[hankyung_scraper.py]

```
from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from webdriver_manager.chrome import ChromeDriverManager

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected_conditions as EC

from bs4 import BeautifulSoup

import logging

from datetime import datetime


class HanKyungScraper:

    def __init__(self, start_date, end_date):

        self.start_date = datetime.strptime(start_date, "%Y%m%d")

        self.end_date = datetime.strptime(end_date, "%Y%m%d")

        options = Options()

        options.add_argument("--headless")

        options.add_argument("--disable-gpu")

        options.add_argument("--no-sandbox")

        options.add_argument("--disable-dev-shm-usage")

        self.driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)

        logging.info("WebDriver initialized.")
```

* **__init__ 메소드**:

start_date와 end_date 문자열을 datetime 객체로 변환, .Selenium WebDriver 옵션을 설정하고 WebDriver를 초기화, WebDriver가 초기화되었음을 로그에 기록

```
    def fetch_page(self):

        try:

            url = "https://www.hankyung.com/all-news/"

            logging.info(f"Fetching URL: {url}")

            self.driver.get(url)

            WebDriverWait(self.driver, 10).until(EC.element_to_be_clickable((By.CSS_SELECTOR, "a.nav-link[data-menu-id='economy']"))).click()

            WebDriverWait(self.driver, 10).until(EC.presence_of_element_located((By.CSS_SELECTOR, "div.day-wrap")))

            return self.driver.page_source

        except Exception as e:

            logging.error(f"Error fetching and navigating the page: {e}")

            return None
```

*** fetch_page 메소드**:

한경 "전체 뉴스" 페이지로 이동하고 "경제" 섹션 링크가 클릭 가능할 때까지 기다렸다가 클릭(인데 작동이 안됨)

페이지가 로드되었음을 나타내는 요소(div.day-wrap)가 나타날 때까지 기다리고, 로드된 페이지의 소스(HTML 콘텐츠)를 반환합니다.

가져오기 및 탐색 과정에서 문제가 발생하면 오류 기록

```
    def parse_articles(self, page_source):
        soup = BeautifulSoup(page_source, 'html.parser')
        articles = []
        for article in soup.select("div.day-wrap"):
            date_element = article.select_one("strong.txt-date")
            if date_element:
                article_date = datetime.strptime(date_element.text.strip(), "%Y.%m.%d")
                if self.start_date <= article_date <= self.end_date:
                    for item in article.select("li[data-aid]"):
                        title_element = item.select_one("h3.news-tit > a")
                        text_element = item.select_one("p.lead")
                        edit_date_element = item.select_one(".datetime .txt-date")  # Selector for edited date

                        if title_element and text_element:
                            title = title_element.text.strip()
                            href = title_element.get("href")
                            date = date_element.text.strip()
                            article_text = text_element.text.strip()
                            date_edit = edit_date_element.text.strip() if edit_date_element else "No edit date"  # Handle cases where no edit date is provided

                            articles.append({
                                "date": date,
                                "date_edit": date_edit,
                                "href": href,
                                "title": title,
                                "article": article_text
                            })
        return articles
```

parse_articles 메소드:

BeautifulSoup을 사용하여 페이지의 HTML 파싱, 기사를 저장할 빈 리스트를 초기화, div.day-wrap 클래스를 가진 요소를 반복하여 각 기사를 찾습니다.

기사의 발행 날짜를 추출, 기사 날짜가 지정된 범위(start_date에서 end_date) 내에 있으면 각 기사를 추가로 처리:

제목, 링크, 기사 본문 및 편집 날짜(있는 경우)를 추출하고 기사 데이터를 articles 리스트에 추가.

```
    def get_articles(self):
        page_source = self.fetch_page()
        all_articles = []
        if page_source:
            all_articles = self.parse_articles(page_source)
            logging.info(f"Parsed {len(all_articles)} articles from the economy section")
        self.driver.quit()
        logging.info(f"Scraping completed. Total articles fetched: {len(all_articles)}")
        return all_articles
```

*** get_articles 메소드**:

fetch_page를 사용하여 페이지 소스를 가져오기

페이지 소스를 성공적으로 가져오면 parse_articles를 사용하여 기사를 파싱.

파싱된 기사 수를 로그에 기록.

WebDriver를 종료.

수집된 총 기사 수를 로그에 기록하고 기사 목록을 반환.

[main.py]

```
from argparse import ArgumentParser
from datetime import datetime
from hankyung_scraper import HanKyungScraper
import json
import logging

def create_parser() -> ArgumentParser:
    today = datetime.today().strftime("%Y%m%d")

    parser = ArgumentParser()
    parser.add_argument("-s", "--start_date", type=str, required=True, help="example: 20240504")
    parser.add_argument("-e", "--end_date", type=str, default=today, help=f"example: {today}")
    parser.add_argument("-o", "--output", type=str, default="output.json", help="output json file path")
    return parser
```

create_parser 함수

-s, -e 입력된 날짜 받아오고 -o를 통해서 result.json으로 크롤링한 데이터 내보내기

```
if __name__ == "__main__":
    parser = create_parser()
    args = parser.parse_args()

    logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

    scraper = HanKyungScraper(args.start_date, args.end_date)
    articles = scraper.get_articles()

    with open(args.output, "w", encoding="utf-8") as f:
        json.dump(articles, f, ensure_ascii=False, indent=4)

    logging.info(f"Data saved to {args.output}")
```

main부분 → 파서 생성, 인자 파싱, 로깅 설정...

* 미완성 코드... 추후 수정하여 업로드

[아직 구현하지 못한 부분]

1. '경제' 카테고리에 들어가서 기사 크롤링하는 부분

2. '더보기' 버튼을 누르는 메서드를 통해서 더 많은 기사 로딩

3. 수정 날짜 크롤링이 아직 구현 안됨