문서 로더

1. Document 객체 구조

LangChain에서 `Document` **객체**는 모든 데이터의 **기본 단위**이며,
Loader로 불러온 텍스트를 모델이 이해할 수 있는 **표준 구조**로 정리한 클래스이다.

📘 1. 기본 개념

`Document`는 하나의 "문서 조각(chunk)" 또는 "텍스트 단위"를 표현하는 객체이다.
즉, "어디서 왔는지(meta 정보)"와 "무슨 내용인지(content)"를 함께 저장한다.

```python
from langchain_core.documents import Document

doc = Document(
    page_content="이 문서는 LangChain의 Document 객체 설명입니다.",
    metadata={"source": "lecture_note", "page": 1}
)
```

주요 속성

속성명	자료형	설명
`page_content`	`str`	문서의 실제 텍스트 내용
`metadata`	`dict`	문서의 출처, 경로, 페이지 번호, 작성자 등 부가정보

🧱 2. 구조 예시

아래처럼 PDF나 CSV, 웹페이지를 읽으면 LangChain은 내부적으로 이런 Document 리스트를 반환한다.

python

```
[
  Document(
    page_content="1장. LangChain 개요...",
    metadata={"source": "data/guide.pdf", "page": 1}
  ),
  Document(
    page_content="2장. Document Loader의 종류...",
    metadata={"source": "data/guide.pdf", "page": 2}
  ),
]
```

즉, **하나의 파일**이 여러 **Document 조각들**로 나뉘어 저장될 수 있다.

⚙️ 3. 생성 방법

1) 직접 생성

```python
doc = Document(page_content="간단한 문서 예시", metadata={"type": "note"})
```

2) Loader를 통해 자동 생성

```python
from langchain_community.document_loaders import PyPDFLoader

loader = PyPDFLoader("report.pdf")
docs = loader.load()  # List[Document] 반환
```

3) Text Splitter로 나누기

긴 문서는 LLM이 한 번에 처리하기 어렵기 때문에 `TextSplitter`로 나눈다.

```python
from langchain_text_splitters import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
split_docs = splitter.split_documents(docs)
```

→ 이때 각 조각은 **Document 형태**를 그대로 유지한다.

🔍 4. metadata 활용

`metadata`는 나중에 **검색(RAG)**, **출처 표시**, **필터링** 등에 매우 유용하다.

예시:

```python
for d in docs:
    print(d.metadata["source"], "에서 추출한 텍스트 길이:", len(d.page_content))
```

또는 벡터DB에 저장할 때 메타데이터 필터링:

```python
vectorstore.similarity_search(query, filter={"source": "report.pdf"})
```

🧩 5. 실제 활용 흐름

단계	설명	LangChain 클래스
① Document 로드	Loader로 외부 문서 읽기	`PyPDFLoader`, `WebBaseLoader` 등
② Document 분할	긴 텍스트를 여러 chunk로 나누기	`RecursiveCharacterTextSplitter`
③ 임베딩 생성	각 Document를 벡터화	`OpenAIEmbeddings`, `UpstageEmbeddings`
④ 벡터 DB 저장	검색용 인덱스로 저장	`FAISS`, `pgvector`, `Chroma`
⑤ RAG 검색	사용자의 질문과 유사한 Document 검색	`similarity_search()`

6. Document의 핵심 요약

항목	설명
클래스 위치	`from langchain_core.documents import Document`
목적	텍스트와 메타데이터를 함께 저장하는 표준 구조
주요 필드	`page_content`, `metadata`
반환 형태	`List[Document]`
주요 활용	RAG, 요약, 검색, 필터링, 출처 표시 등

2. 문서로더(Document Loader)

LangChain의 **Document Loader**는 외부 문서(파일, 웹, 데이터베이스 등)를 읽어서 LangChain이 처리할 수 있는 **Document 객체(list[Document])** 형태로 변환하는 구성요소이다. 즉, "데이터 수집 단계"의 핵심 도구이다.

주요 Document Loader 종류

📂 1. 파일(File) 기반 Loader

Loader 이름	주요 기능	사용 예시
`TextLoader`	일반 텍스트 파일(`.txt`)을 읽음	단순 텍스트 데이터 로드
`CSVLoader`	CSV 파일을 `pandas` 없이 로드	구조화된 표 데이터 처리
`UnstructuredFileLoader`	다양한 파일 형식(txt, pdf, docx, html 등)을 자동 판별	여러 파일을 한번에 로드할 때
`PDFPlumberLoader` / `PyPDFLoader`	PDF 파일에서 텍스트를 추출	논문, 보고서 요약 등
`Docx2txtLoader`	Word 문서(`.docx`)에서 텍스트 추출	교육 자료나 회의록 로드
`JSONLoader`	JSON 파일의 특정 필드를 선택하여 Document로 변환	API 응답 또는 로그 파일 로드

🌐 2. 웹(Web) / 네트워크 기반 Loader

Loader 이름	주요 기능	사용 예시
`WebBaseLoader`	일반 웹페이지를 가져와 HTML → 텍스트 변환	뉴스, 블로그, 위키 문서 수집
`SitemapLoader`	사이트맵(XML)을 읽어 다수의 URL을 자동 크롤링	전체 웹사이트 콘텐츠 로드
`RecursiveUrlLoader`	링크를 따라가며 여러 페이지를 수집	위키형 문서 탐색용
`YoutubeLoader` / `YoutubeTranscriptLoader`	유튜브 자막을 텍스트로 변환	영상 요약, 강의 콘텐츠 분석
`UnstructuredURLLoader`	HTML, PDF 등 다양한 URL 파일 자동 판별	복합 URL 데이터 처리

🧮 3. 데이터베이스(DB) / 클라우드 기반 Loader

Loader 이름	주요 기능	사용 예시
`SQLDatabaseLoader`	SQL 쿼리 결과를 Document로 변환	MySQL, PostgreSQL 데이터 분석
`MongoDBLoader`	MongoDB 컬렉션 데이터를 Document로	NoSQL 데이터 요약
`NotionDBLoader`	Notion 페이지나 DB 데이터를 로드	사내 문서, 프로젝트 로그
`GoogleDriveLoader`	Google Drive 문서 자동 로드	클라우드 자료 수집
`GoogleDocsLoader`	Google Docs 문서 텍스트 추출	협업 문서 분석
`AirtableLoader`	Airtable 데이터베이스 로드	마케팅 데이터, CRM

💬 4. 특수 포맷 / 멀티미디어 Loader

Loader 이름	주요 기능	사용 예시
`ImageCaptionLoader`	이미지에서 캡션(텍스트) 추출	이미지 요약, 비주얼 콘텐츠 분석
`AudioTranscriptionLoader`	오디오 파일을 STT로 변환	음성 회의록 요약
`MarkdownLoader`	Markdown 문서를 읽어서 텍스트로 변환	GitHub 문서, 기술 블로그
`EPubLoader`	전자책(Epub) 텍스트 추출	책 요약
`HTMLLoader`	HTML 파일의 본문만 텍스트로 추출	웹 문서 분석용

🧰 5. 고급 / 범용 Loader

Loader 이름	주요 기능	사용 예시
`DirectoryLoader`	폴더 내 여러 파일을 일괄 로드	대규모 문서 배치 처리
`RecursiveCharacterTextSplitter`	Loader가 반환한 긴 문서를 청크로 분할 (후처리 단계)	RAG 문서 전처리
`UnstructuredFileIOLoader`	파일 스트림 객체로 로드	API 기반 파일 처리
`PyMuPDFLoader`	PDF를 빠르고 정밀하게 파싱 (좌표, 이미지 포함)	논문, 기술 보고서용

🧩 6. 로더 선택 기준

목적	추천 Loader
일반 텍스트	`TextLoader`
PDF 문서	`PyPDFLoader` 또는 `UnstructuredPDFLoader`
Word 문서	`Docx2txtLoader`
웹사이트	`WebBaseLoader`, `SitemapLoader`
YouTube 영상	`YoutubeTranscriptLoader`
Notion 문서	`NotionDBLoader`
CSV/JSON 데이터	`CSVLoader`, `JSONLoader`
폴더 전체 처리	`DirectoryLoader`

예시 코드

python

```python
from langchain_community.document_loaders import PyPDFLoader, DirectoryLoader

# PDF 한 개 로드
loader = PyPDFLoader("report.pdf")
docs = loader.load()

# 폴더 내 모든 PDF 로드
dir_loader = DirectoryLoader("data/", glob="**/*.pdf", loader_cls=PyPDFLoader)
all_docs = dir_loader.load()

print(len(all_docs), "개의 문서를 로드했습니다.")
print(all_docs[0].page_content[:500])
```

2.1 TextLoader

📘 1. 개요

`TextLoader` 는 LangChain에서 가장 기본적인 **문서 로더(Document Loader)** 로,
일반 텍스트 파일(`.txt`)을 읽어들여 **Document 객체**로 변환한다.

즉,

> "텍스트 파일 → Document(page_content + metadata)"
> 형태로 바꿔주는 가장 단순한 Loader 이다.

2. 기본 사용법

python

```python
from langchain_community.document_loaders import TextLoader

# 텍스트 파일 로드
loader = TextLoader("data/sample.txt", encoding="utf-8")
docs = loader.load()

print(docs[0].page_content)  # 파일의 내용 출력
print(docs[0].metadata)      # 파일 경로 등의 부가정보
```

🧱 3. 반환 구조

`TextLoader`는 **하나의 파일을 읽어 하나의 Document 객체**로 반환한다.
반환값은 항상 `List[Document]` 형태이다.

```python
[
  Document(
    page_content="이 문서는 LangChain TextLoader 예시입니다.\n여러 줄의 텍스트를 포함합니다.",
    metadata={"source": "data/sample.txt"}
  )
]
```

🧩 4. 주요 매개변수

매개변수	기본값	설명
`file_path`	필수	읽어올 텍스트 파일 경로
`encoding`	`"utf-8"`	파일 인코딩 지정 (한글 포함 시 반드시 utf-8 추천)
`autodetect_encoding`	`False`	True로 하면 chardet을 이용해 인코딩 자동 감지

예시:

```python
loader = TextLoader("note.txt", autodetect_encoding=True)
docs = loader.load()
```

🧠 5. 활용 예시

(1) 단일 텍스트 파일 로드

```python
loader = TextLoader("guide.txt")
docs = loader.load()

print(docs[0].page_content[:200])
```

(2) 여러 텍스트 파일을 한 번에 로드

폴더 전체의 `.txt` 파일을 불러올 때는 `DirectoryLoader`와 함께 사용한다.

```python
from langchain_community.document_loaders import DirectoryLoader

loader = DirectoryLoader(
    "data/",
    glob="**/*.txt",
    loader_cls=TextLoader
)

docs = loader.load()
print(len(docs), "개의 파일을 불러왔습니다.")
```

🧮 6. TextLoader의 특징 및 장점

항목	설명
단순성	가장 가볍고 빠른 로더 — 별도 라이브러리 불필요
범용성	텍스트 기반 로그, 자막, 코드, 메모 등 대부분의 .txt 파일에 사용 가능
확장성	다른 Loader (PDF, CSV 등)와 동일한 `Document` 구조로 처리 가능
RAG 전처리용	RAG 파이프라인에서 텍스트 기반 데이터 준비 시 자주 사용됨

7. TextLoader → TextSplitter → Embedding 예제

python

```python
from langchain_community.document_loaders import TextLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS

# 1) 텍스트 로드
loader = TextLoader("data/lecture.txt")
docs = loader.load()

# 2) 청크 단위로 나누기
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
split_docs = splitter.split_documents(docs)

# 3) 임베딩 생성 및 벡터 저장
embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
vectorstore = FAISS.from_documents(split_docs, embeddings)

print("총", len(split_docs), "개의 청크가 저장되었습니다.")
```

🔍 8. 정리

항목	설명
클래스명	`TextLoader`
모듈 경로	`from langchain_community.document_loaders import TextLoader`
입력	`.txt` 파일 경로
출력	`List[Document]`
주요 속성	`page_content`, `metadata`
주요 용도	단일/다수 텍스트 파일 로드, RAG용 데이터 전처리
관련 클래스	`DirectoryLoader`, `RecursiveCharacterTextSplitter`

2.2 DirectoryLoader

이 클래스는 실제 프로젝트에서 **대규모 문서 일괄 처리** 시 거의 항상 사용되는 핵심 도구입니다.

📘 1. 개요

`DirectoryLoader`는 **폴더(디렉터리)** 안의 여러 파일을 자동으로 탐색하여
각 파일을 개별 `Document` 객체로 읽어들이는 **문서 로더(Document Loader)** 이다.

즉,

> "폴더 안의 파일들 → 여러 개의 Document 리스트"
> 형태로 변환해주는 일괄 로더이다.

⚙️ 2. 기본 구조 및 사용법

python

```python
from langchain_community.document_loaders import DirectoryLoader, TextLoader

# data 폴더 내 모든 .txt 파일 로드
loader = DirectoryLoader(
    "data/",
    glob="**/*.txt",          # 하위 폴더 포함 모든 txt 파일 탐색
    loader_cls=TextLoader,    # 각 파일을 처리할 개별 로더 지정
    show_progress=True        # 진행률 표시
)

docs = loader.load()
print(len(docs), "개의 문서를 불러왔습니다.")
```

> 결과: `List[Document]` 형태의 리스트 반환
> 각 요소는 `TextLoader`, `PyPDFLoader` 등이 만든 `Document` 객체

🧱 3. 주요 매개변수

매개변수	자료형	설명
`path`	`str`	탐색할 폴더 경로
`glob`	`str`	파일 패턴 지정 (기본값: `"**/*"`)
`loader_cls`	`BaseLoader` 클래스	각 파일을 읽을 로더 지정 (`TextLoader`, `PyPDFLoader` 등)
`show_progress`	`bool`	True로 설정하면 파일 로딩 진행률 표시
`use_multithreading`	`bool`	병렬 로드 사용 여부 (기본 False)
`silent_errors`	`bool`	True면 파일 오류 발생 시 무시

🧩 4. 예시 코드

(1) 폴더 내 모든 `.txt` 파일 불러오기

python

```python
from langchain_community.document_loaders import DirectoryLoader, TextLoader

loader = DirectoryLoader(
    "data/",
    glob="**/*.txt",
    loader_cls=TextLoader
)

docs = loader.load()
print(len(docs))
print(docs[0].metadata)
```

(2) 폴더 내 모든 PDF 파일 불러오기

python

```python
from langchain_community.document_loaders import DirectoryLoader, PyPDFLoader

loader = DirectoryLoader(
    "reports/",
    glob="**/*.pdf",
    loader_cls=PyPDFLoader
)

docs = loader.load()
print("PDF 문서 개수:", len(docs))
```

(3) 하위 폴더까지 자동 포함

`glob="**/*.txt"` 설정 시,
data 폴더의 **모든 하위 폴더**에 있는 텍스트 파일도 자동 탐색한다.

```pgsql
data/
 ├── notice/
 │    ├── 2023.txt
 │    └── 2024.txt
 ├── policy/
 │    └── guide.txt
 └── summary.txt
```

→ 총 4개의 Document 객체가 생성됨

🧮 5. 반환 구조 예시

python

```
[
  Document(
    page_content="2023년도 공지사항 내용...",
    metadata={"source": "data/notice/2023.txt"}
  ),
  Document(
    page_content="정책 안내문...",
    metadata={"source": "data/policy/guide.txt"}
  )
]
```

🧠 6. DirectoryLoader의 장점

특징	설명
대량 처리	수십~수천 개의 파일을 한 번에 읽어올 수 있음
유연성	어떤 파일 형식이든 해당 Loader 지정 가능 (`TextLoader` , `PyPDFLoader` , `UnstructuredFileLoader` 등)
재사용성	같은 폴더 구조의 데이터셋을 반복 처리할 때 유용
RAG 전처리 핵심 도구	수많은 텍스트 문서를 자동으로 Document로 변환

🔍 7. Text Splitter와 함께 사용 예시

python

```python
from langchain_community.document_loaders import DirectoryLoader, TextLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter

# 1) 폴더 전체 로드
loader = DirectoryLoader("data/", glob="**/*.txt", loader_cls=TextLoader)
docs = loader.load()

# 2) 텍스트 분할
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
split_docs = splitter.split_documents(docs)

print("총", len(split_docs), "개의 청크로 분할되었습니다.")
```

8. 실제 RAG 파이프라인 예시

python

```python
from langchain_community.document_loaders import DirectoryLoader, TextLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS

# 1) 디렉터리 내 모든 txt 로드
loader = DirectoryLoader("knowledge_base/", glob="**/*.txt", loader_cls=TextLoader)
docs = loader.load()

# 2) 문서 분할
splitter = RecursiveCharacterTextSplitter(chunk_size=700, chunk_overlap=100)
split_docs = splitter.split_documents(docs)

# 3) 임베딩 및 벡터 저장
embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
vectorstore = FAISS.from_documents(split_docs, embeddings)

print("총", len(split_docs), "개의 문서 청크가 벡터DB에 저장되었습니다.")
```

🧩 9. 정리 요약

항목	내용
클래스명	`DirectoryLoader`
모듈 경로	`from langchain_community.document_loaders import DirectoryLoader`
입력	폴더 경로 (`path`)
출력	`List[Document]`
주요 옵션	`glob` , `loader_cls` , `show_progress` , `use_multithreading`
사용 목적	여러 파일을 한 번에 로드하여 RAG, 검색, 요약용 데이터로 사용
자주 조합되는 클래스	`TextLoader` , `PyPDFLoader` , `RecursiveCharacterTextSplitter` , `FAISS`

2.3 PyPDFLoader

📘 1. 개요

`PyPDFLoader`는 **PDF 파일을 텍스트로 변환하여** `Document` **객체로 만드는 로더**입니다.
내부적으로 `pypdf` (이전 명칭 `PyPDF2`) 라이브러리를 사용하며,
PDF의 각 페이지를 개별 **Document 객체**로 반환합니다.

즉,

> "PDF 문서 → (페이지별) Document 리스트"

형태로 변환합니다.

2. 기본 사용법

python

```python
from langchain_community.document_loaders import PyPDFLoader

# PDF 파일 로드
loader = PyPDFLoader("data/report.pdf")

docs = loader.load()   # 반환값: List[Document]

print(len(docs), "개의 페이지를 불러왔습니다.")
print(docs[0].page_content[:300])   # 첫 페이지 내용 일부 출력
print(docs[0].metadata)             # 페이지 정보 확인
```

🧱 3. 반환 구조

PDF의 각 페이지가 **별도의 Document 객체**로 만들어집니다.

```python
[
  Document(
    page_content="1페이지 내용...",
    metadata={"source": "data/report.pdf", "page": 0}
  ),
  Document(
    page_content="2페이지 내용...",
    metadata={"source": "data/report.pdf", "page": 1}
  ),
  ...
]
```

즉, 10페이지짜리 PDF라면 `len(docs) == 10` 입니다.

🧩 4. 주요 매개변수

매개변수	기본값	설명
`file_path`	필수	불러올 PDF 파일 경로
`password`	`None`	암호화된 PDF의 비밀번호 지정
`extract_images`	`False`	이미지를 추출할지 여부 (일반적으로 False 유지)

🧠 5. 동작 원리

- `PyPDFLoader`는 `pypdf` 모듈을 이용해 페이지별 텍스트를 추출합니다.
- PDF 내부의 **레이아웃, 표, 이미지** 등은 단순히 텍스트로 인식되므로
  시각적 구조는 유지되지 않습니다.
- 각 페이지에 `metadata`로 `"page": 페이지번호`와 `"source": 경로`가 추가됩니다.

🔍 6. 실습 예제

(1) PDF 1개 로드

```python
from langchain_community.document_loaders import PyPDFLoader

loader = PyPDFLoader("data/sample.pdf")
docs = loader.load()

print("총", len(docs), "페이지 로드됨")
print(docs[0].page_content[:200])
```

---

(2) 여러 PDF 일괄 로드 (DirectoryLoader와 함께)

```python
from langchain_community.document_loaders import DirectoryLoader, PyPDFLoader

loader = DirectoryLoader(
    "data/",
    glob="**/*.pdf",
    loader_cls=PyPDFLoader
)
docs = loader.load()

print("총", len(docs), "개의 문서(페이지) 로드 완료")
```

🧮 7. Text Splitter와 함께 사용 예시

페이지 단위로 읽은 후, 더 세밀하게 청크로 나누고 싶을 때 `TextSplitter`를 이용합니다.

python

```python
from langchain_community.document_loaders import PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter

loader = PyPDFLoader("data/report.pdf")
docs = loader.load()

splitter = RecursiveCharacterTextSplitter(chunk_size=700, chunk_overlap=100)
split_docs = splitter.split_documents(docs)

print("총", len(split_docs), "개의 청크로 분할됨")
```

8. RAG 파이프라인 예제

PDF를 불러와서 FAISS 벡터스토어에 저장하는 전체 흐름입니다.

python

```python
from langchain_community.document_loaders import PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS

# 1) PDF 로드
loader = PyPDFLoader("data/report.pdf")
docs = loader.load()

# 2) 문서 분할
splitter = RecursiveCharacterTextSplitter(chunk_size=700, chunk_overlap=100)
split_docs = splitter.split_documents(docs)

# 3) 임베딩 및 벡터 저장
embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
vectorstore = FAISS.from_documents(split_docs, embeddings)

print("✅ PDF 문서가 벡터 DB로 성공적으로 저장되었습니다.")
```

📊 PDF 파서별 성능 비교 표

구분	🥇 PDFMiner	🥈 PDFPlumber	🥉 PyPDFium2	④ PyMuPDF	⑤ PyPDF2
Medical	1	2	3	4	5
Law	3	1	1	3	5
Finance	1	2	2	4	5
Public	1	1	1	4	5
Sum	**5**	**5**	**7**	**15**	**20**

https://low-waltz-df8.notion.site/PDF-3740de0143e54d7b807811bd105fdcef

📘 해석 요약

항목	설명
PDFMiner / PDFPlumber	가장 경량이며 텍스트 추출 정확도가 높음
PyPDFium2	C++ 기반으로 빠르지만 일부 문서에서 레이아웃 손실
PyMuPDF (fitz)	빠르고 표·이미지 추출도 가능해 고급 분석에 적합
PyPDF2	가장 오래된 라이브러리, 단순한 텍스트 추출엔 충분하지만 느림

[PDF 자료 다운로드 사이트]

한국정보화진흥원(NIA)
https://www.nia.or.kr/site/nia_kor/main.do

KCI (한국학술지인용색인)
https://www.kci.go.kr/kciportal/main.kci

해외 학술 논문 사이트
https://arxiv.org/

공공데이터포털
https://www.data.go.kr

2.4 UnstructuredPDFLoader

📘 1. 개요

`UnstructuredPDFLoader`는
PDF 문서의 **텍스트, 표, 이미지 등 다양한 요소를 구조적으로 추출**하는
LangChain의 고급 문서 로더입니다.

기본적으로 `unstructured` 라이브러리를 사용하며,
`PyPDFLoader`보다 **문단 단위로 더 세밀한 텍스트 추출**이 가능합니다.

⚙️ 2. 기본 사용법

```python
from langchain_community.document_loaders import UnstructuredPDFLoader

loader = UnstructuredPDFLoader("data/sample.pdf")
docs = loader.load()

print(len(docs), "개의 문단을 추출했습니다.")
print(docs[0].page_content[:300])
```

- 반환값: `List[Document]`
- 각 `Document`는 **문단 단위**로 구성됨
- `metadata`에는 페이지 번호, 파일 경로 등이 포함됨

🧱 3. 주요 특징

항목	설명
엔진	`unstructured` 라이브러리 기반
출력 단위	페이지가 아닌 **문단(block)** 단위
텍스트 품질	`PyPDFLoader`보다 자연스러운 문장 단위로 분리
표/이미지 처리	표, 이미지 캡션 등도 일부 추출 가능
속도	`PyPDFLoader`보다 약간 느림
의존성	`unstructured`, `pdfminer.six`, `pypdf`, `pi_heif` 등이 필요

🧠 4. PyPDFLoader와의 차이

구분	PyPDFLoader	UnstructuredPDFLoader
추출 단위	페이지별	문단·텍스트 블록별
텍스트 품질	단순	구조 보존 우수
속도	빠름	느림
표/이미지 처리	불가능	일부 가능
의존성	pypdf	unstructured + pi_heif 등
적합 용도	RAG용 기본 PDF 읽기	보고서, 논문 등 구조적 분석용

unstructured → 내부적으로 pdf2image → 내부적으로 poppler 명령어 (pdftoppm, pdfinfo) 호출

👉 **Windows 환경에는 popplerㅏ 먼저 설치되어 있어야함**

① Poppler 설치 (Windows 기준)

1️⃣ Poppler for Windows 다운로드

공식 배포 페이지 (안전, 유지관리됨):

🔗 https://github.com/oschwartz10612/poppler-windows/releases/ ↗

- "**poppler-xx.0.0-x86_64.zip**" 파일 다운로드
  (예: `poppler-24.02.0-x86_64.zip`)

2️⃣ 압축 해제 후 경로 설정

- 예: `C:\poppler-24.02.0\` 로 압축 해제
- 내부에 `bin\pdfinfo.exe`, `bin\pdftoppm.exe` 등이 있어야 합니다.

3️⃣ 환경 변수(PATH) 등록

Windows 검색창 → "환경 변수 편집" →
"시스템 변수"에서 `Path` 선택 → "새로 만들기" →
`C:\poppler-24.02.0\bin` 추가 → 확인 후 **재부팅 또는 커널 재시작**

4️⃣ 설치 확인 (명령 프롬프트에서 실행)

```bash
pdfinfo -v
```

정상 설치 시 버전 정보가 표시됩니다.

② Tesseract OCR 설치 (OCR 관련 경고용)

오류 메시지 중 다음 부분은 OCR 기능 관련이므로 참고만 해도 됩니다.

```csharp
pytesseract is not installed. Cannot use the ocr_only partitioning strategy.
```

OCR을 쓰려면 아래처럼 설치할 수 있습니다 (선택사항):

- Windows용 Tesseract 설치:
  🔗 https://github.com/UB-Mannheim/tesseract/wiki ↗
- 설치 후 Python 모듈 설치:

```bash
pip install pytesseract
```

> ⚙️ 단순 텍스트 추출만 필요하다면 설치하지 않아도 됩니다.

2.5 WebBaseLoader

📘 1. 개요

`WebBaseLoader`는
URL(웹 페이지 주소) 를 입력받아,
그 웹 페이지의 본문 텍스트를 추출하여 **LangChain의** `Document` **객체**로 변환해주는 로더입니다.

즉,

> "웹 페이지 → 텍스트(Document)"
> 로 바꿔주는 역할을 합니다.

⚙️ 2. 기본 사용법

```python
from langchain_community.document_loaders import WebBaseLoader

# 로더 생성
loader = WebBaseLoader("https://ko.wikipedia.org/wiki/LangChain")

# 웹 페이지 로드
docs = loader.load()

# 결과 확인
print(len(docs), "개의 문서 로드됨")
print(docs[0].page_content[:500])
```

- 입력: URL (하나 또는 여러 개 가능)
- 출력: `List[Document]`
- 자동으로 HTML을 파싱하고 본문 텍스트만 남깁니다.
- 이미지, 링크, 광고 영역 등은 자동으로 제거됩니다.

🧱 3. 반환 구조

`WebBaseLoader`는 아래처럼 `Document` 리스트를 반환합니다.

python

```
[
  Document(
    page_content="LangChain은 대규모 언어모델(LLM)을 위한 프레임워크로...",
    metadata={
      'source': 'https://ko.wikipedia.org/wiki/LangChain',
      'title': 'LangChain - 위키백과'
    }
  )
]
```

🧩 4. 여러 페이지 로드하기

`WebBaseLoader`는 한 번에 여러 개의 URL도 처리할 수 있습니다.

python

```python
urls = [
    "https://ko.wikipedia.org/wiki/OpenAI",
    "https://ko.wikipedia.org/wiki/ChatGPT",
    "https://ko.wikipedia.org/wiki/LangChain"
]

loader = WebBaseLoader(urls)
docs = loader.load()

print("총", len(docs), "개의 웹 문서 로드됨")
```

🧠 5. 주요 특징

항목	설명
입력 형태	하나의 URL 또는 URL 리스트
출력 형태	`List[Document]`
자동 파싱	HTML → 본문 텍스트 자동 추출
인코딩 자동 처리	UTF-8, EUC-KR 등 자동 감지
metadata	URL, title, description 등 포함
의존성	`requests`, `beautifulsoup4`
사용처	뉴스 기사, 블로그, 위키, 공공 웹데이터 등

감사합니다