Chapter 4-2: 마지막 정보 조각, 상세 페이지 본문 수집

Prologue: 목록 너머의 세상으로

지금까지 우리는 채용 공고의 '겉모습(제목, 회사명)'만을 수집해왔습니다. 하지만 진짜 핵심 정보는 상세 페이지 안에 담긴 '본문(Job Description)'에 있습니다. Gemini가 분석할 '재료'가 바로 이 본문입니다.

이번 챕터에서는 기존 크롤러들을 업그레이드하여, 목록 페이지에서 수집한 링크를 타고 **상세 페이지에 직접 방문**한 뒤, 그곳의 **핵심 텍스트 내용을 통째로** 긁어오는 기능을 추가할 것입니다.

1. 학습 목표 (Objectives)

- 기존 크롤러 클래스에 새로운 메서드를 추가하여 기능을 확장할 수 있다.
- Selenium의 `get()` 메서드를 재사용하여 여러 페이지를 순차적으로 방문할 수 있다.
- 각기 다른 사이트의 상세 페이지 구조를 분석하여 본문 내용에 해당하는 선택자를 찾아낼 수 있다.

2. 핵심 개념 (Core Concepts)

2.1 2단계 크롤링 (Two-Phase Crawling)

- **What:** 지금까지는 1단계, 즉 '목록 페이지'만 크롤링했습니다. 이제부터는 2단계 크롤링을 수행합니다.
    1. **1단계:** 목록 페이지에서 모든 공고의 기본 정보(제목, 회사명, **상세 페이지 링크**)를 수집한다.
    2. **2단계:** 수집된 링크들을 하나씩 방문하여, 각 상세 페이지의 '본문'을 추가로 수집한다.
- **Why:** 모든 정보를 한 번에 처리하는 것보다 역할을 명확하게 나눌 수 있어 코드가 깔끔해지고, 특정 상세 페이지에서 오류가 발생하더라도 전체 목록 수집에 영향을 주지 않아 안정적입니다.

[Mission 17] 크롤러에 상세 정보 수집 기능 추가

Step 1: `base_crawler.py`에 새로운 '설계도' 추가

`BaseCrawler`에 "모든 크롤러는 상세 페이지 본문을 가져오는 기능을 가져야 한다"는 새로운 '의무 조항'(`@abstractmethod`)을 추가합니다.

```
# crawlers/base_crawler.py (수정)

# ... (기존 import 및 클래스 정의) ...
class BaseCrawler(ABC):
    # ... (__init__, _setup_driver, _random_sleep 등은 동일) ...

    @abstractmethod
    def crawl(self, keyword: str, pages_to_crawl: int = 1, sort_by: str =
'latest'):
        pass

    # ★★★★★ 새로운 추상 메서드 추가 ★★★★★
    @abstractmethod
    def get_job_description(self, url: str) -> str:
```

```
        """
        주어진 URL의 상세 페이지에 방문하여, 채용 공고의 본문 텍스트를 반환합니
다.
        자식 클래스는 이 메서드를 반드시 구현해야 합니다.
        """
        pass
    # ★★★★★★★★★★★★★★★★★★★★★★★★

    def close_driver(self):
        """드라이버를 안전하게 종료합니다."""
        if self.driver:
            self.driver.quit()
```

Step 2: `wanted_crawler.py`에 상세 정보 수집 기능 '구현'

`BaseCrawler`에 추가된 새로운 의무 조항을 `WantedCrawler`에서 실제로 구현합니다. `crawl` 메서드 아래에 새로운 메서드를 추가해주세요.

```
# crawlers/wanted_crawler.py (기능 추가)

# ... (crawl 메서드 구현은 동일하게 유지) ...

    # ★★★★★ 새로운 메서드 구현 ★★★★★
    def get_job_description(self, url: str) -> str:
        """원티드 상세 페이지의 본문 내용을 수집합니다."""
        try:
            self.driver.get(url)
            self._random_sleep() # 페이지 로딩을 위한 정중한 기다림
            soup = BeautifulSoup(self.driver.page_source, 'lxml')

            # 원티드는 상세 페이지 본문에 매우 안정적인 선택자를 사용합니다.
            content_div = soup.select_one('div[data-cy="job-detail-position-
content"]')

            # .text.strip()으로 깔끔한 텍스트만 추출
            return content_div.text.strip() if content_div else ""
        except Exception as e:
            print(f"  🕵 [상세 정보 수집 오류] {url} 처리 중 문제 발생: {e}")
            return "" # 오류 발생 시 빈 문자열 반환
    # ★★★★★★★★★★★★★★★★★★★★★★★★
```

Step 3: `main.py`의 심장부 개조

`main.py`의 실행 로직을 수정하여, 목록을 가져온 직후 바로 상세 페이지를 방문하여 본문 내용을 수집하고, 그 결과를 `job` 딕셔너리에 추가하도록 변경합니다.

```
# main.py (최종 수정)

# ... (import 구문: WantedCrawler, JobKoreaCrawler, SaraminCrawler,
```

```
analyze_job_posting 등) ...

# 2. 크롤러 실행
all_jobs = []
# 클래스 자체를 리스트에 담아, 필요할 때 객체를 생성하는 방식으로 변경
crawlers_to_run = [WantedCrawler, JobKoreaCrawler, SaraminCrawler]

for CrawlerClass in crawlers_to_run:
    crawler = CrawlerClass() # 루프 안에서 WebDriver를 가진 새 크롤러 객체 생성
    crawler_name = type(crawler).__name__
    print(f"--- {crawler_name} 크롤링 시작 ---")

    try:
        # 1단계: 목록 페이지에서 기본 정보 수집
        # (테스트를 위해 우선 1페이지만, 나중에 pages_to_crawl=3 등으로 늘리세
요)
        crawled_jobs = crawler.crawl(keyword='백엔드', pages_to_crawl=1)

        # 2단계: 각 공고의 상세 페이지에 방문하여 본문 수집
        print(f"  -> {len(crawled_jobs)}개 공고의 상세 정보 수집을 시작합니
다...")
        for i, job in enumerate(crawled_jobs):
            job_link = job['link']
            print(f"    ({i+1}/{len(crawled_jobs)}) {job_link[:50]}...") #
링크 앞부분만 출력
            # 상세 페이지 본문을 가져와 'description' 키에 저장
            job['description'] = crawler.get_job_description(job_link)

        all_jobs.extend(crawled_jobs)
        print(f"--- {crawler_name} 크롤링 완료 ---")

    except Exception as e:
        print(f"  [전체 크롤러 오류] {crawler_name} 실행 중 문제 발생:
{e}")
    finally:
        crawler.close_driver() # 사용이 끝난 크롤러는 항상 드라이버 종료

# 3. Notion 저장 및 Gemini 분석 (for 루프)
for i, job in enumerate(all_jobs):
    # ... (중복 확인 로직은 동일) ...

    # --- '신규' 공고일 경우 ---
    print(f" [{i+1}/{len(all_jobs)}] [신규] {title} -> Gemini 분석 시작...")

    # job 딕셔너리에서 'description' 키의 값을 가져옴
    job_description = job.get('description', "")

    if job_description:
        analysis_result = analyze_job_posting(job_description)
        print("--- Gemini 분석 결과 ---")
        print(analysis_result) # 우선 터미널에 출력
        print("------------------------")
    else:
        print("  - 본문 내용이 없어 Gemini 분석을 건너뜁니다.")
```

```
    # ... (Notion 저장 로직은 동일) ...
```

> **참고:** `JobKoreaCrawler`와 `SaraminCrawler`는 아직 `get_job_description` 메서드를 구현하지 않았기 때문에, 실행 시 에러가 발생할 것입니다. 우선은 `crawlers_to_run = [WantedCrawler]` 로 리스트를 수정하여 원티드만 테스트하는 것을 권장합니다.

다음 행동 계획 (Next Action)

1. ☐ `base_crawler.py`와 `wanted_crawler.py`를 위 가이드에 따라 수정합니다.
2. ☐ `main.py`의 크롤러 실행 로직을 **대대적으로 수정**합니다.
3. ☐ 테스트를 위해 `main.py`의 `crawlers_to_run` 리스트를 `[WantedCrawler]`로 잠시 변경합니다.
4. ☐ `main.py`를 실행하고, **터미널에 Gemini가 분석한 실제 채용 공고 요약 결과가 나타나는지** 확인합니다.