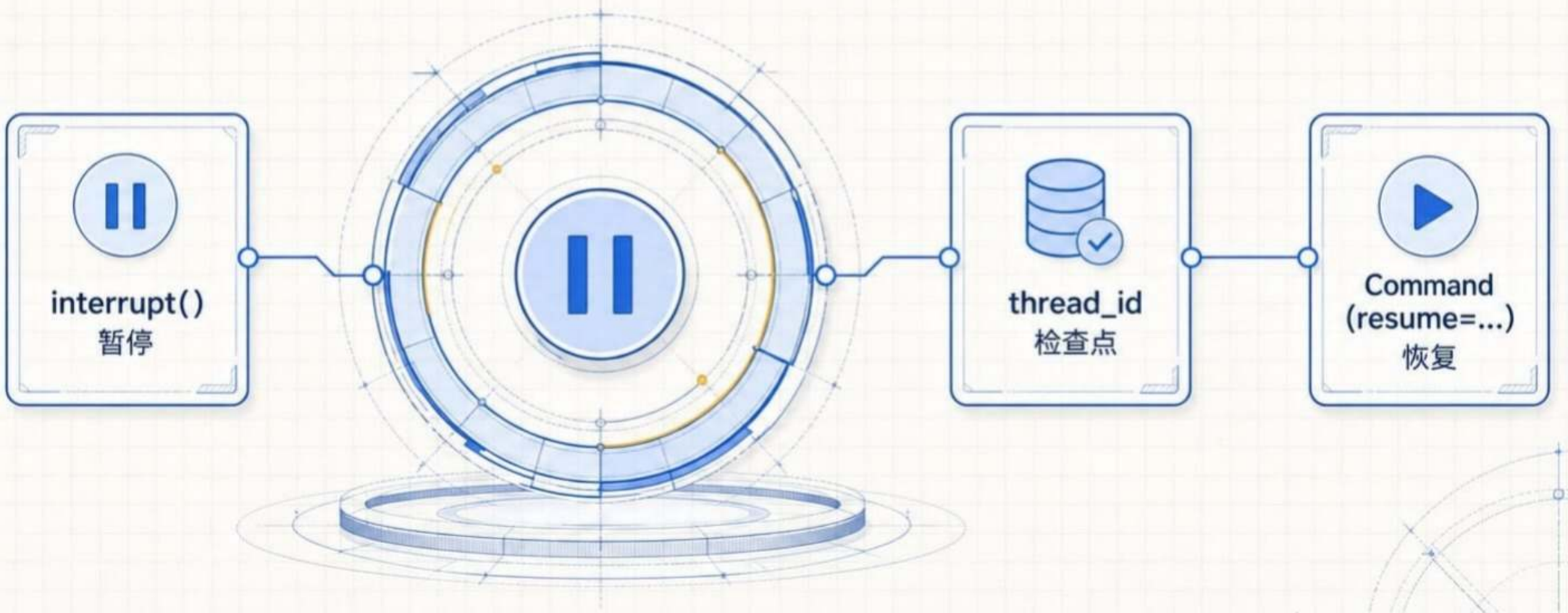
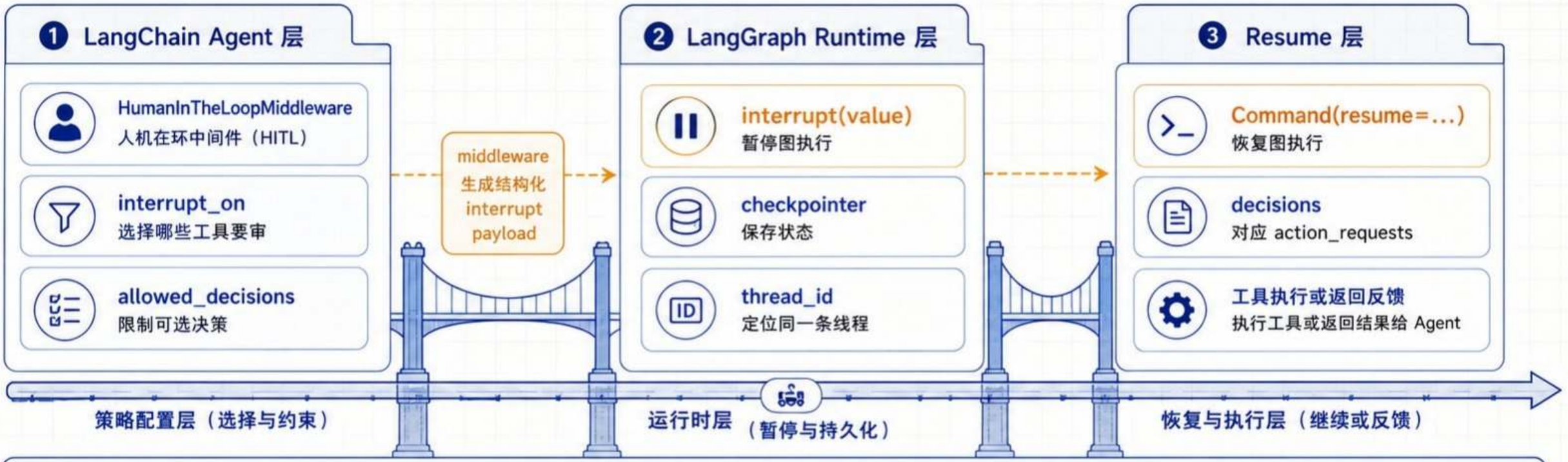
Deep Agents 实战

第 13 讲：HITL 进阶 — 代码与运行时



LangGraph 原语 × LangChain 中间件

HITL middleware 是工具调用层的 interrupt 封装




```
interrupt.value = {
  "action_requests": [...],
  "review_configs": [...],
}
```

```
agent.invoke(Command(resume={"decisions": [...]}),
                     config=config, version="v2")
```

底层只负责暂停与恢复

中间件负责工具审批语义

直接调用 interrupt()

当你要控制节点级流程，而不是工具审批

```python
from langgraph.types import interrupt

@tool
def request_approval(action: str) -> str:
    approval = interrupt({
        "type": "approval_request",
        "action": action,
        "message": f"请审批: {action}",
    })
    return "已批准" if approval.get("approved") else "已拒绝"
```

value：
传给前端审批界面的载荷

返回值：
恢复执行后回到 approval

审批分支：
根据 approved 决定结果

恢复底层中断

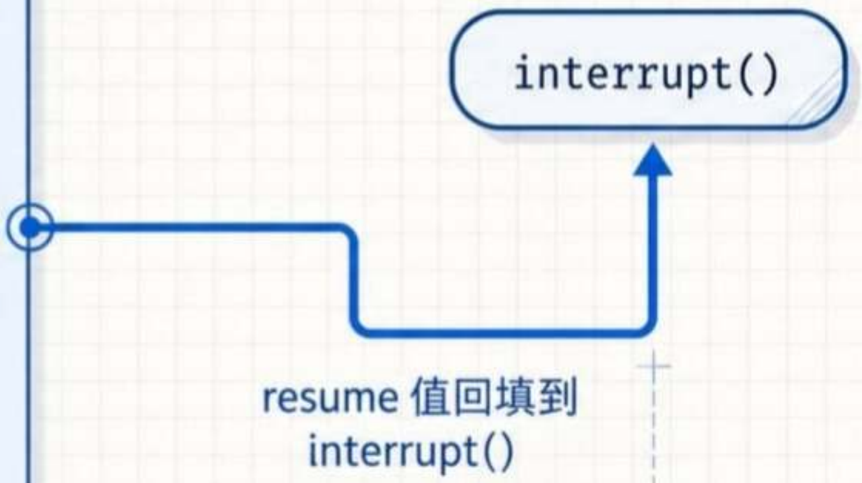
Command(resume=...) 的值会返回给 interrupt()

```python
agent.invoke(
    Command(resume={"approved": True}),
    config=config,
    version="v2",
)
```

通过：继续执行

```python
agent.invoke(
    Command(resume={"approved": False,
                    "reason": "延后执行"}),
    config=config,
    version="v2",
)
```

拒绝：带原因返回



排查暂停在哪里

用 get_state 看 values、next、interrupts

```python
snapshot = graph.get_state(config)

print(snapshot.values)
print(snapshot.next)
print(snapshot.interrupts)


# Platform / SDK
thread_state = client.threads.get_state(
    thread_id="thread-1")
```

thread_id 错了，
会看到另一条线程

恢复时会从节点开头重放

不是从 interrupt() 下一行继续


```python
def node(state):
    before_interrupt()  # resume 后会再跑
    answer = interrupt("是否继续？")
    after_interrupt(answer)
    return state
```

节点开头的代码会再次执行
不会跳过

在此处暂停，等待外部确认

确认后继续执行后续逻辑

规则 1：不要裸 try/except

不要吞掉 interrupt 的特殊异常

错误：裸 except 会吞掉中断

```
try:
    result = interrupt("请审批")
except Exception as e:
    print(e)
```

Exception 会误捕获特殊异常

interrupt 需要向外冒泡

正确：只捕获业务异常

```
try:
    result = interrupt("请审批")
    fetch_data()
except NetworkError as e:
    print(e)
```

只捕获 NetworkError

规则 2：副作用必须幂等

不能重复执行的动作放到 interrupt() 之后

错误：interrupt 前写入

```
def node(state):
    db.create_log("开始")
    approved = interrupt("请审批")
```

副作用提前发生

暂停后可能重放

正确：审批后写入

```
def node(state):
    approved = interrupt("请审批")
    if approved:
        db.create_log("已审批")
```

确认后再写入

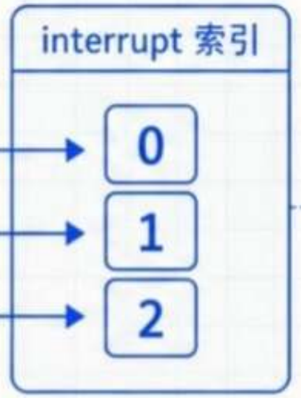
规则 3：不要改变调用顺序

同节点多个 interrupt 按索引匹配

Good

顺序固定，
索引匹配

```
1  name = interrupt("姓名？")
2  age  = interrupt("年龄？")
3  city = interrupt("城市？")
```

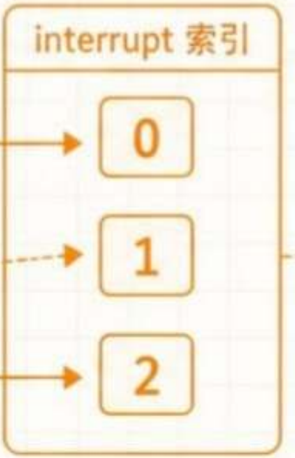


调用顺序稳定，
索引与问题一一对应，
不会错配。

Bad

条件跳过，
导致错配

```
1  name = interrupt("姓名？")
2  if state.get("need_age"):
3      age = interrupt("年龄？")
4  city = interrupt("城市？")
```



当 need_age 为 False 时，
会跳过第 1 个 interrupt。

后续索引前移，
导致与之前保存的索引错配，
恢复对话将出现错误。

规则 4：并行中断用 ID 恢复

界面可排序，恢复值不能错配

```
resume = {
    intr.id: ui_answers[intr.id]
    for intr in stream.interrupts
}

graph.invoke(Command(resume=resume), config=config)
```

用 id 做 key

按 id 取界面答案

排序变化也不影响恢复

审批卡片 A

Interrupt.id = "a17"

请审核申请 A17

同意

审批卡片 B

Interrupt.id = "b42"

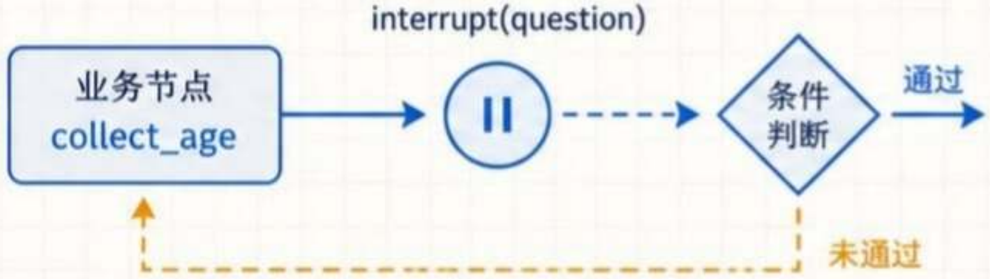
请审核申请 B42

同意

输入验证与静态中断

业务输入用状态回路，调试断点用静态中断

业务输入验证：单次 interrupt + 条件边

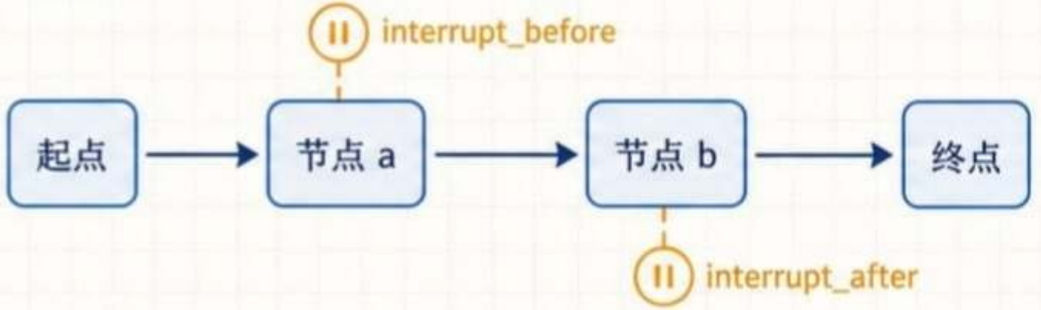



```
answer = interrupt(question)
return {"pending_question": next_question}
```

中断并等待
用户输入

未通过则回到
业务节点重试

调试断点：compile(interrupt_before/after)




```
graph = builder.compile(
    interrupt_before=["node_a"],
    interrupt_after=["node_b"],
    checkpointer=checkpointer,
)
```

在 node_a 之前
暂停

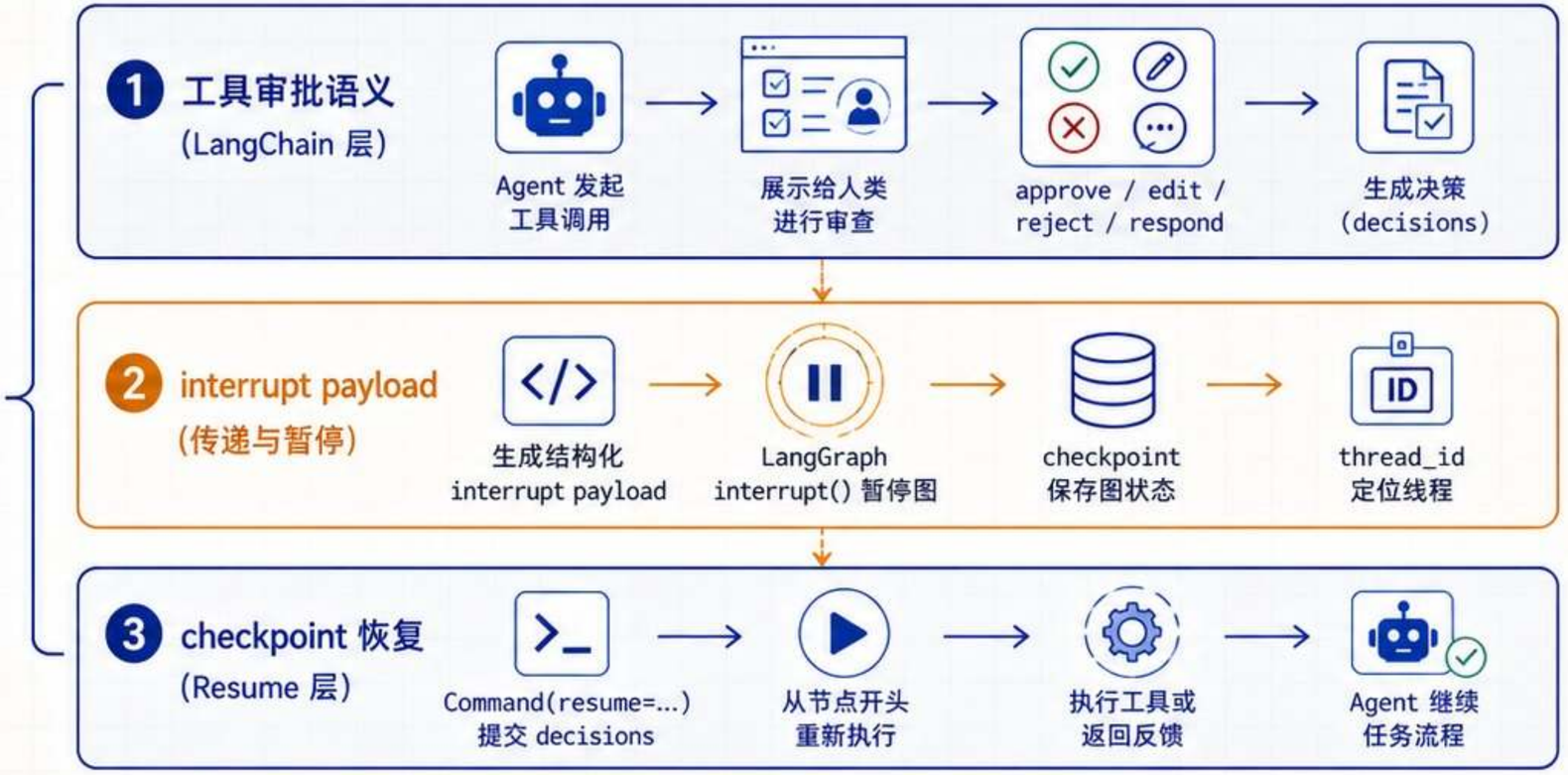
在 node_b 之后
暂停

持久化与恢复
由 checkpointer 管理

本讲回顾

从 HITL 中间件走到底层运行时

- LangChain middleware
  负责工具审批语义
- LangGraph `interrupt()`
  负责暂停图执行
- `Command(resume=...)`
  把人工结果送回节点
- 恢复会从节点开头重放
  要避免提前副作用
- 多个中断并行时
  用 ID 保持恢复值匹配



先保证可恢复，再设计审批体验

下一章： Sandboxes —— 让 Agent 在受控环境中安全执行代码