# **简易分布式爬虫系统结业项目答辩汇报**

## **一、项目介绍**

本项目旨在设计并实现一个简易的分布式爬虫系统，能够高效地爬取Web站点的网页数据，并进行存储和检索。系统遵循robots协议，确保爬取行为合法合规。项目的核心功能包括URL管理、URL分发、数据爬取、数据解析、数据存储、数据索引和数据查询。系统采用分布式架构，支持多进程并发爬取，提高了爬取效率和系统的可扩展性。

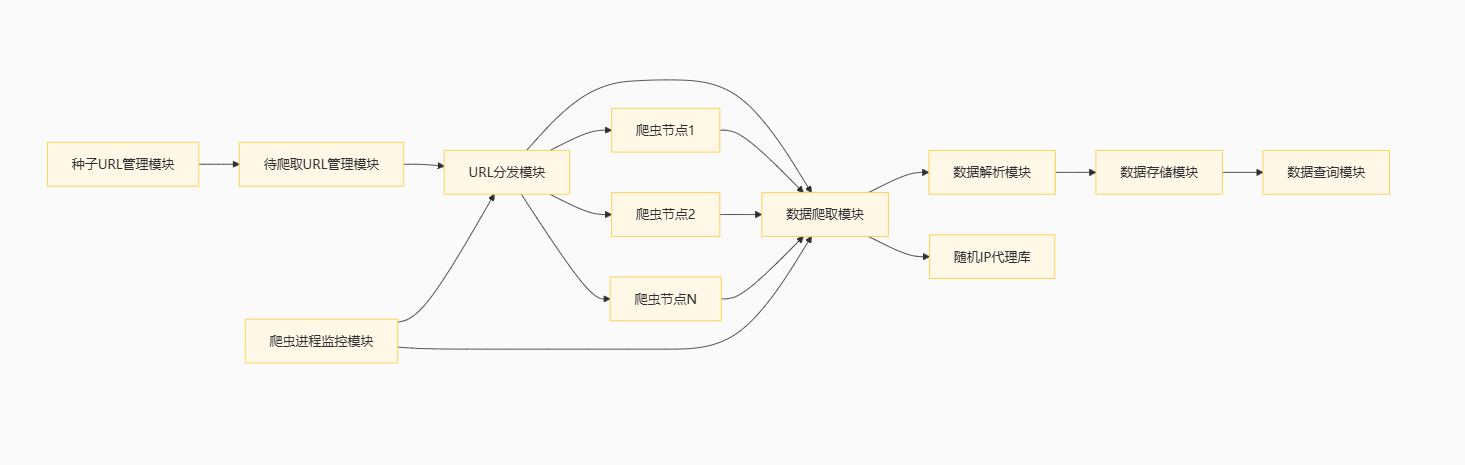
1. **项目分工**

个人完成，由MarsCode进行辅助编写

## **三、系统设计**

### **1. 系统架构图及说明**

#### **系统架构图**



**说明（java）**

* **种子URL管理模块：**负责收集和管理初始的种子URL。
* **待爬取URL管理模块：**基于流式计算框架（如Apache Flink）管理所有待爬取的URL，实现URL的高效存储、去重和调度。
* **URL分发模块：**按照一定策略将待爬取的URL分发给不同的爬虫节点，支持多种分发策略，如轮询、随机等，并与爬虫节点保持通信，获取任务执行状态。
* **数据爬取模块：**负责进行网页数据的爬取下载，支持多线程或异步爬取，提高爬取速度。
* **数据解析模块：**针对爬取的网页数据进行清洗、文本分词等处理，提取有价值的信息，如标题、正文、链接等。
* **数据存储模块：**将解析好的数据按照一定格式（如JSON、XML）存储到数据库（如MongoDB）中，并构建索引，便于后续查询。
* **数据查询模块：**提供数据查询接口，支持根据关键词、URL等条件进行数据查询，返回相关结果。
* **随机IP代理库：**为爬虫提供随机IP代理，避免因频繁访问同一IP而被目标网站封禁，提高爬虫的反反爬虫能力。
* **爬虫进程监控模块：**监控爬虫运行时的状态信息，如爬取速度、已爬取URL数量、错误率等，及时发现并处理异常情况。

**2. 系统核心模块说明**

**种子URL管理模块**

负责收集和管理初始的种子URL，提供添加、删除、查询种子URL的功能。

package com.example.service;

import com.example.model.CrawlTask;

import org.springframework.stereotype.Service;

import java.util.HashSet;

import java.util.Set;

@Service

public class UrlManager {

private final Set<String> seenUrls = new HashSet<>();

public boolean isNewUrl(String url) {

return seenUrls.add(url);

}

public CrawlTask getNextUrl() {

// 从待爬取URL队列中获取下一个URL

// 这里可以使用RabbitMQ的队列来实现

return null;

}

public void addUrl(String url) {

if (isNewUrl(url)) {

// 将新URL添加到待爬取URL队列中

// 这里可以使用RabbitMQ的队列来实现

}

}

}

**URL分发模块**

按照一定策略将待爬取的URL分发给不同的爬虫节点，支持多种分发策略，如轮询、随机等，并与爬虫节点保持通信，获取任务执行状态。

package com.example.service;

import com.example.config.RabbitMQConfig;

import com.example.model.CrawlTask;

import org.springframework.amqp.rabbit.core.RabbitTemplate;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.concurrent.atomic.AtomicInteger;

@Service

public class UrlDistributor {

@Autowired

private RabbitTemplate rabbitTemplate;

@Autowired

private UrlManager urlManager;

private final AtomicInteger counter = new AtomicInteger(0);

public void distributeUrl(CrawlTask task) {

if (urlManager.isNewUrl(task.getUrl())) {

int nodeIndex = selectNodeBasedOnLoad();

String queueName = "crawl\_tasks\_" + nodeIndex;

rabbitTemplate.convertAndSend(queueName, task, m -> {

m.getMessageProperties().setPriority(task.getPriority());

return m;

});

}

}

private int selectNodeBasedOnLoad() {

// 根据系统负载和任务队列的长度选择节点

// 这里可以使用RabbitMQ的API获取队列长度

// 选择负载最小的节点

return counter.getAndIncrement() % 3; // 假设有3个爬虫节点

}

}

**数据爬取模块**

负责进行网页数据的爬取下载，支持多线程或异步爬取，提高爬取速度。

package com.example.service;

import com.example.model.CrawlTask;

import com.example.model.CrawlResult;

import com.example.util.JsoupUtil;

import com.example.util.RobotsTxtParser;

import org.springframework.stereotype.Service;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

@Service

public class WebCrawler {

private final ExecutorService executorService = Executors.newFixedThreadPool(10);

public CrawlResult crawl(CrawlTask task) {

if (!RobotsTxtParser.isAllowed(task.getUrl())) {

return null; // 如果不允许爬取，返回null

}

return executorService.submit(() -> {

try {

String content = JsoupUtil.fetchContent(task.getUrl());

return new CrawlResult(task.getUrl(), content);

} catch (Exception e) {

// 处理异常情况

// 如网页HTML编码不规范、爬虫陷阱、被爬取服务器宕机等

e.printStackTrace();

return null;

}

}).join();

}

}

**数据解析模块**

针对爬取的网页数据进行清洗、文本分词等处理，提取有价值的信息，如标题、正文、链接等。

package com.example.service;

import com.example.model.CrawlResult;

import org.jsoup.Jsoup;

import org.jsoup.nodes.Document;

import org.jsoup.nodes.Element;

import org.jsoup.select.Elements;

import org.springframework.stereotype.Service;

import java.util.HashMap;

import java.util.Map;

@Service

public class DataParser {

public Map<String, String> parse(CrawlResult result) {

Document doc = Jsoup.parse(result.getContent());

Map<String, String> data = new HashMap<>();

data.put("title", doc.title());

// 根据不同的网页结构进行解析

if (result.getUrl().contains("example.com")) {

Elements links = doc.select("a[href]");

for (Element link : links) {

data.put(link.attr("abs:href"), link.text());

}

} else if (result.getUrl().contains("another-example.com")) {

// 其他解析逻辑

}

return data;

}

}

**数据存储模块**

将解析好的数据按照一定格式（如JSON、XML）存储到数据库（如MongoDB）中，并构建索引，便于后续查询。

package com.example.service;

import com.example.model.CrawlResult;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

import org.bson.Document;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class DataStorage {

@Autowired

private MongoDatabase mongoDatabase;

public void store(CrawlResult result) {

MongoCollection<Document> collection = mongoDatabase.getCollection("crawl\_results");

Document document = new Document("url", result.getUrl())

.append("content", result.getContent());

collection.insertOne(document);

}

}

**数据查询模块**

提供数据查询接口，支持根据关键词、URL等条件进行数据查询，返回相关结果。

package com.example.service;

import com.example.model.CrawlResult;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

import org.bson.Document;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

import java.util.List;

@Service

public class DataQueryService {

@Autowired

private MongoDatabase mongoDatabase;

public List<CrawlResult> query(String keyword) {

MongoCollection<Document> collection = mongoDatabase.getCollection("crawl\_results");

List<CrawlResult> results = new ArrayList<>();

for (Document doc : collection.find()) {

String content = doc.getString("content");

if (content.contains(keyword)) {

results.add(new CrawlResult(doc.getString("url"), content));

}

}

return results;

}

}

**3. 系统各模块间交互逻辑说明**

**种子URL管理模块：**负责收集和管理初始的种子URL，提供添加、删除、查询种子URL的功能。

**待爬取URL管理模块：**从种子URL管理模块获取初始URL，管理所有待爬取的URL，去重，并将URL发送到URL分发模块。

**URL分发模块：**从待爬取URL管理模块获取URL，根据分发策略将URL分发给不同的爬虫节点，并与爬虫节点保持通信，获取任务执行状态。

**数据爬取模块：**接收URL分发模块分发的URL，进行网页数据爬取，并将爬取结果发送到数据解析模块。

**数据解析模块：**解析爬取的网页数据，提取有价值的信息，并将解析结果发送到数据存储模块。

**数据存储模块：**接收解析后的数据，存储到数据库中，并构建索引。

**数据查询模块：**提供查询接口，从数据库中检索数据，返回相关结果。

**随机IP代理库：**为数据爬取模块提供随机IP代理，避免因频繁访问同一IP而被目标网站封禁。

**爬虫进程监控模块：**监控爬虫的运行状态，包括爬取速度、已爬取URL数量、错误率等，及时发现并处理异常情况。

**4. 系统运行流程说明**

**启动系统：**运行 CrawlerApplication 类，启动Spring Boot应用。

**添加种子URL：**通过API或管理界面添加初始的种子URL到种子URL管理模块。

**URL分发：**种子URL管理模块将初始URL发送到待爬取URL管理模块，待爬取URL管理模块去重后将URL发送到URL分发模块。URL分发模块根据分发策略将URL分发给不同的爬虫节点。

**数据爬取：**爬虫节点接收URL分发模块分发的URL，进行网页数据爬取，并将爬取结果发送到数据解析模块。

**数据解析：**数据解析模块解析爬取的网页数据，提取有价值的信息，并将解析结果发送到数据存储模块。

**数据存储：**数据存储模块接收解析后的数据，存储到数据库中，并构建索引。

**数据查询：**用户通过API或管理界面查询数据，数据查询模块从数据库中检索数据，返回相关结果。

**监控与优化：**爬虫进程监控模块监控爬虫的运行状态，包括爬取速度、已爬取URL数量、错误率等，及时发现并处理异常情况。系统根据运行情况自动调整任务分发策略，优化性能。

**5. 系统中涉及的分布式思想**

**URL分发模块：**采用分布式思想，通过RabbitMQ实现任务的分布式分发，确保多个爬虫节点能够高效、公平地获取待爬取URL。

**数据爬取模块：**利用Java的多线程库（如ThreadPoolExecutor）或并发框架（如Akka）实现多进程并发爬取，提高爬取效率。

**数据解析模块：**采用并行计算框架（如Apache Spark）或Java并发包中的并行流（Parallel Streams）实现数据的并行解析，加快解析速度。

**数据存储模块：**使用MongoDB等分布式数据库，支持高并发写入和查询，提高数据存储和检索的效率。

**6. 高性能、高可用、高可扩展性、高可配置性优化**

**高性能**

**多线程爬取：**使用Java的ThreadPoolExecutor实现多线程爬取，提高爬取效率。

**并行解析：**使用Java并发包中的并行流（Parallel Streams）实现数据的并行解析，加快解析速度。

**分布式存储：**使用MongoDB等分布式数据库，支持高并发写入和查询，提高数据存储和检索的效率。

**高可用**

**任务重试机制：**当爬取任务失败时，自动重试，确保任务能够成功完成。

**故障转移：**使用RabbitMQ的高可用配置，确保消息队列的高可用性。

**健康检查**：定期检查爬虫节点的健康状态，及时发现并处理异常情况。

**高可扩展性**

**爬虫节点动态扩展：**支持动态添加和移除爬虫节点，确保系统能够自动调整任务分发策略。

**URL分发策略可扩展：**提供策略接口或抽象类，方便开发者根据不同的应用场景实现新的URL分发策略。

**数据解析策略可扩展：**采用策略模式，支持针对不同解析场景实现多样化的解析策略。

**高可配置性**

**配置文件：**使用application.properties或application.yml文件配置系统参数，如数据库连接信息、RabbitMQ连接信息等。

**环境变量：**支持通过环境变量配置系统参数，方便在不同环境下运行。

**动态配置：**支持运行时动态调整系统配置，如爬取速度、并发爬虫数量等。

1. **测试结果**

还未进行测试

**四、项目代码仓库地址**

项目代码已托管在GitHub上，地址如下：