에이전트 상태 관리 설계

LangGraph 기본 Message Channel과 내장 체크포인트 테이블을 쓰지 않는 이유

LHG 2026-08-29

CONTENTS

이 글에서 쓰는 용어

처음 보는 사람도 이해되게 먼저 풀어둔다

- **체크포인트(checkpoint)**: 그래프 실행을 중단했다가 같은 지점에서 다시 시작하도록 남기는 상태 저장본
- **HIL(Human-in-the-loop)**: 그래프가 사용자 승인을 기다리며 멈춰 있는 지점
- `AgentState` : LangGraph가 그래프 실행 중 메모리에서 들고 있는 상태 객체
- **Message channel**: `AgentState` 안에서 대화 메시지만 순서대로 담는 채널. Checkpointer와는 다른 개념
- `MemorySaver` : 같은 프로세스 안에서만 유지되는 LangGraph 기본 체크포인터(재기동하면 사라짐)
- `ag_agent_checkpoint` : 재기동·다중 인스턴스 재개를 위해 우리가 따로 만든 테이블

1. 현재 역할 분리

LangGraph 체크포인트 기능은 그대로 쓰되, 역할을 나눈다

그래프 실행은 LangGraph, 운영은 우리 테이블

LHG

LangGraph가 맡는 것

- 그래프 실행 상태(AgentState)
- 같은 프로세스 안 중단·재개(MemorySaver)

우리 테이블이 맡는 것

- 재기동·다중 인스턴스 재개
- 서비스 운영과 데이터 관리 전반

대화와 실행 이력도 각자 다른 테이블

LHG

재기동·다중 인스턴스

ag_agent_checkpoint

+

대화·재실행

채팅·실행 로그 테이블

- 재기동과 다중 인스턴스 재개는 `ag_agent_checkpoint`가 담당한다
- 대화 이력과 질의 재실행은 채팅·실행 로그 테이블이 담당한다

다섯 가지 역할, 다섯 개의 저장소

- 그래프 실행 상태 → LangGraph `AgentState`
- 같은 프로세스에서 중단·재개 → LangGraph `MemorySaver`
- 재기동·다중 인스턴스 재개 → `ag_agent_checkpoint`
- 세션·질문 순서 → `ag_chat_session_meta`, `ag_chat_turn_meta`
- 질의 재실행·실행 이력 → `ag_chat_turn_meta`, `op_*` 로그

내장 기능을 그대로 쓸지 판단하려면 먼저 이 두 개념부터 구분해야 한다

2. Message channel과 Checkpointer의 관계

내장 기능을 쓰려면 먼저 알아야 하는 관계다

무엇을 담을지 vs 언제·어디에 저장할지

LHG

Message channel

- State 안에 어떤 대화 내용을 담을지 정의

Checkpointer

- Message 포함 전체 State를 언제·어디에 저장·복원할지 담당

AgentState 안에서 Message channel의 위치


```
AgentState
 ├── messages          ← Message channel
 ├── current_step
 ├── next_agent
 ├── retry_count
 └── approval_status
        ↓
   Checkpointer → MemorySaver 또는 DB
```

Message channel은 Checkpointer가 저장하는 State의 일부일 수 있다.

있어도 되고 없어도 되는 조합

- Message channel만 있고 Checkpointer가 없으면 프로세스가 끝난 뒤 대화와 실행 상태를 복원할 수 없다
- Checkpointer가 있다고 Message channel을 반드시 써야 하는 것은 아니다
- Checkpointer는 Message가 없는 구조화된 `AgentState`도 그대로 저장할 수 있다

지금 시스템은 Message channel 없이 `AgentState` 중심으로 실행하고
`MemorySaver` 가 State와 실행 위치를 저장한다.

3. Message channel만으로는 부족한 이유

대화는 저장하지만 실행 상태까지 담지는 못한다

단순 챗봇은 되지만, 대형 멀티에이전트는 안 된다

LHG

단순 챗봇 (Message channel로 충분)

- 질문·답변 저장, 후속 답변, 간단한 도구 호출
- 단순한 대화 중단과 재개

대형 멀티에이전트 (더 필요한 것)

- 실행 중인·다음 에이전트, 작업 단계, Gate 결과
- 승인 종류·재개 위치, 재시도·취소·중복 실행 제어

Message에는 없는, 실행에 필요한 값들

LHG

Message는 대화 내용이지 실행 상태 자체가 아니다

- 지금 실행 중인·다음에 실행할 에이전트, 현재 작업 단계, Gate(안전성 검사) 결과
- 사용자 승인 종류와 재개 위치, 즉 HIL(Human-in-the-loop) 지점
- SQL·대용량 구조화 결과, 채팅 질문 순서와 특정 질문 재실행, 다중 인스턴스 간 상태 공유

"승인"이라는 한 마디로는 부족하다


```
사용자 입력: 승인

→ 계획 승인인가?
→ 보고서 선택인가?
→ NL2SQL 추가 질문에 대한 답인가?
```

현재 단계와 승인 대기 위치가 함께 있어야 정확히 재개할 수 있다.

Message 중심으로 관리하면 생기는 문제

- 모델·프롬프트가 바뀌면 Message 재해석 결과도, 다음 실행 경로도 달라진다
- Message에서 SQL·워커 결과를 다시 꺼내면 구조와 타입이 깨질 수 있다
- 도구 호출을 다시 판단하다가 SQL·외부 API가 중복 실행될 수 있다
- 대화가 길어질수록 체크포인트 저장량과 모델 입력 토큰이 함께 늘어난다
- 오래된 Message를 요약·삭제하면 재개에 필요한 정보까지 같이 사라질 수 있다

Message는 대화 문맥으로,
`AgentState`는 실행 제어로 역할을 나눠 쓴다.

단순 챗봇에는 적합하지만, 대형 멀티에이전트의 실행 상태·재개·분산 제어까지
혼자 맡기에는 한계가 있다.

4. 내장 DB 테이블을 그대로 쓰지 않는 이유

내부 저장소일 뿐, 업무 기능을 대신하는 테이블은 아니다

LangGraph DB checkpointer가 관리해 주는 것

- 체크포인트 스키마 버전 관리
- 그래프 단계별 state와 부가 정보 저장
- state 안 큰 데이터의 분리 저장, 노드 중간 결과 저장

대신 이런 업무 기능은 제공하지 않는다

LHG

- 사용자·세션 소유권 확인, 채팅 화면의 질문 순서
- 특정 질문부터 다시 실행, 이후 질문·로그 정리
- 사용자 승인 응답 선점, 취소·중복 실행 방지

그래프 순서와 채팅 순서는 다르다

한 질문 안에서도 에이전트·Gate를 여러 번 거친다

체크포인트

그래프 실행 순서

≠

채팅 턴

사용자 질문 순서

- 2번 질문을 다시 실행하면 기존 2·3·4번 로그부터 정리해야 한다
- 어떤 체크포인트 묶음이 몇 번 질문인지는 `ag_chat_turn_meta.turn_index`로 구분한다

내장 테이블을 써도 업무 테이블은 남는다

LHG

- 사용자·세션·워크플로우·채팅 순서를 잇는 정보와 인덱스를 추가해야 한다
- 승인 선점·취소용 조건부 처리, 재실행 시 정리 로직도 추가해야 한다
- Java·Python, PostgreSQL·Oracle 간 저장 방식 차이까지 대응해야 한다
- 결국 `ag_chat_turn_meta`, `op_*` 로그와 소유권·취소·선점 로직은 그대로 남는다

내장 테이블을 직접 손대면 버전마다 부담이 따라온다

LHG

- 테이블·컬럼 구조, 인덱스·저장 방식이 바뀌었는지 매번 확인해야 한다
- 기존 체크포인트 데이터, 대기 중인 승인과 호환되는지 확인해야 한다
- 패키지 교체로 끝나지 않고 DB 전환과 회귀 테스트까지 따라온다

대형 멀티에이전트 시스템은 체크포인트 하나에
모든 역할을 몰아넣지 않는다.

5. 내장 테이블 중심으로 바꾸면 생기는 영향

체크포인트 저장 방식 하나만 바뀌는 게 아니다

새로 해야 하는 일

- `MemorySaver`를 DB checkpointer로 교체하고 전용 패키지·테이블을 추가한다
- `AgentState`에 Message channel과 병합·삭제 규칙을 추가한다
- HIL 재개·상태 복원 로직을 checkpoint 조회 방식에 맞춰 다시 짠다
- 기존 HIL·체크포인트 데이터를 새 구조로 옮기는 전환 작업을 한다

그래도 남는 일

- 사용자 소유권·취소·HIL 선점·중복 실행 방지는 별도 구현이 계속 필요하다
- `ag_chat_turn_meta`, `op_*` 로그는 채팅 순서·재실행 때문에 그대로 유지한다
- 재실행 시 checkpoint와 채팅 턴·실행 로그를 함께 정리하는 로직도 새로 맞춰야 한다

버전이 바뀔 때마다 반복되는 검증

- Java·Python이 같은 체크포인트 테이블을 함께 쓰기 어려워 공통 복구 규약을 다시 정한다
- PostgreSQL·Oracle마다 DB checkpointer를 쓸지 별도 구현할지 다시 정한다
- LangGraph 버전이 바뀔 때마다 데이터 호환성과 HIL·재실행 회귀 테스트를 새로 돌린다

한 줄 답변

LangGraph의 네이티브 state와 체크포인트 기능은 그대로 쓴다.
Message와 내장 테이블만으로는 채팅 순서·질의 재실행·사용자 승인·
취소·다중 인스턴스 제어를 처리할 수 없어서
`ag_agent_checkpoint`, `ag_chat_turn_meta`, `op_*` **테이블로 역할을 나눴다.**