Chapter 3-2: 새로운 블록 조립하기 (JobKorea 크롤러 구현)

Prologue: 설계도를 현실로 만드는 즐거움

지난 챕터에서 우리는 `BaseCrawler`라는 견고한 '설계도'를 만들었습니다. 이제 그 설계도가 얼마나 강력한지 직접 증명해볼 시간입니다. 이번 챕터에서는 우리가 직접 분석하여 찾아낸 잡코리아의 '주소(선택자)'를 이용해, `JobKoreaCrawler`라는 새로운 크롤러 블록을 조립할 것입니다.

이 과정을 통해 신호용 님은 우리의 아키텍처가 얼마나 쉽게 새로운 기능을 받아들일 수 있는지, 그리고 **'잘된 설계'가 어떻게 미래의 작업을 즐겁게 만드는지** 직접 체감하게 될 것입니다.

1. 학습 목표 (Objectives)

- 추상 클래스를 상속받아 새로운 기능을 구체화할 수 있다.
- 직접 분석한 CSS 선택자를 코드에 적용하여 원하는 데이터를 정확히 추출할 수 있다.
- 기존 시스템의 코드를 거의 변경하지 않고도 새로운 기능을 원활하게 통합할 수 있다.

---

2. 핵심 개념 (Core Concepts)

2.1 다형성 (Polymorphism): "사용법은 하나, 동작은 각양각색"

다형성은 객체 지향 프로그래밍의 '꽃'이라 불리는 개념입니다. 어려워 보이지만 아주 간단합니다. **"같은 모양의 버튼을 눌렀는데, 기계마다 다른 일을 하는 것"**을 의미합니다.

- **우리의 프로젝트에서:** `main.py`는 `WantedCrawler`든 `JobKoreaCrawler`든 상관없이, 똑같이 `crawler.crawl()`이라는 '시작 버튼'을 누르기만 하면 됩니다. `main.py`는 각 크롤러의 복잡한 내부 사정을 전혀 알 필요가 없습니다. 버튼을 누르면 `WantedCrawler`는 원티드 방식으로, `JobKoreaCrawler`는 잡코리아 방식으로 **알아서** 동작합니다. 이것이 바로 다형성이며, 우리가 `ABC`를 통해 얻는 가장 큰 이점입니다.

2.2 상대 경로 (Relative Path) 다루기

우리가 잡코리아에서 찾은 링크(`href`)는 `/Recruit/GI_Read/...` 처럼 `/`로 시작합니다. 이것은 '상대 경로'라고 불리며, "현재 웹사이트 주소 뒤에 이 경로를 덧붙여라"라는 의미입니다. 따라서 우리는 이 상대 경로 앞에 기본 URL(`https://www.jobkorea.co.kr`)을 직접 합쳐주어 완전한 주소로 만들어야 합니다.

[Mission 10] `JobKoreaCrawler` 구현 및 통합

이제 잡코리아 크롤러를 실제로 만들어 시스템에 연결해봅시다.

> **Step 1: `jobkorea_crawler.py` 파일 생성**
>
> `web-crowler/crawlers/` 폴더 안에, `wanted_crawler.py`와 같은 위치에 `jobkorea_crawler.py` 라는 새 파이썬 파일을 만듭니다.

```
web-crowler/
├── crawlers/
│   ├── __init__.py
│   ├── base_crawler.py
│   ├── wanted_crawler.py
│   └── jobkorea_crawler.py  # <- 새로 추가!
└── ...
```

Step 2: `jobkorea_crawler.py` 코드 작성

새로 만든 파일에 아래 코드를 작성합니다. 우리가 함께 분석한 선택자들이 어떻게 사용되는지 집중해서 봐주세요.

```python
# crawlers/jobkorea_crawler.py

import time
from bs4 import BeautifulSoup
from .base_crawler import BaseCrawler

class JobKoreaCrawler(BaseCrawler):
    """잡코리아 사이트 크롤러"""

    def __init__(self):
        super().__init__("https://www.jobkorea.co.kr")

    def crawl(self, keyword: str = '백엔드'):
        """
        잡코리아에서 주어진 키워드로 채용 정보를 크롤링합니다.
        """
        search_url = f"{self.base_url}/Search/?stext={keyword}"
        self.driver.get(search_url)
        time.sleep(3) # 페이지 로딩 대기

        # 현재 페이지의 HTML을 BeautifulSoup으로 파싱
        soup = BeautifulSoup(self.driver.page_source, 'lxml')

        job_data = []
        # 우리가 찾은 안정적인 기준점: 'CardJob' 컴포넌트
        job_cards = soup.select('div[data-sentry-component="CardJob"]')

        print(f"잡코리아에서 {len(job_cards)}개의 공고 카드를 찾았습니다. 데이터 추출을 시작합니다.")

        for card in job_cards:
            try:
                # 제목과 링크는 첫 번째 주요 링크에서 추출
                title_link_tag = card.select_one('a.title.dev_view') # title 클래스를 가진 a 태그
                if not title_link_tag: # 혹시 title 클래스가 없는 구조일 경우 대비
                    title_link_tag =
```

```
card.select('a[href^="/Recruit/GI_Read"]')[0]

                title = title_link_tag.text.strip()
                relative_link = title_link_tag['href']
                link = self.base_url + relative_link

                # 회사명은 name 클래스를 가진 a 태그에서 추출
                company_tag = card.select_one('a.name.dev_view')
                company = company_tag.text.strip()

                job_data.append({
                    "title": title,
                    "company": company,
                    "link": link,
                    "source": "JobKorea"
                })
            except (AttributeError, IndexError) as e:
                # AttributeError: .text 또는 ['href'] 접근 시 태그가 None일 때 발생
                # IndexError: 리스트 인덱싱 실패 시 발생 (e.g., 광고 카드)
                print(f"  [오류] 특정 공고 카드 처리 중 문제 발생: {e} -> 건너뜁니다.")
                continue # 오류 발생 시 해당 카드만 건너뛰고 계속 진행

        print(f"잡코리아에서 총 {len(job_data)}개의 유효한 공고를 추출했습니다.")
        return job_data
```

> ⚡ **시니어의 팁:** `try-except` 구문을 추가했습니다. 크롤링 시 일부 공고 카드는 우리가 예상치 못한 다른 구조(e.g., 광고)를 가질 수 있습니다. `try-except`는 이런 예외적인 카드 하나 때문에 전체 크롤러가 멈추는 것을 방지하고, "문제가 있는 카드는 건너뛰고 다음 작업을 계속해라" 라고 지시하는 매우 중요한 방어 코드입니다.

Step 3: `main.py`에 새로운 크롤러 통합하기

이제 `main.py` 파일을 열어, 우리가 만든 `JobKoreaCrawler`를 시스템에 연결합니다. **단 두 줄만 추가하면 됩니다!**

```
# main.py

# ... (기존 import 구문들) ...
from crawlers.wanted_crawler import WantedCrawler
from crawlers.jobkorea_crawler import JobKoreaCrawler # <- 1. 잡코리아 크롤러 임포트

# ... (Notion 설정 부분) ...

# 2. 크롤러 실행
all_jobs = []

# 실행할 크롤러들을 리스트에 담습니다.
```

```
crawlers_to_run = [
    WantedCrawler(),
    JobKoreaCrawler() # <- 2. 리스트에 잡코리아 크롤러 추가
]

for crawler in crawlers_to_run:
    try:
        # 어떤 크롤러든 사용법은 동일합니다 (다형성!)
        print(f"--- {type(crawler).__name__} 크롤링 시작 ---")
        crawled_jobs = crawler.crawl(keyword='백엔드')
        all_jobs.extend(crawled_jobs)
        print(f"--- {type(crawler).__name__} 크롤링 완료 ---")
    except Exception as e:
        print(f"{type(crawler).__name__} 실행 중 오류 발생: {e}")
    finally:
        crawler.close_driver()

# 3. Notion에 저장 (기존과 동일)
# ...
```

이제 모든 준비가 끝났습니다. 코드를 수정한 뒤, 지난번처럼 **'검증 가이드'**에 따라 로컬에서 `main.py`와 `test_crawler.py`(잡코리아용으로 수정해서)를 실행하여 원티드와 잡코리아의 공고가 모두 잘 수집되는지 확인해보십시오.

이 미션을 완료하면, 신호용 님의 시스템은 더 이상 '단일 사이트 크롤러'가 아닌, 여러 소스의 데이터를 수집할 수 있는 **'멀티-채널 데이터 수집기'**로 진화하게 됩니다. 결과를 기대하겠습니다.