湖南大學

HUNAN UNIVERSITY

《高级程序设计》
爬虫项目报告

报告题目：爬虫项目报告

学生姓名：郑诗艺

学生学号：202401070210

专业班级：大数据管理与应用 2402 班

学院名称：工商管理学院

指导老师：何卫华

2026 年 5 月 24 日

目录

图表目录

爬虫项目报告

一、项目目标

1.1 功能目标

表 1 功能目标表

功能	描述	优先级
多站点数据爬取	实现豆瓣电影、王者荣耀英雄、中国天气网天气数据的网络爬取，解析网页并提取有效字段	高
命令行菜单交互	提供 CLI 菜单，支持爬取、数据分析、历史数据导入、程序退出等操作	高
JSON 持久化存储	将爬取数据导出为 JSON 文件，支持历史数据导入，实现断点恢复	高
增量抓取与去重	自动过滤已采集数据，避免重复爬取、重复存储	中
数据统计分析	对电影、英雄、天气数据进行简单统计与可视化输出	中
异常与日志处理	统一异常捕获、分层异常处理，集成日志体系记录运行信息	中

1.2 预期效果

(1) 程序可稳定爬取豆瓣电影 Top250、王者荣耀英雄、全国城市天气数据，解析后存入对应实体类；

(2) 支持命令行菜单操作，一键执行爬取、分析、导入历史数据等功能，交互简洁直观；

(3) 爬取数据自动持久化为 JSON 文件，重启程序可一键导入历史数据，无需重复爬取；

(4) 实现增量抓取，重复执行仅获取新增数据，避免冗余；

(5) 网络、解析、IO 等异常友好提示，日志完整记录运行过程，便于调试与排错；

(6) 项目结构规范，采用 MVC、命令模式、策略模式，易于扩展新增爬虫站点。

二、项目进展

2.1 W1

1. 本周任务

(1) 项目选题：确定项目主题为 Java 网络爬虫与数据分析程序，选定三个目标爬取站点：豆瓣电影 Top250、王者荣耀英雄库、全国城市天气。明确整体业务流程：数据爬取 → 数据清洗 → 数据存储 → 统计分析 → 结果展示。

(2) 技术选型与环境搭建：选用 Java 作为开发语言，IDEA 作为开发工具，引入 Jsoup 框架用于网页请求与 HTML 解析；完成项目初始化，导入对应依赖 Jar 包，保证基础开发环境正常运行。

(3) MVC 架构包结构规划：按照 MVC 设计思想划分项目模块并创建对应包：

Model 层：model 包，用于存放数据实体类；

View 层：view 包，负责实现 CLI 控制台交互界面；

Controller 层：controller 包，承担业务调度、上下文管理工作；

基础功能包：crawler（爬虫相关类）、util（通用工具类）。

(4) 实体类开发：在 model 包下创建 Movie 实体类，仅封装电影名称、电影评分两个核心字段，提供成员变量、构造方法及 Getter/Setter 方法，用于封装爬取到的数据。

(5) 豆瓣电影基础爬虫开发：在 crawler 包下编写 MovieCrawler 基础爬虫类，借助 Jsoup 发起网络请求、解析网页标签，定向提取电影名称与评分数据；同时实现基础数据清洗逻辑，去除数据首尾空格、过滤空字符串等无效数据。

(6) 简易 CLI 控制台开发与测试：在 view 包编写基础 CLI 控制台代码，实现指令输入、结果打印功能；运行程序，测试豆瓣电影爬虫能否正常抓取数据，验证单爬虫功能可用。

2. 所学知识

(1) Java 开发环境使用：掌握 JDK 环境变量配置、IDEA 软件安装与基础设置；在 IDE 中新建 Java 项目、划分目录结构，掌握引入第三方 Jar 依赖的操作；理解 Java 代码编译、运行机制，学会使用 IDE 基础功能完成代码编写、运行与简单排错。

(2) 类与对象：类是对现实事物的抽象模板，用于定义属性和行为；对象是类的实例，代表具体个体。掌握类的完整定义格式，学会使用 new 关键字实例化对象，结合项目需求设计 Movie 实体类，完成数据载体的搭建。

(3) 面向对象封装：面向对象三大特性之一，核心是隐藏类内部数据，仅对外暴露合法访问方式。将实体类成员变量私有化，避免外部随意篡改数据；通过构造方法统一初始化对象属性，借助 getter 方法向外提供数据读取入口，提升代码安全性与规范性。

(4) 访问修饰符：学习 public 和 private 两种常用权限修饰符。private 用于修饰成员变量，仅限当前类内部访问，实现数据隐藏；public 用于修饰类、构造方法与 getter 方法，允许项目中所有类访问，保障跨类调用正常进行。

(5) this 关键字：用于区分局部变量与成员变量。在实体类构造方法中，当方法形参名和类的成员变量名一致时，使用 this.成员变量为类属性赋值，明确赋值目标，解决变量名冲突问题。

(6) String 字符串常用方法：掌握 String 类基础用法，重点使用 trim()方法剔除字符串首尾空格，配合条件判断语句过滤空字符串、空白内容，完成爬虫原始数据的基础清洗，保证数据格式统一。

(7) Jsoup 基础使用：了解网络爬虫基本工作原理，学习 Jsoup 核心 API。掌握调用方法发送 HTTP 网络请求、获取网页完整文档对象；学习 HTML 选择器语法，定位页面指定标签，精准提取标签内的文本数据，实现网页数据抓取。

(8) 控制台输入输出：使用 Scanner 类读取控制台输入内容，结合顺序、分支语句编写简易 CLI 交互菜单；掌握控制台信息打印方法，实现人机交互与程序运行结果展示。

3. 遇到的困难

(1) 引入 Jsoup Jar 包后，代码无报错，但运行程序时提示找不到对应类，程序运行失败。

(2) 不熟悉网页结构与 Jsoup 选择器语法，无法准确定位 HTML 标签，爬取不到电影名

称和评分数据。

(3) 初次划分项目包结构，对 MVC 分层逻辑理解不清晰，不清楚各类代码该放置在哪个包下，出现类文件乱放的情况。

4. 如何解决的

(1) 检查项目库设置，确认 Jar 包已成功添加到项目依赖库中，重新设置模块依赖，重启 IDE 后再次运行程序。

(2) 使用浏览器开发者工具查看豆瓣网页源码，分析目标数据所在标签、类名与层级关系，反复调试 Jsoup 选择器，逐步修正语法，实现数据精准提取。

(3) 重新回顾 MVC 各层职责，对照示例梳理 model、crawler、view 等包的作用，按照 “实体类入 model、爬虫类入 crawler、交互代码入 view” 的规则逐一调整文件位置，规范项目结构。

5. AI 是如何帮助的

(1) 知识点答疑：遇到 this 关键字、访问修饰符用法模糊等理论问题时，借助 AI 梳理语法规则、使用场景并搭配简易代码示例，快速理解知识点，弥补课堂学习盲区。

(2) 代码参考与纠错：编写实体类、Jsoup 请求代码、Scanner 控制台逻辑时，参考 AI 提供的标准代码模板，对比自身代码找出语法漏洞、逻辑缺陷；针对 Jsoup 选择器编写、输入流异常等问题，获取针对性修改方案。

(3) 项目结构指导：对 MVC 分包规则不明确时，通过 AI 了解各包功能定位与文件存放规范，快速理清项目整体架构，避免目录混乱。

(4) 问题排查思路：运行代码出现类找不到、数据抓取不全等报错时，AI 帮助分析报错成因，提供分步排查思路，缩短排错耗时，提升开发效率。

2.2 W2

1. 本周任务

(1) 对上周豆瓣电影爬虫代码进行重构，抽取网络请求、页面连接等通用逻辑，搭建抽象父

类 BaseCrawler。

(2) 改造已有 MovieCrawler，使其继承 BaseCrawler，复用父类通用方法，实现代码简化。

(3) 新建 HeroCrawler（王者荣耀英雄爬虫）、WeatherCrawler（全国天气爬虫）两个子类，同样继承抽象父类，完成三大爬虫基础骨架搭建。

(4) 分别编写两个新爬虫的页面解析逻辑，适配对应网站 HTML 结构，统一全项目基础数据清洗规则。

(5) 运用多态特性编写测试代码，使用父类引用指向不同爬虫子类对象，通过统一方式调用爬虫方法，验证多态调用效果。

(6) 进一步优化 MVC 分层结构，明确 Controller 调度逻辑、View 层 CLI 交互职责，完善项目整体架构。

2. 所学知识

(1) 抽象类与抽象方法：抽象类使用 abstract 关键字定义，无法实例化，主要用于抽取多个子类的公共属性与行为；抽象方法仅有方法声明、无方法体，强制子类必须重写实现，适合用来定义爬虫通用规范。

(2) 继承：面向对象三大特性之一，使用 extends 关键字实现类的继承。子类可以直接复用父类中非私有成员，减少重复代码，提升代码复用性与可维护性，本次用于统一三个爬虫的通用网络能力。

(3) 多态：基于继承实现的特性，父类引用指向子类对象，程序运行时会自动调用子类重写后的方法。可以用统一的调用方式操作不同子类对象，降低代码耦合度，适配多爬虫统一调度场景。

(4) 方法重写：子类对从父类继承来的非私有方法进行重新实现，要求方法名、参数列表、返回值与父类保持一致，是实现多态的前提，用于让不同爬虫实现各自独有的解析逻辑。

(5) 访问权限控制：结合继承关系，进一步区分 public、private、protected 修饰符的访问范围，理解不同权限成员在父类与子类之间的访问规则，合理封装通用方法与独有方法。

(6) 代码重构思想：在原有可运行代码基础上，不改变功能的前提下优化代码结构、抽取公共模块，梳理代码层级，让项目结构更清晰，便于后续功能扩展。

3. 遇到的困难

(1) 子类继承父类后，出现成员方法无法访问、权限不足的问题。

(2) 对原有豆瓣爬虫代码完成重构、改为继承结构后，原本可以正常运行的爬虫出现请求失败、数据为空的问题，不清楚是继承关系还是方法调用链路出现问题。

(3) 创建多个子类后，类数量增多，包内文件杂乱，分不清各个爬虫类对应的功能，后期查找和修改代码效率很低。

4. 如何解决的

(1) 复习访问修饰符规则，将父类需要被子类复用的方法调整为 protected 或 public，私有方法仅保留在父类内部使用。

(2) 先单独运行父类网络请求方法，排查网络连接是否正常；再逐级检查子类调用父类方法的代码，发现重构时删减了请求头、页面地址传参等代码。补全缺失代码，分模块依次测试网络请求、数据解析、结果输出，逐一修复问题，保证重构后原有功能不受影响。

(3) 在 crawler 包下按照业务分类新建子包，分别存放电影、英雄、天气对应的爬虫类，同时为每个类、核心方法补充注释说明，规范文件摆放位置，梳理代码目录结构。

5. AI 是如何帮助的

(1) 针对抽象类、继承、多态、方法重写等核心知识点，结合本次爬虫项目场景进行通俗讲解，区分易混淆语法规则，帮助我快速理解理论知识，并理清抽象类设计、类继承的整体思路。

(2) 提供抽象父类、子类继承、方法重写、多态测试的标准代码范例，我对照范例完成 BaseCrawler 及两个新爬虫子类的基础编码，减少语法错误，提升整体开发效率。

(3) 代码出现权限访问报错、方法重写格式错误、多态调用失效等问题时，协助逐行检查代码，定位问题根源，并给出可直接参考的修改方案，快速修复 BUG。

(4) 在代码重构阶段，指导我如何拆分通用逻辑与业务独有逻辑，规划抽象类结构，规避重构过程中容易出现的功能缺失、代码冗余等问题。

(5) 面对新网站 HTML 标签结构不同、Jsoup 选择器匹配失败的问题，根据网页结构协助

优化选择器表达式，缩短调试时间，顺利完成数据解析。

(6) 针对项目文件杂乱、代码结构臃肿等问题，给出目录分层、代码拆分、数组优化多态测试代码等合理化建议，帮助规范项目结构、简化代码逻辑。

2.3 W3

1. 本周任务

(1) 定义爬虫顶层接口 CrawlStrategy，声明 startCrawl()、parse()等核心抽象方法，统一所有爬虫的行为规范。

(2) 修改抽象父类 BaseCrawler，实现 CrawlStrategy 接口，搭建接口 + 继承 + 多态结合的整体爬虫架构。

(3) 按照要求修改接口中的 parse()方法，添加 throws ParseException 异常声明，提前规范网页解析异常的抛出规则。

(4) 完善 MovieCrawler、HeroCrawler、WeatherCrawler 三个子类代码，强制实现接口内全部抽象方法，保证代码符合接口约束。

(5) 优化 CLI 控制台交互界面，补充操作提示、功能说明等内容，提升控制台使用体验。

(6) 整体测试接口、继承、多态组合架构的运行效果，验证三类爬虫均可正常按照接口规范执行爬取逻辑。

2. 所学知识

(1) 接口的定义与使用：接口使用 interface 关键字声明，内部默认包含公共抽象方法，仅用于定义行为标准，不实现具体业务逻辑。接口不能实例化，主要作用是为不同类制定统一的功能规范，实现代码解耦。本周通过定义 CrawlStrategy 接口，为所有爬虫划定统一的执行标准，无论哪种类型的爬虫，都必须实现接口规定的核心方法。

(2) 类实现接口：类通过 implements 关键字实现接口，普通类实现接口后必须重写接口中全部抽象方法；抽象类实现接口可选择性实现方法，未实现的抽象方法将由其子类完成。本次让抽象父类 BaseCrawler 实现爬虫接口，既遵循接口行为约束，又保留抽象类代码复用的特性。

(3) 接口、抽象类与继承组合运用：抽象类偏向抽取通用属性与可复用代码，侧重代码复用；

接口偏向统一行为标准，侧重功能约束。本周将三者结合搭建爬虫架构，以抽象类承载通用网络逻辑，以接口统一爬虫行为，依靠继承与多态实现多子类调度，构建出层次清晰、易于扩展的类结构。

(4) throws 异常声明：在方法签名后使用 throws 关键字，可以声明该方法运行过程中可能抛出的受检异常，提醒方法调用者主动捕获异常或将异常继续向上抛出。本周在接口的 parse()方法中添加异常声明，从顶层统一规范了解析异常的传递规则。

(5) 接口多态：使用接口类型的引用指向接口实现类的对象，运行程序时会自动执行实现类中重写的方法。接口多态进一步降低了代码之间的耦合度，调度逻辑不再依赖具体父类，让整个爬虫架构具备更高的灵活性。

(6) 代码规范性约束：接口相当于开发协议，能够强制所有实现类遵守统一的方法命名、代码结构，搭配异常声明规则，可统一项目编码风格与异常处理逻辑，有效避免代码杂乱，提升项目整体标准化水平。

3. 遇到的困难

(1) 在接口 parse()方法添加 throws ParseException 异常声明后，子类重写对应方法时，异常声明写法混乱，出现编译报错。

(2) 使用接口类型引用调用爬虫对象（接口多态）时，只能调用接口中定义的方法，无法访问抽象父类中独有的通用方法，功能调用受限。

(3) 三个爬虫子类都实现接口后，发现部分辅助逻辑代码高度雷同，多处出现重复代码，后期维护时需要逐个修改，效率很低。

4. 如何解决的

(1) 学习重写方法的异常规则，子类重写方法抛出的异常范围不能大于父类 / 接口声明的异常，统一子类异常声明格式，与接口保持一致，修正语法错误。

(2) 理清引用类型的访问范围，如需调用父类独有方法，合理进行类型向上 / 向下转型；同时调整代码逻辑，将高频使用的通用行为补充至接口中，减少转型操作。

(3) 将重复的代码逻辑抽取到上层抽象父类中，子类在重写接口方法时直接调用父类对应方法，仅保留自身专属的解析逻辑，减少代码冗余。

5. AI 是如何帮助的

(1) 帮助区分接口与抽象类的定位和使用场景，结合爬虫项目分析哪些功能适合定义为接口方法、哪些逻辑适合放在抽象父类，指导精简接口结构，避免接口臃肿、代码重复的问题。

(2) 讲解抽象类实现接口的语法规则，解答抽象类不用实现全部接口方法的疑问，同时给出对应代码示例，快速理清多层类结构的编写逻辑。

(3) 针对重写方法的异常声明规则进行讲解，指出子类异常声明的常见错误，并对照我的代码逐一排查、给出修改方案，解决编译报错问题。

(4) 解释接口引用的访问范围限制，说明向上、向下转型的使用场景与写法，提供实用代码示例，解决无法调用父类独有方法的问题。

(5) 给出代码优化思路，指导我将多个子类中重复的逻辑抽取到抽象父类，减少代码冗余，提升代码复用性与可维护性。

(6) 提供控制台排版的编写思路与参考代码，借助换行符、分隔符优化 CLI 菜单布局，让交互界面清晰规整。

2.4 W4

1. 本周任务

(1) 开发 CrawlerContext 上下文类（Controller 层），搭建策略模式架构，支持动态切换豆瓣、英雄、天气三种爬虫策略。

(2) 引入命令模式，创建顶层 Command 接口，依次开发 MovieCrawlCommand、HeroCrawlCommand、WeatherCrawlCommand 具体命令类，封装各类爬虫的执行逻辑。

(3) 编写命令调用者 CommandInvoker，统一管理所有命令，解耦 CLI 指令触发与底层爬虫执行逻辑。

(4) 补全 HeroCrawler、WeatherCrawler 完整爬取逻辑，实现列表页、详情页数据解析，完成三大站点数据抓取。

(5) 整合 CLI + MVC + 命令模式 + 策略模式整套架构，通过控制台切换指令测试爬虫调度，验证架构可用性。

2. 所学知识

(1) 策略模式：一种行为型设计模式，将不同业务逻辑封装为独立策略类，借助上下文类统一调度，可在运行时动态切换业务实现，有效解耦算法与调用方，提升代码拓展性。

(2) 命令模式：一种行为型设计模式，把客户端请求封装为命令对象，分离指令发起者与具体执行者，统一指令调用入口，方便对各类操作进行管理、扩展与维护。

(3) 设计模式组合应用：掌握策略模式与命令模式协同工作的思路，结合前期接口、继承、多态等面向对象特性，理解多种设计模式在同一项目中的分工与协作方式。

(4) 模块调用与代码解耦：学习多模块项目中类与类之间的调用逻辑，理清不同功能模块的依赖关系，通过设计模式进一步降低代码耦合度，让各模块独立运作又能协同配合。

(5) 集成调试：掌握多模块代码联合调试的方法，跟踪跨类、跨模块的方法调用链路，排查模块衔接处出现的逻辑异常，保障程序整体流转正常。

(6) 输出格式标准化：学习通过封装统一输出方法，规范不同爬虫的数据打印样式与展示结构，让程序输出结果风格统一，提升代码规范性与交互体验。

3. 遇到的困难

(1) 初次搭建策略模式，不清楚上下文类 CrawlerContext 的职责，把对象创建、业务判断逻辑全部写在里面，导致类代码臃肿，切换爬虫策略时逻辑混乱。

(2) 整合多模块代码后，出现类之间循环依赖的情况，编译时直接报错，不清楚模块之间该如何合理传参、互相调用。

(3) 部分命令类在调用爬虫方法时，没有正确处理接口中声明的 ParseException 受检异常，代码直接抛出编译错误。

4. 如何解决的

(1) 明确上下文类仅负责接收并切换策略对象，将对象实例化、核心业务逻辑拆分到对应爬虫类中，简化上下文代码，按照标准流程实现策略切换功能。

(2) 梳理类与类之间的依赖关系，去掉双向引用，通过方法传参的方式传递对象，打破循环依赖，重新编译后报错消除。

(3) 按照异常处理规则，在命令类调用处使用 try-catch 捕获异常，或继续通过 throws 向上抛出，补齐异常处理代码后编译正常。

5. AI 是如何帮助的

(1) 讲解策略模式、命令模式的核心结构与各角色职责，结合本项目爬虫场景梳理执行流程，帮我分清上下文、命令调用者、具体命令类的分工，理清两类设计模式组合使用的逻辑。

(2) 提供策略模式、命令模式的标准代码框架与示例，参考范例快速搭建 CrawlerContext、命令接口及调用者相关代码，减少基础语法与结构错误。

(3) 分析循环依赖、权限访问报错的成因，给出调整类引用关系、修改方法访问修饰符的具体方案，快速解决编译报错。

(4) 针对调用者类分支代码冗余的问题，给出用集合管理命令对象的优化思路与参考代码，简化逻辑并提升代码扩展性。

(5) 结合项目已有的异常规则，讲解受检异常的处理方式，指导在命令类中完成 ParseException 异常捕获或上抛，补齐异常处理代码。

(6) 根据项目现有功能模块，给出分包规划建议，指导按类别划分子包，同时提醒添加代码注释，帮助规整项目目录结构。

2.5 W5

1. 本周任务

(1) 新建 exception 包，创建三大自定义异常类：CrawlerException（爬虫业务异常）、NetworkException（网络请求异常）、ParseException（网页解析异常），补充构造方法与异常描述，搭建完整异常体系。

(2) 梳理全项目代码，在网络请求、页面解析、数据处理、命令执行等环节完善异常抛出、捕获与传递逻辑。

(3) 改造 CrawlCommand 命令类、ScraperService 业务服务类，按照需求添加失败重试逻辑，配置重试次数与时间间隔，应对网络波动、临时访问失败问题。

(4) 在 CLI 视图层增加全局异常捕获，友好展示异常信息，避免程序直接崩溃。

(5) 针对性测试网络中断、页面解析失败等异常场景，验证异常处理与重试机制生效。

2. 所学知识

(1) 自定义异常体系：在 Java 中基于 Exception 基类创建项目专属异常，定义 CrawlerException 作为业务异常父类，再派生出 NetworkException、ParseException 等细分异常，用于精准标识不同业务场景下的错误类型。

(2) 异常抛出与传递机制：在网络请求、页面解析、命令执行等关键环节声明并抛出对应异常，通过 throws 关键字向上传递异常，形成贯穿整个项目的异常处理链路。

(3) 网络请求重试逻辑：通过循环计数、线程休眠实现请求失败重试，配置重试次数与时间间隔，在网络波动、临时访问失败时自动重试，提升程序稳定性。

(4) 全局异常捕获：在 CLI 视图层对所有异常进行集中捕获，区分不同异常类型并输出友好提示信息，避免单一异常导致整个程序崩溃。

(5) 异常场景测试：针对网络中断、页面节点缺失、解析错误等典型异常场景设计测试用例，验证异常分支逻辑是否正常生效，确保异常处理机制的可靠性。

(6) 命令模式与异常处理的结合：在命令类的 execute()方法中整合异常抛出逻辑，在命令调用者处统一捕获并传递异常，实现业务功能与异常体系的无缝融合。

(7) 异常信息规范：为每个自定义异常补充详细的异常描述信息，便于开发过程中快速定位问题原因，提升代码的可维护性。

3. 遇到的困难

(1) 在 BaseCrawler 的 getPage() 方法中抛出了 NetworkException，但调用该方法的上层代码没有声明或捕获异常，出现编译报错。

(2) 为网络请求添加重试逻辑后，重试次数和休眠时间设置不当，要么重试无效，要么请求延迟过高、效率低下。

(3) 在 CLI 视图层做全局异常捕获时，无法区分不同类型的异常，只能输出笼统的错误提示，用户无法判断是网络问题还是解析问题。

(4) 异常信息设置不规范，错误提示模糊，排查问题时无法快速定位是哪个爬虫、哪个环节出现了错误。

4. 如何解决的

(1) 在调用 getPage() 的方法上补充 throws NetworkException 声明，或在调用处使用 try-catch 块捕获异常，确保受检异常的处理符合 Java 语法规范。

(2) 通过 while 循环结合计数器控制重试次数，设置合理的重试次数（如 3 次）和休眠间隔（如 2 秒），同时在每次重试前记录日志，方便观察重试过程。

(3) 使用 instanceof 关键字判断异常类型，针对 NetworkException、ParseException 等不同异常，输出对应的友好提示信息，提升用户体验。

(4) 在抛出异常时补充具体上下文信息，如当前请求的 URL、爬虫名称等，例如 throw new NetworkException("请求失败：" + url, e)，通过异常信息快速定位问题点。

5. AI 是如何帮助的

(1) 协助梳理自定义异常体系结构，明确 CrawlerException 作为业务异常父类、NetworkException 与 ParseException 作为子类的继承关系，指导正确的异常创建语法，避免继承层级错误。

(2) 讲解受检异常的声明与传递规则，指导在 BaseCrawler、CrawlCommand 等方法中补充 throws 异常声明，解决编译报错问题，规范项目异常传递链路。

(3) 提供网络请求重试逻辑的实现思路，指导使用 while 循环结合计数器、Thread.sleep() 编写重试代码，配置合理的重试次数与时间间隔，提升爬虫容错能力。

(4) 指导在 CLI 视图层实现全局异常捕获，讲解 instanceof 关键字的使用，帮助区分不同异常类型并输出对应提示信息，避免程序因单一异常崩溃。

(5) 协助排查异常无法被捕获、异常传递混乱的问题，指导调整代码中异常抛出与捕获的顺序，确保异常能正确向上传递并被上层调用者处理。

(6) 指导优化异常信息编写方式，建议在异常描述中补充 URL、爬虫名称等上下文信息，帮助快速定位问题环节，提升代码可维护性。

(7) 讲解受检异常与非受检异常的区别，协助判断项目中各类异常的类型归属，指导合理选择异常抛出方式，避免不必要的编译负担。

2.6 W6

1. 本周任务

(1) 在项目资源目录添加 logback.xml 配置文件，设置日志级别、输出格式、控制台打印与本地日志文件存储规则。

(2) 为项目所有类添加 Logger 日志成员变量，将原有 System.out 控制台输出全部替换为日志输出，区分 info、error、warn 不同日志级别。

(3) 为所有 CrawlCommand 命令类补充 Logger 成员，并在 execute()方法开头添加指定日志：logger.info("Crawl started for:{}", url)，记录爬虫启动信息。

(4) 结合日志排查爬虫运行隐患，补充请求头、访问延时，降低网站反爬拦截概率。

(5) 统一代码格式，补充关键注释，进一步优化各层代码解耦性。

2. 所学知识

(1) Logback 日志框架配置：指通过 ogback.xml 配置文件定义日志系统的核心规则，包括日志输出的目标（控制台/文件）、日志级别、输出格式、文件存储路径、滚动策略等，实现日志行为的统一管控。

(2) 日志级别：日志框架中用于区分信息重要程度的等级划分，常见级别包括 info（常规运行信息）、warn（潜在风险警告）、error（错误信息），不同级别可独立配置输出规则，便于问题排查。

(3) Logger 对象：日志框架中用于记录日志的核心对象，通过 LoggerFactory.getLogger()获取实例，提供 info()、warn()、error()等方法，替代 System.out 实现标准化日志输出。

(4) 日志上下文记录：在程序关键节点记录与执行流程相关的上下文信息（如请求 URL、操作指令），便于问题发生时快速定位问题环节，提升排查效率。

(5) 日志驱动调试：通过分析日志内容识别程序运行隐患（如请求失败、节点解析异常），针对性优化请求配置（如请求头、访问延时），降低反爬拦截风险。

(6) 代码解耦：指降低模块间的依赖关系，减少不必要的耦合，提升模块的独立性与可扩展性，便于后续功能迭代与维护。

3. 遇到的困难

(1) 对日志级别区分不清晰，info、warn、error 使用混乱，日志输出层级杂乱，不利于问题排查。

(2) 项目中 System.out 语句与新日志代码共存，控制台输出重复、信息混乱，无法实现标准化日志管理。

(3) Logback 配置文件书写不规范、存放路径错误，导致日志框架加载失败，项目无法输出日志。

(4) 不熟悉 Logger 对象使用规范，容易在方法内部重复创建日志实例，造成资源冗余。

4. 如何解决的

(1) 明确各级日志使用规范，普通运行流程使用 info、潜在风险提示使用 warn、程序异常报错使用 error，统一全项目日志输出标准。

(2) 批量遍历项目所有类，删除全部原生控制台输出语句，统一替换为对应级别的日志输出，彻底规范项目输出方式。

(3) 将 logback.xml 放置在项目标准资源目录下，修正配置标签语法，正确设置控制台输出、文件输出、日志级别等节点内容，使配置文件正常加载生效。

(4) 遵循开发规范，在每个类中静态声明唯一 Logger 日志对象，全局统一复用，避免重复创建对象浪费系统资源。

5. AI 是如何帮助的

(1) 指导 logback.xml 配置文件的编写规范，讲解文件标准存放路径、各配置节点含义与语法要求，协助排查配置错误，保证日志框架正常加载运行。

(2) 讲解不同日志级别的适用场景，区分 info、warn、error 的使用边界，协助统一全项目日志输出规范。

(3) 说明 Logger 对象的声明与使用规范，建议在类中静态定义日志实例，避免重复创建对象，同时指导批量替换 System.out 语句，完成输出方式统一。

(4) 讲解日志占位符的语法规则与使用方法，示范动态参数的日志打印写法，保证日志信息

记录完整。

(5) 讲解通过日志分析程序运行状态的思路，结合日志内容给出优化请求头、设置访问延时的建议，规避反爬风险。

(6) 讲解代码耦合的判断依据与解耦基本思路，指导在不改动原有业务逻辑的前提下，精简模块依赖，提升代码独立性。

2.7 W7

1. 本周任务

(1) 引入 FastJSON2 依赖，编写工具类 DataUtil，实现 JSON 序列化持久化导出，将爬取的实体数据保存为本地 JSON 文件。

(2) 在工具类中新增文件读取方法，实现数据导入、恢复历史会话功能，支持加载本地历史爬取数据。

(3) 项目中所有文件流、网络流统一使用 try-with-resources 语法，实现资源自动关闭，完成安全资源管理。

(4) 新建 repository 数据仓库层，统一管理数据读写操作，并为 Repository 层增加防御性检查，校验空对象、空集合、非法数据，规避空指针与脏数据问题。

(5) 开发增量抓取功能：读取本地历史数据，对比数据唯一标识，仅抓取新增数据，彻底避免重复爬取与数据冗余。

(6) 全流程联调测试：CLI 指令 → 命令调用 → 策略切换 → 增量判断 → 数据清洗 → JSON 导出 → 数据导入，验证整套链路稳定运行。

2. 所学知识

(1) FastJSON2 序列化与反序列化：JSON 序列化是将 Java 对象、集合数据转换为标准 JSON 字符串的过程，用于数据持久化存储；反序列化是将本地 JSON 文件内容重新解析为 Java 对象，实现数据读取与恢复。

(2) 本地文件持久化存储：将程序运行产生的内存数据，以文件形式保存到本地磁盘，实现程序重启后数据不丢失，完成数据持久化会话保留。

(3) 会话数据恢复机制：通过读取本地已保存的持久化文件，将历史数据重新加载进程序内

存，恢复上一次程序运行的数据状态。

(4) try-with-resources 自动资源管理：JDK 提供的自动资源释放语法，可自动关闭所有实现 AutoCloseable 接口的流资源，无需手动 close，避免 IO 资源泄露与文件占用问题。

(5) 分层架构 ——Repository 数据仓库层：专门用于统一管理项目所有数据读写、文件操作、持久化操作，分离业务层与 IO 层代码，实现职责解耦。

(6) 防御性编程：在数据读写前对空对象、空集合、空字符串、非法数据进行预先校验，提前拦截异常数据，防止空指针异常与脏数据入库。

(7) 增量数据抓取原理：通过读取历史数据唯一标识，与新爬取数据做比对过滤，只保留新增数据，避免全量重复爬取，解决数据冗余叠加问题。

3. 遇到的困难

(1) JSON 反序列化读取本地文件时，容易因文件损坏、格式不规范导致解析失败，程序抛出解析异常。

(2) 传统 IO 流写法需要手动关闭资源，容易遗忘关闭，造成资源泄露、文件被占用、无法二次写入等问题。

(3) 对象转 JSON 时出现字段丢失、格式错乱、多余空字段等问题，无法正常生成规范 JSON 文件。

(4) 初次实现增量抓取逻辑时，无法正确比对历史数据，导致每次爬取依旧重复写入大量冗余数据。

4. 如何解决的

(1) 在读取方法中增加异常捕获与容错处理，识别损坏文件时给出提示，文件不存在时返回空集合，保证程序不会崩溃。

(2) 全部改用 try-with-resources 语法管理所有文件流、字符流，程序执行结束自动释放资源，彻底解决资源泄露问题。

(3) 熟悉 FastJSON2 序列化机制，规范实体类结构，配置格式化输出规则，保证所有字段正常序列化，生成标准、整洁的 JSON 文件。

(4) 读取历史数据唯一标识并存入集合，新数据通过过滤比对，只保留从未爬取过的新增数

据。

5. AI 是如何帮助的

(1) 指导 FastJSON2 依赖引入与环境配置，讲解 JSON 序列化、反序列化核心用法，示范实体对象转 JSON、JSON 还原对象的标准代码，解决数据导出格式错乱、字段丢失问题。

(2) 讲解文件 IO 读写规范，指导封装统一的数据导入、导出工具方法，实现本地数据持久化与历史会话恢复功能。

(3) 科普 try-with-resources 自动资源管理机制，协助改造全部文件流、网络流代码，规避资源泄露、文件占用等 IO 隐患。

(4) 指导理解 Repository 分层思想，协助搭建数据仓库层架构，统一收拢所有数据读写逻辑，实现业务代码与数据存储代码解耦。

(5) 讲解防御性编程思想，指导在数据操作入口添加空对象、空集合、非法数据校验逻辑，有效预防空指针异常和脏数据问题。

(6) 梳理增量抓取的实现思路，指导通过唯一标识比对、集合去重的方式过滤重复数据，成功实现增量爬取，解决数据冗余问题。

(7) 协助梳理全项目执行链路，讲解 CLI 指令、命令模式、策略模式、数据持久化的联动逻辑，指导全流程联调，排查并修复隐性 Bug，保证整套流程稳定运行。

2.8 W8

1. 本周任务

(1) 读取本地 JSON 历史数据，借助 Java 集合框架（List、Map）对三类数据开展多维度统计分析。

(2) 电影数据分析：基于名称、评分两个字段，统计电影平均评分和 8.5 分以上的电影数量。

(3) 天气数据分析：统计各天气类型的城市数量、最高温、最低温与平均温度。

(4) 英雄数据分析：统计所有英雄数量。

(5) 完善 CLI 主菜单，新增数据分析操作指令，整合项目全部业务功能。

(6) 开展全场景功能测试，修复遗留 Bug，优化程序运行效率与控制台交互体验。

2. 所学知识

(1) Java 集合框架：List 集合用于有序存储批量实体对象，支持遍历、筛选操作；Map 集合可按照指定字段对数据分组归类，配合集合遍历、条件判断、元素检索等方法，完成批量数据的提取与汇总。

(2) 数值统计计算：基于实体类中的数值型字段进行运算，包含遍历数据集求取平均值、筛选指定数值区间内的数据、查找数据集最大值与最小值、统计集合元素总数量等基础数据计算方法。

(3) 多维度数据分析：结合不同业务数据的字段特征制定统计规则，针对不同类型数据设计差异化统计逻辑，从多个角度挖掘已有数据信息。

(4) 控制台交互拓展：在原有 CLI 菜单体系中新增功能选项与对应指令，绑定指令与数据分析逻辑，整合爬取、存储、分析等全部业务功能，完善整体交互流程。

3. 遇到的困难

(1) 利用 Map 对天气数据分组时，分组规则设计不合理，出现分类错乱、数据归属错误的问题。

(2) 项目功能整合后，整体运行流程变长，部分模块执行速度慢，程序运行效率偏低。

(3) 本地 JSON 数据为空或数据量较少时，直接执行统计逻辑会触发空指针、除数为零等异常。

4. 如何解决的

(1) 明确分组依据，以天气类型作为 Map 的键，对应城市数据集合作为值，遍历数据时按规则完成归类，核对分组结果确保准确。

(2) 梳理代码逻辑，删减重复遍历、重复读取文件等冗余操作，精简执行步骤，提升程序整体运行速度。

(3) 在统计代码执行前增加数据校验，判断集合是否为空，针对无数据、数据不足等边界场景做容错处理，避免程序报错。

5. AI 是如何帮助的

(1) 讲解 List、Map 集合的遍历、筛选与分组用法，示范遍历统计、条件筛选的代码写法，协助梳理统计逻辑，解决数据重复计算、漏统计的问题。

(2) 提供浮点数格式化处理方案，指导对平均评分等计算结果进行小数位数保留，规范数值展示格式。

(3) 讲解 Map 集合分组实现思路，指导以业务字段作为键、对应数据集合作为值完成归类，修正分组错乱、数据归属错误等问题。

(4) 协助梳理 CLI 菜单分支与指令跳转逻辑，核对指令和功能代码的绑定关系，修复指令失效、指令冲突等问题。

(5) 指导添加边界条件判断，针对空集合、数据量不足等场景编写容错代码，规避空指针、除数为零等运行异常。

(6) 分析代码中重复遍历、重复读取文件等冗余逻辑，给出代码精简优化建议，提升程序整体运行效率。

(7) 提供控制台排版优化思路，指导使用换行、分隔符、文字标注等方式分区展示统计结果，提升界面可读性与交互体验。

(8) 协助划分测试模块与测试场景，结合日志排查隐性问题，定位代码缺陷并给出对应的修复方案，保障全功能稳定运行。

三、项目结构

3.1 最终包结构

本项目采用 MVC 分层架构结合多种设计模式构建，整体结构清晰、职责边界明确，实现了业务逻辑与用户交互、数据存储的解耦。

1. 分层设计：通过 view（CLI 交互）、controller（上下文管理）、command（命令调度）、crawler（业务实现）、model（数据模型）、util（通用工具）、exception（异常体系）的分层组织，使不同模块各司其职，降低耦合度，便于维护与扩展。
2. 设计模式应用：使用命令模式封装所有用户操作，将请求、执行与撤销解耦，支持 CLI 指令的灵活扩展与管理；使用策略模式封装不同数据源的爬虫逻辑，统一 BaseCrawler

接口，新增爬虫无需修改现有业务代码。

3. 可扩展性与可维护性：新增业务只需添加对应 Command 和 Crawler 实现，无需改动核心调度逻辑；统一的 DataUtil 工具类与自定义异常体系，保证了数据持久化与错误处理的一致性。
4. 配置与日志管理：通过 logback.xml 集中配置日志规则，实现运行状态的可追踪；.gitignore 规范版本控制，避免无关文件污染仓库。

```
CrawlerMain2
├── .idea # IDEA 配置文件
├── out # 编译输出目录
└── src
├── command # 命令模式模块
│ ├── AnalyzeCommand
│ ├── CommandInvoker
│ ├── CrawlCommand
│ ├── HeroCrawlCommand
│ ├── MovieCrawlCommand
│ ├── WeatherAnalyzeCommand
│ └── WeatherCrawlCommand
├── controller # 控制器与上下文
│ └── CrawlerContext
├── crawler # 策略模式爬虫实现
│ ├── BaseCrawler
│ ├── Crawler
│ ├── HeroCrawler
│ ├── MovieCrawler
│ └── WeatherCrawler
├── exception # 自定义异常体系
├── model # 数据实体类
│ ├── Hero
│ ├── Movie
│ └── Weather
├── util # 工具类
│ └── DataUtil # JSON 导入导出、增量去重、文件 IO
├── view # 视图层（CLI 菜单交互）
│ └── CrawlerView
├── CrawlerMain # 程序入口
└── logback.xml # 日志配置
├── .gitignore # Git忽略文件配置
└── CrawlerMain2.iml # IDEA模块配置
```

图 1 最终包结构

3.2 类图

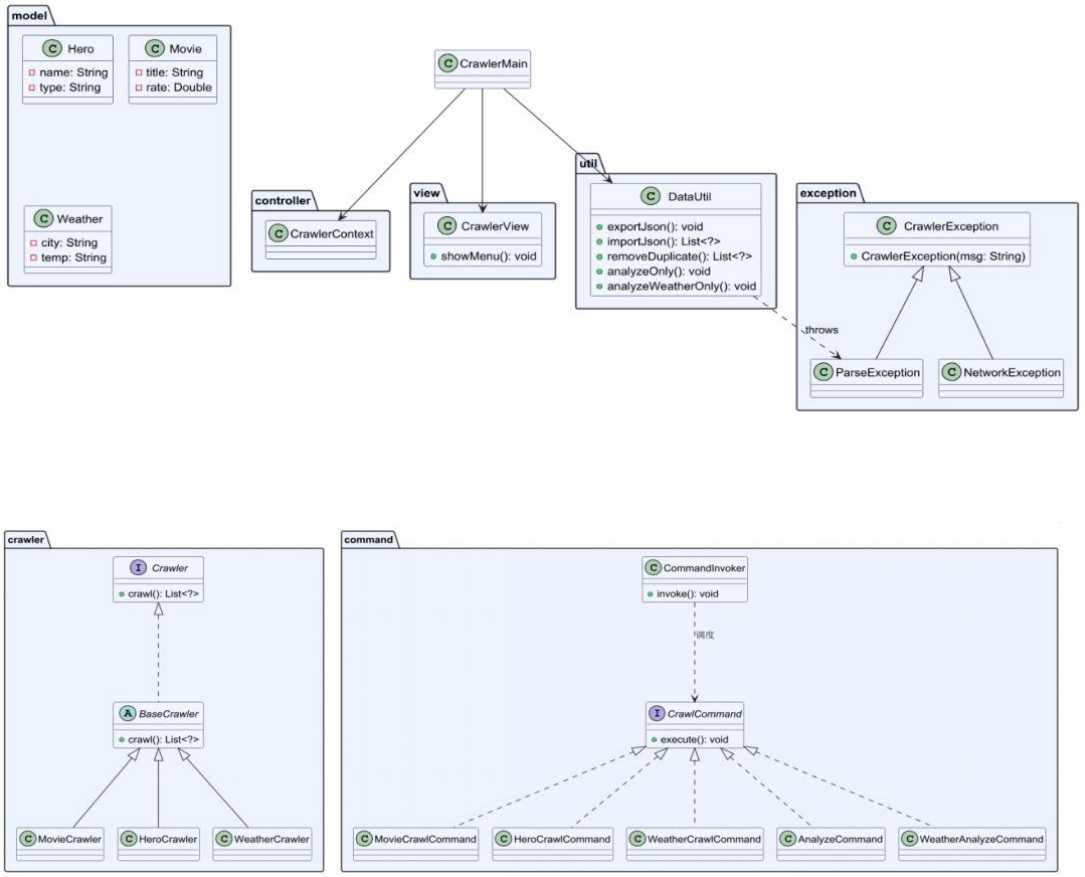

图 2 类图

类图说明：

1. crawler 包（策略模式）

Crawler 为顶层爬虫接口，定义 crawl() 方法规范数据爬取行为；BaseCrawler 为抽象类，实现 Crawler 接口并封装网络请求、重试、解析等通用爬取逻辑；MovieCrawler、HeroCrawler、WeatherCrawler 继承该抽象类，分别实现豆瓣电影、王者荣耀英雄、全国天气数据的专属爬取策略，符合策略模式设计思想。

2. command 包（命令模式）

CrawlCommand 为命令顶层接口，定义 execute() 统一执行方法；各类具体业务命令实现该接口，封装爬取、数据分析、数据导入等操作；CommandInvoker 作为命令调用者，统一调度各类命令，实现请求发起与业务执行解耦，满足命令模式设计要求。

3. model 包（数据模型层）

包含 Movie、Hero、Weather 三个实体模型类，通过私有属性存储电影、英雄、天气的业务数据，作为系统的数据载体，完成数据封装与传递。

4. MVC 支撑模块

controller 包的 CrawlerContext 管理程序全局上下文数据；view 包的 CrawlerView 实现 CLI 交互菜单展示、用户输入接收等视图功能；util 包的 DataUtil 封装 JSON 文件导入导出、数据去重、数据分析等通用工具方法，为业务层提供数据持久化与统计支持。

5. exception 包（异常处理模块）

自定义 CrawlerException 作为基础异常父类，派生出 NetworkException 网络异常、ParseException 数据解析异常，统一捕获并处理网络故障、JSON 格式错误等爬虫运行异常，提升系统鲁棒性。

6. 主入口类

CrawlerMain 为程序启动入口，关联调用上下文、视图、工具类与命令调度器，串联各功能模块，驱动系统整体运行。

四、成果展示

4.1 运行截图

4.1.1 程序启动

程序启动后，控制台首先输出日志信息，表明程序以“CLI+MVC+命令模式+策略模式”的架构成功启动。随后系统进入 CLI 交互主菜单，向用户展示全部可执行功能选项，包括爬取豆瓣电影、爬取王者荣耀英雄、爬取全国天气、电影&英雄数据分析、天气数据分析、导入历史数据以及退出程序，如图 3 所示。用户可通过输入对应数字指令，快速触发不同业务流程，实现了简洁直观的人机交互入口。

图 3 程序启动与 CLI 主菜单界面

4.1.2 执行爬虫

1. 爬取豆瓣电影 TOP250

用户输入操作指令 1 后，程序进入豆瓣电影 TOP250 的爬取流程，如图 4 所示。执行过程如下：程序首先读取本地历史电影数据（共 250 条），随后启动爬虫任务。通过日志可以清晰看到，程序按分页策略依次请求豆瓣电影 TOP250 的不同页面（`start=0/25/50...`），记录每一次请求的 URL 与状态。爬取完成后，系统自动将数据写入本地文件，并以 JSON 格式导出，最终提示本次无新增数据，说明增量去重逻辑已生效。

整个过程由日志系统完整记录，包含数据导入、请求过程、爬取结果、文件保存等关键节点，确保流程可追溯。

```
请输入操作指令：1
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by com.alibaba.fastjson2.util.UnsafeUtils (file:/C:/
WARNING: Please consider reporting this to the maintainers of class com.alibaba.fastjson2.util.UnsafeUtils
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
20:31:55.401 [INFO] command.MovieCrawlCommand - 导入历史电影数据：250条
20:31:55.446 [INFO] crawler.MovieCrawler - 开始爬取豆瓣电影Top250
20:31:55.447 [INFO] crawler.BaseCrawler - 第1次请求页面：https://movie.douban.com/top250?start=0
20:31:57.678 [INFO] crawler.BaseCrawler - 第1次请求页面：https://movie.douban.com/top250?start=25
20:31:58.859 [INFO] crawler.BaseCrawler - 第1次请求页面：https://movie.douban.com/top250?start=50
20:32:00.133 [INFO] crawler.BaseCrawler - 第1次请求页面：https://movie.douban.com/top250?start=75
20:32:01.242 [INFO] crawler.BaseCrawler - 第1次请求页面：https://movie.douban.com/top250?start=100
20:32:07.387 [INFO] crawler.BaseCrawler - 第1次请求页面：https://movie.douban.com/top250?start=225
20:32:08.498 [INFO] crawler.MovieCrawler - 豆瓣电影爬取完成，共250条数据
20:32:08.509 [INFO] util.DataUtil - 文件保存成功：电影数据.txt
20:32:08.575 [INFO] util.DataUtil - ✅ JSON文件导出成功：movie.json
20:32:08.575 [INFO] command.MovieCrawlCommand - 电影爬取完成，本次新增：0条
电影爬取完成
20:32:08.575 [INFO] view.CrawlerView - 电影爬取完成
```

图 4 豆瓣爬取过程截图

爬取结果通过 movie.json 文件持久化存储如图 5 所示，JSON 文件中以数组形式保存了结构化的电影数据，每条记录包含电影标题、字符串类型评分、数值类型评分三个字段，如《肖申克的救赎》《霸王别姬》《泰坦尼克号》《阿甘正传》等数据已被完整序列化导出，如图 6 所示。

图 5 爬取生成的电影数据 JSON 文件

图 6 电影数据 JSON 文件内容

2. 爬取王者荣耀英雄

用户输入指令 2 后，程序执行王者荣耀英雄数据爬取流程，如图 7 所示。首先导入本地历史数据，随后调用 HeroCrawler 发起对英雄列表页面的请求，完成数据解析后，将爬取结果写入文本文件并导出为 JSON 格式文件。日志显示本次共爬取 93 条数据，因与历史数据完全一致，本次新增数据为 0。

```
===== 爬虫CLI交互菜单 =====
1. 爬取豆瓣电影
2. 爬取王者荣耀英雄
3. 爬取全国天气
4. 电影&英雄数据分析（仅统计，不存储）
5. 天气数据分析（天气类型、最高/最低/平均温）
6. 导入历史数据
0. 退出程序
请输入操作指令：2
20:53:47.328 [INFO] command.HeroCrawlCommand - 导入历史英雄数据：93条
20:53:47.344 [INFO] crawler.HeroCrawler - 开始爬取王者荣耀英雄数据
20:53:47.344 [INFO] crawler.BaseCrawler - 第1次请求页面：https://pvp.qq.com/web201605/herolist.shtml
20:53:47.546 [INFO] crawler.HeroCrawler - 英雄爬取完成，共93条数据
20:53:47.548 [INFO] util.DataUtil - 文件保存成功：英雄数据.txt
20:53:47.586 [INFO] util.DataUtil - ✅ JSON文件导出成功：hero.json
20:53:47.586 [INFO] command.HeroCrawlCommand - 英雄爬取完成，本次新增：0条
英雄爬取完成
20:53:47.586 [INFO] view.CrawlerView - 英雄爬取完成
```

图 7 王者荣耀英雄爬取过程截图

下方展示了生成的 hero.json 文件，作为爬取结果的持久化载体，为后续数据分析提供了结构化数据源。

图 8 爬取生成的英雄数据 JSON 文件

图 9 英雄数据 JSON 文件内容

3. 爬取全国天气

用户输入指令 3 后，程序执行全国城市天气数据爬取流程如图 10 所示，首先导入本地历史天气数据（共 34 条），随后启动天气爬虫，依次请求多个城市的天气页面 URL。爬取完成后，将数据写入文本文件并导出为 weather.json 文件。日志显示本次共爬取 34 条数据，且与历史数据完全一致，本次新增数据为 0。

```
请输入操作指令：3
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by com.alibaba.fastjson2.util.UnsafeUtils (file:/C:/U
WARNING: Please consider reporting this to the maintainers of class com.alibaba.fastjson2.util.UnsafeUtils
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
21:19:32.182 [INFO] command.WeatherCrawlCommand - 导入历史天气数据：34条
21:19:32.210 [INFO] crawler.WeatherCrawler - 开始爬取全国城市实时温度数据
21:19:32.210 [INFO] crawler.BaseCrawler - 第1次请求页面：https://www.weather.com.cn/weather/101010100.shtml
21:19:33.576 [INFO] crawler.BaseCrawler - 第1次请求页面：https://www.weather.com.cn/weather/101020100.shtml
21:19:34.307 [INFO] crawler.BaseCrawler - 第1次请求页面：https://www.weather.com.cn/weather/101030100.shtml
21:19:35.054 [INFO] crawler.BaseCrawler - 第1次请求页面：https://www.weather.com.cn/weather/101040100.shtml
21:19:36.088 [INFO] crawler.BaseCrawler - 第1次请求页面：https://www.weather.com.cn/weather/101090101.shtml
21:19:57.659 [INFO] crawler.BaseCrawler - 第1次请求页面：https://www.weather.com.cn/weather/101320101.shtml
21:19:58.548 [INFO] crawler.BaseCrawler - 第1次请求页面：https://www.weather.com.cn/weather/101330101.shtml
21:19:59.325 [INFO] crawler.BaseCrawler - 第1次请求页面：https://www.weather.com.cn/weather/101340101.shtml
21:20:00.351 [INFO] crawler.WeatherCrawler - 实时天气爬取完成，共34条数据
21:20:00.355 [INFO] util.DataUtil - 文件保存成功：天气数据.txt
21:20:00.403 [INFO] util.DataUtil - ✅ JSON文件导出成功：weather.json
21:20:00.404 [INFO] command.WeatherCrawlCommand - 天气爬取完成，本次新增：0条
天气爬取完成
21:20:00.404 [INFO] view.CrawlerView - 天气爬取完成
```

图 10 中国各省份天气爬取过程截图

weather.json 文件以标准 JSON 数组格式存储了结构化的天气数据，每条记录包含城市名、省份、天气状况、数值温度（tempNum）与文本温度（temperature）等字段，如哈尔滨、南京、杭州、合肥等城市的天气信息已被完整序列化导出，如图 11 所示。

图 11 天气数据 JSON 文件内容

4.1.3 数据分析

数据分析模块包含电影、英雄与天气三类数据的统计功能，用户通过 CLI 菜单输入对应指令即可触发。

1. 电影、英雄数据分析

输入指令 4 后，程序读取本地 JSON 文件中的电影与英雄数据，完成统计计算：电影平均评分为 8.95 分，其中 8.5 分以上的电影共 249 部；王者荣耀英雄总数为 93 个。整个过程仅进行内存计算，不修改或存储原始数据。

```
===== 爬虫CLI交互菜单 =====
1. 爬取豆瓣电影
2. 爬取王者荣耀英雄
3. 爬取全国天气
4. 电影&英雄数据分析（仅统计，不存储）
5. 天气数据分析（天气类型、最高/最低/平均温）
6. 导入历史数据
0. 退出程序
请输入操作指令：4
21:43:40.323 [INFO] util.DataUtil - ===== 电影&英雄数据分析（仅统计，不存储） =====
电影平均评分：8.95
8.5分以上电影数量：249
英雄总数量：93
21:43:40.355 [INFO] util.DataUtil - 电影&英雄数据分析结束
21:43:40.355 [INFO] command.AnalyzeCommand - 电影&英雄数据分析命令执行完成（仅统计）
```

图 12 电影与英雄数据分析结果

2. 天气数据分析

输入指令 5 后，程序对全国天气数据进行多维度统计：按天气类型统计城市数量（如多云 14 个、小雨 7 个、中雨 4 个等），并计算出最高温度 28℃、最低温度 7℃、平均温

度 20.2℃。

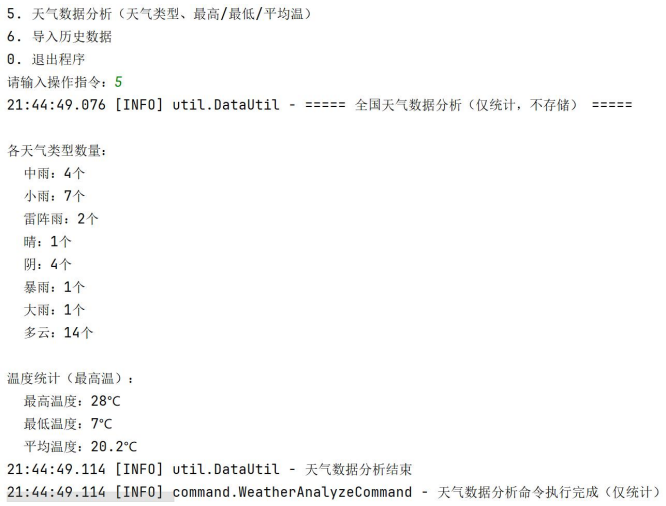

图 13 天气数据分析结果

4.1.4 导入 JSON

用户在 CLI 菜单中输入指令 6，触发历史数据导入功能。程序读取本地存储的 JSON 文件，成功加载电影、英雄与天气三类数据，并在控制台输出导入结果：电影数据 250 条、英雄数据 93 条、天气数据 34 条。该功能验证了 JSON 文件反序列化与会话恢复逻辑的正确性，确保程序可基于历史数据继续执行后续的分析或增量爬取任务。

图 14 历史数据导入功能执行截图

4.1.5 异常处理

1. 网络异常

在爬虫程序运行过程中，若出现网络异常（如域名解析失败、连接超时等），程序会触发自定义的重试与错误处理机制。日志显示，当用户执行电影爬取指令时，请求豆瓣电影页面时抛出 java.net.UnknownHostException 异常，表明域名解析失败。程序按照预设策略进行了重试，在 3 次请求失败后终止爬取流程，并输出错误提示信息：“网络异常：页面请求重试 3 次后仍失败”，同时将错误信息反馈至视图层，告知用户请求失败的 URL。该场景验证了项目中网络异常处理、重试机制与错误日志记录功能的有效性，保障了程序在不稳定网络环境下的健壮性。

```
请输入操作指令：1
21:48:15.714 [INFO] command.MovieCrawlCommand - 导入历史电影数据：250条
21:48:15.716 [INFO] crawler.MovieCrawler - 开始爬取豆瓣电影Top250
21:48:15.716 [INFO] crawler.BaseCrawler - 第1次请求页面：https://movie.douban.com/top250?start=0
21:48:15.719 [ERROR] crawler.BaseCrawler - 请求页面失败，剩余重试次数：2
java.net.UnknownHostException Create breakpoint : movie.douban.com
    at java.base/sun.nio.ch.NioSocketImpl.connect(NioSocketImpl.java:574)
    at java.base/java.net.SocksSocketImpl.connect(SocksSocketImpl.java:284)
    at java.base/java.net.Socket.connect(Socket.java:668)
    at java.base/sun.security.ssl.SSLSocketImpl.connect(SSLSocketImpl.java:304)
    at java.base/sun.net.NetworkClient.doConnect(NetworkClient.java:161)
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:516)
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:604)
Caused by: java.net.UnknownHostException Create breakpoint : movie.douban.com
    at java.base/sun.nio.ch.NioSocketImpl.connect(NioSocketImpl.java:574)
    at java.base/java.net.SocksSocketImpl.connect(SocksSocketImpl.java:284)
    at java.base/java.net.Socket.connect(Socket.java:668)
    at java.base/sun.security.ssl.SSLSocketImpl.connect(SSLSocketImpl.java:304)
    at java.base/sun.net.NetworkClient.doConnect(NetworkClient.java:161)
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:516)
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:604)
    at java.base/sun.net.www.protocol.https.HttpsClient.<init>(HttpsClient.java:206)
<11 个折叠帧>
    ... 5 common frames omitted
网络异常：页面请求重试3次后仍失败：https://movie.douban.com/top250?start=0
21:48:19.735 [INFO] view.CrawlerView - 网络异常：页面请求重试3次后仍失败：https://movie.douban.com/top250?start=0
```

图 15 网络异常场景处理截图

2. 数据解析异常

在执行 “导入历史数据” 指令时，程序读取 JSON 文件过程中触发了解析异常。日志显示 util.DataUtil 模块抛出 JSON 格式解析错误，提示 syntax error，表明文件内容存在语法错误（如括号不匹配、格式不规范）。程序捕获该异常并输出错误日志，明确错误类型与位置信息，实现了数据解析阶段的异常捕获与日志记录，保障了程序在读取损坏或格式错误文件时的稳定性，避免因文件异常导致程序崩溃。

```
===== 爬虫CLI交互菜单 =====
1. 爬取豆瓣电影
2. 爬取王者荣耀英雄
3. 爬取全国天气
4. 电影&英雄数据分析（仅统计，不存储）
5. 天气数据分析（天气类型、最高/最低/平均温）
6. 导入历史数据
0. 退出程序
请输入操作指令：6
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by com.alibaba.fastjson2.util.UnsafeUtils (file:/C:/Us
WARNING: Please consider reporting this to the maintainers of class com.alibaba.fastjson2.util.UnsafeUtils
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
21:57:31.970 [ERROR] util.DataUtil - JSON格式解析错误：syntax error : {, offset 3, character {, line 2, column 4, fa
```

图 16 JSON 数据解析异常处理截图

3. 输入指令非法异常

在 CLI 交互菜单中，当用户输入非数字字符（如字母 a）作为操作指令时，程序触发了输入指令非法异常处理逻辑。控制台立即提示“请输入数字！”，并记录该提示信息，验证了输入校验机制的有效性。该机制能够对用户输入进行合法性校验，拒绝非预期输入，避免程序因无效指令而崩溃，提升了交互界面的健壮性与用户体验。

```
===== 爬虫CLI交互菜单 =====
1. 爬取豆瓣电影
2. 爬取王者荣耀英雄
3. 爬取全国天气
4. 电影&英雄数据分析（仅统计，不存储）
5. 天气数据分析（天气类型、最高/最低/平均温）
6. 导入历史数据
0. 退出程序
请输入操作指令： a
请输入数字!
21:59:36.735 [INFO] view.CrawlerView - 请输入数字!
```

图 17 CLI 非法输入异常处理截图

4.2 功能测试

表 2 功能测试表

功能	测试结果	备注
程序启动与 CLI 菜单交互	正常	可正常启动程序，展示功能选择菜单，支持指令输入
豆瓣电影数据爬取	正常	成功获取电影名称、评分等信息，支持增量去重抓取

功能	测试结果	备注
王者荣耀英雄数据爬取	正常	成功获取英雄名称、类型信息，过滤重复数据
全国天气数据爬取	正常	可获取多城市天气、温度数据，支持增量更新
电影 & 英雄数据统计分析	正常	自动计算电影平均分、高分影片数量、英雄总数
天气数据统计分析	正常	统计天气类型分布、最高 / 最低 / 平均温度
JSON 数据导出持久化	正常	将爬取数据格式化导出为本地 JSON 文件
历史 JSON 数据导入	正常	读取本地历史数据，恢复会话，支持格式校验
网络异常捕获与重试	正常	断网时自动重试请求，最终抛出自定义网络异常
JSON 解析异常处理	正常	JSON 格式错误时拦截并抛出自定义解析异常
用户非法输入校验	正常	输入非数字指令时提示合法输入，拦截非法操作

五、总结

本项目以“多数据源爬虫与数据分析系统”为目标，基于 MVC 分层架构，结合命令模式、策略模式完成了整体设计与实现，最终达成了预期开发目标。系统支持豆瓣电影、王者荣耀英雄、全国天气三类数据的爬取，通过 FastJSON2 实现数据的序列化与持久化，并提供了数据分析、历史数据导入等配套功能，同时构建了完善的异常处理与日志记录体系，保障了程序的稳定性与可维护性。

在开发过程中，我不仅掌握了 Java 网络请求、HTML 解析、JSON 序列化与数据分析的核心技术，更深入理解了面向对象设计模式在实际项目中的应用价值。通过命令模式封装 CLI 交互指令，实现了用户请求与业务逻辑的解耦；通过策略模式统一多爬虫的实现规范，降低了新增数据源的开发成本；通过对网络异常、数据解析异常、非法输入等场景的捕获与处理，有效提升了系统的健壮性，避免了程序因边界情况而崩溃。同时，日志系统的搭建让程序运行过程可追溯，为调试与问题排查提供了重要支撑。

本次项目也暴露出一些待改进之处：目前爬虫未配置代理池与请求间隔策略，大规模请求时存在被反爬限制的风险；数据分析模块仅支持基础统计功能，缺乏可视化呈现能力；单元测试覆盖度不足，部分边界场景未经过充分验证。后续可通过引入代理池与请求随机化策略提升爬虫稳定性，集成图表库实现数据可视化，补充单元测试用例，进一步完善系统功能。

总体而言，本次项目开发让我从“功能实现”转向“工程化设计”，深刻体会到分层架构、设计模式、异常处理与日志系统对项目质量的影响，为后续的 Java 开发实践积累了宝贵经验。