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

---

2. `도구 바인딩` *(BindingTools)*

1) `LLM`에 `도구 바인딩` *(Binding Tools)*

- `LLM` 모델 → `도구` *(tool)* 를 호출 → **`chat`** **요청 시** **`LLM`** **모델에** **`도구 스키마`** ***(tool schema)*** **를** **`전달`****해야 함**
- **`LangChain Chat Model`**:
  - `.bind_tools()` 메서드 구현 → `도구 호출` *(tool callinf)* 기능 지원
  - `LangChain 도구 객체`, `Pydantic 클래스` 또는 `JSON 스키마 목록`을 `수신`하고 공급자별 예상 형식으로 `채팅 모델`에 `바인딩` (binding)
- `바인딩`된 `Chat Model`의 `후속 호출`은 `모델 API`에 대한 `모든 호출`에 `도구 스키마`를 `포함함`

- **`환경설정`**

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv

# API 키 정보 로드
load_dotenv()                           # True
```

```
from langsmith import Client
from langsmith import traceable

import os

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if os.getenv('LANGCHAIN_API_KEY') else "설정되지 않음" # API 키 값은 직접 출력하지 않음

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchain_project:
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{langchain_tracing_v2}')")
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status}")
    print("  -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.")
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확인하세요:")
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있지 않습니다 (현재: '{langchain_tracing_v2}').")
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

```

```

```
# ========================================
# 경고 메시지 무시
# ========================================
import os
os.environ['TOKENIZERS_PARALLELISM'] = 'false'
```

```
import sys
from pathlib import Path

# 루트 디렉토리를 Python 경로에 추가
root_dir = Path().absolute().parent
sys.path.append(str(root_dir))

print(f"✅ 루트 디렉토리 추가: {root_dir}")

# config import
from config import get_llm, get_embeddings
from config import gpt_5_nano, gpt_5_mini

print("✅ config.py import 성공!")
```

- 응답 시간: 4.7s
- ✅ 루트 디렉토리 추가: `20250727-langchain-note`
- ✅ config.py import 성공!

```
# ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
# Cell 2: 테스트
# ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
# 방법 1: 직접 호출
response = gpt_5_nano.invoke("대한민국의 수도는?")
print(response.content)

# 방법 2: 함수 호출
llm = get_llm("gpt_5_nano")
response = llm.invoke("안녕하세요!")
print(response.content)
```

- 응답 - 5.6s

```
서울(서울특별시)입니다.
안녕하세요! 만나서 반가워요. 무엇을 도와드릴까요? (번역, 글쓰기, 공부, 코딩, 정보 검색 등 뭐든 괜찮아요.)
```

---

2) `LLM에 바인딩할 Tool 정의`

- 실험을 위한 도구 (`tool`) 정의하기
  - **`get_word_length`**: 단어의 길이를 반환하는 함수
  - **`add_function`**: 두 숫자를 더하는 함수
  - **`naver_news_crawl`**: 네이버 뉴스 기사를 크롤링 하여 본문 내용을 반환하는 함수

- **`참고`**
  - **`@tool` 데코레이터** = 도구 정의에서 사용
  - `docstring` = *가급적 `영어`로 작성하는 것을 `권장`*

```
# ================================
# 1. 환경 설정 및 필수 라이브러리 임포트
# ================================
import os
import re
import requests
```

```
from bs4 import BeautifulSoup
from langchain_core.tools import tool
```

```
# 도구 정의하기

@tool
def get_word_length(word: str) -> int:
    """Returns the length of a word."""
    return len(word)


@tool
def add_function(a: float, b: float) -> float:
    """Adds two numbers together."""
    return a + b


@tool
def naver_news_crawl(news_url: str) -> str:
    """Crawls a 네이버 (naver.com) news article and returns the body content."""
    # HTTP GET 요청 보내기
    response = requests.get(news_url)

    # 요청이 성공했는지 확인
    if response.status_code == 200:
        # BeautifulSoup을 사용하여 HTML 파싱
        soup = BeautifulSoup(response.text, "html.parser")

        # 원하는 정보 추출
        title = soup.find("h2", id="title_area").get_text()
        content = soup.find("div", id="contents").get_text()
        cleaned_title = re.sub(r"\n{2,}", "\n", title)
        cleaned_content = re.sub(r"\n{2,}", "\n", content)
    else:
        print(f"HTTP 요청 실패. 응답 코드: {response.status_code}")

    return f"{cleaned_title}\n{cleaned_content}"


tools = [get_word_length, add_function, naver_news_crawl]
```

3) `bind_tools()` 로 LLM 에 도구 바인딩

- `llm` 모델에 `bind_tools()` 적용 → 도구 바인딩하기

```
from langchain_openai import ChatOpenAI
from config import gpt_5_nano, embeddings

# 모델 생성하기
gpt_5_nano = gpt_5_nano

# 도구 바인딩
gpt_5_nano_with_tools = gpt_5_nano.bind_tools(tools)
```

- 실행결과 확인하기
- 결과 = `tool_calls` 에 저장됨
  - **`.tool_calls` 확인 → 도구 호출 결과 확인 가능**
- 참고
  - `name` = 도구의 이름
  - `args` = 도구에 전달되는 인자

```
# 실행 결과
gpt_5_nano_with_tools.invoke("What is the length of the word 'teddynote'?").tool_calls
```

- `tool_calls` 확인하기 (2.9s)

```
[{'name': 'get_word_length',
'args': {'word': 'teddynote'},
```

```
'id': 'call_fk8ZhLcVyKucjiyfnELn4Els',
'type': 'tool_call'}]
```

- 다음으로 `llm_with_tools` + `JsonOutputToolsParser` → `tool_calls`를 `parsing`하여 결과 확인하기

```
from langchain_core.output_parsers.openai_tools import JsonOutputToolsParser

# 도구 바인딩 + 도구 파서
chain_gpt_5_nano = gpt_5_nano_with_tools | JsonOutputToolsParser(tools=tools)

# 실행 결과
tool_call_results = chain_gpt_5_nano.invoke("What is the length of the word 'teddynote'?")    # 3.3s
```

```
print(tool_call_results)
```

- 응답

```
[{'args': {'word': 'teddynote'}, 'type': 'get_word_length'}]
```

```
print(type(tool_call_results))      # <class 'list'>
```

```
print(tool_call_results, end="\n\n==========\n\n")

# 첫 번째 도구 호출 결과
single_result = tool_call_results[0]

# 도구 이름
print(single_result["type"])

# 도구 인자
print(single_result["args"])
```

- 응답

```
[{'args': {'word': 'teddynote'}, 'type': 'get_word_length'}]

# ==========

get_word_length
{'word': 'teddynote'}
```

- 도구 이름과 일치하는 도구를 찾아 실행하기

```
tool_call_results[0]["type"], tools[0].name
```

- 응답

```
('get_word_length', 'get_word_length')
```

- `execute_tool_calls` 함수 → 도구를 찾아 `args` 전달하여 도구 실행
  - `type` → 도구 이름
  - `args` → 도구 인자

```
def execute_tool_calls(tool_call_results):
    """
    도구 호출 결과를 실행하는 함수

    :param tool_call_results: 도구 호출 결과 리스트
    :param tools: 사용 가능한 도구 리스트
    """
    # 도구 호출 결과 리스트를 순회
    for tool_call_result in tool_call_results:
        # 도구의 이름과 인자를 추출
        tool_name = tool_call_result["type"]
        tool_args = tool_call_result["args"]
```

```
        # 도구 이름과 일치하는 도구를 찾아 실행
        # next() 함수를 사용하여 일치하는 첫 번째 도구를 찾음
        matching_tool = next((tool for tool in tools if tool.name == tool_name), None)

        if matching_tool:
            # 일치하는 도구를 찾았다면 해당 도구 실행
            result = matching_tool.invoke(tool_args)
            # 실행 결과 출력
            print(f"[실행도구] {tool_name}\n[실행결과] {result}")
        else:
            # 일치하는 도구를 찾지 못했다면 경고 메시지 출력
            print(f"경고: {tool_name}에 해당하는 도구를 찾을 수 없습니다.")


# 도구 호출 실행
# 이전에 얻은 tool_call_results를 인자로 전달하여 함수를 실행함
execute_tool_calls(tool_call_results)
```

- 응답

```
[실행도구] get_word_length
[실행결과] 9
```

4) `bind_tools` + `Parser` + `Execution`

- 일련의 과정을 한 번에 실행해보기
  - `llm_with_tools` : 도구를 바인딩한 모델
  - `JsonOutputToolsParser` : 도구 호출 결과를 파싱하는 파서
  - `execute_tool_calls` : 도구 호출 결과를 실행하는 함수

- 흐름 정리
  - 모델에 도구를 바인딩
  - 도구 호출 결과를 파싱
  - 도구 호출 결과를 실행

```
from langchain_core.output_parsers.openai_tools import JsonOutputToolsParser

# bind_tools + Parser + Execution
chain_gpt_5_nano = gpt_5_nano_with_tools | JsonOutputToolsParser(tools=tools) | execute_tool_calls
```

```
# 실행 결과_1
chain_gpt_5_nano.invoke("What is the length of the word 'teddynote'?")
```

- 응답 - (3.9s)

```
[실행도구] get_word_length
[실행결과] 9
```

```
# 실행 결과_2
chain_gpt_5_nano.invoke("114.5 + 121.2")
```

```
print(114.5 + 121.2)
```

- 응답 (3.4s)

```
[실행도구] add_function
[실행결과] 235.7
235.7
```

```
# 실행 결과_3
chain_gpt_5_nano.invoke(
```

```
    "뉴스 기사 내용을 크롤링해줘: https://n.news.naver.com/mnews/hotissue/article/092/0002347672?type=series&cid=2000065"
)
```

- 응답 (3.8s)

```
[실행도구] naver_news_crawl
[실행결과] [미장브리핑] 9월 미국 CPI 주목…3분기 S&P500 실적 발표

            ▲10일(현지시간) 미국 9월 소비자물가지수(CPI) 발표 예정. 고용 지표가 양호하게 나온 가운데 물가 지표 주목. 9월 미국 비농업고용 25
(사진=이미지투데이)▲11월 미국 연방준비제도(연준) 공개시장위원회(FOMC) 에서 0.50%p 인하 기대가 크케 후퇴한 가운데, 9일에는 FOMC 의사록 공개. 
손희연 기자(kunst@zdnet.co.kr)
Copyright © ZDNet Korea. All rights reserved. 무단 전재 및 재배포 금지.

이 기사는 언론사에서 IT 섹션으로 분류했습니다.
기사 섹션 분류 안내
기사의 섹션 정보는 해당 언론사의 분류를 따르고 있습니다. 언론사는 개별 기사를 2개 이상 섹션으로 중복 분류할 수 있습니다.
닫기
기자 프로필
손희연 기자
손희연 기자
지디넷코리아
구독
구독중
원·달러 환율 1450원대로 급락…"가용수단 적극 활용 대처"
[미장브리핑] 다우 사상 최고치…소프트뱅크, 엔비디아 지분 전량 매각
구독
지디넷코리아 구독하고 메인에서 바로 만나보세요!구독하고 메인에서 만나보세요!
구독중
지디넷코리아 구독하고 메인에서 바로 만나보세요!구독하고 메인에서 만나보세요!
언론사홈
지디넷코리아
            주요뉴스해당 언론사에서 선정하며 언론사(아웃링크)로 이동합니다.
KB금융, 광주 1인 여성 자영업자에 월 100만원 지원
아파트만? 이제 빌라 주담대도 갈아타자
카카오뱅크, 6일 새벽 금융거래 일시 중단
우리은행 올해만 세 번째 금융사고…"허위서류로 55억 대출"
지디넷코리아 '홈페이지'
QR 코드를 클릭하면 크게 볼 수 있어요.
QR을 촬영해보세요.
지디넷코리아 '홈페이지'
닫기
네이버 채널 구독하기
지디넷코리아 언론사가 직접 선정한 이슈

이슈
반도체 전쟁
"내년 D램 가격 50% 상승…삼성·SK 수혜 지속"
이슈
AI 핫트렌드
"AI 썼더니 생산성 올랐다" 82%…2026 소셜 미디어 마케팅 지형도
이슈
트럼프 2.0시대
"관세 해법은 유럽"…현대차, EU 공략 위한 신규 기술센터 가동
이전
다음
이 기사를 추천합니다
기사 추천은 24시간 내 50회까지 참여할 수 있습니다.
닫기
쏠쏠정보
0
흥미진진
0
공감백배
0
분석탁월
0
후속강추
```

```
0


기자 구독 후 기사보기
구독 없이 계속 보기
```

---

5) `bind_tools` > `Agent & AgentExecutor`로 대체

- **`bind_tools()` = 모델에 사용할 수 있는 스키마(도구)를 제공**
- **`AgentExecutor` = 실제로 `llm` 호출, 올바른 `도구`로 `라우팅`, `실행`, `모델 재호출` 등을 위한 실행 루프를 생성**
- 참고
  - `Agent`, `AgentExecutor` 에 대해서는 다음 장에서 자세히 다룰 예정

- **`langchain` 버전이 달라 해당 메소드, 임포트 경로 존재하지 않음**

문제 원인	상태
`langchain.agents.tool_calling` 은 존재하지 않음	🔥 100% 오류 발생
최신 LangChain은 기능이 다른 위치로 이동함	✔
올바른 import → `from langchain.agents import create_tool_calling_agent`	✔

- LangChain 릴리즈 순서
  - 0.1.x — agents v2 등장
  - 0.2.x — tool calling agent 기본화
  - 0.3.x — more restructuring
  - 1.x.x — 완전히 다른 구조 (LangGraph 중심, Agent API 대폭 축소)
  - 현재 사용 버전 = 1.0.5 :

> **`자동 Tool Calling Agent 기능이 제거된 상태`**

- 🔥 그러면 현 환경에서 Tool Calling Agent를 어떻게 쓰는가?
  - **`LangGraph` 기반으로 `직접 구성` 해야 함**
    - LangChain 1.x에서는 Agent API 대부분이 `LangGraph 라이브러리`로 이동됨.
    - 예전 방식은 제거됨.
      - ❌ `langchain.agents.tool_calling`
      - ❌ `create_tool_calling_agent`
      - ❌ `AgentExecutor`

```
try:
    from langgraph.prebuilt import create_react_agent
    print("Import 성공")
except ImportError as e:
    print("Import 실패:", e)
```

```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_openai import ChatOpenAI
from config import gpt_5_nano, embeddings
from langchain.tools import tool
from langgraph.prebuilt import create_react_agent

# ------------------------------
# 1. 모델
# ------------------------------
gpt_5_nano = gpt_5_nano   # 그대로 유지
```

```
# ------------------------------
# 2. Tools (이전 셀에서 만든 것 재사용)
# ------------------------------
tools = [get_word_length, add_function, naver_news_crawl]
```

```
# ------------------------------
# 3. Agent 생성 (LC 1.0.5 방식)
# create_tool_calling_agent → ❌ 없음
# create_react_agent → ⭕ Jay 환경에서 정상 작동
# ------------------------------
agent = create_react_agent(
    model=gpt_5_nano,       # ❌ llm=llm X → model=llm ✅
    tools=tools,
)
```

```
# ------------------------------
# 5. Agent 실행 (AgentExecutor 존재 ❌)
# LangGraph agent는 invoke()로 사용
# ------------------------------
def run_agent(user_input: str):
    result = agent.invoke(
        {
            "messages": [{"role": "user", "content": user_input}],
            "remaining_steps": 1
        }
    )
    # 반환값 dict → "output" 키 확인
    if isinstance(result, dict) and "output" in result:
        return result["output"]
    return str(result)
```

```
# Agent 실행하기_1
response = run_agent("How many letters in the word `teddynote`?")
print(response)
```

- 출력 (3.6s)

```
{'messages': [HumanMessage(content='How many letters in the word `teddynote`?', additional_kwargs={}, response_me
```

```
# Agent 실행하기_2
response2 = run_agent("114.5 + 121.2 + 34.2 + 110.1 의 계산 결과는?")
```

```
# 결과 확인
print(response2)
```

- 출력 (6.5s)

```
{'messages': [HumanMessage(content='114.5 + 121.2 + 34.2 + 110.1 의 계산 결과는?', additional_kwargs={}, response_me
```

```
print("==========\n")
print(114.5 + 121.2 + 34.2 + 110.1)
```

- 응답

```
==========

380.0
```

```
# Agent 실행하기_3

response3 = run_agent("뉴스 기사를 요약해 줘: https://n.news.naver.com/mnews/hotissue/article/092/0002347672?type=series&c
print(response3)
```

- 응답 (19.0s)

```
{'messages': [HumanMessage(content='뉴스 기사를 요약해 줘: https://n.news.naver.com/mnews/hotissue/article/092/0002347
```

- `Human Message`. `AI Message`, `Usage_message`, `meta_message` 등이 모두 출력됨 → **`content`** 만 출력 후 파싱 기능 추가되면 될 듯
  - **`ToolMessage`**의 **`content`**
  - **`JSON`** 파싱

```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_openai import ChatOpenAI
from config import gpt_5_nano, embeddings
from langchain.tools import tool
from langgraph.prebuilt import create_react_agent
import json

# ----------------------------
# 1. 모델
# ----------------------------
gpt_5_nano = gpt_5_nano   # 그대로 유지
```

```
# ----------------------------
# 2. Tools (이전 셀에서 만든 것 재사용)
# ----------------------------
tools = [get_word_length, add_function, naver_news_crawl]
```

```
# ----------------------------
# 3. Agent 생성 (LC 1.0.5 방식)
# create_tool_calling_agent → ❌ 없음
# create_react_agent → ⭕ Jay 환경에서 정상 작동
# ----------------------------
agent = create_react_agent(
    model=gpt_5_nano,       # ❌ llm=llm X → model=llm ✅
    tools=tools,
)
```

```
# ------------------------------------------
# 3. 실행 함수 (ToolMessage content + JSON 파싱)
# ------------------------------------------
def run_agent_tool_message(user_input: str, parse_json: bool = True):
    """
    Agent 실행용 함수 (ToolMessage 기준)

    Parameters:
    - user_input: str → AI에게 질문할 텍스트
    - parse_json: bool → ToolMessage content를 JSON으로 파싱 여부

    Returns:
    - dict(JSON) 또는 str
    """
    # Agent 호출
    result = agent.invoke({
        "messages": [{"role": "user", "content": user_input}],
        "remaining_steps": 1
    })

    # ToolMessage 객체가 있다고 가정
    # Jay 환경 v2 기준, tools 호출 시 result.tools[0].content로 접근 가능
    tool_messages = getattr(result, "tools", [])
    if not tool_messages:
        return None  # ToolMessage 없음

    tool_content = tool_messages[0].content

    # JSON 파싱
    if parse_json:
        try:
            return json.loads(tool_content)
        except json.JSONDecodeError:
            return {"raw_content": tool_content}

    return tool_content
```

- 위의 함수 결과 출력
- `agent.invoke*()` 가 반환하는 원본 객체 구조 그대로
- 즉, `content` 추출 단계가 빠짐

```
{'messages': [HumanMessage(content='How many letters in the word `teddynote`?', additional_kwargs={}, response_met
```

- **`content` = `.content` 속성으로 접근해야 추출이 가능함**

```
def run_agent_get_content(user_input: str):
    result = agent.invoke({
        "messages": [{"role": "user", "content": user_input}],
        "remaining_steps": 1
    })

    # 메시지 리스트 가져오기
    messages = result.get("messages", [])

    # 마지막 AIMessage content만 추출
    for msg in reversed(messages):
        if msg.__class__.__name__ == "AIMessage":
            return msg.content

    return None  # AIMessage 없으면 None
```

4) `bind_tools` + `Parser` + `Execution`

```
# Agent 실행하기_1
response = run_agent_get_content("How many letters in the word `teddynote`?")
print(response)
```

- 응답 (3.2s) : 9

```
# Agent 실행하기_2
response2 = run_agent_get_content("114.5 + 121.2 + 34.2 + 110.1 의 계산 결과는?")
```

```
# 결과 확인
print(response2)
print("==========\n")
```

- 응답 (6.0s)

```
380.0 입니다.

간단한 단계:
- 114.5 + 121.2 = 235.7
- 34.2 + 110.1 = 144.3
- 235.7 + 144.3 = 380.0

print("==========\n")
print(114.5 + 121.2 + 34.2 + 110.1)
```

```
# Agent 실행하기_3

response3 = run_agent_get_content("뉴스 기사를 요약해 줘: https://n.news.naver.com/mnews/hotissue/article/092/0002347672?t
print(response3)
```

- 응답 (30.3s)

```
다음은 해당 기사(미장브리핑 관련 요약)의 핵심 내용입니다.

간단 요약
- 9월 미국 물가 지표 발표를 앞두고 고용 호조와 함께 물가 핵심 지표의 둔화 가능성에 주목. 3분기 S&P 500 기업들의 실적 발표가 본격적으로 시작되며,

주요 포인트
- 고용·임금
- 9월 비농업고용 +25만4천명 증가(시장 예측 상회).
- 실업률 4.1%로 2개월 연속 하락.
- 평균 시급 YoY +4%로 5월 이후 최고 수준.
- 물가 지표 전망
- 9월 헤드라인 CPI: 전년 동월 대비 2.3% 증가 예상(8월은 2.6%).
- 전월 대비로는 9월도 둔화 기대(8월은 0.2% 둔화 관측).
- 핵심 CPI는 8월과 비슷한 수준으로 관측(다른 수치는 3.2% 근처를 유지 기대).
- 생산자 물가(PPI)
- 9월 발표 예정. 8월은 1.7% 증가로 반등 추세가 꺾이는 양상.
- 연준(FOMC) 방향
- 11월 FOMC에서 0.50%포인트 인하 기대가 다소 약화.
- 9일 FOMC 의사록 공개가 예정되어 있어 지난해 대규모 인하의 배경과 인플레이션 전망에 대한 논의가 주목.
- 3분기 실적 주도권
- S&P 500 기업의 3분기 실적 발표가 시작되면, 평균 이익 증가율은 전년 동기 대비 약 +4.6%로 추정(하지만 증가폭은 둔화될 것으로 보임)
```