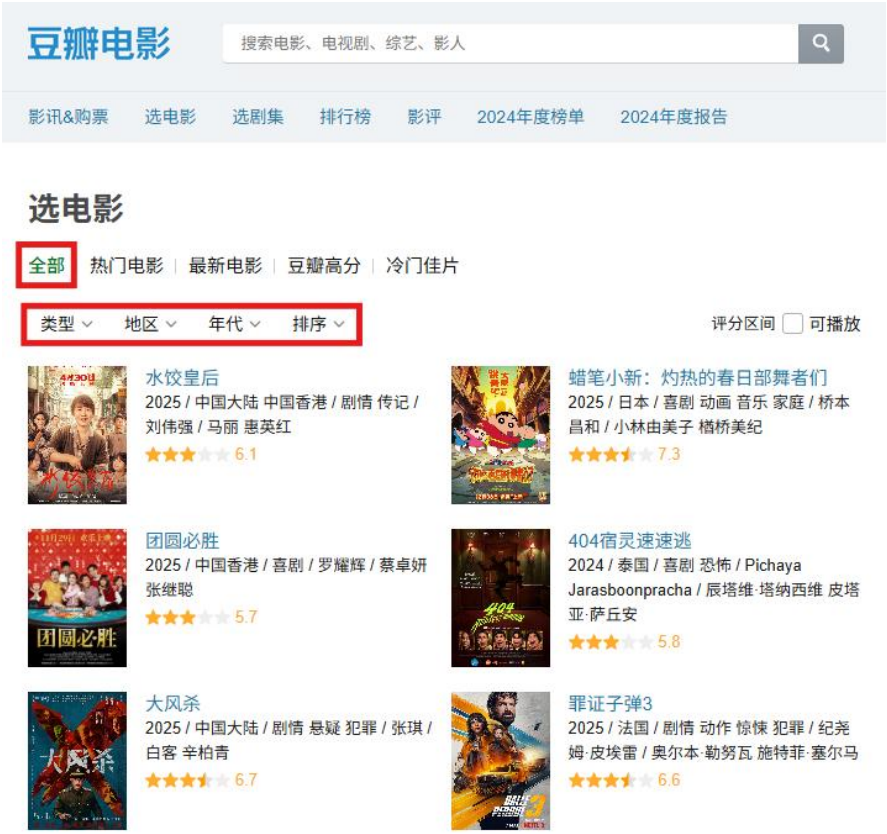
设置项目名称为 DoubanMovie（你可以取自己喜欢的名称），存放地址等，然后点击 Finish。

2. 理解豆瓣的通讯协议

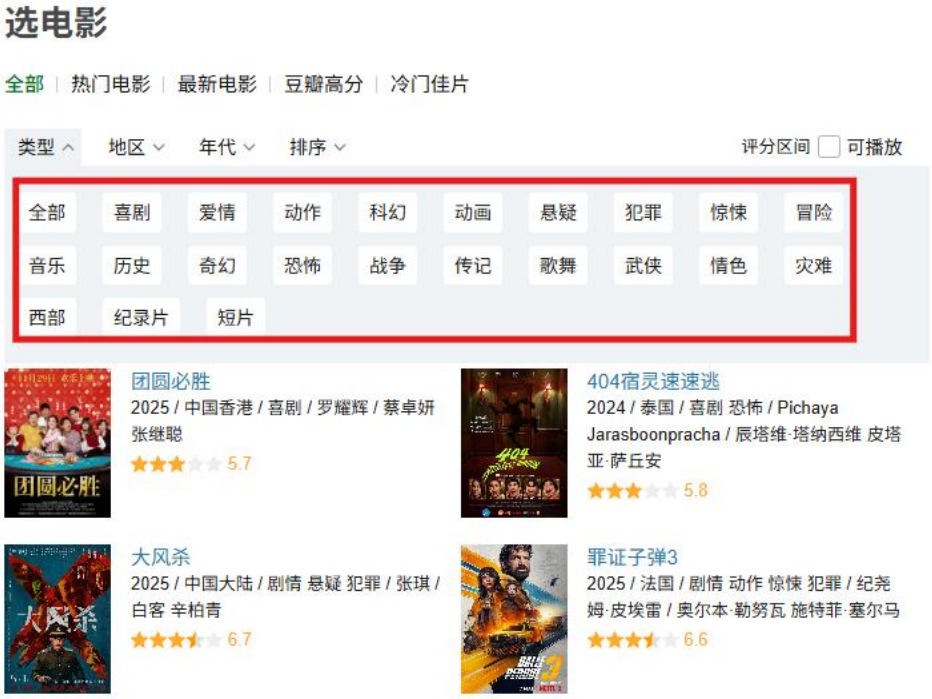
我们使用 Chrome 浏览器访问豆瓣电影的网站：https://movie.douban.com/explore ，或者 Firefox

浏览器也可以，操作方法类似。

点击界面上的“全部”标签，下方的“类型”、“地区”、“年代”、“排序”会变成有下拉框选项的形式。



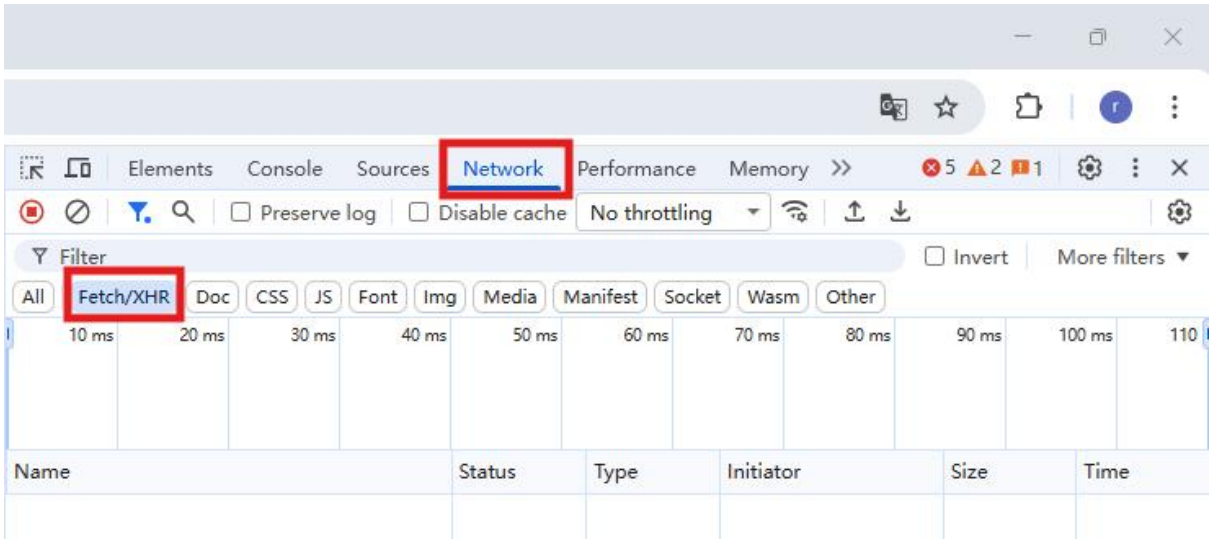
点击某个选项比如“类型”，会出现浮动选择项目，列出可以选择的内容如图



我们计划来用鸿蒙实现一个客户端，从豆瓣服务器上下载电影指定类型电影的数据，并展示在鸿蒙手机界面上。

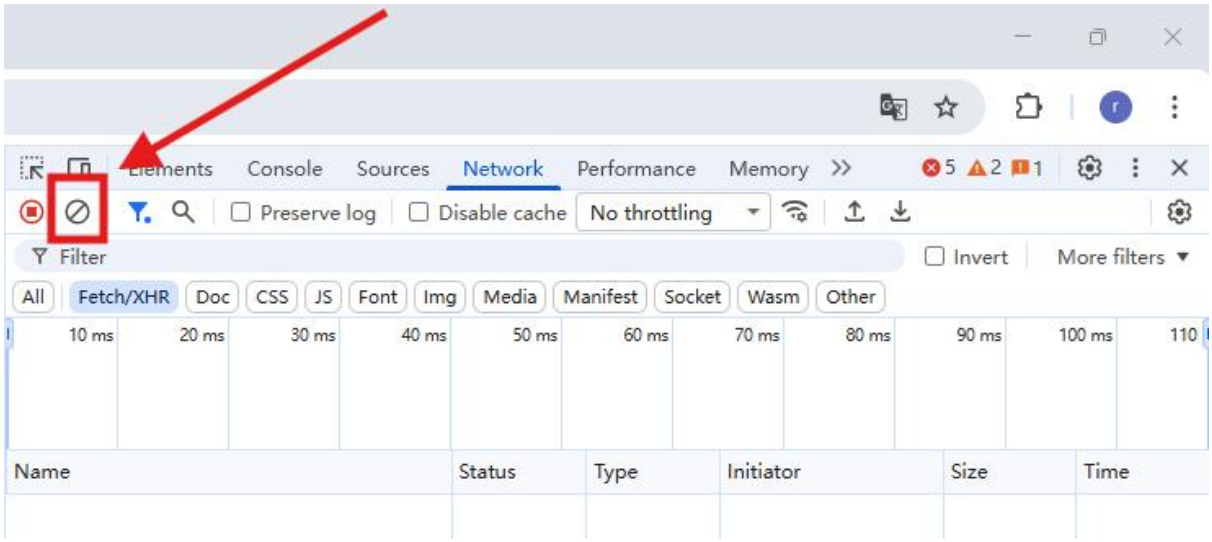
2.1 抓取通讯数据

在 Chrome 浏览器中点击 F12 打开 开发者工具，选择其中的“网络”标签，继续选择下面过滤器中的“Fetch/XHR”标签。“网络”标签的功能可以查看浏览器与服务器之间的通讯数据，并且选中可以仅仅过滤 AJAX 请求之类的异步接口请求的数据。



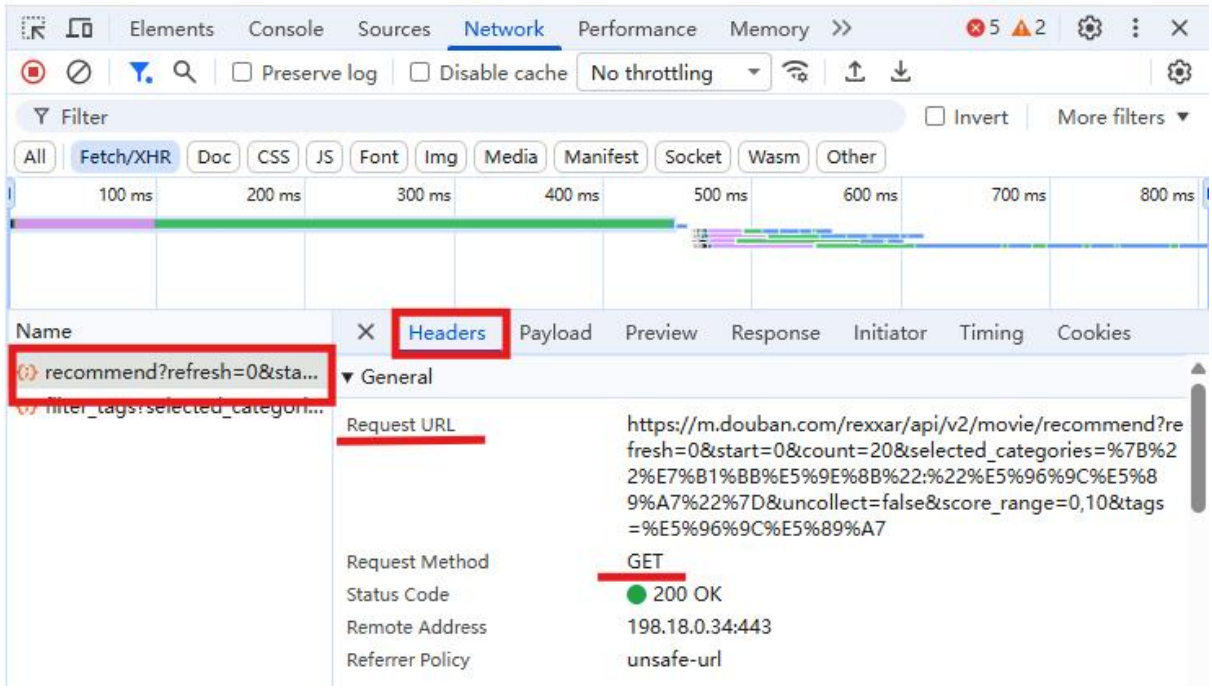
当我们浏览网页过程中，开发者工具记录了太多的通讯数据时，可以点击左上角的“清理”图标来

清除记录的数据，保持界面清爽便于找到我们关注的信息。



我们在豆瓣电影界面选择某一个类型的电影，比如“喜剧”类型，界面会列出所有喜剧类型的电影，同时我们可以看到开发者工具中记录到了两次会话的数据。点击第一条“recommend”开头的会话，选择“标头”标签，我们看到是一条向一个服务器地址发送 GET 命令请求数据的记录，请求地址是：

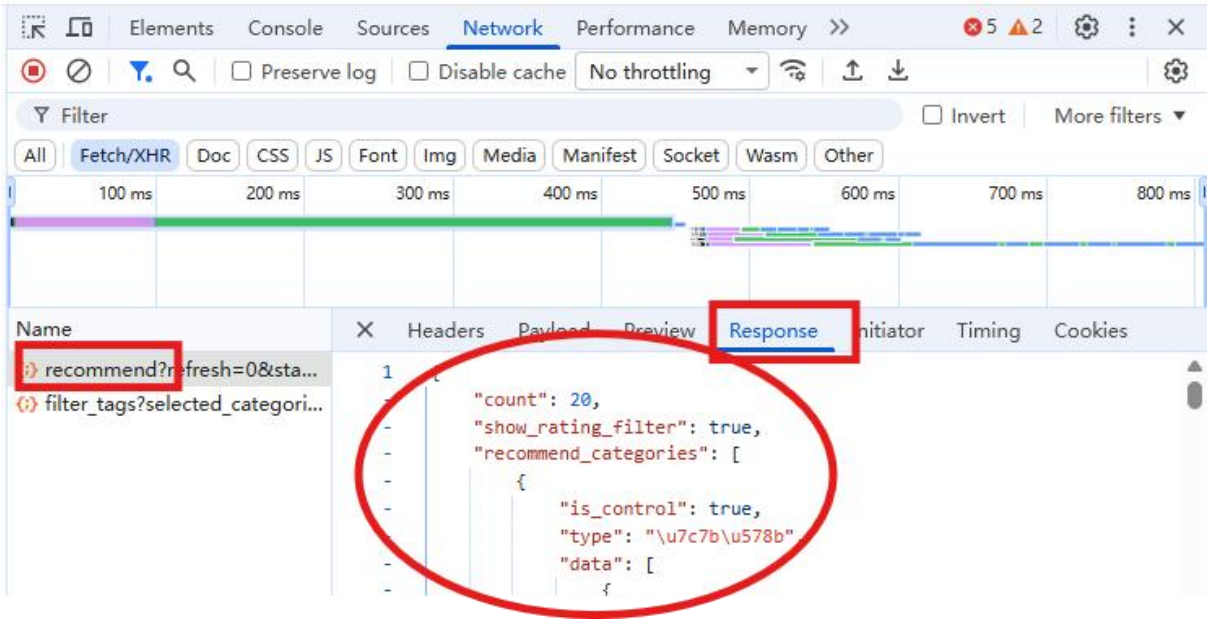
https://m.douban.com/rexxar/api/v2/movie/recommend?refresh=0&start=0&count=20&selected_categories=%7B%22%E7%B1%BB%E5%9E%8B%22:%22%E5%96%9C%E5%89%A7%22%7D&uncollect=false&score_range=0,10&tags=%E5%96%9C%E5%89%A7



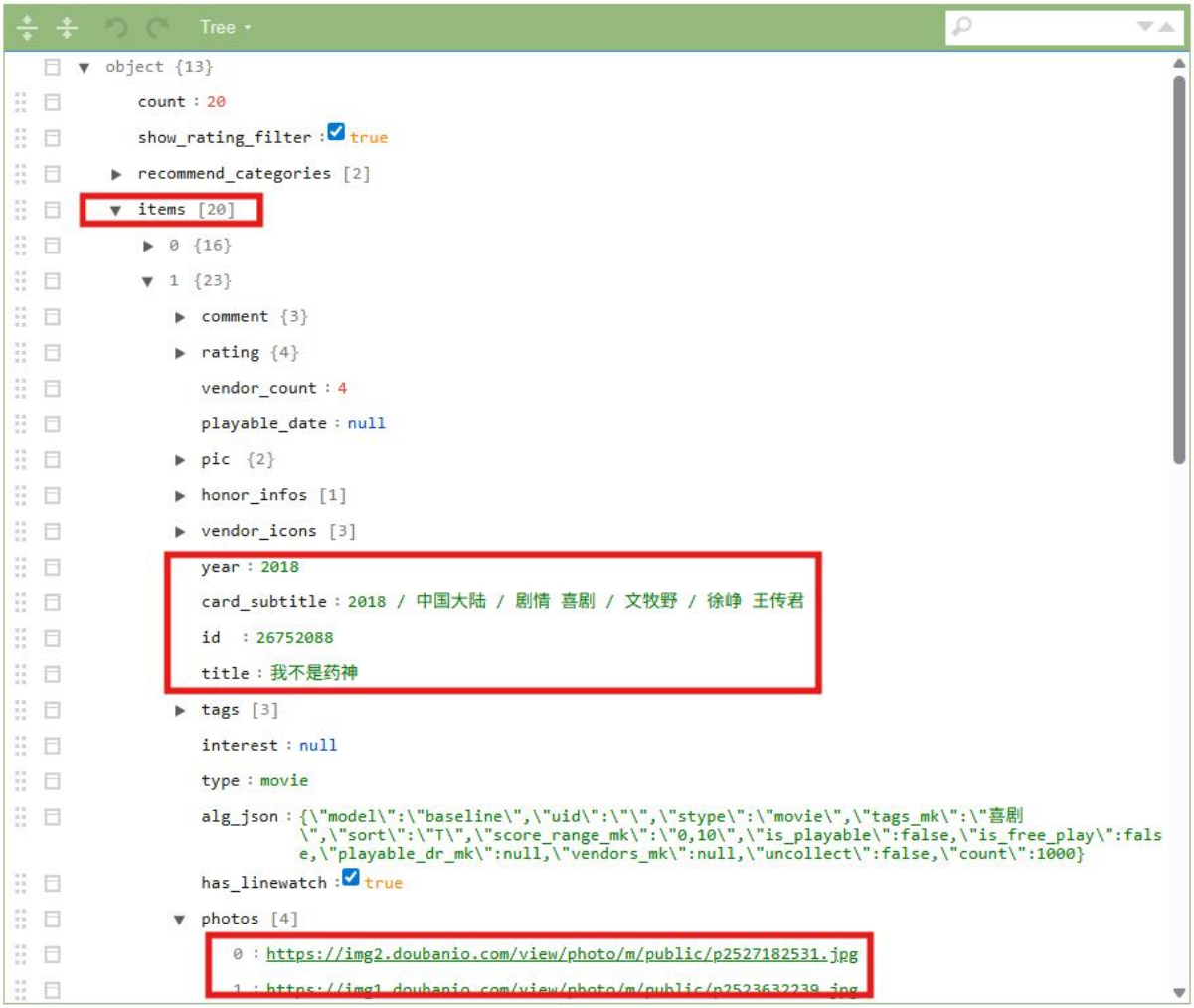
请求地址中，有很多转义后的字符不能直接显示，我们到站长工具箱（https://tool.chinaz.com/tools/urlencode.aspx）对被请求的 URL 地址进行解码获得如下内容：

```
https://m.douban.com/rexxar/api/v2/movie/recommend?refresh=0&start=0&count=20&selected_categories={"类型":"喜剧"}&uncollect=false&score_range=0,10&tags=喜剧
```

看起来，正是我们发出的请求，然后我们点击“响应”标签查看服务端返回的数据。我们可以看到，返回的内容是一段JSON格式的信息。



复制这段JSON文本内容，转到“菜鸟工具箱”解析和观察这段JSON的内容：https://www.jyshare.com/front-end/53/ 。通过分析返回的内容，我们确定就是网站上电影列表上的数据。



观察返回的数据，主要包括如下的字段：

count: 返回的记录条数

items: 返回的电影的清单列表

recommend_categories: 推荐的分类列表;

total: 该分类下所有的电影条数

电影列表在消息体的 items 字段下，共有 20 条记录。每条电影记录又包括如下字段：

id: 电影的 ID 字符串;

title: 电影的标题;

card_subtitle: 简短的介绍信息;

pic: 电影海报的列表;

rating: 电影评分信息;

某些字段比如 rating 评分，还包括更细节的信息；某些字段的内容例如 pic 照片则是一个数组，在此不再详细说明。

通过回看请求地址及其参数，结合分析服务端返回的数据，我们可以确定这个 https://m.douban.com/rexxar/api/v2/movie/recommend 地址就是我们寻找的豆瓣电影数据服务端接口。

通过整理我们可以得知请求参数如下：

start：翻页时候，从哪一条记录开始返回数据

count：期望返回数据的条数

selected_categories：选择的电影类别参数

tags：选中的标签;

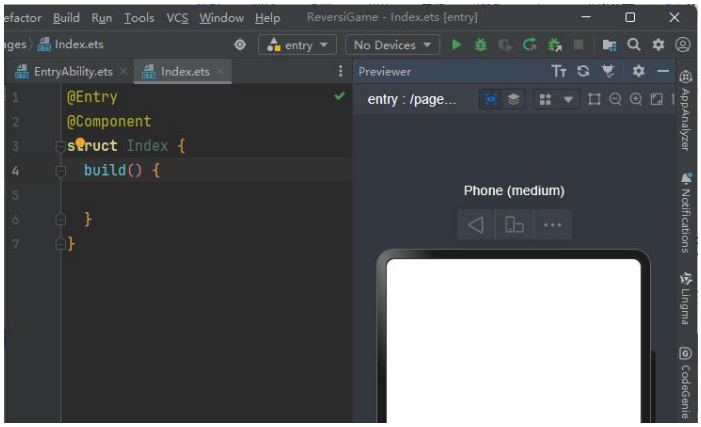
搜集到以上的信息后，我们就可以开始开发自己的豆瓣电影客户端了。

2.2 开始开发

对 Index.ets 进行一些修改：

- 清理界面

在 Index.ets 文件的 Index 结构中，首先删除掉 message 这个成员变量，清理掉 build() 函数中原来绘制 “Hello World”的代码，此时刷新 Previewer,界面会变成空白。



在界面构建函数 build() 中绘制一个 Button 控件，代码如下：

```
@Entry
```

```
@Component
struct Index {
  @State counter: number = 0;
  build() {
    Column() {
      Button() {
        Text('下载电影')
          .fontSize(20)
      }
      .margin({ top: 30 })
      .width('60%')
      .height(50)
      .fontColor(Color.White)
      .onClick(() => {
      })
    }
    .width('100%')
    .height('100%')
  }
}
```

修改 onClick 函数的响应代码，内容如下：

```
.onClick(async () => {
       const                                  url                                    =
'https://m.douban.com/rexxar/api/v2/movie/recommend?refresh=0&start=0&count=20
&selected_categories=%7B%22%E7%B1%BB%E5%9E%8B%22:%22%E5%96%9C%E5%89%A7%22%7D&u
ncollect=false&score_range=0,10&tags=%E5%96%9C%E5%89%A7';
       let httpRequest = http.createHttp();

       console.info(`Starting network request to: ${url}`);
       try {
           const response = await httpRequest.request(url, {
               method: http.RequestMethod.GET,
               header: {
                 'accept': 'application/json, text/plain, */*',
                 'accept-language': 'zh-CN,zh;q=0.9',
                 'origin': 'https://movie.douban.com',
                 'priority': 'u=1, i',
                 'referer': 'https://movie.douban.com/explore',
                 'sec-ch-ua': '"Google Chrome";v="143", "Chromium";v="143", "Not
A(Brand";v="24"',
                 'sec-ch-ua-mobile': '?0',
                 'sec-ch-ua-platform': '"Linux"',
```

```
                'sec-fetch-dest': 'empty',
                'sec-fetch-mode': 'cors',
                'sec-fetch-site': 'same-site',
                'user-agent':     'Mozilla/5.0     (X11;     Linux     x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36'
            }
        });

        if (response.responseCode === http.ResponseCode.OK) {
            let   resultStr   =   typeof   response.result   ===   'string'   ?
response.result : JSON.stringify(response.result);
            console.info(`Request successful. Received data: ${resultStr}`);
        } else {
            console.info(`Request       failed.       Response       code:
${response.responseCode}, message: ${JSON.stringify(response.header)}`);
        }
    } catch (err) {
        console.info(`Request error: ${JSON.stringify(err)}`);
    } finally {
        httpRequest.destroy();
    }
  })
```

为了这段代码能正常运行，记得在文件开始引入 `import http from '@ohos.net.http';`

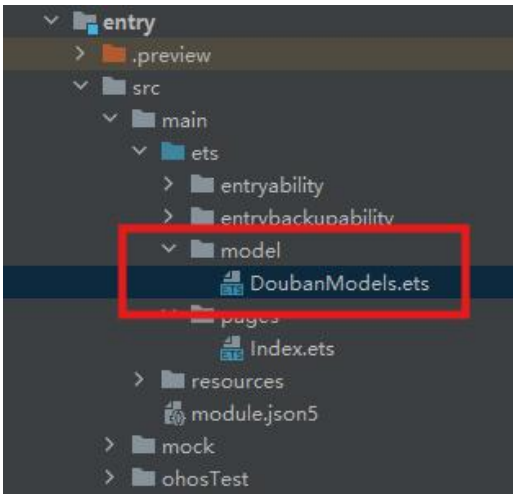
这段代码功能是响应按钮的 onClick 消息，创建一个 httpRequest 对象并修改 http 的 header 信息，伪装成浏览器发送 GET 请求获取豆瓣网站的电影清单。此时在 previewer 中是可以正确运行程序，点击按钮后，可以在 log 中看到输出日志的。注意我们是如何拼接请求 URL 的，同时注意为了伪装成正常的访问，我们修改了 user-agent， origin，referer 等参数。

```
Starting network request to: https://m.douban.com/rexxar/api/v2/movie/recommend?refresh=0&start=0&count=20&se
Request successful. Received data: {"count": 20, "show_rating_filter": true, "recommend_categories": [{"is_co
```

尝试在模拟器中运行程序，看看能否正确运行。如果你看不到正确的返回数据，而是看到了 `Request error: {"code":201,"message":"Permission denied"}` ，应该怎么解决这个问题呢？

接下来我们需要解析服务端返回的信息。参考我们在前面分析服务端返回的内容，我们可以创建一些数据模型来保存这些信息。首先在 ets 目录下创建 model 目录，并在 model 目录下创建

DoubanModels.ets 文件，用来保存数据模型。



数据模型文件内容如下，这些都是我们通过分析返回数据的结构获得的，注意各个模型之间的层级关系：

```
export interface DoubanResponse {
  count: number;
  items: MovieItem[];
  recommend_categories: RecommendCategory[];
  total: number;
}

export interface MovieItem {
  id: string;
  title: string;
  card_subtitle: string;
  pic: Pic;
  rating: Rating;
}

export interface Rating {
  count: number;
  max: number;
  star_count: number;
  value: number;
}

export interface Pic {
  large: string;
  normal: string;
}
```

```
export interface CategoryDataItem {
  default: boolean;
  text: string;
}

export interface RecommendCategory {
  is_control: boolean;
  type: string;
  data: CategoryDataItem[];
}
```

在 Index.ets 中引入这个文件

```
import { DoubanResponse, MovieItem } from '../model/DoubanModels';
```

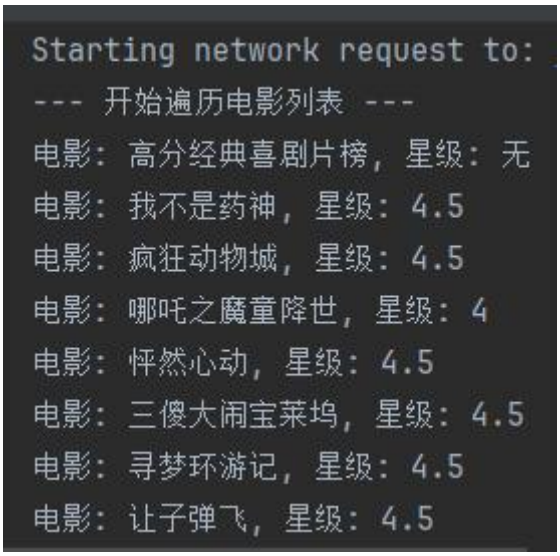
紧接着修改 onClick 函数中，收到服务端返回信息后的处理部分代码，替换原来的代码块为如下内容：

```
import http from '@ohos.net.http';
import { DoubanResponse, MovieItem } from '../model/DoubanModels';

@Entry
@Component
struct Index {
  @State counter: number = 0;
  build() {
    Column() {
      Button() {
        Text('下载电影')
          .fontSize(20)
      }
      .margin({ top: 30 })
      .width('60%')
      .height(50)
      .fontColor(Color.White)
      .onClick(async () => {
        const url =
'https://m.douban.com/rexxar/api/v2/movie/recommend?refresh=0&start=0&count=20
&selected_categories=%7B%22%E7%B1%BB%E5%9E%8B%22:%22%E5%96%9C%E5%89%A7%22%7D&u
ncollect=false&score_range=0,10&tags=%E5%96%9C%E5%89%A7';
        let httpRequest = http.createHttp();

        console.info(`Starting network request to: ${url}`);
```

```
        try {
            const response = await httpRequest.request(url, {
                method: http.RequestMethod.GET,
                header: {
                  'accept': 'application/json, text/plain, */*',
                  'accept-language': 'zh-CN,zh;q=0.9',
                  'origin': 'https://movie.douban.com',
                  'priority': 'u=1, i',
                  'referer': 'https://movie.douban.com/explore',
                  'sec-ch-ua': '"Google Chrome";v="143", "Chromium";v="143", "Not
A(Brand";v="24"',
                  'sec-ch-ua-mobile': '?0',
                  'sec-ch-ua-platform': '"Linux"',
                  'sec-fetch-dest': 'empty',
                  'sec-fetch-mode': 'cors',
                  'sec-fetch-site': 'same-site',
                  'user-agent':     'Mozilla/5.0     (X11;     Linux     x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36'
                }
            });

            if (response.responseCode === http.ResponseCode.OK) {
                let   resultStr   =   typeof   response.result   ===   'string'   ?
response.result : JSON.stringify(response.result);
                const doubanData: DoubanResponse = JSON.parse(resultStr);
                if (doubanData && doubanData.items) {
                  console.info('--- 开始遍历电影列表 ---');
                  doubanData.items.forEach((item: MovieItem) => {
                    console.info(`   电   影   :   ${item.title},   星   级   :
${item.rating?.star_count ?? '无'}`);
                  });
                } else {
                  console.info('未找到电影项目或返回数据格式不正确');
                }
            } else {
                console.info(`Request       failed.       Response       code:
${response.responseCode}, message: ${JSON.stringify(response.header)}`);
            }
        } catch (err) {
            console.info(`Request error: ${JSON.stringify(err)}`);
        } finally {
            httpRequest.destroy();
        }
      })
```

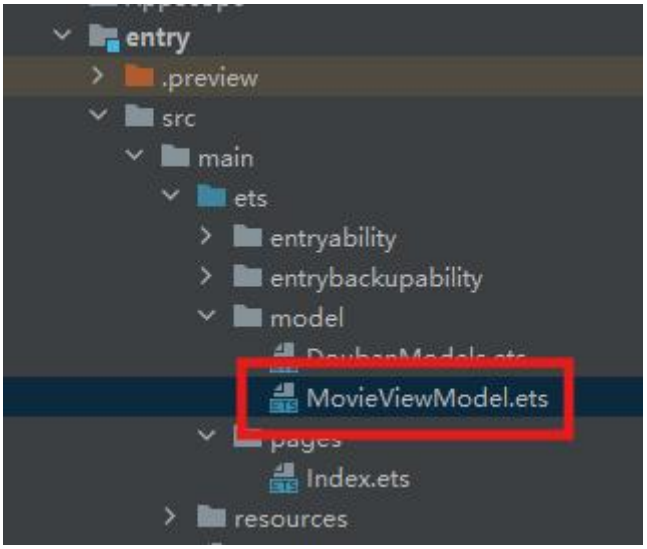
```
    }
    .width('100%')
    .height('100%')
  }
}
```

此时执行程序，我们可以看到代码中使用 JSON.parse(resultStr) 将服务端返回的 JSON 字符串转换成了对象并赋值给了我们刚刚定义的 DoubanResponse 数据模型的对象，然后日志遍历这个对象的 items 属性中的所有电影，打印了解析出来的每一部电影的标题和评分。同样，如果我们的模型中，各个对象、以及对象的属性字段设计的正确，电影的其他信息也都能正确的被解析和打印出来。



至此我们已经能够正确处理基本的 HTTP 请求，并且能解析 JSON 数据格式。不过，我们的最终目标还有很长的路要走。

我们需要做界面、逻辑、数据分离的程序，因此还要对项目做更深入的改造。我们在 model 目录下创建一个 MovieViewModel.ets 文件，用来封装和管理电影网络客户端所有属性、方法，并且准备用来创建数据链接关系。



MovieViewModel.ets 文件的代码内容如下，注意类是被 @Observed 修饰的，说明当数据变化的时候，界面 UI 中用 @State 修饰的的变量可以响应其中的数据变化并通知界面更新：

```
import http from '@ohos.net.http';
import { DoubanResponse, MovieItem, RecommendCategory } from './DoubanModels';

@Observed
export class MovieViewModel {
  movies: MovieItem[] = [];
  typeCategory: RecommendCategory | null = null;
  regionCategory: RecommendCategory | null = null;
  selectedType: string = '喜剧';
  selectedRegion: string = '华语';
  isLoading: boolean = false;
  isRefreshing: boolean = false;
  private start: number = 0;
  private count: number = 20;
  private total: number = 0;

  fetchData(isRefresh: boolean): void {
    console.info(`[DoubanMovie VM] fetchData called. isRefresh: ${isRefresh},
isLoading: ${this.isLoading}, movies.length: ${this.movies.length}`);
    if (this.isLoading) {
      console.info(`[DoubanMovie VM] Already loading, returning.`);
      return;
    }
    if (isRefresh) {
      this.start = 0;
    }
    this.isLoading = true;
```

```
    let httpRequest = http.createHttp();

    const selectedCategories: Record<string, string> = {
      "类型": this.selectedType === '全部类型' ? '' : this.selectedType,
      "地区": this.selectedRegion === '全部地区' ? '' : this.selectedRegion
    };

    const activeTags: string[] = [];
    if (this.selectedType !== '全部类型') {
      activeTags.push(this.selectedType);
    }
    if (this.selectedRegion !== '全部地区') {
      activeTags.push(this.selectedRegion);
    }

    const tags = activeTags.join(',');
    const url =

`https://m.douban.com/rexxar/api/v2/movie/recommend?refresh=0&start=${this.start}&count=${this.count}&selected_categories=${encodeURIComponent(JSON.stringify(selectedCategories))}&uncollect=false&score_range=0,10&tags=${encodeURIComponent(tags)}`;
    console.info(`[DoubanMovie VM] Requesting URL: ${url}`);

    httpRequest.request(
      url,
      {
        method: http.RequestMethod.GET,
        header: {
          'accept': 'application/json, text/plain, */*',
          'accept-language': 'zh,zh-CN;q=0.9,en;q=0.8',
          'origin': 'https://movie.douban.com',
          'priority': 'u=1, i',
          'referer': 'https://movie.douban.com/explore',
          'sec-ch-ua': '"Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"',
          'sec-ch-ua-mobile': '?0',
          'sec-ch-ua-platform': '"Linux"',
          'sec-fetch-dest': 'empty',
          'sec-fetch-mode': 'cors',
          'sec-fetch-site': 'same-site',
          'user-agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36'
```

```
        }
      },
      (err, data) => {
        this.isLoading = false;
        if (!err) {
          console.info(`[DoubanMovie VM] Request successful. Data result type:
${typeof data.result}`);
          try {
            const doubanResponse: DoubanResponse = JSON.parse(data.result as
string);
            console.info(`[DoubanMovie VM] JSON parsed successfully. Items length:
${doubanResponse.items.length}, Categories length:
${doubanResponse.recommend_categories.length}`);
            if (isRefresh) {
              this.movies = doubanResponse.items;
              if (!this.typeCategory || !this.regionCategory) {
                this.typeCategory = doubanResponse.recommend_categories.find(c =>
c.type === '类型') ?? null;
                this.regionCategory = doubanResponse.recommend_categories.find(c
=> c.type === '地区') ?? null;
              }
            } else {
              this.movies.push(...doubanResponse.items);
            }
            this.start += this.count;
            this.total = doubanResponse.total;
            console.info(`[DoubanMovie VM] Movies array updated. New length:
${this.movies.length}, total: ${this.total}`);
          } catch (e) {
            console.error(`[DoubanMovie VM] JSON parsing failed:
${JSON.stringify(e)}`);
          }
        } else {
          console.error(`[DoubanMovie VM] Request failed:
${JSON.stringify(err)}`);
        }
        httpRequest.destroy();
      }
    );
  }

}
```

修改 Index.ets，删除其中调用网络请求的部分代码，改为使用 MovieViewModel.ets 中的数据，完整代码如下：

```
import { MovieItem } from '../model/DoubanModels';
import { MovieViewModel } from '../model/MovieViewModel';

@Entry
@Component
struct Index {
  @State vm: MovieViewModel = new MovieViewModel();

  build() {
    Column() {
      Button('下载电影')
        .fontSize(20)
        .margin({ top: 30 })
        .width('60%')
        .height(50)
        .fontColor(Color.White)
        .onClick(async () => {
          await this.vm.fetchData(true);
          if (this.vm.movies && this.vm.movies.length > 0) {
            this.vm.movies.forEach((item: MovieItem) => {
              console.info(`    电    影    :    ${item.title},    星    级    :
${item.rating?.star_count ?? '无'}`);
            });
          } else {
            console.info('未获取到电影数据或列表为空');
          }
        })
    }
    .width('100%')
    .height('100%')
  }
}
```

完成后的界面还是只有一个按钮，但是网络请求部分的功能已经完全剥离了。接着我们修改 Index.ets ，删除在 log 中通过日志显示返回数据的方法，改为在 UI 中增加一个简单的列表，并且在列表中显示返回的电影数据。完整代码如下，注意：

```
import { MovieItem } from '../model/DoubanModels';
import { MovieViewModel } from '../model/MovieViewModel';
```

```
@Entry
@Component
struct Index {
  @State vm: MovieViewModel = new MovieViewModel();

  build() {
    Column({ space: 10 }) {
      Row({ space: 10 }) {
        Button('刷新电影列表')
          .fontSize(20)
          .margin({ top: 30 })
          .width('50%')
          .height(50)
          .fontColor(Color.White)
          .onClick(() => {
            this.vm.fetchData(true);
          })
        Button('清除')
          .fontSize(20)
          .margin({ top: 30 })
          .height(50)
          .fontColor(Color.White)
          .onClick(() => {
            this.vm.movies = [];
          })
      }

      List({ space: 10 }) {
        ForEach(this.vm.movies, (item: MovieItem) => {
          ListItem() {
            Text(item.title + '，评分：' + item.rating?.star_count)
              .fontSize(18)
              .width('100%')
              .padding(10)
          }
        }, (item: MovieItem) => item.id)
      }
      .layoutWeight(1)
      .width('90%')
      .divider({ strokeWidth: 1, color: '#e0e0e0' })
    }
    .width('100%')
    .height('100%')
    .padding({ bottom: 10 })
```

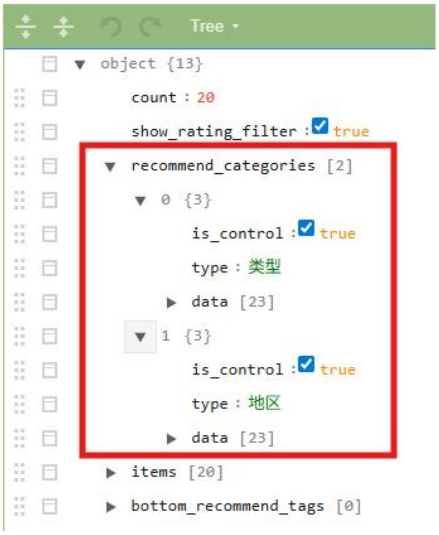
```
  }
}
```

完成后运行代码，在 previewer 中查看：

至此，我们的电影客户端有了一个初步雏形，具备了基本的功能。

2.3 动态数据

接下来我们要把界面做的更精美一些，并且使用下拉框来选择不同的电影类型。首先要解决的是下拉框里面的数据应该显示什么分类。通过我们前面对服务端返回数据的分析，我们可以知道，服务端返回的结构体中，有一个“recommend_categories”字段的数组，里面的数组 1、2 分别包含了两个标签“类型”和“地区”，里面的数据就包括了网站中电影的分类数据。我们就动态的使用这些服务端返回的数据，构造我们的下拉框。

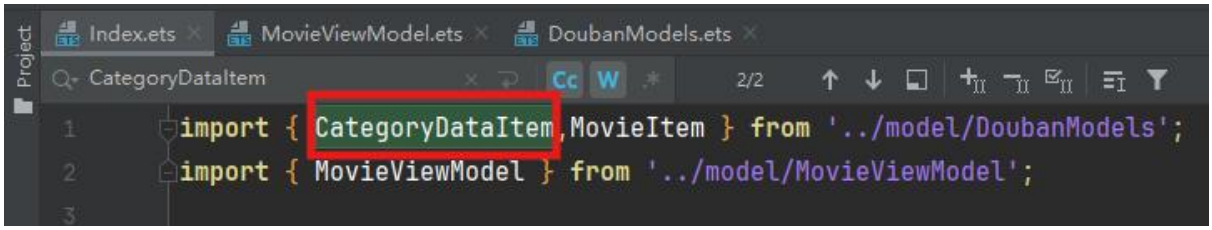


回到 Index.ets 文件，将原本界面上的“刷新电影列表” 和“清除”这个 ROW 全部删除掉，替换为下面的代码：

```
      Row() {
        if (this.vm.typeCategory && this.vm.regionCategory) {
          Row({ space: 5 }) {
            Text('类型:')
            Select(this.vm.typeCategory?.data.map((item: CategoryDataItem) => ({
              value: item.text,
              text: item.text
            } as SelectOption)) ?? [])
              .value(this.vm.selectedType           ===           ''           ?
(this.vm.typeCategory?.data[0]?.text ?? '') : this.vm.selectedType)
              .selected(this.vm.selectedType      ===      ''      ?      0      :
(this.vm.typeCategory?.data.findIndex(item       =>       item.text       ===
this.vm.selectedType) ?? -1))
              .onSelect(async (index: number) => {
                console.info(`[DoubanMovie  Index]  Type  selected.  New  index:
${index}`);
                this.vm.selectedType    =    index    ===    0    ?    ''    :
this.vm.typeCategory?.data[index]?.text ?? '';
                await this.vm.fetchData(true);
              })
              .fontColor(Color.Black)
              .selectedOptionFontColor(Color.Blue)
              .backgroundColor(Color.White)
          }.alignItems(VerticalAlign.Center)
        }
      }
      .width('100%')
```

```
      .justifyContent(FlexAlign.SpaceEvenly)
      .padding(10)
      .height(60)
      .backgroundColor('#f0f0f0')
```

由于引入了 CategoryDataItem 这个数据类型，因此要从我们之前解析的代码中导入类型：



接着在 Index.ets 中增加如下的代码，使得界面显示的时候，就异步调用 ViewModel 中的 fetchData 下载数据并显示在界面上：

```
async aboutToAppear() {
  await this.vm.fetchData(true);
}
```

如果一切正常，刷新界面会更新为如下状态：

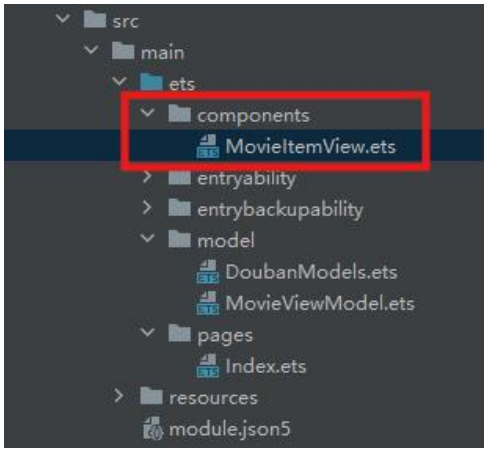
- 

重复上面的步骤，并分析示例代码，增加“地区”下拉框。这部分代码需要自行分析后实现。完成后界面如下图：

2.4 自定义组件

当前界面中 List 项目显示的还是非常粗糙原始的手动拼接的 Text，非常原始，我们需要构造出类似官网上面电影项目的界面元素，包含剧照、电影名、简介、星级、评分等信息。

要完成这个需求，我们需要自定义自己的视图组件，替换掉 List 组件中的 Text 组件。首先在 ets 目录下创建一个 components 文件夹，用于保存自定义组件。接在在该目录下创建一个 MovieItemView.ets 文件。



文件的内容如下：

```
import { MovieItem } from '../model/DoubanModels';

@Component
export struct MovieItemView {
  @Prop movie: MovieItem;

  build() {
    Column() {
      Row() {
        Image(this.movie.pic?.normal ?? '')
          .width(100)
          .height(150)
```

```
          .objectFit(ImageFit.Contain)
          .margin({ right: 10 })

        Column() {
          Text(this.movie.title)
            .fontSize(18)
            .fontWeight(FontWeight.Bold)
            .fontColor('#ff2157d4')
          Text(this.movie.card_subtitle)
            .fontSize(15)
            .fontColor(Color.Black)
            .margin({ top: 8, bottom: 8 })
            .maxLines(3)
            .textOverflow({ overflow: TextOverflow.Ellipsis })
          Row() {
            Text(`${this.movie.rating?.value ?? 'N/A'}`)
              .fontSize(15)
              .fontColor(Color.Orange)
              .margin({ left: 5 })
          }
          .alignItems(VerticalAlign.Center)
        }.alignItems(HorizontalAlign.Start).layoutWeight(1)
      }.padding(10)

      Divider()
    }
  }
}
```

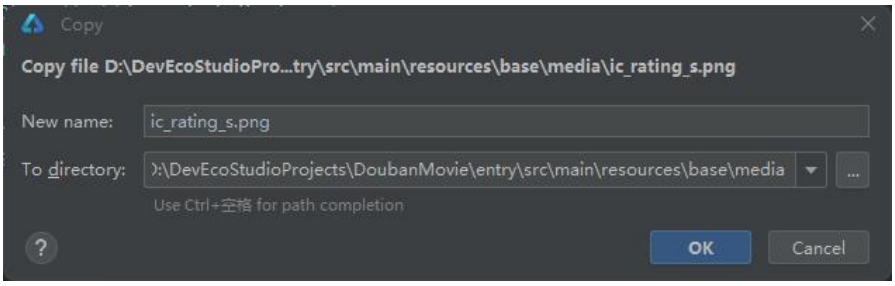
记得要导入我们刚刚创建的 MovieItemView 组件，然后修改 Index.ets 中 ListItem 中的内容，将平淡无奇的 Text 替换成如下我们自定义的组件：

```
MovieItemView({ movie: item })
```

此时刷新界面，会看到如下界面，已经成功 80%了，但是电影的星级图标还不能正常显示。另外也不具备翻页追加数据和刷新等待图标的功能：

星级的图片，通过分析豆瓣网页，我们找到一个文件 ic_rating_s.png

下载这个文件，并且复制到资源目录下：



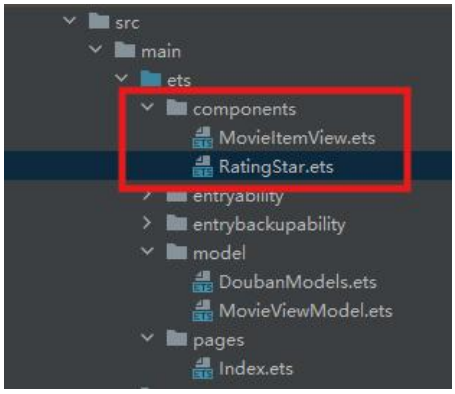
我们分析这个图片的信息：

可以看到图像的分辨率为 55 * 121，可以得知图片宽度 55，高度 121。继续分析图片可以得知，图

片可以可以分割为 11 行，分别标识了 5 星好评到 0.5 星好评工 10 个档位，以及没有星级的一个档位，一共 11 个档位。继续计算可以得知，每一个档位的星级占用了 121/11 = 11 像素高度。

所以，每一个星级的占用的像素是 55 * 11 分辨率。

基于上述信息，我们创建一个新的自定义组件来显示星级，文件名为：RatingStar.ets



这个自定义组件的内容如下，尤其要注意这个组件中有个 @Prop rating: number; 属性。这是需要传递给这个自定一组件的星级参数。绘制组件的时候，会根据 rating 参数自动裁剪合适的星级图片并渲染显示：

理解代码中 build 函数的 Stack 组件和 Image 组件如何配合工作裁剪图片的。

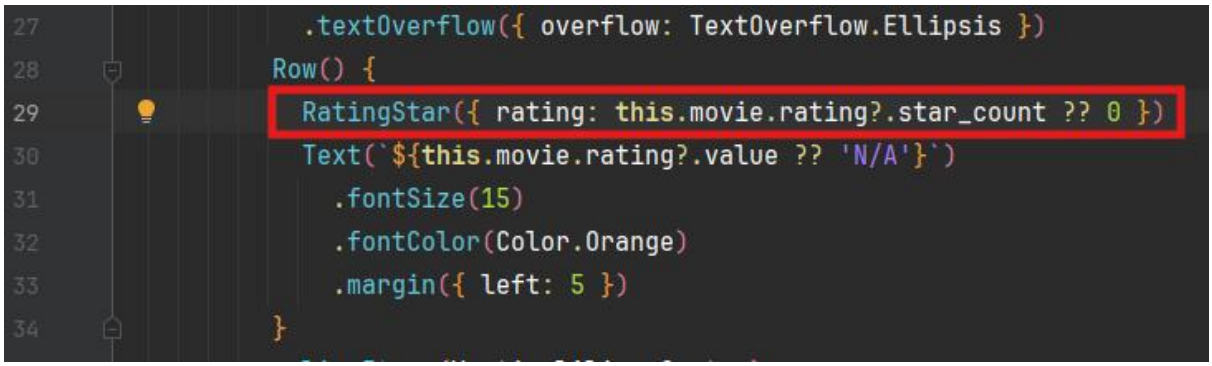
```
@Component
export struct RatingStar {
  private static readonly STAR_HEIGHT: number = 11;
  private static readonly STAR_IMG_WIDTH: number = 55;
  private static readonly STAR_IMG_HEIGHT: number = 121; // 11 rows * 11vp/row

  @Prop rating: number;

  calculateStarRowOffset(ratingValue: number): number {
    // 检查无效值
    if (ratingValue == null || isNaN(ratingValue)) {
      return 0;
    }

    // 限制评分在有效范围内 (0-5)
    const clampedRating = Math.min(Math.max(ratingValue, 0), 5);

    // 计算星图偏移量
    const starIndex = Math.round(clampedRating * 2);
    const rowIndex = -(10 - starIndex) * RatingStar.STAR_HEIGHT;
    return rowIndex;
```

```
  }

  build() {
    Stack() {
      Image($r('app.media.ic_rating_s'))
        .width(RatingStar.STAR_IMG_WIDTH)
        .height(RatingStar.STAR_IMG_HEIGHT)
        .offset({
          y: this.calculateStarRowOffset(this.rating ?? 0)
        })
    }
    .align(Alignment.Top)
    .width(RatingStar.STAR_IMG_WIDTH)
    .height(RatingStar.STAR_HEIGHT)
    .clip(true)
  }
}
```

在 MovieItemView.ets 电影信息组件中，导入我们新增的星级显示组件

```
import { RatingStar } from './RatingStar';
```

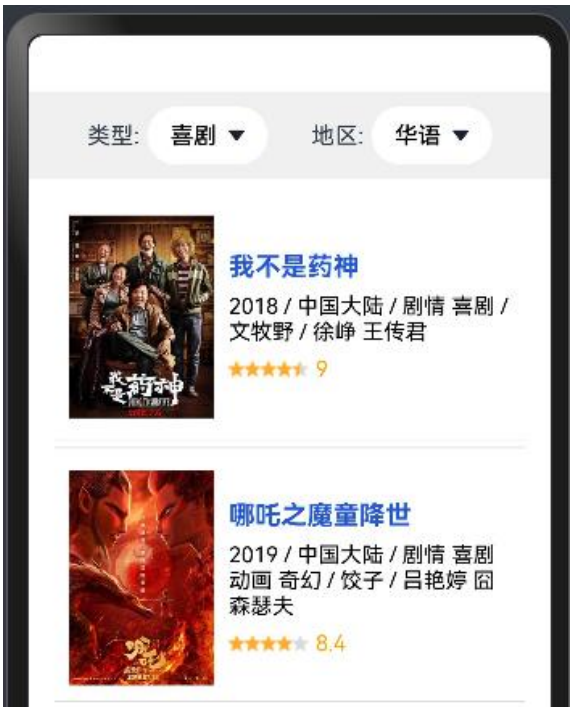
并且在显示电影评分的组件前面，增加一行代码，用我们新增的 RatingStar 自定一组件显示电影的星级：



代码：

```
 RatingStar({ rating: this.movie.rating?.star_count ?? 0 })
```

保存代码后刷新界面，我们就得到了能显示星级的电影信息：



至此我们就得到了一个能根据类别查询电影信息，并且显示丰富界面元素的电影客户端。但是目前的功能，每次查询我们只能获得一页内容。

2.5 载入更多信息

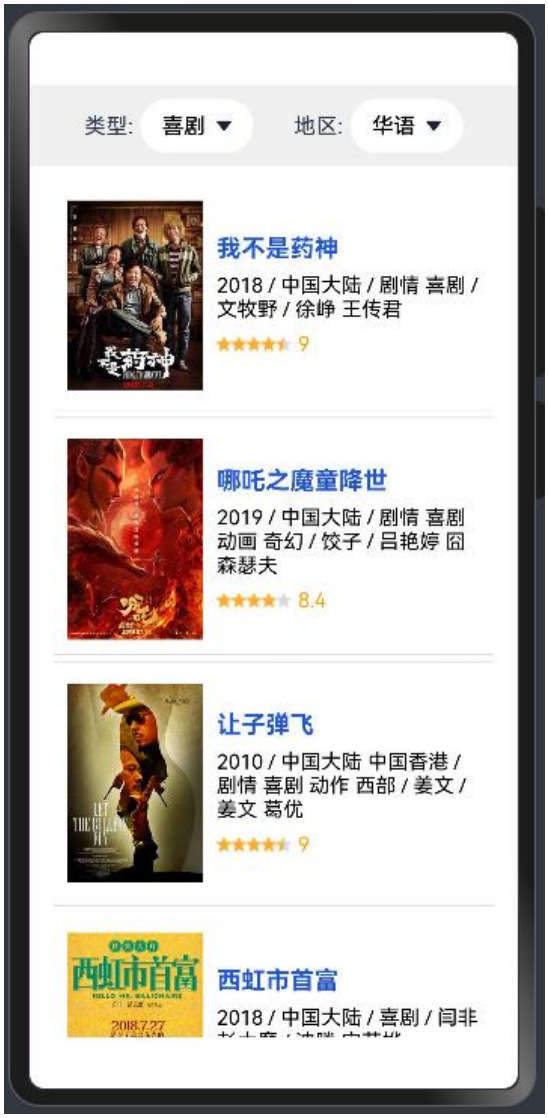
为了能够在 List 划到底部的时候，自动翻页增加更多的电影信息，我们需要在 MovieViewModel.ets 增加一个 loadMore 函数，代码内容如下，注意理解其中翻页追加数据的逻辑是如何实现的：

```
  async loadMore() {
    if (this.start < this.total) {
      await this.fetchData(false);
    }
  }
```

然后给 Index.ets 中的 List 组件，增加一个滚动响应函数，当检测到当前 List 滚动到底部的时候，异步的调用 loadMore 函数去翻页并载入数据。

```
          .onScrollIndex(async (start, end) => {
            if (!this.vm.isLoading && end === this.vm.movies.length - 1) {
              await this.vm.loadMore();
            }
          })
```

到目前为止，一个功能完备的通过网络查询和下载电影数据，并使用自定义组件显示电影信息的豆瓣电影客户端，就基本完成了：



五、实验注意事项

1. 注意教师的操作演示。

2. 学生机与教师机内网连通，能接收和提交实验结果。

3. 按实验要求输入测试数据，并查看输出结果是否符合实验结果。

六、思考题

1. 通过这个实验，你学到了什么？