Chapter 3-4: 더 깊은 바다로, 페이지네이션 (Pagination) 정복

Prologue: 숨겨진 보물 지도를 펼치다

지금까지 우리는 해변에 밀려온 조개(첫 페이지의 데이터)만 주워왔습니다. 하지만 진짜 보물은 저 깊은 바닷속, 여러 페이지에 걸쳐 잠들어 있습니다. 이번 챕터에서는 **페이지네이션(Pagination)**, 즉 여러 페이지를 순서대로 항해하며 데이터를 수집하는 기술을 구현할 것입니다.

이 미션을 완료하면, 신호용 님의 크롤러는 비로소 '수박 겉핥기' 수준을 넘어, 각 채용 사이트가 가진 **데이터의 99% 이상을 수집할 수 있는 진정한 '탐사선'**으로 거듭나게 됩니다.

1. 학습 목표 (Objectives)

- 웹사이트의 페이지 이동 방식을 분석하고 규칙을 파악할 수 있다.
- URL 파라미터를 변경하는 방식으로 여러 페이지를 순차적으로 크롤링할 수 있다.
- 기존 클래스 구조를 수정하여, 크롤링할 페이지 수를 동적으로 제어할 수 있다.

2. 핵심 개념 (Core Concepts)

2.1 페이지네이션(Pagination)의 두 가지 유형

1. **URL 파라미터 기반 (e.g., 잡코리아):**
    - 가장 간단하고 크롤링하기 쉬운 방식입니다. URL 주소 끝에 `?page=1`, `&Page_No=2` 와 같은 파라미터를 변경하여 다음 페이지로 이동합니다. 우리는 이 숫자만 `for` 루프를 통해 1부터 5까지 증가시키면, 아주 쉽게 여러 페이지를 방문할 수 있습니다.
2. **무한 스크롤 (Infinite Scroll) (e.g., 원티드):**
    - 사용자가 페이지 맨 아래로 스크롤하면, JavaScript가 새로운 데이터를 '뒤에 덧붙이는' 방식입니다. 별도의 페이지 URL이 존재하지 않습니다. 이 방식은 우리가 이미 `while` 루프와 `window.scrollTo`를 통해 구현해 놓았으므로, 사실상 원티드는 이미 페이지네이션이 구현된 상태나 마찬가지입니다.

2.2 유연한 설계: 크롤링 깊이 제어하기

"무조건 100페이지까지 다 긁어와!" 라고 코드를 짜는 것은 비효율적입니다. 테스트할 때는 2페이지만, 실제 운영에서는 10페이지만 긁어오고 싶을 수 있습니다. 따라서 크롤링할 페이지 수를 **함수의 파라미터**로 만들어, 외부에서 쉽게 제어할 수 있도록 설계하는 것이 중요합니다.

[Mission 12] 크롤러에 페이지네이션 엔진 장착하기

이제 우리의 크롤러들이 더 깊이 잠수할 수 있도록 코드를 수정해봅시다.

> **Step 1: `base_crawler.py`의 '설계도' 변경**

먼저, 모든 크롤러가 '페이지 수'라는 개념을 알아들을 수 있도록 `BaseCrawler`의 `crawl` 메서드에 `pages_to_crawl` 이라는 파라미터를 추가합니다.

```
# crawlers/base_crawler.py (수정)

# ... (다른 코드는 동일) ...
    @abstractmethod
    def crawl(self, keyword: str, pages_to_crawl: int = 1): # <-
pages_to_crawl 파라미터 추가
        """
        크롤링 프로세스를 시작하는 메인 메서드. (의무 조항)
        ...
        """
        pass
# ...
```

Step 2: `jobkorea_crawler.py`에 페이지네이션 로직 구현

잡코리아 크롤러가 `for` 루프를 돌며 여러 페이지를 방문하도록 코드를 대폭 수정합니다.

```
# crawlers/jobkorea_crawler.py (수정)

# ... (import 구문) ...

class JobKoreaCrawler(BaseCrawler):
    # ... (__init__ 메서드는 동일) ...

    def crawl(self, keyword: str = '백엔드', pages_to_crawl: int = 3): # <-
pages_to_crawl 구현
        """
        잡코리아에서 주어진 키워드로 지정된 페이지 수만큼 채용 정보를 크롤링합니
다.
        """
        print(f"잡코리아에서 '{keyword}' 키워드로 {pages_to_crawl} 페이지까지
크롤링을 시작합니다.")
        all_job_data = []

        # 1부터 pages_to_crawl 숫자까지 루프를 돕니다.
        for page in range(1, pages_to_crawl + 1):
            # URL에 페이지 번호를 추가합니다.
            search_url = f"{self.base_url}/Search/?stext={keyword}&Page_No=
{page}"
            self.driver.get(search_url)
            time.sleep(3)

            print(f"  - {page} 페이지 처리 중...")
            soup = BeautifulSoup(self.driver.page_source, 'lxml')
            job_cards = soup.select('div[data-sentry-component="CardJob"]')

            if not job_cards: # 페이지에 공고 카드가 더 이상 없으면
```

```
                print(f"  - {page} 페이지에 더 이상 공고가 없어 크롤링을 중단합
니다.")
                break # 루프를 빠져나갑니다.

            for card in job_cards:
                # ... (이전의 카드에서 데이터 추출하는 로직은 동일) ...
                try:
                    title_span =
card.select_one('span[class*="Typography_variant_size18"]')
                    title = title_span.text.strip()
                    link_tag = title_span.find_parent('a')
                    relative_link = link_tag['href']
                    link = self.base_url + relative_link
                    company_span =
card.select_one('span[class*="Typography_variant_size16"]')
                    company = company_span.text.strip()

                    all_job_data.append({
                        "title": title,
                        "company": company,
                        "link": link,
                        "source": "JobKorea"
                    })
                except Exception:
                    continue

        print(f"잡코리아에서 총 {len(all_job_data)}개의 유효한 공고를 추출했습니
다.")
        return all_job_data
```

Step 3: `wanted_crawler.py` **최종 최적화**

신호용 님의 발견을 바탕으로, `WantedCrawler`가 불필요한 클릭 없이 가장 빠르고 안정적인 경로로 데이터에 접근하도록 코드를 업그레이드합니다.

```
# crawlers/wanted_crawler.py (최종 최적화 버전)

import time
from bs4 import BeautifulSoup
from .base_crawler import BaseCrawler

class WantedCrawler(BaseCrawler):
    # ... (__init__ 메서드는 동일) ...

    def crawl(self, keyword: str = '백엔드', pages_to_crawl: int = 1):
        print(f"원티드에서 '{keyword}' 키워드로 크롤링을 시작합니다...")

        # 신호용 님의 발견: URL 직접 접근으로 안정성 및 속도 향상
        # '포지션' 탭의 URL로 바로 접근하여 불필요한 클릭 과정을 생략합니다.
        target_url = f"{self.base_url}/search?query={keyword}&tab=position"
        self.driver.get(target_url)
```

```
        time.sleep(3) # 페이지 전체 로딩 대기

        # 무한 스크롤로 데이터 수집 (기존 로직과 동일)
        print("  - 무한 스크롤을 시작합니다.")
        last_height = self.driver.execute_script('return
document.body.scrollHeight')
        while True:
            self.driver.execute_script('window.scrollTo(0,
document.body.scrollHeight);')
            time.sleep(2)
            new_height = self.driver.execute_script('return
document.body.scrollHeight')
            if new_height == last_height:
                print("  - 스크롤이 페이지 끝에 도달했습니다.")
                break
            last_height = new_height

        # 데이터 추출 로직 (이전과 동일)
        job_data = []
        soup = BeautifulSoup(self.driver.page_source, 'lxml')
        job_cards = soup.select("div[role='listitem'] a")

        for card in job_cards:
            try:
                title =
card.select_one("strong[class*='JobCard_title']").text
                company_name =
card.select_one("span[class*='CompanyName']").text
                link = "https://www.wanted.co.kr" + card['href']
                job_data.append({"title": title, "company": company_name,
"link": link, "source": "Wanted"})
            except Exception:
                continue # 광고 등 다른 구조의 카드는 건너뜀

        print(f"원티드에서 총 {len(job_data)}개의 공고를 찾았습니다.")
        return job_data
```

Step 4: `main.py`에서 크롤링 깊이 지시하기

이제 `main.py`에서 각 크롤러에게 "몇 페이지까지 긁어와!" 라고 지시할 수 있습니다.

```
# main.py (수정)

# ... (이전 코드) ...
for crawler in crawlers_to_run:
    try:
        print(f"--- {type(crawler).__name__} 크롤링 시작 ---")
        # crawl 메서드에 pages_to_crawl 인자를 전달합니다.
        # 테스트 시에는 2, 실제 운영 시에는 5~10 정도로 설정할 수 있습니다.
        crawled_jobs = crawler.crawl(keyword='백엔드', pages_to_crawl=2)
        all_jobs.extend(crawled_jobs)
        # ... (이후 코드는 동일) ...
```

다음 행동 계획 (Next Action)

1. ☐ 위 가이드에 따라 `base_crawler.py`, `jobkorea_crawler.py`, `wanted_crawler.py`, `main.py`를 모두 수정합니다.
2. ☐ `main.py`를 로컬에서 실행하여 잡코리아 크롤러가 여러 페이지를 순회하며 데이터를 더 많이 가져오는지 확인합니다. 터미널 로그에 "1 페이지 처리 중...", "2 페이지 처리 중..." 이라는 메시지가 순서대로 뜨는지 확인해주세요.
3. ☐ Notion DB에 더 많은 양의 데이터가, 중복 방지 로직에 따라 신규 공고만 정확히 저장되는지 최종 확인합니다.

이 미션을 성공적으로 마치면, 우리는 비로소 '데이터의 양'이라는 첫 번째 관문을 통과하게 됩니다. 결과를 기다리겠습니다.