

RAG・AIエージェント[実践] 入門5章プレゼンテーション

2201110172

村上弘幸

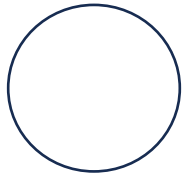
5 章LangCain Expression Language(LCEL)徹底解説 キーワード

- LCEL • Chain • |
- Runnable
- RunnableLambda • RunnableParallel
- RunnablePassthrough

注意事項！



・ RAG ・ AIエージェント



・ LangCain

LCELの基本的な実装

- Prompt template
- Chat model
- Output parser

の3つを連結する

```
chain = prompt | model | output_parser
```

LCELの基本的な実装

- Prompt template
- Chat model
- Output parser

の3つのコンポーネントは

Runnableという

抽象基底クラスを継承しています。

↓ **RunnableSequence**になる(Runnableの1種)

chain = prompt | model | output_parser

LCELの基本的な実装

Runnableの実行方法：

： invoke

```
1 chain = prompt | model | output_parser
2
3 output = chain.invoke({"dish": "カレー"})
4 print(output)
```

⇒ カレーのレシピを考えました！以下の材料と手順で美味しいカレーを作ってください。

材料（4人分）

- 鶏肉（もも肉または胸肉）：400g
- 玉ねぎ：2個
- にんじん：1本

```
1 chain = prompt | model | output_parser
2
3 for chunk in chain.stream({"dish": "カレー"}):
4     print(chunk, end="", flush=True)
```

… カレーのレシピを考えてみましょう。以下は基本的なチキンカレーのレシピです。

```
1 chain = prompt | model | output_parser
2
3 outputs = chain.batch([{"dish": "カレー"}, {"dish": "うどん"}])
4 print(outputs)
```

⇒ ['カレーのレシピをご紹介します！これは基本的なチキンカレーのレシピで

： stream

： batch

LCELの基本的な実装

Runnableなら繋げることができる！

例えばchain同士も繋げることができる

↓ ステップバイステップで考えるchainと
結論を抽出するchain

```
cot_summarize_chain = cot_chain | summarize_chain
```

使い方が見えてきた？

LCELの実践的な実装:RunnableLambda

RunnableLambda — 任意の関数をRunnableにする

LCELの実践的な実装:RunnableLambda

```
from langchain_core.runnables import RunnableLambda
```

```
def upper(text: str) -> str:  
    return text.upper()
```

```
chain = prompt | model | outputparser | RunnableLambda(upper)
```

```
output = chain.invoke({"input": "Hello!"})  
print(output)
```

LCELの実践的な実装:RunnableLambda

```
from langchain_core.runnables import chain
```

```
@chain
```

```
def upper(text: str) -> str:  
    return text.upper()
```

```
chain = prompt | model | outputparser | upper
```

```
output = chain.invoke({"input": "Hello!"})  
print(output)
```

LCELの実践的な実装:RunnableLambda

```
def upper(text: str) -> str:  
    return text.upper()
```

↓ 「|」 で自動的に変換

```
chain = prompt | model | outputparser | upper
```

```
output = chain.invoke({"input": "Hello!"})  
print(output)
```

LCELの実践的な実装:RunnableLambda

RunnableLambda — 任意の関数をRunnableにする

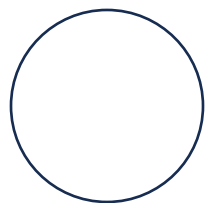


関数をchainに含めることができる

注意事項！ ※入力と出力の型に注意



```
chain = prompt | model | upper(入力 : str)
```

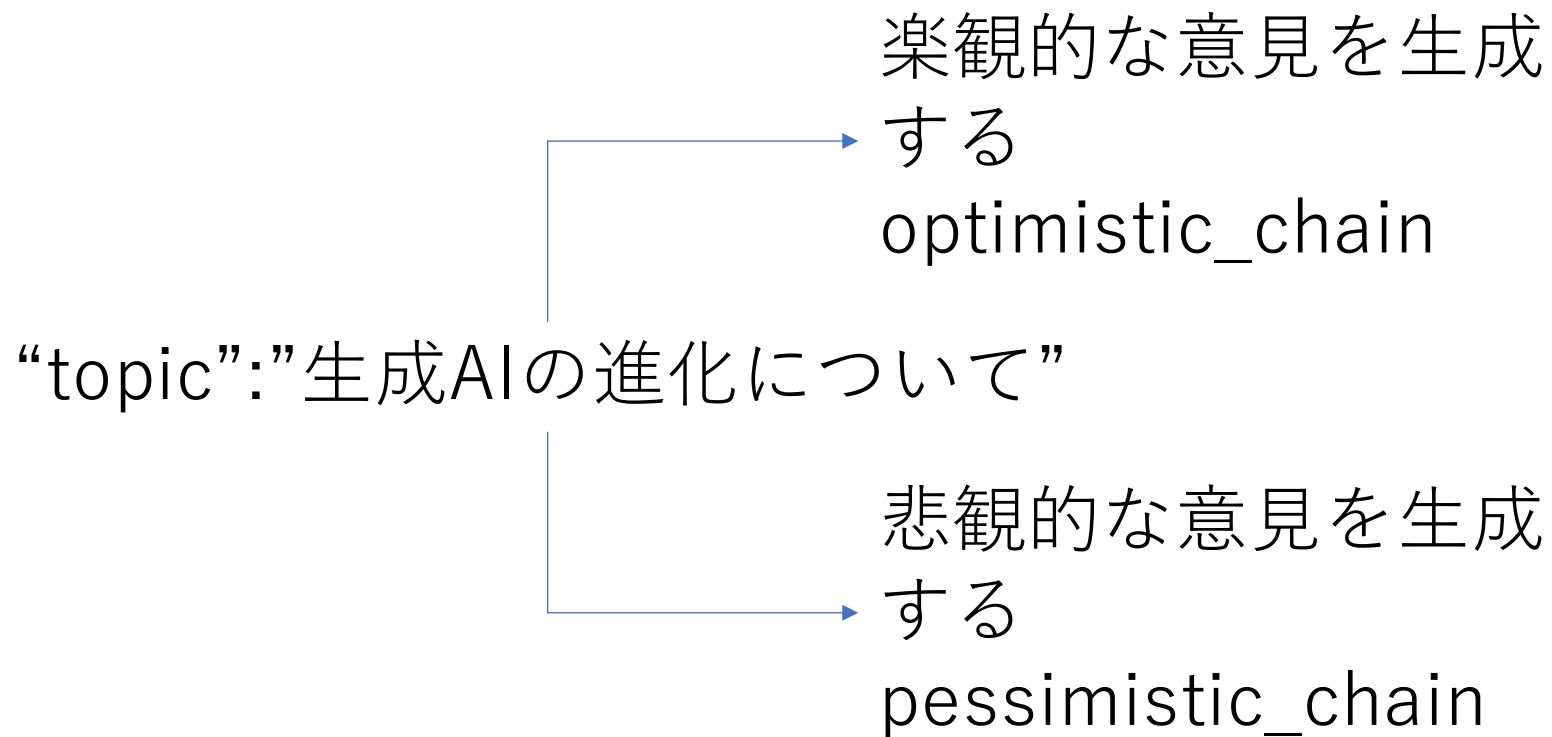


```
chain = prompt | model | output_parser | upper (入力 : str)
```

LCELの実践的な実装:RunnableParallel

RunnableParallel—複数のRunnableを並列に繋げる

LCELの実践的な実装:RunnableParallel

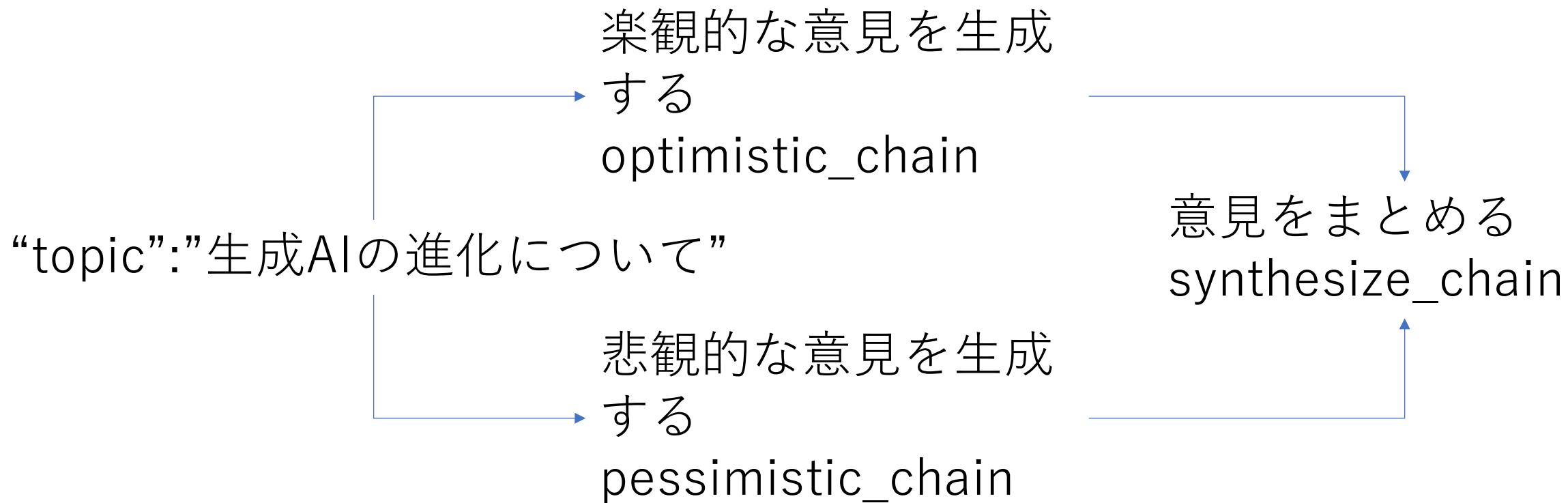


LCELの実践的な実装:RunnableParallel

```
1 import pprint
2 from langchain_core.runnables import RunnableParallel
3
4 parallel_chain = RunnableParallel(
5     {
6         "optimistic_opinion": optimistic_chain,
7         "pessimistic_opinion": pessimistic_chain,
8     }
9 )
10
11 output = parallel_chain.invoke({"topic": "生成AIの進化について"})
12 pprint.pprint(output)
```

```
{'optimistic_opinion': '生成AIの進化は本当に素晴らしいですね！技術が進むことで、私たちの生活がより便利で豊か  
'pessimistic_opinion': '生成AIの進化は確かに目覚ましいものがありますが、その裏には多くの懸念が潜んでいます。'}
```

LCELの実践的な実装:RunnableParallel



LCELの実践的な実装:RunnableParallel

```
3 synthesize_prompt = ChatPromptTemplate.from_messages(  
4     [  
5         ("system", "あなたは客観的AIです。2つの意見をまとめてください。"),  
6         ("human", "楽観的意見: {optimistic_opinion}と悲観的意見: {pessimistic_opinion}"),  
7     ]  
8 )  
9  
10 synthesize_chain = (  
11     RunnableParallel(  
12         {  
13             "optimistic_opinion": optimistic_chain,  
14             "pessimistic_opinion": pessimistic_chain,  
15         }  
16     )  
17     | synthesize_prompt  
18     | model  
19     | output_parser  
20 )  
21  
22 output = synthesize_chain.invoke({"topic": "生成AIの進化について"})  
23 print(output)
```

生成AIの進化については、楽観的な意見と悲観的な意見が存在します。楽観的な見方では、生成AIが私たちの生活を便利で
一方で、悲観的な見方では、生成AIの進化が仕事の喪失や情報の信頼性の低下を引き起こすリスクがあることが指摘されて
総じて、生成AIの進化には多くの可能性と同時にリスクが伴い、私たちがどのようにこの技術を活用するかが重要であると

LCELの実践的な実装:RunnableParallel

```
synthesize_chain = (
```

```
{
```

↓ dictのkeyがstrで、値がRunnableは自動的にRunnableParallelに変換

```
    "optimistic_opinion": optimistic_chain,
```

```
    "pessimistic_opinion": pessimistic_chain,
```

```
}
```

```
| synthesize_prompt
```

```
| model
```

```
| output_parser
```

```
)
```

RunnableLambdaとの組み合わせ※**難しい**

Itemgetter : Python標準ライブラリで提供されている関数
dictなどから値を取り出す関数を簡単に作る

RunnableLambdaとの組み合わせ

```
from operator import itemgetter
synthesize_prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "あなたは客観的AIです。{topic}について2つの意見をまとめてください。"),
        ("human", "楽観的意見: {optimistic_opinion}¥n悲観的意見: {pessimistic_opinion}", ),
    ]
)
synthesize_chain = (
    {
        "optimistic_opinion": optimistic_chain,
        "pessimistic_opinion": pessimistic_chain,
        "topic": itemgetter("topic"),
    }
    | synthesize_prompt | model | output_parser
)
output = synthesize_chain.invoke({"topic": "生成AIの進化について"})
print(output)
```

The diagram illustrates the data flow in the RunnableLambda chain. Yellow arrows show the following paths:

- From the `topic` input in the `synthesize_chain` dictionary to the `optimistic_opinion` and `pessimistic_opinion` outputs.
- From the `optimistic_opinion` and `pessimistic_opinion` outputs to the `synthesize_prompt` step.
- From the `synthesize_prompt` step to the `model` step.
- From the `model` step to the `output_parser` step.

LCELの実践的な実装:RunnableParallel

RunnableParallel—複数のRunnableを並列に繋げる

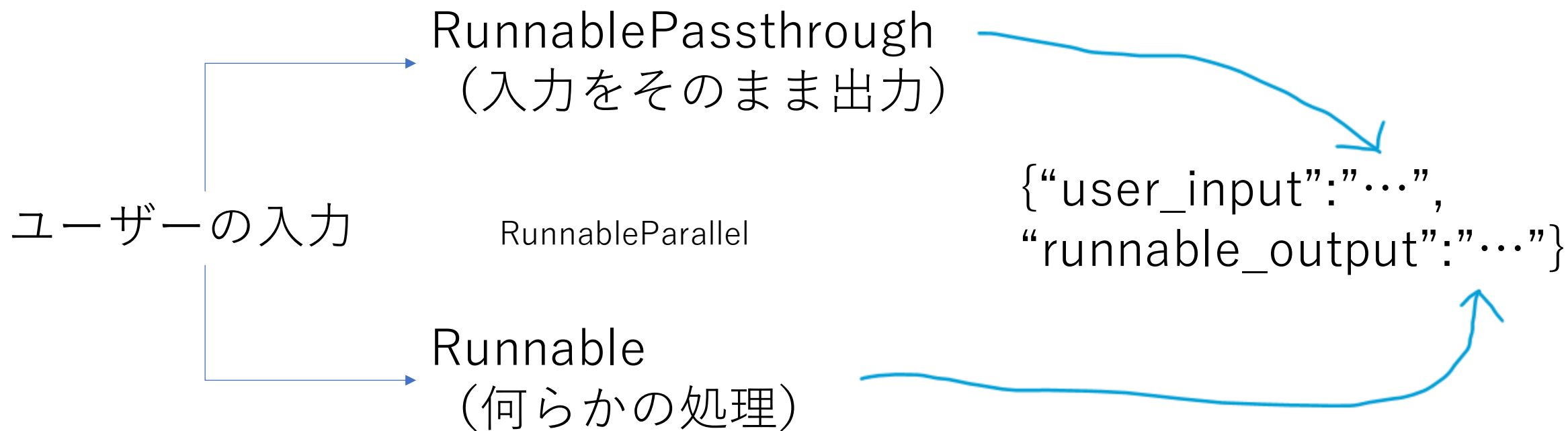


Chainの幅を広げることができる

LCELの実践的な実装:RunnablePassthrough

RunnablePassthrough — 入力をそのまま出力する

LCELの実践的な実装:RunnableParallel




LCELの実践的な実装:RunnablePassthrough

```
prompt = ChatPromptTemplate.from_template(
    "¥ 以下の文脈だけを踏まえて質問に回答してください。
    文脈: "" {context} "" 質問: {question} ")
model = ChatOpenAI(model_name="gpt-4o-mini", temperature=0)

retriever = TavilySerchAPIRetriever(k=3)

chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt | model | StrOutputParser()
)
output = chain.invoke("東京の今日の天気は？")
print(output)
```

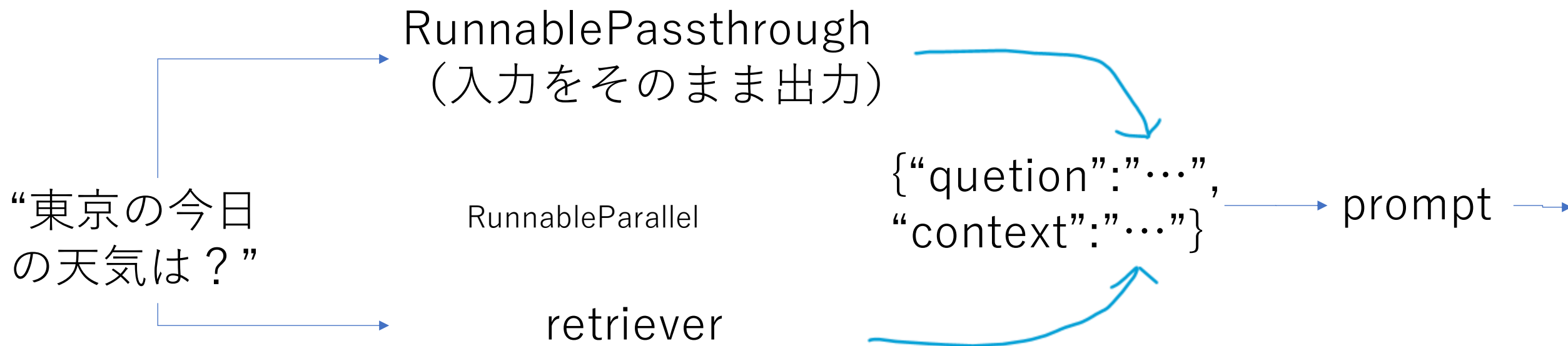


LCELの実践的な実装:RunnablePassthrough

```
7 prompt = ChatPromptTemplate.from_template("""  
8 以下の文脈だけを踏まえて質問に回答してください。  
9  
10 文脈: """  
11 {context}  
12 """  
13  
14 質問: {question}  
15 """)  
16 model = ChatOpenAI(model_name="gpt-4o-mini", temperature=0)  
17 retriever = TavilySearchAPIRetriever(k=3)  
18  
19 chain = (  
20     {"context": retriever, "question": RunnablePassthrough()}  
21     | prompt  
22     | model  
23     | StrOutputParser()  
24 )  
25  
26 output = chain.invoke("東京の今日の天気は？")  
27 print(output)
```

東京の今日、2025年2月8日（土）の天気は晴れで、最高気温は9℃、最低気温は-1℃です。降水確率は10%となっています。また、北西の風がやや強く吹く予想です。

LCELの実践的な実装:RunnableParallel



LCELの実践的な実装:RunnablePassthrough

RunnablePassthroughのクラスメソッド、
Runnableのインスタンスメソッド

assign : RunnableParallelの出力に値を追加する

Runnableのインスタンスメソッド

pick : RunnableParallelやRunnableの一部をピックアップする

LCELの実践的な実装:RunnablePassthrough

```
chain =  
    { "question": RunnablePassthrough(), "context": retriever, }  
    | RunnablePassthrough.assign(answer=prompt | model |  
    StrOutputParser())  
  
output = chain.invoke("東京の今日の天気は？")  
pprint.pprint(output)
```

LCELの実践的な実装:RunnablePassthrough

```
1 chain = {
2     "question": RunnablePassthrough(),
3     "context": retriever,
4 } | RunnablePassthrough.assign(answer=prompt | model | StrOutputParser())
5
6 output = chain.invoke("東京の今日の天気は？")
7 pprint(output)
```

```
{
  'answer': '東京の今日、2月8日（土）の天気は晴れで、最高気温は9℃、最低気温は-1℃です。降水確率は10%となっています。北西の風がやや強く吹いています。',
  'context': [
    Document(metadata={'title': '東京都の天気 - 日本気象協会 tenki.jp', 'source': 'https://tenki.jp/forecast/3/16/', 'score': 0.8875269, 'image_urls': []}),
    Document(metadata={'title': '東京（東京）の天気 - Yahoo!天気・災害', 'source': 'https://weather.yahoo.co.jp/weather/jp/13/4410.html', 'score': 0.734139, 'image_urls': []}),
    Document(metadata={'title': '東京の天気 - ウェザーニューズ', 'source': 'https://weathernews.jp/onebox/tenki/tokyo/', 'score': 0.734139, 'image_urls': []})
  ],
  'question': '東京の今日の天気は？'
}
```

LCELの実践的な実装:RunnablePassthrough

```
chain = (  
    RunnableParallel(  
        {  
            "question": RunnablePassthrough(),  
            "context": retriever,  
        }  
    )  
    .assign(answer=prompt | model | StrOutputParser())  
    .pick(["context", "answer"])  
)
```

```
output = chain.invoke("東京の今日の天気は？")  
pprint.pprint(output)
```


LCELの実践的な実装:RunnablePassthrough

```
4 chain = (  
5     RunnableParallel(  
6         {  
7             "question": RunnablePassthrough(),  
8             "context": retriever,  
9         }  
10    )  
11    .assign(answer=prompt | model | StrOutputParser())  
12    .pick(["context", "answer"])  
13 )  
14  
15 output = chain.invoke("東京の今日の天気は?")  
16 pprint(output)
```

```
{  
  'answer': '東京の今日、2月8日（土）の天気は晴れで、最高気温は9℃、最低気温は-1℃です。降水確率は10%となっています。北西の風がやや強く吹いています。',  
  'context': [  
    Document(metadata={'title': '東京都の天気 - 日本気象協会 tenki.jp', 'source': 'https://tenki.jp/forecast/3/16/', 'score': 0.8875269, 'image': 'https://tenki.jp/images/forecast/3/16/tenki.jpg', 'url': 'https://tenki.jp/forecast/3/16/'}, content='東京都の天気 - 日本気象協会 tenki.jp'),  
    Document(metadata={'title': '東京（東京）の天気 - Yahoo!天気・災害', 'source': 'https://weather.yahoo.co.jp/weather/jp/13/4410.html', 'score': 0.734139, 'image': 'https://weather.yahoo.co.jp/weather/jp/13/4410.html'}, content='東京（東京）の天気 - Yahoo!天気・災害'),  
    Document(metadata={'title': '東京の天気 - ウェザーニュース', 'source': 'https://weathernews.jp/onebox/tenki/tokyo/', 'score': 0.734139, 'image': 'https://weathernews.jp/onebox/tenki/tokyo/'}, content='東京の天気 - ウェザーニュース')  
  ]  
}
```

LCELの実践的な実装:RunnablePassthrough

RunnablePassthrough — 入力をそのまま出力する



Chain内の入力、出力を操作して幅を広げる

まとめ

LangCain Expression Language(LCEL)とは
LLMへのプロンプトを多彩、柔軟にデザインできる機能
である

まとめ

LangCain Expression Language(LCEL)の特徴

- ・ 各コンポーネントをライブラリのように使うだけでも便利
- ・ LangSmithのトレースとの連携などをより有効活用できる
- ・ LCELを使いこなすにはライブラリだけでなく、体系的に学ぶことが重要
- ・ LangCainを使うからといって、全ての処理をLCELで記述しなくてもいい（通常のプログラミング）

使ってみてください。

おわり