
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

9. `사용자 정의 제네레이터 (generator)`

1) `사용자 정의 제네레이터` *(generator)*

- **`제너레이터 함수`** = **`yield`** **키워드 사용,** **`이터레이터`** **처럼** **`동작`** **하는 함수**
  - `LCEL` 파이프라인에서 사용 가능
  - 제너레이터: **`Iterator`**`[`**`Input`**`]` -> **`Iterator`**`[`**`Output`**`]`
  - 비동기 제너레이터: **`AsyncIterator`**`[`**`Input`**`]` -> **`AsyncIterator`**`[`**`Output`**`]`
- **`유용한 경우`**
  - 사용자 정의 출력 파서 구현
  - 이전 단계의 출력을 수정하면서 스트리밍 기능 유지
- **`쉼표`** **로** **`구분`** **된** **`목록`** **에 대한** **`사용자 정의 출력 파서`** **구현해보기**

```
from typing import Iterator, List

from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import chain
from langchain_google_genai import ChatGoogleGenerativeAI

from dotenv import load_dotenv
import os

# LLM 초기화
# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

# LLM 생성하기
gemini_lc = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
    temperature=0,
)
```

- 기본 `LLM` 생성하기 (`gemini_lc`) - `gemini-2.5.flash-lite`

```
E0000 00:00:1760011465.829097 2338920 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste
```

```
# 프롬프트 생성
prompt = ChatPromptTemplate.from_template(
    # 주어진 회사와 유사한 5개의 회사를 쉼표로 구분된 목록으로 작성하세요.
    "Write a comma-separated list of 5 companies similar to: {company}"
)

# chain 생성하기 - 프롬프트와 모델을 연결하고 문자열 출력 파서를 적용
str_chain = prompt | gemini_lc | StrOutputParser()
```

- **`stream`** 으로 실행 → 결과 출력
  - *결과 = 실시간으로 생성됨*

```
# 데이터 스트리밍하기

for chunk in str_chain.stream({"company": "Google"}):
```

```
    # 각 청크를 출력하고, 줄 바꿈 없이 버퍼를 즉시 플러시함
    print(chunk, end="", flush=True)
```

- 데이터 스트리밍하기 - (`1.2s`)
- **`Microsoft`**, **`Amazon`**, **`Apple`**, **`Meta`**, **`IBM`**

- **`invoke()`** → 실행 결과 확인하기

```
# 체인에 데이터를 invoke 해보기

str_chain.invoke({"company": "Google"})
```

- `str_chain.invoke()` - (`0.6s`)
- **`'Microsoft, Amazon, Apple, Meta, IBM'`**

- **`split_into_list()`** = `LLM` 토큰의 이터레이터를 입력으로 받음 → 쉼표로 구분된 문자열 리스트의 이터레이터를 반환함
  - `generator` = 마지막 청크를 `yield`하여 반환함

```
# 입력으로 llm 토큰의 반복자를 받아 쉼표로 구분된 문자열 리스트로 분할하는 사용자 정의 파서

def split_into_list(input: Iterator[str]) -> Iterator[List[str]]:

    # 쉼표가 나올 때까지 부분 입력을 보관함
    buffer = ""

    for chunk in input:
        # 현재 청크를 버퍼에 추가함
        buffer += chunk

        # 버퍼에 쉼표가 있는 동안 반복함
        while "," in buffer:
            # 버퍼를 쉼표로 분할함
            comma_index = buffer.index(",")

            # 쉼표 이전의 모든 내용을 반환함
            yield [buffer[:comma_index].strip()]

            # 나머지는 다음 반복을 위해 저장함
            buffer = buffer[comma_index + 1 :]

    # 마지막 청크를 반환함
    yield [buffer.strip()]
```

- **`str_chain`** 변수의 문자열 → **`파이프(|) 연산자`** → **`split_into_list`** 함수에 전달
  - **`split_into_list`** 함수 = `문자열`을 `리스트`로 `분할`하는 역할
  - `분할된 리스트` = **`list_chain`** 변수에 할당

```
# 문자열 체인을 리스트로 분할함

list_chain = str_chain | split_into_list
```

- **`list_chain`** 객체의 **`stream`** **메서드 호출 → 입력 데이터 `{"animal": "bear"}`에 대한 출력을 생성함**
  - **`stream`** **메서드 = 출력을 `청크`(`chunk`) 단위로 반환하며, 각 `청크`는 `반복문`을 통해 `처리`됨**
  - 각 `청크`는 `print 함수`를 사용하여 즉시 출력
  - `flush=True` 옵션 → *버퍼링 없이 출력이 즉시 이루어짐*

- **`list_chain`을 사용 → 입력 데이터에 대한 출력 생성 → 출력을 청크 단위로 처리 → 즉시 출력하는 과정**

```
# 생성한 list_chain → 스트리밍되는지 확인하기
for chunk in list_chain.stream({"company": "Google"}):
    print(chunk, flush=True)                                    # 각 청크를 출력하고, 버퍼를 즉시 플러시
```

- `list_chain` 스트리밍⭕ - (`0.7s`)

```
['Microsoft']
['Amazon']
['Apple']
```

```
['Meta']
['IBM']
```

- **`list_chain`에 `invoke()`로 데이터 주입 → 작동 확인하기**

```
# list_chain 에 데이터를 invoke하기

list_chain.invoke({"company": "Google"})
```

- `list_chain.invoke()` - (`0.8s`)

```
['Microsoft', 'Amazon', 'Apple', 'Meta', 'IBM']
```

---

2) 비동기 (`Asynchronous`)

- **`asplit_into_list`** 함수
  - 비동기 제너레이터(`AsyncIterator`[`str`]) 입력으로 받음 → 문자열 리스트의 비동기 제너레이터(`AsyncIterator`[`List`[`str`]])를 반환

```
from typing import AsyncIterator


# 비동기 함수 정의
async def asplit_into_list(input: AsyncIterator[str]) -> AsyncIterator[List[str]]:

    buffer = ""

    # `input`은 `async_generator` 객체이므로 `async for`를 사용
    async for chunk in input:

        buffer += chunk

        while "," in buffer:

            comma_index = buffer.index(",")

            yield [
                buffer[:comma_index].strip()            # 쉼표를 기준으로 분할하여 리스트로 반환
            ]
            buffer = buffer[comma_index + 1:]

    yield [buffer.strip()]                              # 남은 버퍼 내용을 리스트로 반환
```

```
# alist_chain 과 asplit_into_list 를 파이프라인으로 연결

alist_chain = str_chain | asplit_into_list
```

- **비동기 함수 → 스트리밍**

```
# async for 루프 → 데이터 스트리밍하기

async for chunk in alist_chain.astream({"company": "Google"}):
    # 각 청크를 출력하고 버퍼를 비우기
    print(chunk, flush=True)
```

- `async for 루프.astream` - (`1.0s`)

```
['Microsoft']
['Amazon']
['Apple']
['Meta']
['IBM']
```

- **비동기 `chain` → 데이터를 `invoke()` → 동작 확인해보기**

```
# 리스트 체인을 비동기적으로 호출해보기

await alist_chain.ainvoke({"company": "Google"})
```

- `await alist_chain.ainvoke()` - (`0.5s`)

```
['Microsoft', 'Amazon', 'Apple', 'Meta', 'IBM']
```

- next: ***10. Runtime Arguments 바인딩***
