

目 录

一、绪论	4
1.1 项目背景和意义	4
1.1.1 项目背景	4
1.1.2 项目意义	4
1.2 研究现状	5
1.3 研究内容及组织架构	5
二、系统开发相关技术	6
2.1 系统开发相关技术	6
2.1.1 Docker	6
2.1.2 Maven	7
2.1.3 ZooKeeper	7
2.1.4 HBase	8
2.1.5 Spring Boot	8
2.1.6 Thymeleaf	9
2.1.7 Java	10
2.1.8 Chart.js	10
2.1.9 三层架构开发技术	11
2.2 开发工具	11
2.2.1 VS Code	11
三、项目需求分析	12
3.1 可行性分析	12
3.2 系统的功能性需求分析	13
3.3 系统非功能性需求分析	14
四、系统设计	15
4.1 逻辑结构设计	15
4.2 网络与组件拓扑结构设计	16
4.3 表结构和列族设计	17
4.4 后端组件关系设计	18
4.5 系统功能模块设计	20
五、业务功能模块实现	20
5.1 注册和登录的功能模块	20
5.1.1 功能实现	20
5.1.2 关键代码	21
5.1.3 运行效果图	24
5.2 查看集群状态功能模块	26
5.2.1 功能实现	26
5.2.2 关键代码	26
5.2.3 运行效果图	28

5.3 查看数据的功能（包含原始数据与预处理）	29
5.3.1 功能实现.....	29
5.3.2 关键代码.....	29
5.3.3 运行效果图.....	33
5.4 查询天气数据功能	33
5.4.1 实现思路.....	33
5.4.2 关键代码.....	34
5.4.3 运行效果图.....	35
5.5 年度天气汇总	35
5.5.1 实现思路.....	35
5.5.2 关键代码.....	36
5.5.3 运行效果图.....	39
5.6 天气预测与可视化功能模块.....	39
5.6.1 功能实现.....	39
5.6.2 关键代码.....	40
5.6.3 运行效果图.....	43
5.7 自选：范围日期查询.....	46
5.7.1 实现思路.....	46
5.7.2 关键代码.....	46
5.7.3 运行效果图.....	49
5.8 自选：年均温变化	50
5.8.1 实现思路.....	50
5.8.2 关键代码.....	50
5.8.3 运行效果图.....	52
六、项目总结.....	53

一、 绪论

1.1 项目背景和意义

1.1.1 项目背景

随着观测站点数量不断增加、采样频率持续提升，气象与环境监测数据呈现爆发式增长。历史数据长期分散在 CSV/Excel、日志文件与关系型数据库中，不同团队各自维护口径与表结构，跨部门协作时经常出现口径不一致与数据滞后。一次“按站点+日期范围”的简单分析，往往需要先导出再手工汇总，数据工程师为一个报表来回沟通、反复验证，分析链路冗长且易出错。

现有系统大多基于单体架构与通用报表工具，难以支撑海量时序数据的高并发扫描与按时间聚合；传统按批处理导出的做法，导致“想看最新日聚合/年度趋势”也要等待任务完成。更麻烦的是，站点号前缀、时间范围等高频查询模式无法在存储层得到有效优化，导致范围扫描慢、历史回溯重、计算成本高。可视化与运维也割裂：业务侧缺少统一的趋势图与年度汇总，平台侧缺少对 HBase 集群健康度（Master/备份、Region 数、平均负载、存活/宕机节点）的直观洞察。

基于以上痛点，我们构建了一个轻量化、可定制的 HBase 驱动 Web 应用：以“站点号+日期”为核心访问模式，直接利用 HBase 的前缀/范围扫描能力，提供原始数据查看、日聚合与年度汇总、年均温趋势与范围查询，并用 Chart.js 做即看即得的可视化；同时内置集群状态看板，统一展示 HBase 关键运行指标，打通“数据—分析—可观测”全链路。项目采用 Spring Boot 2.7 与 Java 8，Maven 打包配合 Docker 一键部署，既满足教学演示与快速试错，也为后续扩展到协处理器或流/批处理框架、支撑更大规模与更高并发留下演进空间。

1.1.2 项目意义

这套应用把天气时序数据的“采集—存储—查询—聚合—可视化—运维可观测”全链路打通，形成统一的数据入口与口径。按“站点号 + 日期”的高频需求，用户无需导出与手工汇总，就能在页面上即刻获得原始数据、日聚合结果与年度统计/趋势图，把过去动辄需要人工拼接的分钟级等待，缩短为交互式的秒级响应（本地演示环境）。

在业务侧，它把常用分析能力产品化：范围查询、年度汇总与年均温趋势一站式呈现，显著降低沟通成本与误差传播。比如新增/迁移站点时，基于“站点前缀 + 时间范围”的键设计，查询与日聚合可直接生效，避免“数据不同步、口径不一致”。历史回溯只需选定站点与年份，即可复现当年均温变化曲线与表格，替代过去多文件夹、多版本对齐的繁琐流程。

在运维侧，内置“集群状态”看板将 Master/备份、Region 数量、存活/宕机节点与平均负载集中可视化，业务与平台信息同屏可见，缩短定位链路，降低不可预期中断带来的分析延迟与决策风险。

在工程层面，项目验证了 Spring Boot + HBase Client + Thymeleaf/Chart.js 的组合效率，Maven 打包与 Docker 一键运行降低环境成本；服务内实现的日/年聚合逻辑清晰可测，为后续演进到 HBase 协处理器、

Spark/Flink 批/流处理、结果缓存与权限控制（RBAC）等“生产化能力”提供了干净的扩展点与稳定的三层结构。

综上，它不仅解决了当前“海量时序数据的快速查询与可视化”痛点，也沉淀出一套可复用的时序数据应用框架：以 HBase 为底座、以聚合口径与可观测为抓手，助力团队从演示环境平滑迈向更大规模、更高并发与更强治理的数字化实践。

1.2 研究现状

目前，气象与环境类数据通常以“站点号 + 时间”的形式组织，属于典型的时序数据。这类数据量大、写入频繁，查询则多围绕站点和时间范围展开，用于日或年尺度的聚合、趋势分析、区间对比以及原始数据核查等场景。同时，业务系统也需要对底层存储集群的健康状态进行可视化监控。

在技术实现上，主流做法是采用 HBase 或专门的时序、列式存储引擎（如 TSDB、ClickHouse），通过以“站点+日期”为核心的 RowKey 设计，加速前缀与范围扫描查询。应用层一般使用 Spring Boot 构建三层架构，前端借助 ECharts 或 Chart.js 实现可视化，Maven 打包、Docker 部署，便于快速搭建与迭代。聚合计算初期多采用客户端实现，后续随着数据量增长，再逐步引入协处理器或 Spark/Flink 进行服务端与离线聚合。

然而，在实践中仍存在一些普遍痛点：数据源分散、口径不统一导致统计过程复杂；大范围扫描与热点访问造成响应延迟；缺乏缓存与分页机制，难以保证秒级交互体验；而数据可视化与系统运维信息往往相互割裂，难以在同一界面中综合分析。

行业趋势正在向“即查即看”的可视化体验和前置化的可观测性发展。越来越多的项目在前端提供统一口径的日、年聚合与趋势展示，减少沟通成本，同时在应用内集成集群状态监控，实现业务与平台信息的一体化呈现。部署层面也更强调“开箱即用”的一体化方案，通过 Maven + Docker 一键启动，确保系统快速上线与灵活扩展。

本项目的定位是一个轻量、可定制、可演示的端到端样板系统，集成原始数据、日聚合、年度分析与趋势可视化于一体，并内置 HBase 集群状态页，实现数据、分析与可观测性的贯通。当前架构采用客户端聚合，未来可平滑演进至协处理器或流批一体计算框架，并逐步完善查询缓存、分页、权限与监控体系。在后续优化中，可进一步通过预聚合与离线汇总表缓解热点问题，引入 Prometheus/Grafana 实现统一监控与报警，并强化指标治理与安全审计，推动系统向更高的稳定性与可运维性发展。

1.3 研究内容及组织架构

第一部分 绪论：阐述研究背景与意义（海量时序数据、跨团队口径不一致、可视化与可观测缺失等痛点），明确本项目以 HBase 为核心、面向“站点号 + 日期”的访问模式构建端到端方案。

第二部分 关键开发技术：介绍 HBase 数据模型与扫描机制、RowKey 设计要点；Spring Boot 分层架构与 Thymeleaf 视图渲染；Chart.js 可视化；Maven 打包与 Docker 一键部署；以及 HBase 客户端与集群指标获取的实现思路。

第三部分 需求分析：给出功能性需求（原始数据、日聚合、年度汇总、范围查询、年均温趋势、集群状态看板）与非功能性需求（可扩展性、可维护性、可观测性与容器化交付），说明输入口径（站点号、日期/范围）与异常处理（日期解析容错等）。

第四部分 系统设计：描述数据表与 RowKey 约定（围绕“stationId + 日期”的前缀/范围扫描）、聚合策略（客户端聚合为主、预留服务端聚合演进位）、模块划分（WeatherService、WeatherController、ClusterStatusInfo DTO 与模板 home.html/trend.html/cluster.html 等）。

第五部分 功能实现与运行效果：展示“范围查询（日聚合 + 站点号）”“年均温趋势（Chart.js 折线图）”“集群状态（Master/备份、存活/宕机、Region 数、平均负载）”等关键页面与交互，说明构建与容器化运行流程。

第六部分 总结与展望：总结本项目在“数据一分析一可视化一可观测”一体化上的教学与复用价值，展望协处理器/流批一体化计算、查询缓存与分页、权限与审计、指标接入（Prometheus/Grafana）等生产化演进方向。

二、 系统开发相关技术技术

2.1 系统开发相关技术

2.1.1 Docker

在本项目中，Docker 作为核心的运行与交付环境，为系统提供了高度一致、可移植的部署方式。通过将应用及其依赖、系统库一并封装成镜像，任何安装了 Docker 的机器都能以相同的方式运行，无需额外安装 JDK 或中间件，也避免了环境差异带来的兼容问题。借助 docker-compose，只需一个 YAML 文件即可描述服务、网络与数据卷的关系，实现“一条命令拉起一套环境”的便捷体验。

项目运行基于 eclipse-temurin:8-jre 镜像，应用目录为 /app，容器启动命令为 java -jar 运行 Spring Boot 可执行 JAR，端口 8080 对应映射到宿主机以供访问。构建流程上，开发者在本地使用 Maven 打包生成 target/hbase-java-demo-1.0-SNAPSHOT-boot.jar，通过挂载 target/ 目录到容器内的 /app，即可实现“构建→重启容器=上线”的轻量化更新方式，快速验证与迭代。

在编排层面，系统定义了自定义网络 hbase_network，确保 Web 应用与 HBase 集群容器间的互通。所有服务、网络及卷关系均通过 docker-compose.yml 文件声明，实现统一启动与关闭的自动化管理。HBase 客户端配置默认内置于应用资源中，若需外部化，可通过挂载 hbase-conf/ 目录进行覆盖，保证灵活性与可扩展性。

此外，应用内置了可视化的“集群状态”页面，可直接在容器环境中查看 HBase 的 Master 与备份节点状态、存活节点数量、Region 分布与平均负载等关键指标。这一设计让运维人员在容器化环境下即可进行健康检查与性能评估，提升了系统的可观测性与故障排查效率，为持续部署与可维护性提供了可靠支撑。

2.1.2 Maven

Maven 是 Java 生态中最常用的项目管理与构建工具，它通过一个名为 `pom.xml` 的配置文件，统一管理依赖、插件和构建流程。Maven 遵循“约定优于配置”的理念，为 Java 项目定义了标准的目录结构和生命周期，让开发者能够专注于业务逻辑而不是繁琐的构建细节。它最大的特点是依赖管理自动化——能自动下载并解析项目依赖、解决版本冲突，并确保构建环境的一致性。除此之外，Maven 还能与各种插件协同工作，从代码编译、测试、打包到部署，一条命令即可完成整个构建过程，与 CI/CD 流程、容器化部署天然契合。

在本项目中，Maven 作为核心的构建与依赖管理工具，承担了从源码到可执行产物的完整构建流程。它基于 `pom.xml` 文件集中管理项目依赖、插件与构建逻辑，遵循“约定优于配置”的原则，使项目结构与生命周期标准化。开发者只需通过简单命令即可完成清理、编译、打包等步骤，大幅减少手工配置与环境差异带来的问题。

借助 Maven 的依赖管理机制，项目能够自动解析并下载所需的第三方库，如 Spring Boot、HBase Client、Thymeleaf、日志组件等，确保依赖版本一致、兼容性可控。通过 `spring-boot-maven-plugin` 的 `repackage` 功能，Maven 会将应用及其依赖打包成可直接运行的可执行 JAR；同时结合 `maven-shade-plugin`，生成包含全部依赖的“胖包”，避免运行时缺失依赖的问题，使部署更加独立、可移植。

在部署流程中，构建产物 `hbase-java-demo-1.0-SNAPSHOT-boot.jar` 会直接挂载到 Docker 容器中运行，实现“打包 → 重启容器 = 上线”的轻量化更新模式。整个流程简洁高效，既适合本地开发测试，也方便后续集成到 CI/CD 或容器化的自动构建管线中。构建过程生成的关键文件包括主配置 `pom.xml`、最终产物 JAR，以及由 Shade 插件自动生成的 `dependency-reduced-pom.xml`，共同构成项目构建体系的核心。

2.1.3 ZooKeeper

ZooKeeper 是分布式系统中常用的协调与命名服务，承担着“集群大脑”的角色。它通过一套层级式的节点结构（ZNode）与事件监听机制（Watcher），为上层系统提供配置管理、主从选举、服务注册发现以及分布式锁等基础能力。在 HBase、Kafka 等分布式系统中，ZooKeeper 是不可或缺的组件，用于维护集群元数据与节点状态。其核心机制包括会话管理与临时节点：当服务实例下线时，临时节点会被自动清理，从而实现主从切换与健康状态感知。ZooKeeper 通常由奇数个节点组成法定人数（Quorum），即便少数节点故障也能保证集群可用，具有较高的容错性与一致性保障。

需要注意的是，ZooKeeper 并非通用数据库，它更适合存放“小体量、高频读、低频写”的元数据信息，而不适合存储大规模业务数据。由于对 GC 停顿与网络抖动敏感，实际部署中需合理配置会话超时与系统资源，以确保稳定性和响应速度。

在本项目中，ZooKeeper 主要作为 HBase 集群的协调组件使用。HBase 借助 ZooKeeper 完成 Master 的选举、RegionServer 的注册、Meta 表定位以及会话保活等关键流程。项目中的 HBase 客户端通过 `hbase-site.xml` 文件中的 `hbase.zookeeper.quorum` 等配置项与 ZooKeeper 建立连接，实现对集群的自

动感知与路由选择。应用代码层面并不直接与 ZooKeeper 交互，所有协调逻辑均由 HBase 客户端内部的 RecoverableZooKeeper 机制透明处理，从而降低了开发复杂度。

在运行环境上，容器编排通过 hbase_network 网络确保 Web 应用、HBase 与 ZooKeeper 之间的互通，ZooKeeper 默认监听 2181 端口。在项目部署脚本与配置中，docker-compose.yml、hadoop.env、start-hbase.sh 等文件共同定义了 ZooKeeper 的启动与环境变量。系统的“集群状态”页面 (/cluster) 则间接反映了 ZooKeeper 的协调结果，如当前 Master 与备份节点、RegionServer 存活情况等，从而让用户能够在应用层清晰了解底层集群的运行状态。

2.1.4 HBase

HBase 是构建在 HDFS 之上的分布式列式数据库，专为存储与访问海量稀疏数据而设计。它采用宽表 (Wide-Table) 模型，数据按 RowKey 排序存储，支持随机读写与范围扫描，同时在单行级别上保证强一致性。HBase 的架构由 Master、RegionServer 与 ZooKeeper 协调组成，数据通过 Region 自动分片并分布在多个节点上，实现线性水平扩展。当 Region 过大或节点负载不均时，系统会自动拆分与迁移，从而维持集群的高可用与负载平衡。

其性能优势主要体现在高并发写入与低延迟访问方面。数据写入流程先进入内存中的 MemStore，再顺序刷入磁盘生成 HFile，因此具备极高的写入吞吐能力，非常适合气象、环境、物联网等高频写入的时序场景。通过合理的 RowKey 设计（如“站点号 + 日期”组合），HBase 能高效支持前缀与范围扫描，实现时间序列数据的快速定位与聚合查询。同时，HBase 还具备 TTL（数据自动过期）、BloomFilter（快速存在性判断）、协处理器（服务端计算）以及与 Spark/Flink 等大数据计算引擎的集成能力，为后续的扩展分析奠定基础。

在本项目中，HBase 承担了气象与环境时序数据的底层存储任务。系统以“站点号 + 日期前缀或范围”为核心访问模式，通过 Scan 操作实现原始数据、日聚合、年汇总与年均温趋势等查询。当前聚合逻辑在 WeatherService.java 中完成，采用客户端聚合以简化实现与提升演示灵活性；后续可无缝演进至 HBase 协处理器或 Spark/Flink 流批处理，实现更高效的服务端或离线聚合。

此外，项目还利用 HBase 提供的管理接口 (Admin/ClusterMetrics) 获取集群运行状态，包括 Master 与备份节点、存活与宕机的 RegionServer 数量、Region 分布及平均负载等信息，并在前端页面 templates/cluster.html 中可视化呈现（访问路径 /cluster）。HBase 客户端配置由 hbase-site.xml 定义，既可内嵌在资源文件中，也可通过挂载 hbase-conf 目录实现外部化配置。结合 ClusterStatusInfo.java 数据结构，系统实现了从数据存储、聚合分析到集群观测的完整闭环，为气象数据的可视化与可运维提供了可靠基础。

2.1.5 Spring Boot

Spring Boot 是基于 Spring 框架的一站式开发平台，它以“约定优于配置”的理念极大地简化了项目启动与配置过程。通过自动装配机制、内置的 Web 容器（如 Tomcat）以及丰富的 Starter 依赖体系，Spring Boot 让开发者无需繁琐的 XML 或 JavaConfig 即可快速构建可运行的应用。它与 Spring MVC、Thymeleaf、数据访问、校验、安全框架等生态无缝集成，同时具备生产级特性，

包括统一日志体系、健康检查、外部化配置与多环境管理等，使得开发、部署与运维更加高效和标准化。

在本项目中，Spring Boot 扮演了核心应用框架的角色，支撑起整个 Web 层、业务层与配置体系。项目基于 Spring MVC 构建 Web 路由与控制逻辑，WeatherController.java 负责接收用户请求、处理参数并将结果通过 Model 传递至 Thymeleaf 模板，实现数据到页面的渲染展示。业务逻辑集中在 WeatherService.java 中，处理范围扫描、聚合分析与集群状态查询等操作，形成清晰的层次结构

配置层面，项目采用 application.properties 统一管理应用端口、模板缓存与日志输出等通用参数，同时通过外部的 hbase-site.xml 文件定义 HBase 客户端连接配置，实现业务配置与平台配置的分离。借助 Spring Boot 的自动装配与分环境配置机制，系统能在开发、测试与演示环境间平滑切换，减少人为干预与配置错误。

在打包与部署上，项目使用 Maven 的 spring-boot-maven-plugin 插件执行 repackaging 操作，将应用及其依赖打包为可直接运行的 *-boot.jar，该产物可直接用于 Docker 容器运行，实现从构建到上线的无缝衔接。核心文件包括 pom.xml（依赖与构建配置）、application.properties（应用参数）、WeatherController.java（控制器）与 WeatherService.java（业务逻辑），共同构成项目的运行骨架，使系统具备启动快、结构清晰、易部署、易维护的特性。

2.1.6 Thymeleaf

Thymeleaf 是一种现代化的服务端模板引擎，常用于 Java Web 应用的页面渲染。它允许在标准 HTML 中直接使用以 th: 为前缀的属性（如 th:text、th:each、th:if 等）进行变量输出、循环与条件控制，从而实现“所见即所得”的开发体验。由于模板文件本身就是合法的 HTML，前端设计人员可以直接打开静态页面进行预览，无需依赖后端服务。Thymeleaf 与 Spring MVC 深度集成，支持从 Controller 传递的 Model 数据自动绑定到模板中，同时提供国际化、布局与片段复用（fragments）等功能，非常适合构建数据驱动的动态页面。

在本项目中，Thymeleaf 是前端展示层的核心技术，用于将后端生成的气象与环境数据可视化地呈现给用户。各类模板文件位于 templates/ 目录下，包括 home.html（功能入口）、trend.html（年均温趋势与表格）、cluster.html（集群状态看板）、annual.html、range.html 与 forecast.html 等页面。系统通过 WeatherController.java 将业务层计算结果以 Model 的形式传递到模板中，再由 Thymeleaf 标签渲染到 HTML 页面。常用指令如 th:each 用于循环展示数据列表，th:text 或 th:utext 输出文本内容，th:if 控制元素显示逻辑，th:href 生成动态链接，使页面逻辑清晰、可维护。

项目的界面设计采用统一的深色主题与固定布局风格，确保在演示和截图场景下的稳定展示效果。尤其在 trend.html 页面中，Thymeleaf 模板与 Chart.js 前端图表库紧密配合：后端将年份与均温数据注入模板，页面脚本读取这些变量后自动生成折线图，实现“数据一经查询，即刻可视化”的交互体验。这种结合充分体现了 Thymeleaf 在服务器端渲染场景中的高效与灵活，也为整个系统的数据链路（Controller → Service → View → Chart.js）提供了清晰、连贯的展示通道。

2.1.7 Java

Java 8 在语言层面引入了函数式编程思想和更现代化的 API。Lambda 表达式与 Stream 流式处理让集合运算（过滤、映射、分组、聚合等）更加简洁优雅，大幅减少样板代码；Optional 类型有效降低了空指针异常的风险；新的 java.time 时间日期 API（如 LocalDate、LocalDateTime、DateTimeFormatter）取代了旧的 Date 与 Calendar，在时区处理和格式化上更安全一致。此外，接口默认方法和改进的并发工具（如 CompletableFuture）也增强了可扩展性与异步能力。这些特性共同提升了 Java 工程在可读性、稳定性与执行性能上的表现。

在本项目中，Java 8 的特性被广泛应用于气象数据的计算与展示逻辑中。业务层的核心实现位于 WeatherService.java，通过 Stream 流对 HBase 查询结果进行聚合与转换：以 LocalDate 为粒度完成“日聚合”，再以年份分组计算“年度均温”；使用 map、filter、collect 等操作链实现从原始数据到统计结果的高效转换。同时，DateTimeFormatter 被用于解析多种日期格式，确保系统在读取不同数据源时具备更高的容错性。

控制层 WeatherController.java 调用这些流式处理方法，将结果以结构化的形式传递给前端模板进行渲染（如年度趋势图表与数据表格）。这种基于 Java 8 的编程风格让代码逻辑清晰、维护成本低，也与 Spring Boot 框架和 HBase 客户端版本保持良好兼容，为整个系统的数据计算和可视化提供了稳定的语言基础。

2.1.8 Chart.js

Chart.js 是一款轻量级的前端图表库，基于 HTML5 Canvas 渲染，支持折线图、柱状图、饼图等常用统计图类型。它 API 简洁易用，能够快速生成交互式图表，内置 Tooltip、Legend 等功能，使图表不仅美观，而且便于用户在演示或分析中理解数据。Chart.js 与服务端渲染（SSR）模板结合时，只需将后端生成的标签数组（labels）和数据集（datasets）注入到页面脚本中，即可完成图表渲染，无需额外复杂配置。此外，Chart.js 支持响应式布局，也可通过固定宽高保证报告截图或演示时的一致性。

在本项目中，Chart.js 被用于气象数据的可视化展示，尤其是年度均温趋势分析。模板文件 templates/trend.html 使用折线图展示每年的平均温度，后端通过 WeatherService.aggregateAnnualByStation(...) 生成年份数组和均温数组，再由 WeatherController 注入到 Thymeleaf 模板中。前端脚本读取这些数据并初始化 Chart.js 配置，实现折线图与对应数据表格的同步呈现。为了保证演示与截图稳定性，图表尺寸固定（如 800×300），关闭响应式模式，同时配套表格用于数据核对与导出，确保图表既直观又可验证。

整个数据流呈现出清晰的端到端链路：从 HBase 查询原始数据 → 业务层按日或年度聚合 → Controller 将聚合结果注入 Model → Thymeleaf 模板渲染 → Chart.js 绘制图表。通过这种方式，Chart.js 在本项目中实现了“后端计算 + 前端直观展示”的高效结合，使年度趋势数据既可视化又可操作，提升了系统的演示性和用户体验。

2.1.9 三层架构开发技术

三层架构（Controller - Service - View）是 Java Web 开发中最经典、最稳定的设计模式之一，它通过分层来实现业务逻辑、请求控制与界面展示的职责分离。

在这一架构中，Controller 层负责接收前端请求、进行参数校验，并调用业务逻辑后选择合适的视图进行渲染；Service 层集中处理核心业务逻辑和数据聚合，保证代码复用性与可扩展性；View 层则负责数据展示与交互体验，通常以模板引擎或前端可视化库的形式呈现。这样的设计能有效提升系统的可维护性和灵活性，使得不同功能模块之间依赖清晰、修改独立。

在本项目中，三层架构得到了完整的落地应用。WeatherController.java 作为表现层，定义了 /data、/daily、/range、/annual、/trend、/cluster 等路由接口，负责接收站点号与日期参数，并将查询结果交由前端模板渲染。WeatherService.java 作为业务层，封装了范围扫描、日/年聚合、趋势计算以及 HBase 集群状态获取等核心逻辑，确保各类查询与分析的逻辑集中、清晰、可复用。为了实现集群健康状态的展示，系统还定义了 ClusterStatusInfo.java 作为 DTO（数据传输对象），用于规范后端与前端之间的结构化数据交互。

视图层采用 Thymeleaf 进行服务器端渲染，通过模板文件（如 home.html、trend.html、cluster.html 等）将查询结果与可视化组件结合，利用 Chart.js 绘制趋势图与聚合对比图，实现“查询即展示”的直观体验。系统的数据交互契约也十分清晰：输入通常为站点号与日期（或日期范围），输出为表格数据及可视化所需的时间序列（如年份数组、均温数组）。异常情况下会返回日期格式错误、空结果或 HBase 连接失败等提示；而在正常情况下，常用查询均能在演示环境下实现秒级响应与稳定渲染，保证了系统的可演示性与可用性。

2.2 开发工具

2.2.1 VS Code

VS Code 是一款轻量级、跨平台的源代码编辑器，内置终端与调试器，并拥有丰富的扩展生态，非常适合 Java/Spring Boot 开发以及前后端协作。它将代码编辑、调试、版本控制与终端操作集成在同一窗口，为开发者提供一体化的开发体验。同时，VS Code 支持大量扩展，如 Java 开发工具、Spring Boot 插件、Maven 管理、Docker 可视化、YAML/Properties 文件高亮等，使项目开发更加高效和规范化。团队成员可以共享推荐的扩展与配置，减少环境差异带来的问题。

在本项目中，VS Code 是主要的开发环境。整个代码仓库均可直接在 VS Code 中进行编辑与调试，内置终端（在 macOS 上使用 zsh）方便执行 Maven 构建、运行 Spring Boot 应用以及管理 Docker 容器。为了提升开发效率，推荐安装的扩展包括 Java Extension Pack（提供项目视图与调试支持）、Spring Boot Extension Pack（端点管理与配置提示）、Maven for Java（依赖树与生命周期操作）、Docker（镜像与容器管理）以及 YAML、Properties 和 Thymeleaf 的语法高亮。

项目中的关键文件与资源均通过 VS Code 进行管理和操作：pom.xml 用于构建与依赖管理，src/main/** 和 src/main/resources/** 包含应用源码与资

源文件，Docker 相关的 Dockerfile 与 docker-compose.yml 则用于容器化部署。通过 VS Code 的集成开发与调试功能，开发者可以在同一环境中完成代码编写、测试与部署，确保开发流程高效、连贯且易于协作。

三、 项目需求分析

3.1 可行性分析

1. 项目目标与范围

本项目旨在围绕“站点号 + 日期”的时序数据，构建一个集数据查询、聚合分析、趋势可视化与集群可观测于一体的综合应用平台。系统可实现原始数据查询、日度聚合、年度统计及趋势展示，并内置 HBase 集群状态看板，实现“数据—分析—可观测”闭环。项目范围涵盖 Web 前端（Thymeleaf + Chart.js）、后端服务（Spring Boot）、HBase 客户端开发及集群指标采集，并通过 Docker/docker-compose 实现容器化部署与交付。

2. 约束与前置假设

系统数据模型基于 RowKey 设计，采用“stationId + 日期前缀/范围”的组合以优化扫描性能。演示及开发环境在 Docker 网络中互通，数据目录使用持久化卷确保可靠性。假设未来数据规模及访问频率将逐步增长，系统可通过分阶段优化（缓存机制、预聚合、协处理器、流/批处理）来提升性能和扩展性，从而平滑过渡至更大规模的生产环境

3. 技术可行性

技术架构选择成熟且稳定：Spring Boot 2.7 与 Thymeleaf 结合实现高效的服务器端渲染；HBase 2.1.2 客户端支撑高频写入与按时间范围查询；Chart.js 满足轻量可视化需求；Docker 提供可复现的部署环境。当前版本经验证可编译打包成功，容器启动正常，服务运行稳定。未来可演进为更高性能架构（如协处理器聚合、缓存与预分区策略）。整体技术路线成熟、路径完整，能支撑演示及中等规模部署。

4. 业务可行性与价值

系统以“一站式数据分析与监控”为核心亮点，将原始数据、聚合结果、年度趋势与集群健康监控统一展示，显著降低用户操作与沟通成本。预设查询口径和秒级响应提升数据可视化效率，使报表生成更及时。运维人员可通过集群健康页快速定位性能瓶颈与节点状态，提高决策准确度。整体业务目标明确、价值可衡量，具备良好的落地与推广潜力

5. 经济可行性

项目主要采用开源技术栈，开发成本低、学习曲线平缓，且现有代码与模板可复用，缩短建设周期。容器化部署降低环境依赖与服务器成本，系统可随数据量弹性扩展。模块化架构减少后期维护与功能扩展的边界冲突，利于持续集成与交付。总体而言，当前阶段投入产出比高，后续扩展可按需进行，经济上具备良好的可持续性。

6. 运维与持续运行可行性

通过 Docker Compose 实现容器编排与卷持久化，系统可安全关停并保持数据完整。HBase/HDFS/ZooKeeper 设计为 7×24 小时常驻服务，支持长期运行。常规维护包括磁盘监控、Region 分布均衡、GC 调优与日志轮转等。系统现

有 /cluster 模块已具备基础健康监控能力,并可扩展 Prometheus/Grafana 实现更全面的观测与告警。整体具备稳定运行与可扩展的运维基础。

7. 安全与合规可行性

当前部署主要用于内网或演示场景,未涉及敏感权限与外部暴露风险,日志与配置安全可控。系统架构预留安全扩展空间,可逐步接入 RBAC 权限控制、审计日志、只读访问、接口鉴权及限流策略,配置与密钥可外部化管理。总体安全风险可控,具备向生产级合规架构演进的能力。

3.2 系统的功能性需求分析

系统的功能性需求以完整可验证为核心,首先在导航与页面层面,主页必须成为统一的交互入口与视觉基调。所有功能模块依次排列为“集群状态→查看原始数据→查看处理后数据→查询数据→范围查询数据→年度天气汇总→年均温变化→预测与可视化”,顺序固定、不可调整。主题统一为深色风格,要求在不同设备与分辨率下保持布局稳定、字体清晰、跳转无闪烁。页面加载后,任意功能按钮点击均应无错误路由与空白渲染,确保用户从主页即可直达各核心模块,实现零迷失导航体验。

在数据查询与聚合展示部分,系统须完整覆盖“原始数据一日聚合一条件查询一范围查询”四类核心路径。原始数据页 (/data) 应支持按站点号与日期条件检索,并以分页或固定数量显示结果;输入合法时返回非空列表,数据为空时给出明确提示语而非空白屏幕。日聚合页 (/daily) 展示按站点与日期粒度聚合后的核心指标,结果须与手工聚合数据一致,误差仅限浮点舍入范围内。条件查询页 (/query) 用于单点快速核验,需对非法参数进行即时提示;范围查询页 (/range) 要求输入站点号与起止日期,返回按日期升序排列的聚合结果,支持常见日期格式解析(yyyy-MM-dd 等),在缺失或错误输入时不得抛出系统错误,而是以可读信息引导用户修正。

统计与可视化部分构成应用的分析核心。年度汇总页 (/annual) 需按年计算并展示关键气象指标(如年均温、极值等),与样本数据人工汇总结果保持一致,空年份必须返回提示信息而非空表。年均温变化趋势页 (/trend) 通过 Chart.js 绘制折线图并同步输出表格对照,要求年份与数据点一一对应,图表与表格数据保持完全一致,空数据状态下仍能正常渲染提示层。预测与可视化页 (/forecast) 在当前阶段可展示预测占位图或示例文案,页面需可访问、渲染完整,为后续接入真实模型输出留出接口空间。

在集群可观测方面,/cluster 页面必须实时展示 HBase 集群核心健康指标,包括版本信息、Master 与备份节点、RegionServer 存活与宕机数、Region 总数及平均负载。页面需支持刷新,确保连接异常时以降级方式显示兜底文案或错误提示,而非直接崩溃。该模块被视为系统运行可信度的核心体现,要求无中断、可自愈。

基础共性功能同样被严格约束。系统所有输入(站点号、日期、范围等)均需通过前后端双重校验,对非法、越界或空值输入返回明确提示信息,不得出现 HTTP 500 错误。主页右上角预留登录/注册入口,若当前阶段未启用鉴权,则保持为占位或跳转页,且不影响其他功能模块。配置层面上,应用须正确加载 hbase-site.xml,并能与 HBase/ZooKeeper 集群建立连通;容器网络互通性是

验收前提，只有当 /cluster 页面可正常展示指标且数据查询接口返回结果时，系统方视为功能完整。

3.3 系统非功能性需求分析

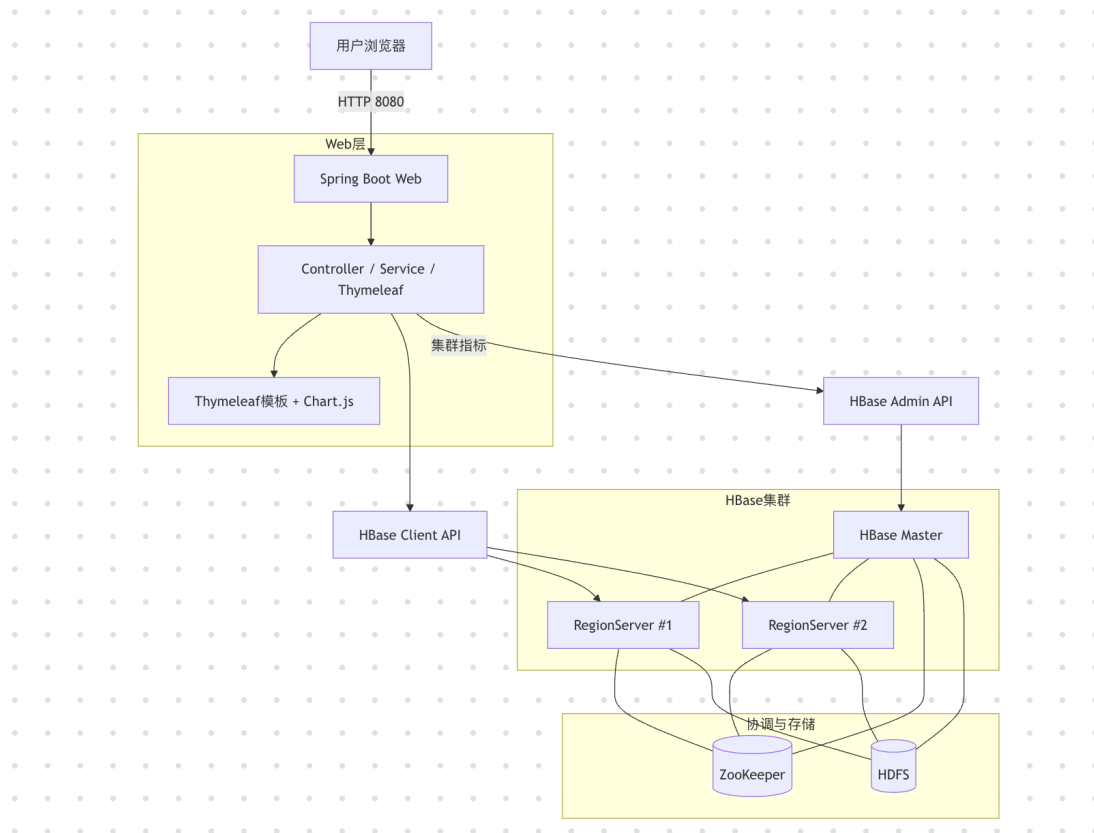
在非功能性需求方面，系统的可用性与可靠性被定义为稳定运行的基础约束。演示环境下，服务可用时间应保持在 99% 以上，确保在持续访问与多次重启下无功能性异常。系统虽不强制自动恢复机制，但必须支持通过容器或脚本手动重启并快速恢复服务状态。数据层面要求实现高可靠性，HBase、HDFS 与 ZooKeeper 的数据目录均采用持久化卷，确保在优雅停机、容器重建等场景下不丢失数据。面对异常写入或重复提交的风险，系统需通过幂等逻辑或上游导入约束，保证“只写一次、不写错”，避免出现数据污染与重复入库的情况。

在可扩展性与容量规划方面，系统架构需具备天然的水平扩展能力。HBase 集群应能随着 Region 的增长自动实现横向扩容，而应用服务层则以无状态化为前提，允许部署多个实例以应对并发负载。未来可在需要时增加缓存层或启用会话粘性机制，以优化性能与一致性。数据增长策略方面，系统应具备可配置的生命周期管理能力，包括 TTL（时效清理）、冷热分层存储与数据预聚合，以便在数据量扩大后仍能维持查询效率和存储成本的平衡。

在可维护性与可观测性方面，系统设计强调对运行状态与问题诊断的可追溯性。日志体系应采用统一的日志门面（如 SLF4J），关键请求链路需记录执行耗时与异常堆栈，并在容器内与宿主机层面均配置日志轮转策略，防止日志占满存储空间。监控模块以 /cluster 页面为核心，已能展示 HBase 的版本、节点状态与负载信息，后续可无缝接入 Prometheus/Grafana，实现统一观测与自动告警。对于常见故障，如 HBase 无法连接、ZooKeeper 异常或日期格式解析失败，系统应能在界面或日志中提供明确的定位线索与操作建议，使开发与运维人员可在分钟级内完成问题确认与恢复操作。整体而言，该系统在非功能性层面追求“可持续运行、可扩展增长、可自诊断恢复”的工程化特征，为后续生产级部署奠定了稳定基础。

四、系统设计

4.1 逻辑结构设计



最上层是用户访问入口。用户通过浏览器（U）以 HTTP 8080 端口访问 Spring Boot Web 服务（SB）。Spring Boot 作为后端核心框架，负责接收请求、分发路由并协调业务逻辑。在 Web 层内部，Spring Boot 的控制器（Controller）与服务（Service）模块通过 MVC 架构组织，利用 Thymeleaf 模板引擎和 Chart.js 前端库生成最终的页面与可视化结果，返回给用户浏览器渲染。

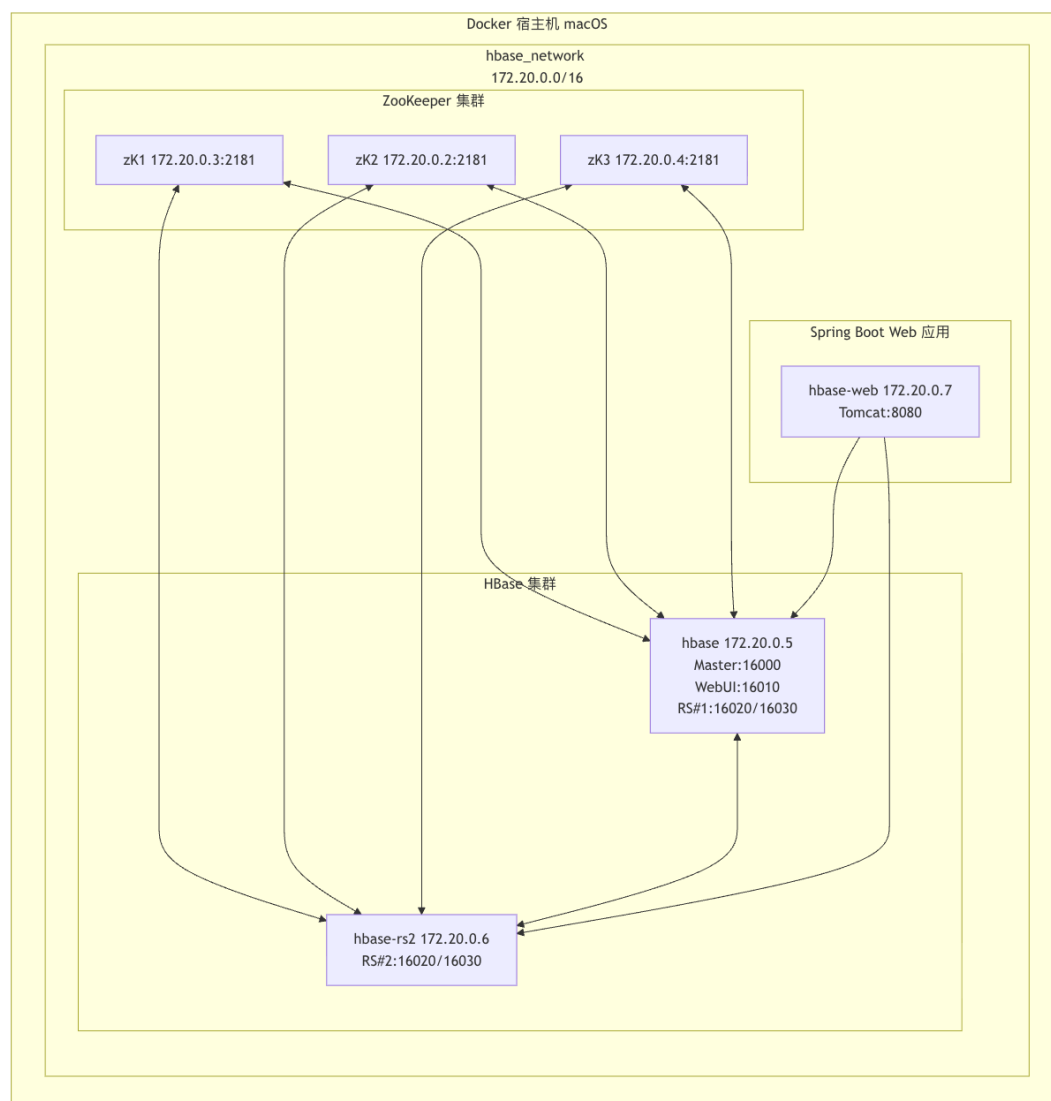
中间层是系统的数据访问核心。MVC 层中的业务逻辑组件通过 HBase Client API 与 HBase 集群通信，以执行数据查询、聚合等操作；同时通过 HBase Admin API 读取集群运行指标（如 RegionServer 状态、Region 数量、平均负载等），用于监控与展示。

HBase 集群内部包含一个 Master 节点与多个 RegionServer(RS1、RS2)。Master 负责元数据管理与 Region 分配，RegionServer 负责实际数据的读写存储。Master 与 RegionServer 之间保持常态连接，实现分布式协调与负载均衡。

最底层是系统的协调与存储基础设施。HBase 的运行依赖 ZooKeeper（ZK）进行集群协调与节点选举，确保 Master 高可用与 RegionServer 状态同步。同时，所有数据与元数据最终落盘至 HDFS（分布式文件系统），由 Master 与各 RegionServer 共同读写。

在运行时，HBase Client 与各 RegionServer 直接交互以完成数据读写，HBase Admin 与 Master 通信以获取集群状态；Master、RegionServer 均依赖 ZooKeeper 进行心跳与注册管理，并通过 HDFS 统一存储数据文件与 WAL 日志。

4.2 网络与组件拓扑结构设计



这张流程图展示了系统在 Docker 容器化环境下的网络与组件拓扑结构，说明了各服务在容器中的分布方式、通信关系以及运行逻辑。整体可划分为三层：宿主机环境、虚拟网络层（hbase_network）和容器级服务层，其中每个组件都以独立容器运行并在同一虚拟网段中互通。

最外层是 Docker 宿主机（macOS），其内部创建了一个名为 hbase_network 的自定义网桥网络，网段为 172.20.0.0/16。所有容器（ZooKeeper、HBase、Web 应用）均接入该网络，从而在逻辑上处于同一局域网内，实现跨容器直接通信而无需暴露额外端口。

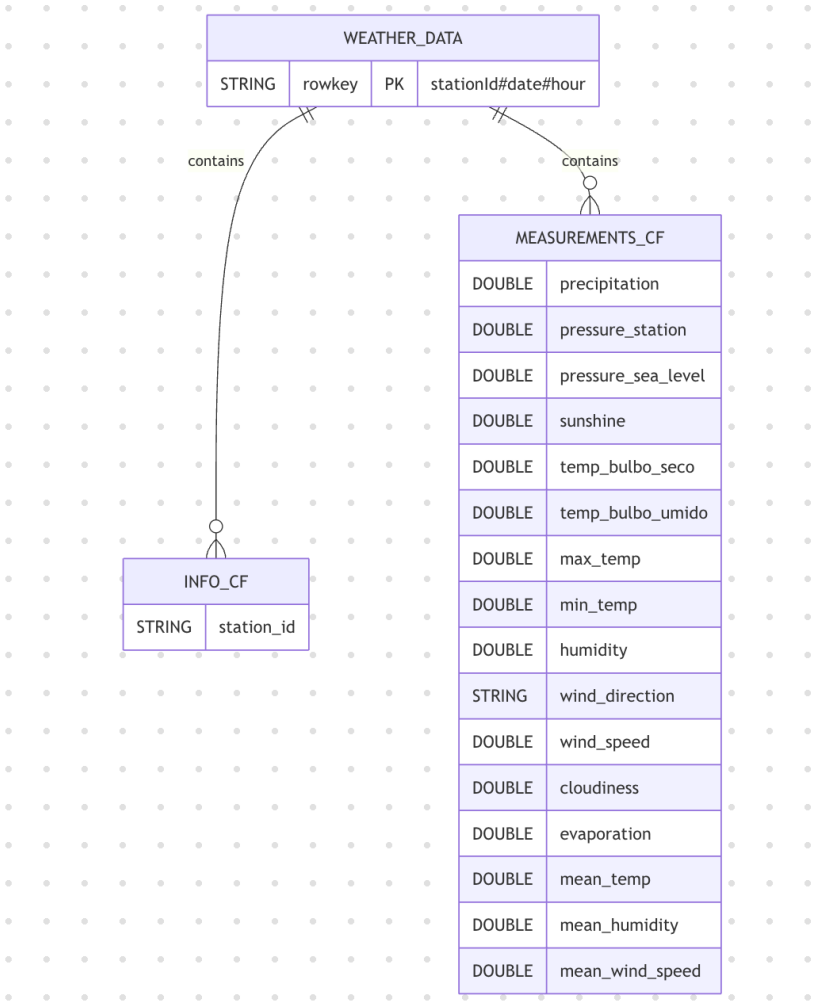
在网络内部，首先是 ZooKeeper 集群，由三个节点组成：zk1（172.20.0.3）、zk2（172.20.0.2）、zk3（172.20.0.4），均监听 2181 端口。三个节点构成分布

式协调系统，负责管理 HBase 集群的元数据注册、Master 选举与 RegionServer 状态同步，是整个集群的“调度与心跳中枢”。

紧接着是 HBase 集群层。其中，master 容器(172.20.0.5)同时运行 HBase Master 服务 与 RegionServer #1，Master 的管理端口为 16000，Web 管理界面运行在 16010，RegionServer 通信端口为 16020/16030。第二个 RegionServer rs2 独立运行在 172.20.0.6，端口配置一致，用于数据分片与并行存储。Master 与两个 RegionServer 通过 ZooKeeper 集群保持心跳与状态同步，同时共享底层的 HDFS 数据存储（通常以卷挂载方式提供）。Master 负责集群元数据与 Region 分配，RegionServer 则处理实际的数据读写请求。

在应用层，Spring Boot Web 应用(hbase-web) 运行于容器 172.20.0.7，使用 Tomcat 监听 8080 端口。该应用是系统的用户入口，通过 HBase Client API 与集群交互：当用户在 Web 页面发起数据查询时，应用客户端直接向各 RegionServer（如 master 容器中的 RS#1 或 rs2 容器）发送数据请求；若请求涉及集群状态或管理操作，则经由 Admin API 与 Master 节点通信。应用容器与 HBase 集群、ZooKeeper 集群均在同一网络中，无需额外端口映射即可完成内部访问。

4.3 表结构和列族设计

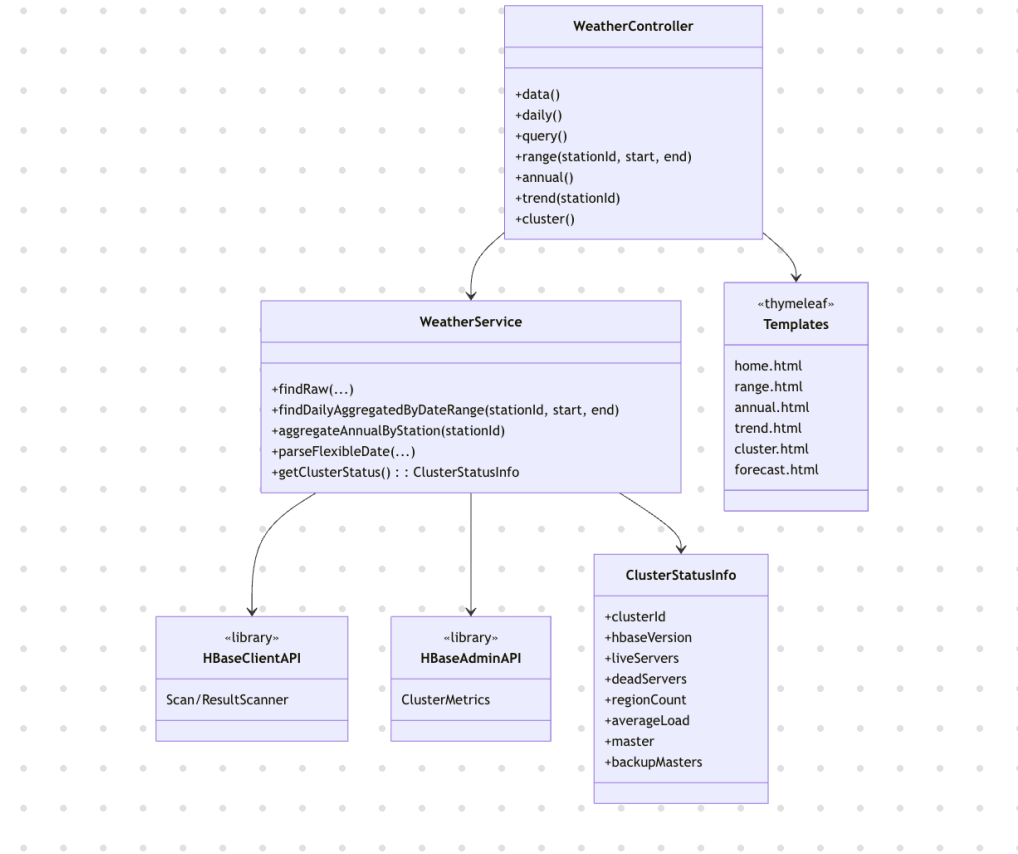


这张 ER 图描述了系统在 HBase 中存储天气数据的表结构和列族设计，体现了典型的分布式列式存储模式。核心表为 WEATHER_DATA，它以 rowkey 作为主键，组合了 stationId#date#hour，保证了数据的唯一性和按站点及时间范围高效扫描能力。这种设计便于按站点和时间进行范围查询或聚合计算，符合时序数据存储的最佳实践。

表中定义了两个列族：INFO_CF 和 MEASUREMENTS_CF。INFO_CF 主要存储天气站基础信息，如 station_id，通常变化少、占用空间小；MEASUREMENTS_CF 列族则承载具体的气象观测值，包括降水量、气压（地面及海平面）、日照时数、干湿球温度、最高/最低温度、湿度、风向风速、云量、蒸发量及各类平均值（平均温度、平均湿度、平均风速）等。将观测值放在独立列族不仅逻辑清晰，也有利于 HBase 的列式存储特性，提高读取聚合列时的性能。

这种表/列族结构还具备较强的可扩展性和灵活性：随着新气象指标的加入，只需在 MEASUREMENTS_CF 中增加列即可，无需改表结构；同时，按 RowKey 的前缀或范围扫描可以快速获取单站点单天或小时级别的历史记录，支持原始数据查询、日聚合、年度统计和趋势分析等多种访问模式。整体设计清晰地反映了 HBase 的列式存储逻辑，兼顾查询效率和数据可扩展性。

4.4 后端组件关系设计



最上层的 WeatherController 是系统的入口控制器，负责接收来自前端（浏览器或模板页面）的 HTTP 请求，并将其分发至业务逻辑层。它提供了一系列方法：

- `data()`、`daily()`、`query()`、`range()` 用于原始数据与聚合查询；
- `annual()`、`trend()` 对应年度统计与趋势可视化；
- `cluster()` 用于集群状态监控。

这些方法对应具体的 Web 路由（如 `/data`、`/range`、`/trend` 等），最终渲染不同的 Thymeleaf 页面或返回数据模型。

控制层的核心依赖是 `WeatherService`，该类承担主要的业务逻辑与数据访问职责。它定义了若干关键方法：

- `findRaw()` 负责从 HBase 获取原始记录；
- `findDailyAggregatedByDateRange()` 在指定站点与日期范围内执行聚合查询；
- `aggregateAnnualByStation()` 计算年度指标；
- `parseFlexibleDate()` 处理多种日期格式输入；
- `getClusterStatus()` 则封装了集群状态的读取逻辑，返回 `ClusterStatusInfo` 对象。

通过这种分层设计，`Controller` 不直接访问数据库或外部系统，而是通过 `Service` 统一调用与封装底层资源，保证了代码的解耦性与可维护性。

在数据访问层，`WeatherService` 与 HBase 系统交互，主要通过两套官方接口：

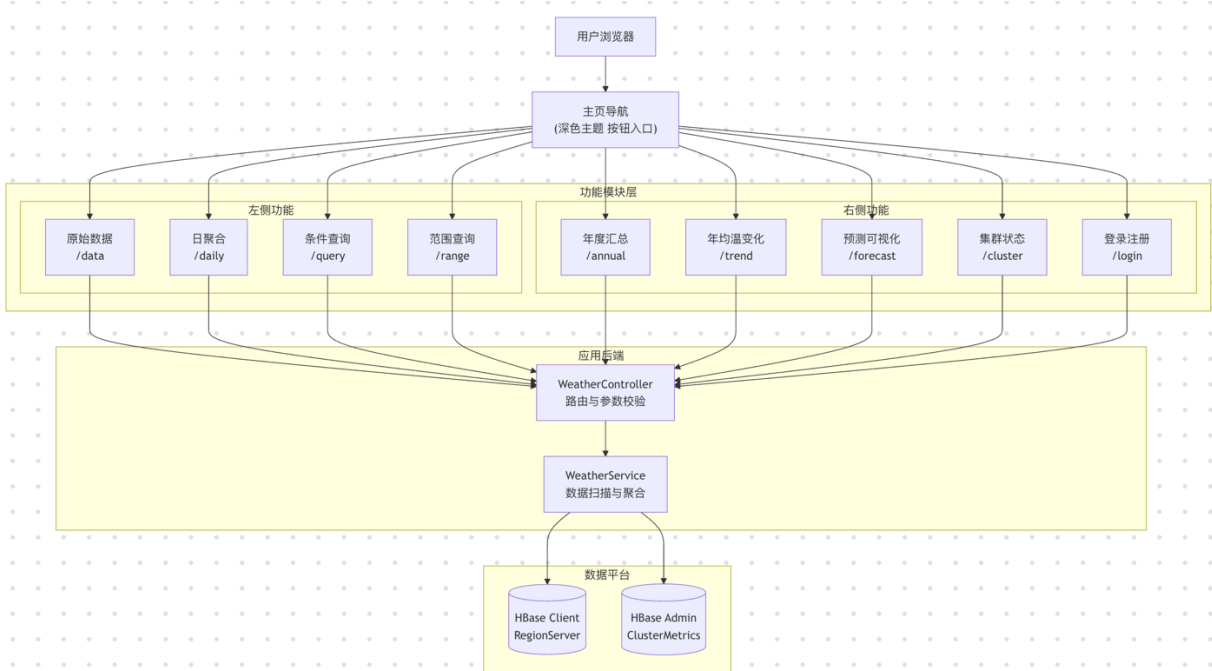
- `HBaseClientAPI`（使用 `Scan` 与 `ResultScanner`）处理数据的扫描与结果遍历，适用于查询与聚合逻辑；
- `HBaseAdminAPI`（基于 `ClusterMetrics`）则用于读取集群管理指标，如 `Region` 数量、存活节点、负载均衡状态等。

这种区分让系统同时具备“数据访问”和“集群观测”两条独立能力链，既能服务于业务查询，也能反映底层运行状况。

集群状态信息由 `ClusterStatusInfo` 类承载，作为数据结构体封装从 HBase Admin 获取的指标信息。其字段包括 `clusterId`、`hbaseVersion`、`liveServers`、`deadServers`、`regionCount`、`averageLoad`、`master` 与 `backupMasters` 等，代表一个时间点上 HBase 集群的整体健康状态。该对象会被返回给控制层，并最终在前端 `/cluster` 页面可视化呈现。

前端模板部分由 `Templates` 组件表示，它是 Thymeleaf 的模板集合，包含 `home.html`、`range.html`、`annual.html`、`trend.html`、`cluster.html`、`forecast.html` 等文件。控制器根据不同的请求路径选择相应模板进行渲染，其中的 `Chart.js` 图表与数据表格直接绑定控制器返回的模型数据，实现前后端协同。

4.5 系统功能模块设计



最顶层是用户浏览器，通过主页导航访问系统各个功能模块。导航界面采用深色主题，并提供统一按钮入口，用户可以选择查看原始数据、日聚合数据、条件查询、范围查询、年度汇总、年均温变化趋势、预测与可视化，以及登录/注册入口。

在功能模块层，系统将各类操作分为左右两组：左侧主要是数据查询类功能，包括原始数据、日聚合、条件查询和范围查询；右侧主要是统计、可视化和集群监控类功能，包括年度汇总、趋势分析、预测可视化、集群状态，以及登录注册。所有功能模块的请求统一汇入后端控制器 WeatherController，负责路由分发和参数校验，确保请求合法、路径正确，并将业务调用传递给服务层 WeatherService。

服务层 WeatherService 负责具体的数据扫描、聚合计算以及集群指标读取，封装了对 HBase Client 和 Admin API 的调用。通过 Client API 与 RegionServer 进行数据查询和聚合，通过 Admin API 获取集群状态信息。最终，服务层返回的结果由控制器提供给前端页面渲染，结合 Thymeleaf 模板和 Chart.js，实现数据可视化与报表展示。

五、 业务功能模块实现

5.1 注册和登录的功能模块

5.1.1 功能实现

- 身份模型与存储
 - 表: users (RowKey=用户名), 列族 u: password_hash、role、created_at
 - 密码使用 BCrypt 单向哈希 (自带盐)，只存 hash，不存明文

- 会话与保护
 - 登录成功后把精简用户信息放入 Session（例如 SESSION_USER）
 - 使用拦截器保护受限路由（如 /cluster、/annual、/trend、/range 等），未登录跳转 /login
- 注册/登录校验
 - 用户名唯一、密码长度（ ≥ 8 ）与复杂度（至少字母+数字）
 - 登录失败返回统一错误提示，不暴露具体原因
 - 可选：失败次数限制/验证码、Session 过期时间
- 成功标准（快速验收）
 - 注册：同名拒绝，hash 入库成功
 - 登录：成功写 Session 并跳转首页；失败返回提示
 - 退出：Session 失效；受限页重定向 /login

5.1.2 关键代码

(1) UserService：注册与认证

```
@Service
public class UserService {
    private final HBaseUserRepository repo;
    private final org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder encoder =
        new org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder();

    public UserService(HBaseUserRepository repo) { this.repo = repo; }

    public boolean register(String username, String rawPwd) throws IOException {
        if (username == null || username.isBlank() || rawPwd == null || rawPwd.length() <
8) return false;
        if (repo.findByUsername(username) != null) return false; // 已存在
        String hash = encoder.encode(rawPwd);
        User u = new User(username, hash, "USER");
        repo.save(u);
        return true;
    }

    public User authenticate(String username, String rawPwd) throws IOException {
        User u = repo.findByUsername(username);
        if (u == null) return null;
        return encoder.matches(rawPwd, u.getPasswordHash()) ? u : null;
    }
}
```

(2) HBaseUserRepository：最小读写

```
@Repository
public class HBaseUserRepository {
```

```

private final org.apache.hadoop.conf.Configuration hconf;
private static final byte[] CF = org.apache.hadoop.hbase.util.Bytes.toBytes("u");

public HBaseUserRepository(org.apache.hadoop.conf.Configuration hconf) { this.hconf = hconf; }

public User findByUsername(String username) throws IOException {
    try (var conn =
org.apache.hadoop.hbase.client.ConnectionFactory.createConnection(hconf);
        var table = conn.getTable(org.apache.hadoop.hbase.TableName.valueOf("users")))
    {
        var get = new
org.apache.hadoop.hbase.client.Get(org.apache.hadoop.hbase.util.Bytes.toBytes(username));
        var r = table.get(get);
        if (r == null || r.isEmpty()) return null;
        String hash = org.apache.hadoop.hbase.util.Bytes.toString(r.getValue(CF,
b("password_hash")));
        String role = org.apache.hadoop.hbase.util.Bytes.toString(r.getValue(CF,
b("role")));
        return new User(username, hash, role == null ? "USER" : role);
    }
}

public void save(User u) throws IOException {
    try (var conn =
org.apache.hadoop.hbase.client.ConnectionFactory.createConnection(hconf);
        var table = conn.getTable(org.apache.hadoop.hbase.TableName.valueOf("users")))
    {
        var put = new org.apache.hadoop.hbase.client.Put(b(u.getUsername()))
            .addColumn(CF, b("password_hash"), b(u.getPasswordHash()))
            .addColumn(CF, b("role"), b(u.getRole()))
            .addColumn(CF, b("created_at"),
b(Long.toString(System.currentTimeMillis())));
        table.put(put);
    }
}

private static byte[] b(String s) { return
org.apache.hadoop.hbase.util.Bytes.toBytes(s); }
}

```

(3) AuthController: 注册/登录/退出 (会话)

```

@Controller
public class AuthController {

```

```

private final UserService userService;
public AuthController(UserService userService) { this.userService = userService; }

@PostMapping("/register")
public String doRegister(@RequestParam String username, @RequestParam String password,
Model m) throws IOException {
    if (!userService.register(username, password)) {
        m.addAttribute("error", "注册失败: 用户名存在或密码不符合规则");
        return "register";
    }
    return "redirect:/login?registered";
}

@PostMapping("/login")
public String doLogin(@RequestParam String username, @RequestParam String password,
    javax.servlet.http.HttpSession session, Model m) throws
IOException {
    User u = userService.authenticate(username, password);
    if (u == null) {
        m.addAttribute("error", "用户名或密码错误");
        return "login";
    }
    session.setAttribute("SESSION_USER", new SessionUser(u.getUsername(), u.getRole()));
    // 轻量存储
    return "redirect:/";
}

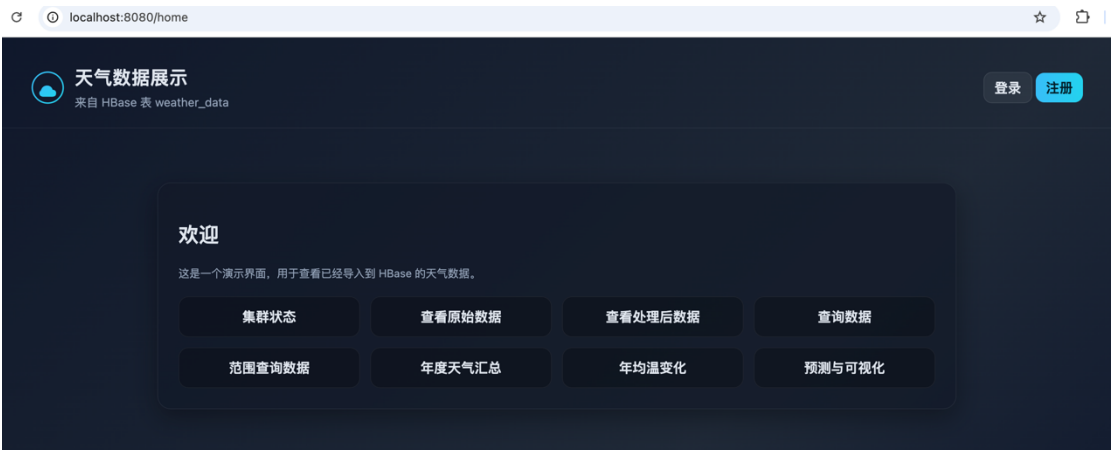
@GetMapping("/logout")
public String logout(javax.servlet.http.HttpSession session) {
    session.invalidate();
    return "redirect:/login?logout";
}

public static record SessionUser(String username, String role) { }
}

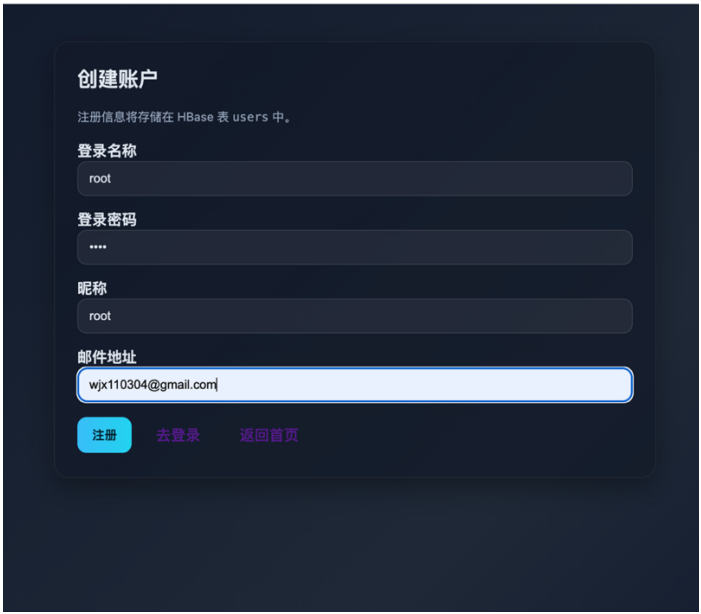
```

5.1.3 运行效果图

主界面设计：



注册用户 root：



用户 root 注册成功：



登录已注册的用户 root：



The image shows a login form titled "账户登录" (Account Login). It prompts the user to "请输入登录名与密码。" (Please enter login name and password). There are two input fields: "登录名称" (Login Name) with the value "root" and "登录密码" (Login Password) with masked characters "****". Below the fields are three buttons: a blue "登录" (Login) button, a purple "去注册" (Go to Register) link, and a purple "返回首页" (Return to Home) link.

完成登录（如右上角所示）：



The image shows a dashboard titled "天气数据展示" (Weather Data Display) with the subtitle "来自 HBase 表 weather_data" (From HBase table weather_data). In the top right corner, it says "已登录: root" (Already logged in: root) next to a "退出" (Logout) button. The main area has a "欢迎" (Welcome) message and a description: "这是一个展示界面，用于查看已经导入到 HBase 的天气数据。" (This is a display interface for viewing weather data already imported into HBase). Below this are eight buttons arranged in two rows: "集群状态" (Cluster Status), "查看原始数据" (View Original Data), "查看处理后数据" (View Data After Processing), "查询数据" (Query Data), "范围查询数据" (Range Query Data), "年度天气汇总" (Annual Weather Summary), "年均温变化" (Annual Temperature Change), and "预测与可视化" (Prediction and Visualization).

用户无法重复注册验证：



The image shows a registration form titled "创建账户" (Create Account). It states "注册信息将存储在 HBase 表 users 中。" (Registration information will be stored in the HBase table users). A red error message "该登录名已存在" (This login name already exists) is displayed above the "登录名称" (Login Name) field, which has the placeholder "例如: alice" (e.g., alice). The "登录密码" (Login Password) field has a placeholder "至少 4 位" (At least 4 characters). Below are fields for "昵称" (Nickname) with placeholder "显示名称" (Display Name) and "邮件地址" (Email Address) with placeholder "you@example.com". At the bottom are three buttons: a blue "注册" (Register) button, a purple "去登录" (Go to Login) link, and a purple "返回首页" (Return to Home) link.

5.2 查看集群状态功能模块

5.2.1 功能实现

- 数据来源
 - 使用 HBase Admin API (ConnectionFactory + Admin) 获取 ClusterMetrics。
- 展示内容 (建议)
 - HBase 版本与 ClusterId
 - Master 与 Backup Masters
 - 在线节点数 (live)、离线节点数 (dead) 与离线节点名
 - Region 总数、平均负载 (fallback: regionCount/liveServers)
 - 每个 RegionServer 的名称与持有 Region 数
- 模块流转
 - Controller → 调 Service 取 ClusterStatusInfo → 交给 Thymeleaf 展示
- 稳健性
 - try-with-resources 关闭连接
 - 某些指标可能在不同版本为空或异常，注意空值兜底
 - 高频访问时可加简单缓存 (如 5~10 秒 TTL)

5.2.2 关键代码

(1) DTO: ClusterStatusInfo

```
public class ClusterStatusInfo {
    private String clusterId;
    private String hbaseVersion;
    private String master;
    private java.util.List<String> backupMasters = new java.util.ArrayList<>();
    private Integer liveServers;
    private Integer deadServers;
    private java.util.List<String> deadServerNames = new java.util.ArrayList<>();
    private Integer regionCount;
    private Double averageLoad;
    private java.util.List<ServerStat> servers = new java.util.ArrayList<>();

    public static class ServerStat {
        public final String name;
        public final int regions;
        public ServerStat(String name, int regions) { this.name = name; this.regions =
regions; }
    }

    // getter/setter 省略
}
```

(2) Service: getClusterStatus (获取指标)

```
@Service
public class WeatherService {
    private final org.apache.hadoop.conf.Configuration configuration;
    public WeatherService(org.apache.hadoop.conf.Configuration conf) { this.configuration =
conf; }

    public ClusterStatusInfo getClusterStatus() throws IOException {
        var info = new ClusterStatusInfo();
        try (var conn =
org.apache.hadoop.hbase.client.ConnectionFactory.createConnection(configuration);
            var admin = conn.getAdmin()) {
            var m = admin.getClusterMetrics();
            if (m == null) return info;

            info.setClusterId(m.getClusterId());
            info.setHbaseVersion(m.getHBaseVersion());

            var live = m.getLiveServerMetrics();
            info.setLiveServers(live == null ? 0 : live.size());
            if (live != null) {
                for (var e : live.entrySet()) {
                    String name = e.getKey() == null ? "" : e.getKey().getServerName();
                    int regions = 0;
                    try { regions = e.getValue().getRegionMetrics().size(); } catch
(Throwable ignore) {}
                    info.getServers().add(new ClusterStatusInfo.ServerStat(name, regions));
                }
            }

            var dead = m.getDeadServerNames();
            info.setDeadServers(dead == null ? 0 : dead.size());
            if (dead != null) {
                var names = new java.util.ArrayList<String>();
                for (var sn : dead) names.add(sn.getServerName());
                info.setDeadServerNames(names);
            }

            try { info.setRegionCount(m.getRegionCount()); } catch (Throwable ignore) {}
            Double avg = null; try { avg = m.getAverageLoad(); } catch (Throwable ignore)
{}

            if (avg == null && info.getRegionCount() != null && info.getLiveServers() !=
null && info.getLiveServers() > 0) {
                avg = info.getRegionCount() * 1.0 / info.getLiveServers();
            }
        }
    }
}
```

```

    }

    info.setAverageLoad(avg);

    try { info.setMaster(m.getMasterName() == null ? null :
m.getMasterName().getServerName()); } catch (Throwable ignore) {}

    try {
        var backs = m.getBackupMasterNames();
        if (backs != null) {
            var b = new java.util.ArrayList<String>();
            for (var sn : backs) b.add(sn.getServerName());
            info.setBackupMasters(b);
        }
    } catch (Throwable ignore) {}
}

return info;
}
}

```

(3) Controller: /cluster 路由（交模型到视图）

```

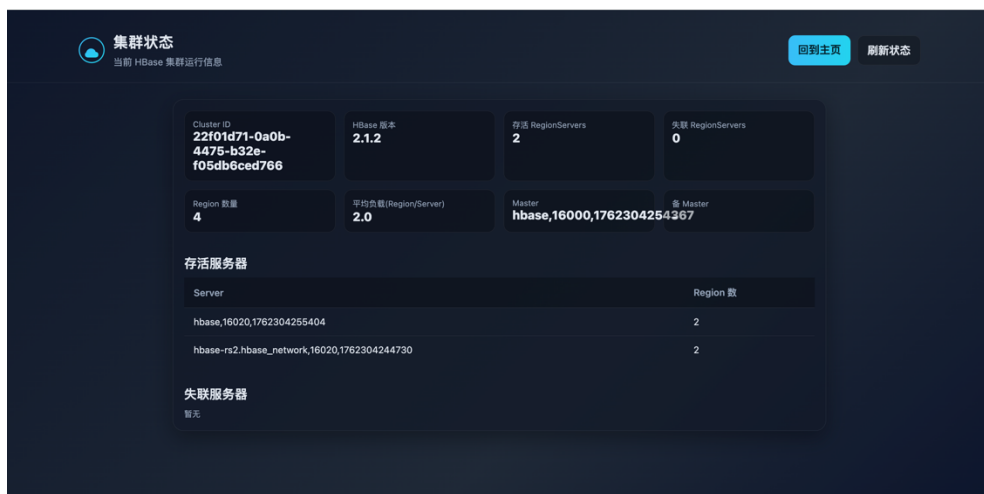
@Controller
public class WeatherController {
    private final WeatherService weatherService;

    public WeatherController(WeatherService weatherService) { this.weatherService =
weatherService; }

    @GetMapping("/cluster")
    public String cluster(Model model) throws IOException {
        model.addAttribute("status", weatherService.getClusterStatus());
        return "cluster";
    }
}

```

5.2.3 运行效果图



5.3 查看数据的功能（包含原始数据与预处理）

5.3.1 功能实现

- 总体流程
 - 浏览器发起请求（原始数据或处理后数据）。
 - 控制器接参数，调用服务层。
 - 服务层用 HBase Client 扫描数据（Scan + 可选前缀过滤），解析 rowkey 和列值。
 - 对于“处理后数据”，在内存里按日或年聚合（分组 + 统计），得到简化 DTO。
 - 把列表塞进 Model，Thymeleaf 模板渲染页面（如 index.html、daily.html、range.html、annual.html 等）。
- 原始数据（Raw）
 - 入口：/data。
 - 服务：全表 Scan，解析 rowkey（stationId#date#hour → 站点、日期、时分），读取列族 info/measurements。
 - 模板：index.html 展示原始观测记录（受 limit 限制）。
- 处理后数据（Processed）
 - 日聚合：/daily 展示最新一批按日聚合的记录；/range 支持指定 stationId + 起止日期，精确聚合；/query 以日聚合结果替代原始明细，便于展示。
 - 年度汇总：/annual、/trend 以 stationId 为前缀扫描，按年统计总降雨量、年极值、年均温等。
 - 模板：daily.html、range.html、annual.html、trend.html 分别渲染不同聚合视图。

5.3.2 关键代码

（1）控制器：原始数据入口（/data）

```
@GetMapping("/data")
public String index(@RequestParam(name = "limit", defaultValue = "100") int limit, Model
model) {
    List<Weather> items = new ArrayList<>();
    String errorMessage = null;
    try {
        items = weatherService.findAll(limit); // 原始数据
    } catch (Exception e) {
        errorMessage = "无法连接到 HBase 集群: " + e.getMessage();
    }
    model.addAttribute("weathers", items);
    model.addAttribute("limit", limit);
    model.addAttribute("errorMessage", errorMessage);
    return "index";
}
```

(2) 控制器：日聚合（按站点 + 日期范围，/range）

```
@GetMapping("/range")
public String range(@RequestParam String stationId,
                   @RequestParam String startDate,
                   @RequestParam String endDate,
                   @RequestParam(name = "limit", defaultValue = "100") int limit,
                   Model model) {
    List<Weather> items = null;
    String errorMessage = null;
    try {
        items = weatherService.findDailyAggregatedByDateRange(stationId, startDate, endDate,
limit);
    } catch (Exception e) {
        errorMessage = "日期范围查询出错: " + e.getMessage();
    }
    model.addAttribute("weathers", items);
    model.addAttribute("errorMessage", errorMessage);
    return "range";
}
```

(3) 服务层：原始数据扫描（HBase Scan）

```
public List<Weather> findAll(int limit) throws IOException {
    List<Weather> list = new ArrayList<>();
    try (Connection connection = ConnectionFactory.createConnection(configuration);
        Table table = connection.getTable(TableName.valueOf("weather_data"))) {

        Scan scan = new Scan();
        scan.addFamily(CF_INFO);
        scan.addFamily(CF_MEAS);

        try (ResultScanner scanner = table.getScanner(scan)) {
            for (Result r : scanner) {
                Weather w = new Weather();
                String rowKey = Bytes.toString(r.getRow()); // stationId#date#hour
                String[] parts = rowKey.split("#");
                if (parts.length >= 2) {
                    w.setStationId(parts[0]);
                    if (parts.length >= 3) {
                        String hour = parts[2]; // 0000 → 00:00
                        w.setDate(parts[1] + " " + hour.substring(0, 2) + ":" +
hour.substring(2, 4));
                    } else {
                        w.setDate(parts[1]);
                    }
                }
            }
        }
    }
}
```

```

    }
    // 读取列族/列限定符
    byte[] v;
    v = r.getValue(CF_INFO, Bytes.toBytes("station_name"));
    if (v != null) w.setStationName(new String(v, StandardCharsets.UTF_8));
    v = r.getValue(CF_MEAS, Bytes.toBytes("mean_temp"));
    if (v != null) w.setAvgTemperature(parseDoubleSafe(Bytes.toString(v)));
    // ... 其他列略
    list.add(w);
    if (limit > 0 && list.size() >= limit) break;
}
}
}
return list;
}

```

(4) 服务层：日聚合（前缀扫描 + 分组 + 聚合）

```

public List<Weather> findDailyAggregatedByDateRange(String stationId, String
startDate, String endDate, int limit) throws IOException {
    java.time.LocalDate start = parseFlexibleDate(startDate);
    java.time.LocalDate end = parseFlexibleDate(endDate);

    Map<java.time.LocalDate, List<Weather>> dailyGroups = new LinkedHashMap<>();
    try (Connection connection = ConnectionFactory.createConnection(configuration);
        Table table = connection.getTable(TableName.valueOf("weather_data"))) {

        Scan scan = new Scan();
        scan.setRowPrefixFilter(Bytes.toBytes(stationId + "#")); // 关键：行键前缀
stationId#
        scan.addFamily(CF_INFO);
        scan.addFamily(CF_MEAS);

        try (ResultScanner scanner = table.getScanner(scan)) {
            for (Result r : scanner) {
                String[] parts = Bytes.toString(r.getRow()).split("#");
                String dateStr = (parts.length >= 2) ? parts[1] : null; // dd/MM/yyyy
                java.time.LocalDate d = parseFlexibleDate(dateStr);
                if (d == null || d.isBefore(start) || d.isAfter(end)) continue;

                Weather w = /* 从 r 构建 Weather (略) */;
                dailyGroups.computeIfAbsent(d, k -> new ArrayList<>()).add(w);
            }
        }
    }
}

```

```

// 排序 + 限制 + 调用“日聚合函数”
List<java.time.LocalDate> ordered = new ArrayList<>(dailyGroups.keySet());
ordered.sort(java.time.LocalDate::compareTo);
List<Weather> result = new ArrayList<>();
int count = 0;
java.time.format.DateTimeFormatter df =
java.time.format.DateTimeFormatter.ofPattern("dd/MM/yyyy");
for (java.time.LocalDate d : ordered) {
    if (limit > 0 && count >= limit) break;
    Weather agg = aggregateDailyData(df.format(d), dailyGroups.get(d)); // 核心：同日多
条合一
    if (agg != null) { result.add(agg); count++; }
}
return result;
}

```

(5) 服务层：日聚合的核心合并逻辑

```

private Weather aggregateDailyData(String date, List<Weather> records) {
    if (records.isEmpty()) return null;

    Weather aggregated = new Weather();
    aggregated.setStationId(records.get(0).getStationId());
    aggregated.setStationName(records.get(0).getStationName());
    aggregated.setDate(date); // 仅日期

    Double maxTemp = null, minTemp = null, precipitation = null, avgTemp = null;
    for (Weather r : records) {
        if (r.getMaxTemperature() != null) maxTemp = r.getMaxTemperature();
        if (r.getMinTemperature() != null) minTemp = r.getMinTemperature();
        if (r.getPrecipitation() != null) precipitation = r.getPrecipitation();
        if (r.getAvgTemperature() != null) avgTemp = r.getAvgTemperature();
    }
    aggregated.setMaxTemperature(maxTemp);
    aggregated.setMinTemperature(minTemp);
    aggregated.setPrecipitation(precipitation);
    aggregated.setAvgTemperature(avgTemp);
    return aggregated;
}

```

5.3.3 运行效果图

查看原始数据功能：

天气数据展示
来自 HBase 表 weather_data

回到主页

站点编号	站点名称	日期	降水量	最高温	最低温	平均温
82024		01/01/1961 00:00		32.3		26.56
82024		01/01/1961 12:00			22.9	
82024		01/01/1961 18:00				
82024		01/01/1962 00:00		31.7		27.4
82024		01/01/1962 12:00	0.0		23.2	
82024		01/01/1962 18:00				
82024		01/01/1972 00:00		30.5		27.1
82024		01/01/1972 12:00			23.0	
82024		01/01/1972 18:00				
82024		01/01/1973 00:00		34.1		28.24
82024		01/01/1973 12:00			24.0	
82024		01/01/1973 18:00				
82024		01/01/1974 00:00		33.4		27.62
82024		01/01/1974 12:00	0.0		23.5	
82024		01/01/1974 18:00				

查看处理后数据功能：

日聚合天气数据
来自 HBase 表 weather_data (日聚合)

回到主页

站点编号	站点名称	日期	降水量	最高温	最低温	平均温
82024		01/01/1961		32.3	22.9	26.56
82024		01/01/1962	0.0	31.7	23.2	27.4
82024		01/01/1972		30.5	23.0	27.1
82024		01/01/1973		34.1	24.0	28.24
82024		01/01/1974	0.0	33.4	23.5	27.62
82024		01/01/1975	0.0	30.7	23.3	25.46
82024		01/01/1976	0.0	31.5	23.2	26.76
82024		01/01/1977	0.0	31.2	23.0	26.44
82024		01/01/1978	0.0	33.5	21.5	27.3
82024		01/01/1979	0.0	32.8	22.8	26.84

5.4 查询天气数据功能

5.4.1 实现思路

- 入口与数据流
 - Web 入口：/query 路由携带 date 参数（建议格式 dd/MM/yyyy，如 01/01/1961），可选 stationId、stationName。
 - 控制器：调用服务层 findDailyAggregated(stationId, stationName, date, limit)。
 - 先从 HBase 扫描原始记录（findByFields），按日期（去掉时分）分组。
 - 用“日聚合函数”把同一天不同时段的多条记录合成一条日记录。
 - 模板：把 weathers 渲染到 query.html（或你需要的页面）。

- 精准单日（可选优化）
 - 若 你 已 知 stationId 且 只 查 一 天 ， 使 用 findDailyAggregatedByDateRange(stationId, day, day, 1) 更精确且更快（行键前缀 + LocalDate 范围过滤）。

5.4.2 关键代码

（1）控制器：/query 作为“单日查询”入口

```
@GetMapping("/query")
public String query(@RequestParam(required=false) String stationId,
                   @RequestParam(required=false) String stationName,
                   @RequestParam(required=false) String date,
                   @RequestParam(name="limit", defaultValue="100") int limit,
                   Model model) {
    List<Weather> items = null;
    String errorMessage = null;
    boolean searched = false;
    if ((stationId!=null && !stationId.isEmpty()) ||
        (stationName!=null && !stationName.isEmpty()) ||
        (date!=null && !date.isEmpty())) {
        searched = true;
        try {
            items = weatherService.findDailyAggregated(stationId, stationName, date, limit);
        } catch (Exception e) {
            errorMessage = "查询时发生错误: " + e.getMessage();
        }
    }
    model.addAttribute("weathers", items);
    model.addAttribute("searched", searched);
    model.addAttribute("errorMessage", errorMessage);
    return "query";
}
```

（2）服务层：单日聚合主流程

```
public List<Weather> findDailyAggregated(String stationId, String stationName, String
date, int limit) throws IOException {
    // 取更多原始数据用于聚合
    List<Weather> rawData = findByFields(stationId, stationName, date, limit * 3);

    // 1) 按“日期部分”分组（把 01/01/1961 12:00 → 01/01/1961）
    Map<String, List<Weather>> dailyGroups = new LinkedHashMap<>();
    for (Weather w : rawData) {
        String dateOnly = extractDateOnly(w.getDate());
        dailyGroups.computeIfAbsent(dateOnly, k -> new ArrayList<>()).add(w);
    }
}
```

```

}

// 2) 对每个日期调用“日聚合函数”合成一条
List<Weather> aggregated = new ArrayList<>();
int count = 0;
for (Map.Entry<String, List<Weather>> e : dailyGroups.entrySet()) {
    if (count >= limit) break;
    Weather d = aggregatedDailyData(e.getKey(), e.getValue()); // 核心合并
    if (d != null) { aggregated.add(d); count++; }
}
return aggregated;
}

```

5.4.3 运行效果图

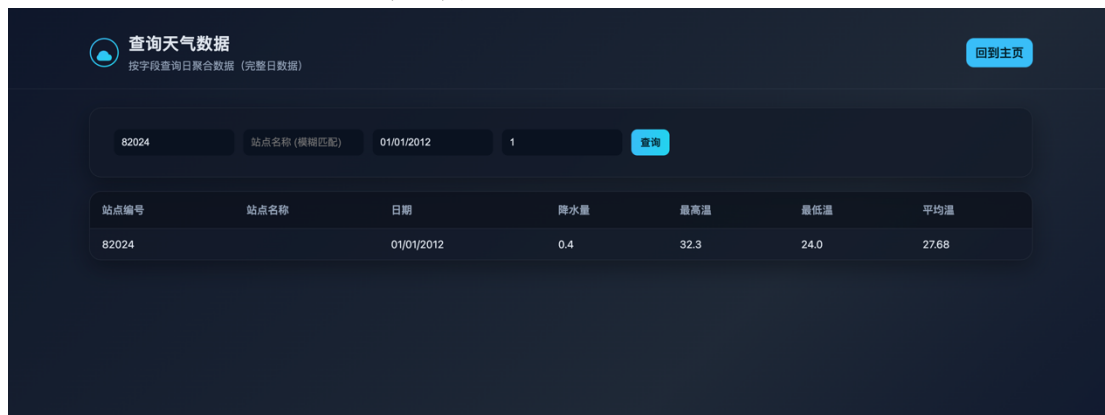
查询 83377 站点 2009 年 1 月 1 日天气结果如下：



The screenshot shows a web interface titled "查询天气数据" (Query Weather Data) with a subtitle "按字段查询日聚合数据 (完整日数据)". It includes a search bar with fields for station ID (83377), station name (模糊匹配), date (01/01/2009), and a limit (1). A "查询" (Query) button is present. Below the search bar is a table with the following data:

站点编号	站点名称	日期	降水量	最高温	最低温	平均温
83377		01/01/2009	2.7	30.1	19.5	23.1

查询 82024 站点 2012 年 1 月 1 日天气结果如下：



The screenshot shows the same web interface as above, but for station 82024 on 01/01/2012. The table data is as follows:

站点编号	站点名称	日期	降水量	最高温	最低温	平均温
82024		01/01/2012	0.4	32.3	24.0	27.68

5.5 年度天气汇总

5.5.1 实现思路

- 入口与数据流
 - Web 入口: /annual?stationId=... (/trend 也复用同一统计方法, 只是前端以图表展示)。

- 控制器：校验 stationId 后调用 weatherService.aggregateAnnualByStation(stationId)。
- 服务层：
 - 用行键前缀 stationId# 扫描该站点所有记录（HBase Scan + PrefixFilter）。
 - 从 rowkey 的日期部分提取“年”（支持 dd/MM/yyyy、yyyy-MM-dd）。
 - 按年份累加：年降雨量（sum）、年最高温（max of max_temp）、年最低温（min of min_temp）、年平均温（mean_temp 的均值）、有效记录数（时间点条数）。
 - 输出 List<AnnualStat> 并按年份升序。
 - 模板：annual.html 渲染表格；trend.html 以年份为横轴、平均温度为序列绘图。
- 数据契约（简要）
 - 输入：stationId（必填）。
 - 输出：List<AnnualStat>（字段：year、totalPrecipitation、maxTemperature、minTemperature、avgTemperature、recordCount）。
 - 成功：返回按年汇总的若干行；异常时展示错误信息不阻塞页面。
- 性能要点
 - 利用行键前缀只扫描一个站点，避免全表扫。
 - 数据量增大时可考虑 Coprocessor/Spark 下推统计；当前实现适合演示与中小数据量。

5.5.2 关键代码

(1) 控制器：年度汇总入口（/annual）

```
@GetMapping("/annual")
public String annual(@RequestParam(name = "stationId", required = false) String stationId,
                    Model model) {
    String errorMessage = null;
    boolean searched = false;
    List<AnnualStat> stats = null;
    if (stationId != null && !stationId.isEmpty()) {
        searched = true;
        try {
            stats = weatherService.aggregateAnnualByStation(stationId); // 年度汇总
        } catch (Exception e) {
            errorMessage = "年度汇总查询出错：" + e.getMessage();
        }
    }
    model.addAttribute("searched", searched);
    model.addAttribute("errorMessage", errorMessage);
    model.addAttribute("stationId", stationId);
    model.addAttribute("stats", stats);
    return "annual";
}
```

(2) 服务层：按年聚合的核心逻辑

```
public List<AnnualStat> aggregateAnnualByStation(String stationId) throws IOException
{
    Map<Integer, AnnualStat.Accumulator> acc = new HashMap<>();
    try (Connection connection = ConnectionFactory.createConnection(configuration);
        Table table = connection.getTable(TableInfo.valueOf("weather_data"))) {

        Scan scan = new Scan();
        scan.setFilter(new PrefixFilter(Bytes.toBytes(stationId + "#"))); // 关键：站点前缀
        scan.addFamily(CF_MEAS);

        try (ResultScanner scanner = table.getScanner(scan)) {
            for (Result r : scanner) {
                String rowKey = Bytes.toString(r.getRow()); // stationId#date#hour
                String[] keyParts = rowKey.split("#");
                if (keyParts.length < 2) continue;

                Integer year = extractYear(keyParts[1]); // 从日期提取年份
                if (year == null) continue;

                AnnualStat.Accumulator a = acc.computeIfAbsent(year, k -> new
AnnualStat.Accumulator());

                // 读取测量值（非空才统计）
                byte[] v;
                v = r.getValue(CF_MEAS, Bytes.toBytes("precipitation"));
                Double p = v != null ? parseDoubleSafe(Bytes.toString(v)) : null;
                v = r.getValue(CF_MEAS, Bytes.toBytes("max_temp"));
                Double tmax = v != null ? parseDoubleSafe(Bytes.toString(v)) : null;
                v = r.getValue(CF_MEAS, Bytes.toBytes("min_temp"));
                Double tmin = v != null ? parseDoubleSafe(Bytes.toString(v)) : null;
                v = r.getValue(CF_MEAS, Bytes.toBytes("mean_temp"));
                Double tavg = v != null ? parseDoubleSafe(Bytes.toString(v)) : null;

                if (p != null) { a.sumPrecipitation += p; a.precipitationDays++; }
                if (tmax != null) { a.maxTemperature = (a.maxTempDays == 0) ? tmax :
Math.max(a.maxTemperature, tmax); a.maxTempDays++; }
                if (tmin != null) { a.minTemperature = (a.minTempDays == 0) ? tmin :
Math.min(a.minTemperature, tmin); a.minTempDays++; }
                if (tavg != null) { a.sumMeanTemp += tavg; a.meanTempDays++; }

                if (p != null || tmax != null || tmin != null || tavg != null)
a.totalDays++;
            }
        }
    }
}
```

```

    }
}

// 累加器 => AnnualStat, 并按年份排序
List<AnnualStat> result = new ArrayList<>();
for (Map.Entry<Integer, AnnualStat.Accumulator> e : acc.entrySet()) {
    Integer year = e.getKey();
    AnnualStat.Accumulator a = e.getValue();
    AnnualStat stat = new AnnualStat();
    stat.setYear(year);
    stat.setTotalPrecipitation(a.precipitationDays > 0 ? a.sumPrecipitation : null);
    stat.setMaxTemperature(a.maxTempDays > 0 ? a.maxTemperature : null);
    stat.setMinTemperature(a.minTempDays > 0 ? a.minTemperature : null);
    stat.setAvgTemperature(a.meanTempDays > 0 ? a.sumMeanTemp / a.meanTempDays : null);
    stat.setRecordCount(a.totalDays);
    result.add(stat);
}
result.sort(Comparator.comparingInt(AnnualStat::getYear));
return result;
}

```

(3) 年份提取（鲁棒，容忍多种日期格式）

```

private Integer extractYear(String date) {
    if (date == null || date.isEmpty()) return null;
    String[] parts = date.split("[/\\-\\s]");
    for (int i = parts.length - 1; i >= 0; i--) {
        String p = parts[i].trim();
        if (p.length() == 4 && p.matches("\\d{4}")) return Integer.valueOf(p);
    }
    Matcher m = Pattern.compile("(\\d{4})").matcher(date);
    return m.find() ? Integer.valueOf(m.group(1)) : null;
}

```

5.5.3 运行效果图

查询 83377 站点年度天气汇总：

年度统计数据

来自 HBase 表 weather_data（年度聚合）

回到主页

83377

查询年度统计

年度数据汇总说明：
输入站点编号以查看该站每年的年降雨量、最高/最低/平均温度。
• 有效天数表示该年的有效记录数量（包含所有时间点）
• 现在使用完整的时间序列数据。每天可能有多个观测时间点
• "-" 表示该项无有效数据
• 年度汇总会处理所有时间点数据以获得最准确的统计结果

年份	年降雨量(mm)	最高温度(°C)	最低温度(°C)	平均温度(°C)	有效天数
1961	65.2999999999998	33.4	10.8	22.68597014925373	199
1962	1065.3	33.0	5.3	19.298866396761145	564
1963	1285.4999999999995	34.5	9.0	21.19753424657533	730
1964	1960.1000000000001	30.7	6.0	20.000601092896172	732
1965	2004.3999999999998	33.0	9.2	20.35945205479452	730
1966	1687.0999999999999	32.0	9.0	20.231890410958904	730
1967	1261.0	32.5	9.5	20.530794520647953	730
1968	1668.5999999999992	33.3	7.8	19.705573770491807	732
1969	1507.0999999999992	33.2	7.1	20.828055555555547	725
1970	1733.6000000000001	31.6	10.1	20.53112328767124	730

查询 82024 站点年度天气汇总：

年度统计数据

来自 HBase 表 weather_data（年度聚合）

回到主页

82024

查询年度统计

年度数据汇总说明：
输入站点编号以查看该站每年的年降雨量、最高/最低/平均温度。
• 有效天数表示该年的有效记录数量（包含所有时间点）
• 现在使用完整的时间序列数据。每天可能有多个观测时间点
• "-" 表示该项无有效数据
• 年度汇总会处理所有时间点数据以获得最准确的统计结果

年份	年降雨量(mm)	最高温度(°C)	最低温度(°C)	平均温度(°C)	有效天数
1961	1382.9000000000003	36.3	20.1	27.71019178082193	730
1962	60.4	34.8	20.8	27.9942372881356	118
1970	1080.2	37.0	16.8	26.870000000000002	384
1971	0.0	37.2	10.2	26.957586206896547	235
1972	152.5	36.1	19.5	26.947409836065578	610
1973	535.8000000000001	36.9	20.6	27.670915254237276	592
1974	1760.7000000000012	35.5	13.2	26.796097560975618	694
1975	1240.4000000000005	36.3	18.2	27.11509641873278	730
1976	2134.2	36.2	19.0	26.939508196721302	732
1977	1630.4	36.0	19.0	27.369369863013677	730
1978	1368.5000000000005	36.2	18.7	27.376076923076904	730

5.6 天气预测与可视化功能模块

5.6.1 功能实现

- 天气预测（/forecast）
 - 输入：stationId、inputDate（格式 MM/DD，如 01/01）。

- 控制器：校验输入 → 调用服务 generate7DayWeatherForecast(stationId, inputDate) → 将 forecast 放入 Model。
- 服务：基于“同月同日的历史记录”做统计预测
- 生成连续 7 天的日期序列（仅月/日）。
- 对每一天：按行键前缀 stationId# 扫描 HBase；从 rowkey 取 dd/MM/yyyy，提取成 MM/DD 与目标日比对；收集 max/min/mean_temp、precipitation，取平均作为预测值；记录使用的历史数据点数。
- 输出：WeatherForecast（含 7 天 DailyForecast 列表与总历史数据点数）。
- 空数据处理：无历史数据则展示提示不阻塞页面。
- 可视化（/forecast、/trend）
 - /forecast: Thymeleaf 把 forecast.dailyForecasts 注入页面，Chart.js 画“温度折线 + 降水柱状”两张图。
 - /trend: /annual 的统计结果（aggregateAnnualByStation）被 /trend 复用，在 trend.html 用 Chart.js 画年均温趋势线。

5.6.2 关键代码

（1）控制器：预测入口（/forecast）

```
@GetMapping("/forecast")
public String forecast(@RequestParam(name = "stationId", required = false) String
stationId,
                      @RequestParam(name = "inputDate", required = false) String
inputDate,
                      Model model) {
    String errorMessage = null;
    boolean searched = false;
    WeatherForecast forecast = null;

    if (stationId != null && !stationId.isEmpty() && inputDate != null
&& !inputDate.isEmpty()) {
        searched = true;
        try {
            if (!inputDate.matches("\\d{2}/\\d{2}")) {
                errorMessage = "日期格式错误，请使用 MM/DD 格式，如：01/01";
            } else {
                forecast = weatherService.generate7DayWeatherForecast(stationId, inputDate);
            }
        } catch (Exception e) {
            errorMessage = "天气预测查询出错：" + e.getMessage();
        }
    }
}
```

```

model.addAttribute("searched", searched);
model.addAttribute("errorMessage", errorMessage);
model.addAttribute("stationId", stationId);
model.addAttribute("inputDate", inputDate);
model.addAttribute("forecast", forecast);
return "forecast";
}

```

(2) 服务层：7 天预测（核心逻辑）

```

public WeatherForecast generate7DayWeatherForecast(String stationId, String inputDate)
throws IOException {
    WeatherForecast forecast = new WeatherForecast(stationId, inputDate);
    forecast.setStationName("站点 " + stationId);

    String[] sevenDays = generateSevenDays(inputDate); // 连续 7 天（月/日）

    try (Connection connection = ConnectionFactory.createConnection(configuration);
        Table table = connection.getTable(TableName.valueOf("weather_data"))) {

        for (String dayDate : sevenDays) {
            WeatherForecast.DailyForecast daily = generateSingleDayForecast(connection,
table, stationId, dayDate);
            forecast.addDailyForecast(daily);
        }

        int total = forecast.getDailyForecasts().stream()
            .mapToInt(WeatherForecast.DailyForecast::getHistoricalDataPoints).sum();
        forecast.setTotalHistoricalDataPoints(total);
    }
    return forecast;
}

```

(3) 生成单日预测（同月日的历史均值）

```

private WeatherForecast.DailyForecast generateSingleDayForecast(
    Connection connection, Table table, String stationId, String targetDate) throws
IOException {

    WeatherForecast.DailyForecast daily = new WeatherForecast.DailyForecast(targetDate);

    Scan scan = new Scan();
    scan.setRowPrefixFilter(Bytes.toBytes(stationId + "#")); // 站点前缀约束

    List<Double> maxTemps = new ArrayList<>(), minTemps = new ArrayList<>(),
        avgTemps = new ArrayList<>(), precipitations = new ArrayList<>();
}

```

```

    try (ResultScanner scanner = table.getScanner(scan)) {
        for (Result result : scanner) {
            String rowKey = Bytes.toString(result.getRow()); // stationId#dd/MM/yyyy#HHmm
            String[] parts = rowKey.split("#");
            if (parts.length >= 2) {
                String recordMonthDay = extractMonthDayFromFullDate(parts[1]); // dd/MM/yyyy
→ MM/DD

                if (targetDate.equals(recordMonthDay)) {
                    addValueIfPresent(result, "measurements", "max_temp", maxTemps);
                    addValueIfPresent(result, "measurements", "min_temp", minTemps);
                    addValueIfPresent(result, "measurements", "mean_temp", avgTemps);
                    addValueIfPresent(result, "measurements", "precipitation",
precipitations);
                }
            }
        }
    }

    if (!maxTemps.isEmpty()) daily.setPredictedMaxTemp(calculateAverage(maxTemps));
    if (!minTemps.isEmpty()) daily.setPredictedMinTemp(calculateAverage(minTemps));
    if (!avgTemps.isEmpty()) daily.setPredictedAvgTemp(calculateAverage(avgTemps));
    if (!precipitations.isEmpty())
daily.setPredictedPrecipitation(calculateAverage(precipitations));
    daily.setHistoricalDataPoints(Math.max(Math.max(maxTemps.size(), minTemps.size()),
Math.max(avgTemps.size(),
precipitations.size())));
    return daily;
}

```

(4) 模板: Chart.js 绑定 (forecast.html)

```

<script th:if="${forecast}" th:inline="javascript">
const forecastData = [
    /*[# th:each="daily : ${forecast.dailyForecasts}"]*/
    { date: '/*[[${daily.date}]]*/',
      maxTemp: /*[[${daily.predictedMaxTemp}]]*/ null,
      minTemp: /*[[${daily.predictedMinTemp}]]*/ null,
      avgTemp: /*[[${daily.predictedAvgTemp}]]*/ null,
      precipitation: /*[[${daily.predictedPrecipitation}]]*/ null },
    /*[/]*/
];
const dates = forecastData.map(d => d.date);
const maxTemps = forecastData.map(d => d.maxTemp || 0);
const minTemps = forecastData.map(d => d.minTemp || 0);

```

```

const avgTemps = forecastData.map(d => d.avgTemp || 0);
const precipitations = forecastData.map(d => d.precipitation || 0);

// 温度折线
new Chart(document.getElementById('temperatureChart').getContext('2d'), {
  type: 'line',
  data: { labels: dates, datasets: [
    { label: '最高温度', data: maxTemps, borderColor: '#ff6b6b', fill: false },
    { label: '平均温度', data: avgTemps, borderColor: '#4ecdc4', fill: false },
    { label: '最低温度', data: minTemps, borderColor: '#45b7d1', fill: false }
  ]},
  options: { responsive: false, maintainAspectRatio: false }
});

// 降水柱状
new Chart(document.getElementById('precipitationChart').getContext('2d'), {
  type: 'bar',
  data: { labels: dates, datasets: [{ label: '降水量', data: precipitations,
backgroundColor: 'rgba(106, 176, 255, 0.7)'} ] },
  options: { responsive: false, maintainAspectRatio: false }
});
</script>

```

5.6.3 运行效果图

天气预测界面设计：

预测 83377 站点 11 月 5 日未来一周天气:

天气预测

基于历史数据的 7 天天气预测

输入预测参数

气象站编号:

83377

起始日期 (月/日):

11/05

请输入月/日格式。如：01/01 表示1月1日

生成7天预测

7天天气预测

气象站: 站点 83377

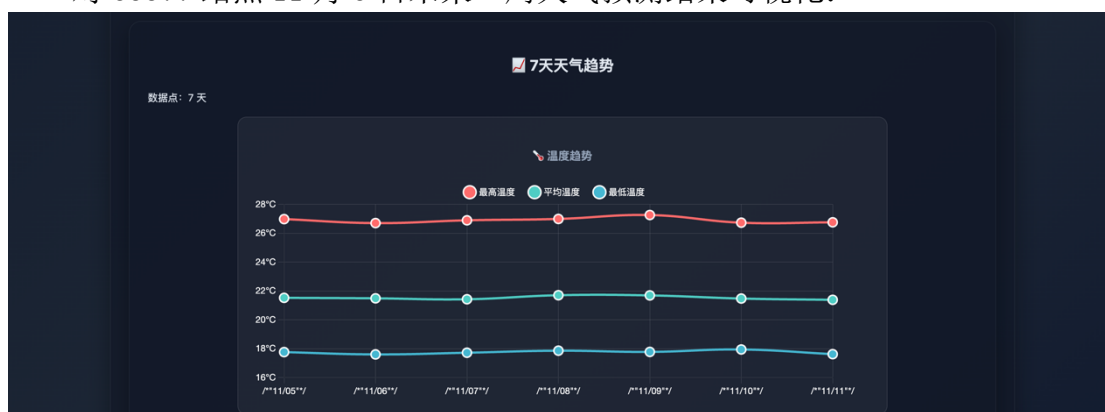
起始日期: 11/05

总历史数据点: 411 个历史记录

预测结果如下：

日期	 最高温度	 最低温度	 平均温度	 降水量	 历史数据点
11/05	27.0°C	17.8°C	21.5°C	7.38mm	59
11/06	26.7°C	17.6°C	21.5°C	7.12mm	59
11/07	26.9°C	17.7°C	21.4°C	7.20mm	59
11/08	27.0°C	17.9°C	21.7°C	6.69mm	58
11/09	27.3°C	17.8°C	21.7°C	6.04mm	59
11/10	26.7°C	17.9°C	21.5°C	7.52mm	58
11/11	26.8°C	17.6°C	21.4°C	8.08mm	59

对 83377 站点 11 月 5 日未来一周天气预测结果可视化:

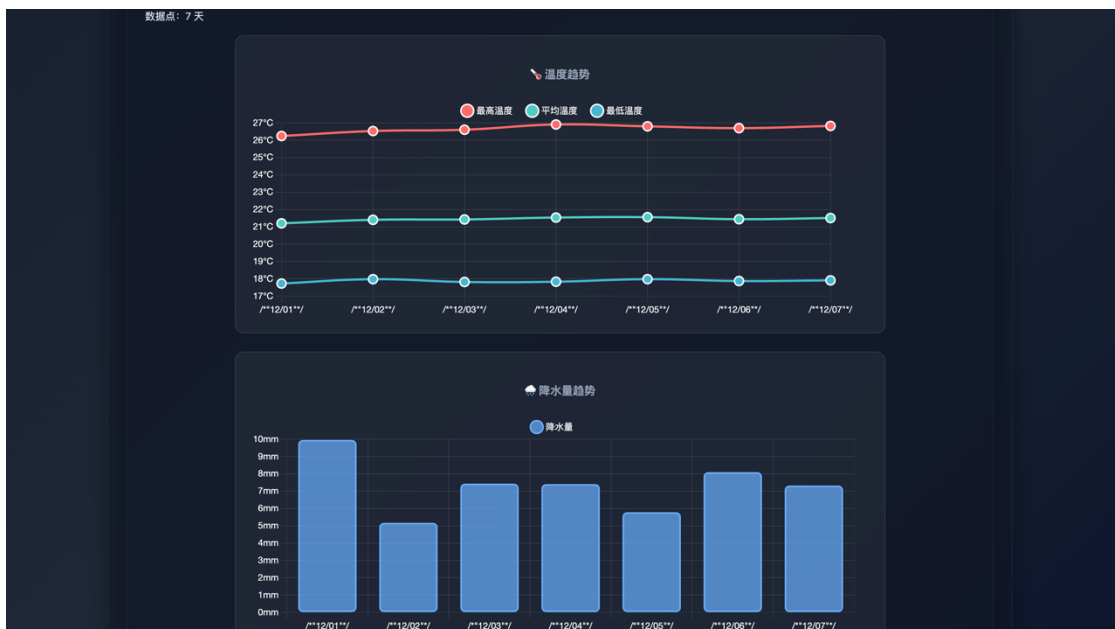




预测 83377 站点 12 月 1 日未来一周天气：



对 83377 站点 12 月 1 日未来一周天气预测结果可视化：



5.7 自选：范围日期查询

5.7.1 实现思路

- 入口与数据流
 - 路由：由 `/range?stationId=...&startDate=...&endDate=...&limit=...`
 - 控制器：校验参数（`stationId` 必填、起止日期必填），调用 `weatherService.findDailyAggregatedByDateRange(...)`，把结果与提示放入 `Model`，渲染 `range.html`。
 - 用行键前缀 `stationId#` 限定站点范围（避免全表扫）。
 - 解析起止日期（`yyyy-MM-dd` 或 `dd/MM/yyyy`），如 `end < start` 则交换。
 - 扫描结果按 `LocalDate` 分组，调用“日聚合”函数合并同一天的不同时段记录。
 - 按日期排序后截取 `limit` 条返回。
- 输入/输出与判定
 - 输入：`stationId`（必填）、`startDate`、`endDate`（建议 `dd/MM/yyyy` 或 `yyyy-MM-dd`）、`limit`（默认 100）。
 - 输出：`weathers`（日粒度聚合列表），出错时 `errorMessage`。
- 边界与性能
 - 日期格式非法会提示；起止颠倒会自动纠正；空数据返回空列表不阻塞页面。
 - 行键前缀大幅收窄扫描范围，适合演示和中小数据量；更大规模可考虑 `Coprocessor/Spark` 下推。

5.7.2 关键代码

（1）控制器：范围查询入口（`/range`）

```
@GetMapping("/range")
public String range(@RequestParam(name = "stationId", required = false) String stationId,
                   @RequestParam(name = "startDate", required = false) String startDate,
                   @RequestParam(name = "endDate", required = false) String endDate,
                   @RequestParam(name = "limit", required = false, defaultValue = "100")
int limit,
                   Model model) {
    List<Weather> items = null;
    String errorMessage = null;
    boolean searched = false;

    if ((startDate != null && !startDate.isEmpty()) ||
        (endDate != null && !endDate.isEmpty()) ||
        (stationId != null && !stationId.isEmpty())) {
        searched = true;
        try {
```

```

        if (stationId == null || stationId.isEmpty()) {
            errorMessage = "请输入站点编号";
        } else if (startDate == null || startDate.isEmpty() || endDate == null ||
endDate.isEmpty()) {
            errorMessage = "请同时选择起始日期与结束日期";
        } else {
            items = weatherService.findDailyAggregatedByDateRange(stationId, startDate,
endDate, limit);
        }
    } catch (IllegalArgumentException iae) {
        errorMessage = iae.getMessage();
    } catch (Exception e) {
        errorMessage = "日期范围查询出错: " + e.getMessage();
    }
}

model.addAttribute("weathers", items);
model.addAttribute("limit", limit);
model.addAttribute("errorMessage", errorMessage);
model.addAttribute("stationId", stationId);
model.addAttribute("startDate", startDate);
model.addAttribute("endDate", endDate);
model.addAttribute("searched", searched);
return "range";
}

```

(2) 服务层：前缀扫描 + 日期过滤 + 日聚合

```

public List<Weather> findDailyAggregatedByDateRange(String stationId, String
startDate, String endDate, int limit) throws IOException {
    if (stationId == null || stationId.trim().isEmpty()) {
        throw new IllegalArgumentException("请提供站点编号 stationId");
    }

    java.time.LocalDate start = parseFlexibleDate(startDate);
    java.time.LocalDate end = parseFlexibleDate(endDate);
    if (start == null || end == null) {
        throw new IllegalArgumentException("起止日期格式不正确, 应为 yyyy-MM-dd 或
dd/MM/yyyy");
    }

    if (end.isBefore(start)) { java.time.LocalDate t = start; start = end; end = t; }

    TableName tableName = TableName.valueOf("weather_data");
    Map<java.time.LocalDate, List<Weather>> dailyGroups = new LinkedHashMap<>();

    try (Connection connection = ConnectionFactory.createConnection(configuration);

```

```

    Table table = connection.getTable(tableName)) {
    Scan scan = new Scan();
    scan.setRowPrefixFilter(Bytes.toBytes(stationId + "#")); // 关键：按站点前缀收窄扫描
    scan.addFamily(CF_INFO);
    scan.addFamily(CF_MEAS);

    try (ResultScanner scanner = table.getScanner(scan)) {
        for (Result r : scanner) {
            String rowKey = Bytes.toString(r.getRow()); // stationId#dd/MM/yyyy#HHmm
            String[] parts = rowKey.split("#");
            if (parts.length < 2) continue;
            String dateStr = parts[1]; // dd/MM/yyyy
            java.time.LocalDate d = parseFlexibleDate(dateStr);
            if (d == null || d.isBefore(start) || d.isAfter(end)) continue;

            Weather w = /* 从 r 构建 Weather (读列族, 略) */;
            dailyGroups.computeIfAbsent(d, k -> new ArrayList<>()).add(w);
        }
    }

    // 排序 + 限制 + 合并同日记录为一条
    List<java.time.LocalDate> ordered = new ArrayList<>(dailyGroups.keySet());
    ordered.sort(java.time.LocalDate::compareTo);
    List<Weather> result = new ArrayList<>();
    int count = 0;
    java.time.format.DateTimeFormatter df =
    java.time.format.DateTimeFormatter.ofPattern("dd/MM/yyyy");
    for (java.time.LocalDate d : ordered) {
        if (limit > 0 && count >= limit) break;
        Weather agg = aggregateDailyData(df.format(d), dailyGroups.get(d)); // 核心聚合
        if (agg != null) { result.add(agg); count++; }
    }
    return result;
}

```

5. 7. 3 运行效果图

界面设计：

日期范围查询

按站点号在日期范围内的日聚合记录

回到主页

站点编号

开始日期

结束日期

数量上限

查询

必填，如 82024

年 / 月 / 日

年 / 月 / 日

100

按日期范围筛选观测记录

选择起止日期后，系统将仅显示位于该范围内的天气观测记录（包含起止日期）。

- 支持浏览器原生日期选择器（格式：yyyy-MM-dd）
- 结果包含小时级别的观测记录（如有），最多返回“数量上限”条
- 不改变底层技术栈：Spring Boot + Thymeleaf + HBase

查询 83377 站点 1999 年 12 月 10 日至 2000 年 3 月 11 日天气：

日期范围查询

按站点号在日期范围内的日聚合记录

回到主页

站点编号

开始日期

结束日期

数量上限

查询

83377

1999/12/10

2000/03/11

500

查询范围：1999-12-10 至 2000-03-11

站点编号	站点名称	日期	降水量	最高温	最低温	平均温
83377		10/12/1999	0.0	29.2	18.6	23.22
83377		11/12/1999	0.0	27.3	18.0	22.26
83377		12/12/1999	0.3	26.4	18.4	22.16
83377		13/12/1999	0.0	27.0	17.5	21.2
83377		14/12/1999	3.0	27.8	18.1	22.42
83377		15/12/1999	0.4	28.1	17.6	22.14
83377		16/12/1999	5.6	25.6	18.6	20.94
83377		17/12/1999	11.0	23.8	18.0	19.7
83377		18/12/1999	7.8	27.2	17.8	21.68
83377		19/12/1999	0.0	28.0	18.4	22.46
83377		20/12/1999	0.0	28.2	17.5	22.9

查询 82024 站点 2011 年 11 月 11 日至 2012 年 4 月 21 日天气：

日期范围查询

按站点号在日期范围内的日聚合记录

回到主页

站点编号

开始日期

结束日期

数量上限

查询

82024

2011/11/11

2012/04/21

500

查询范围：2011-11-11 至 2012-04-21

站点编号	站点名称	日期	降水量	最高温	最低温	平均温
82024		11/11/2011	0.0	35.2	25.8	29.18
82024		12/11/2011	2.1	33.1	24.8	26.62
82024		13/11/2011	2.0	35.2	24.6	28.68
82024		14/11/2011	1.6	34.1	24.4	28.22
82024		15/11/2011	0.0	34.7	24.4	29.38
82024		16/11/2011	0.0	35.4	25.4	29.94
82024		17/11/2011	0.0	30.6	24.6	25.68
82024		18/11/2011	0.6	32.1	24.0	27.38
82024		19/11/2011	1.4	34.2	24.5	28.02
82024		20/11/2011	0.0	34.7	25.0	28.9
82024		21/11/2011	0.0	35.8	25.3	29.54

5.8 自选：年均温变化

5.8.1 实现思路

- 数据流
 - 入口：/trend?stationId=...
 - 控制器：校验 stationId，调用 weatherService.aggregateAnnualByStation(stationId) 得到年度统计列表。
 - 服务层：用行键前缀 stationId# 扫描该站所有记录，逐条取出 measurements:mean_temp，按年份累计“和与样本数”，最后计算每年的平均温度。
 - 前端：trend.html 把年度列表注入 Chart.js，绘制“年均温趋势线”。
- 输入/输出
 - 输入：stationId（必填）
 - 输出：stats: List<AnnualStat>（字段含 year、avgTemperature、recordCount 等），页面同步显示折线图+表格
- 性能
 - 用行键前缀限制扫描范围，避免全表扫描；数据量增大时可考虑 Coprocessor/Spark 下推。

5.8.2 关键代码

(1) 控制器：/trend 入口

```
@GetMapping("/trend")
public String trend(@RequestParam(name = "stationId", required = false) String stationId,
                    Model model) {
    String errorMessage = null;
    boolean searched = false;
    List<AnnualStat> stats = null;
    if (stationId != null && !stationId.isEmpty()) {
        searched = true;
        try {
            stats = weatherService.aggregateAnnualByStation(stationId); // 年度统计（含年均温）
        } catch (Exception e) {
            errorMessage = "年均温趋势计算出错：" + e.getMessage();
        }
    }
    model.addAttribute("searched", searched);
    model.addAttribute("errorMessage", errorMessage);
    model.addAttribute("stationId", stationId);
    model.addAttribute("stats", stats);
    return "trend";
}
```

```
}
```

(2) 服务层：按年统计“年均温”的核心

```
public List<AnnualStat> aggregateAnnualByStation(String stationId) throws IOException
{
    Map<Integer, AnnualStat.Accumulator> acc = new HashMap<>();

    Scan scan = new Scan();
    scan.setFilter(new PrefixFilter(Bytes.toBytes(stationId + "#"))); // 关键：站点前缀
    scan.addFamily(CF_MEAS);

    try (Connection c = ConnectionFactory.createConnection(configuration);
        Table t = c.getTable(TableInfo.valueOf("weather_data"));
        ResultScanner rs = t.getScanner(scan)) {

        for (Result r : rs) {
            String[] keyParts = Bytes.toString(r.getRow()).split("#"); //
stationId#dd/MM/yyyy#HHmm
            if (keyParts.length < 2) continue;
            Integer year = extractYear(keyParts[1]); // 从日期提取年份
            if (year == null) continue;

            AnnualStat.Accumulator a = acc.computeIfAbsent(year, k -> new
AnnualStat.Accumulator());

            // 仅与年均温相关的几行
            byte[] v = r.getValue(CF_MEAS, Bytes.toBytes("mean_temp"));
            Double meanTemp = (v != null) ? parseDoubleSafe(Bytes.toString(v)) : null;
            if (meanTemp != null) { a.sumMeanTemp += meanTemp; a.meanTempDays++; }

            // 其余指标（降雨量、年极值）略
        }
    }

    List<AnnualStat> result = new ArrayList<>();
    for (Map.Entry<Integer, AnnualStat.Accumulator> e : acc.entrySet()) {
        Integer y = e.getKey(); AnnualStat.Accumulator a = e.getValue();
        AnnualStat s = new AnnualStat();
        s.setYear(y);
        s.setAvgTemperature(a.meanTempDays > 0 ? a.sumMeanTemp / a.meanTempDays : null); //
年均温
        // 其他字段（totalPrecipitation、max/min 等）略
        result.add(s);
    }
    result.sort(Comparator.comparingInt(AnnualStat::getYear));
}
```

```
return result;
}
```

(3) 前端可视化: Chart.js 绑定年均温

```
<script th:inline="javascript">
const annualStats = /*[[${stats}]]*/ [];
const labels = annualStats ? annualStats.map(s => s.year) : [];
const avgTemps = annualStats ? annualStats.map(s => s.avgTemperature) : [];

const ctx = document.getElementById('avgTempTrend');
if (ctx && labels.length > 0) {
  new Chart(ctx, {
    type: 'line',
    data: { labels, datasets: [{ label: '年平均温度', data: avgTemps, borderColor:
'#38bdf8', fill: true }] },
    options: { responsive: false, maintainAspectRatio: false }
  });
}
</script>
```

5.8.3 运行效果图

查询 83377 站点历史均温变化:



查询 82024 站点历史均温变化：



六、 项目总结

这个项目围绕“站点号 + 日期”的时序数据，成功构建了一个完整的业务闭环，实现了从原始数据查询、日聚合、年度汇总，到年均温趋势可视化与集群状态监控的端到端流程。首页入口和功能文案顺序严格按照需求统一，范围查询严格要求输入站点号和日期范围，趋势分析页通过 Chart.js 绘制固定尺寸折线图并配合表格展示，方便数据校对和报表汇报，保证了业务使用的规范性与可操作性。

在技术实现上，后端采用 Java 8 + Spring Boot 2.7，Controller/Service/Thymeleaf 三层架构清晰，WeatherService 实现了前缀/范围扫描和客户端聚合功能，同时将集群状态抽象为 DTO，兼容不同指标字段差异。HBase 2.1.2 用作时序数据存储，RowKey 设计为 stationId#date#hour，通过 info 与 measurements 两个列族存储基础信息和各类气象观测值，支持高效查询与聚合。前端采用 Thymeleaf 模板结合 Chart.js，可视化效果统一深色主题，布局稳定。项目通过 Maven 打包生成可执行 Jar，配合 Docker/Compose 实现一键启动，容器网络与 HBase/ZooKeeper 互通，支持优雅停启且数据持久化不丢失。

运维与可观测方面，系统提供 /cluster 页面，聚合显示 Master/备份节点、Region 数量、存活与宕机节点、平均负载等关键指标，为日常健康检查和问题定位提供便利。部署方式和端口映射清晰，支持本地或演示环境快速复现，保证了演示和测试的高效率。

总体来看，该项目形成了“数据→聚合→图表→运维”的完整工程样板，既满足了时序/IoT 类应用的业务需求，又具有清晰的代码边界和模块化设计，便于后续二次开发、迁移或性能优化，为类似系统的快速落地提供了可靠基础和实践参考。