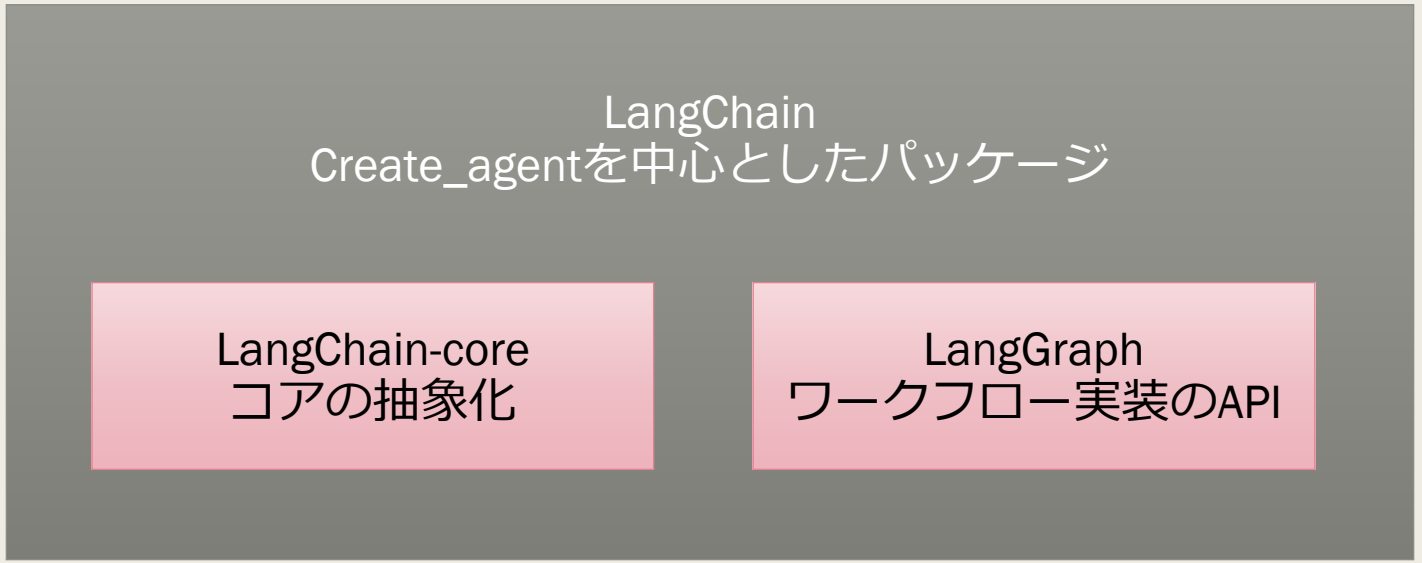
LANGCHAIN V1.2 概要

エージェント開発を支える共通機能の整理

LangChain v1.2の概要

簡単にエージェントをつくりたい：LangChainだけで実装
ワークフローをつくりたい： LangGraphも利用する



Langchain v1.2のまとめ

- create_react_agent → create_agentに変更
- エージェントのインプットとアウトプットを構造化
- Middleware機能の追加
- 機能のフェードアウト
  - *Prompt template/Output parser/LCEL*　が公式ページ例からなくなった
- LangGraphでLocalServerにできる
- LangChainの脆弱性

エージェントのインプットとアウトプットを構造化

- 出力（特に「構造化出力 / structured output」）の取り扱いが大きく改善され、モデルによる構造化データ生成がメインループで直接行われるよう
  - ①**データ損失防止**：出力ミスによる情報欠落のリスクが低減
  - ②**直接利用可能な形式**：解析不要でデータとしてそのまま扱える
  - ③**バリデーションによる安心感**：スキーマに沿った出力なので、型に合わない結果はエラーとして検出される
- Standard Content Blocks（標準コンテンツブロック）」という考え方により、モデルに関係なく reasoning（思考部分）、text（自然言語出力）、tool outputs（ツールからの出力）などを構造化して扱う仕組みが整備

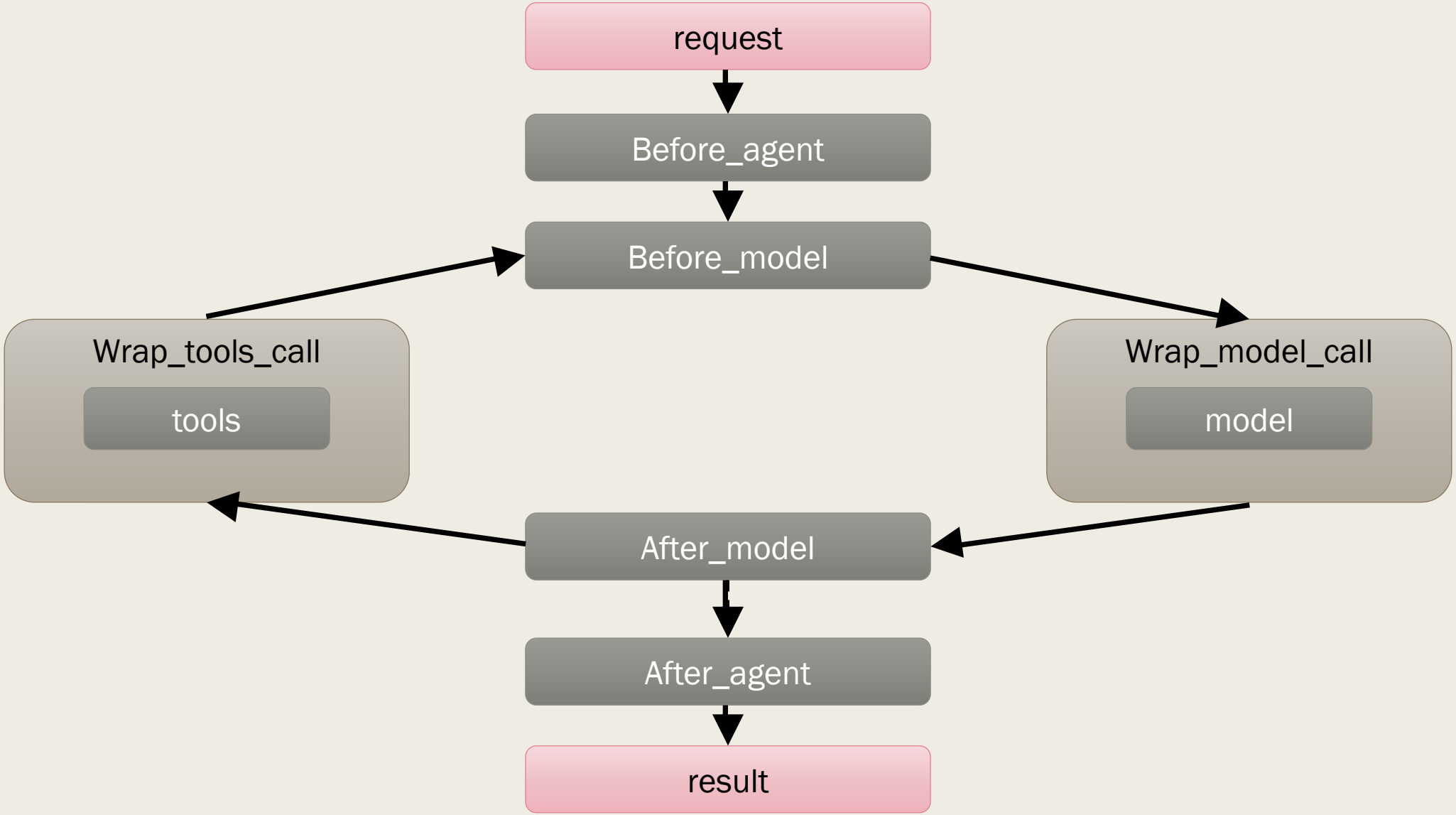
structured outputについて

- **ProviderStrategy**はモデルプロバイダ側でネイティブに対応している場合に有効で、OpenAIやGroKのような一部モデルが直接スキーマ対応して出力
- **ToolStrategy**はモデルがネイティブ対応しないときに用いられ、LangChainが内部的にツール呼び出し戦略で構造化出力
- 通常、schemaを直接指定するだけでモデルが対応すればProviderStrategyが自動的に使われ、対応モデルがない場合はToolStrategyにフォールバックする

Built-in Middleware とは？

- LangGraph 上でエージェント／ワークフローを実行するときに、
  モデル呼び出しやツール呼び出しの「前後」に共通処理を挟み込むための仕組み
- 各ノードのロジックを書き換えずに、横断的な関心事を後付けできる：
  - コンテキスト管理（要約・編集・古い情報のクリアなど）
  - ガバナンス／安全性（*Human-in-the-loop*、*PII* マスキング、モデレーションなど）
  - 信頼性／コスト管理（リトライ、フォールバック、回数制限、プロンプトキャッシュなど）
  - 開発者向け機能（シェル実行、ファイル検索、ツール選択・エミュレーションなど）
- 使い方のイメージ：
  - *graph* 実行時に *middleware=[...]* を指定すると、対象イベントごとに自動で処理が走る

Middlewareの概要図



Built-in Middleware の全体像

- 目的別カテゴリ
  - *コンテキスト・タスク管理*
  - *ガバナンス・安全性・コンプライアンス*
  - *信頼性・コスト最適化*
  - *開発者生産性・ツール連携*
- Provider 別
  - *Provider-agnostic：ほとんどのミドルウェア（Summarization, HITL, Call limit, Fallback, PII など）*
  - *Anthropic 向け：Prompt caching, Bash tool, Text editor, Memory, File search など*
  - *OpenAI 向け：Content moderation（OpenAIModerationMiddleware）*

コンテキスト・タスク管理系⭐

- SummarizationMiddleware ⭐
  - *会話履歴がコンテキスト上限に近づいたら、古いメッセージを自動要約*
  - *直近メッセージは保持しつつ、過去を「要約１メッセージ」に圧縮*
- ContextEditingMiddleware（例: ClearToolUsesEdit）
  - *古いツール実行結果だけを削除／プレースホルダ置換してトークン削減*
- TodoListMiddleware
  - *ToDo リスト作成・更新機能を付与し、マルチステップタスクを管理*
- （Anthropic）Claude Memory / Text Editor / State File Search
  - *長期メモリやテキスト編集・仮想ファイル検索で長期タスクやコード編集をサポート*

ガバナンス・安全性・コンプライアンス系 ⭐

- HumanInTheLoopMiddleware ⭐
  - *ツール実行前に処理を一時停止し、「承認・編集・却下」を人が判断*
  - *DB 書き込み、メール送信、決済など高リスク操作のガードレール*
- PIIMiddleware ⭐
  - *メールアドレスやクレカ番号などの PII を検出し、マスク・伏字・ブロックなどで処理*
  - *入力・出力・ツール結果のどこに適用するかを制御可能*
- OpenAIModerationMiddleware（OpenAI 専用）
  - *OpenAI モデレーション API で入力・出力・ツール結果をチェックし有害コンテンツを制御*

信頼性・コスト最適化系 ⭐

- ModelFallbackMiddleware
  - *メインモデル失敗時に別モデルへ自動フォールバック*
  - *モデル障害時のレジリエンス向上・コスト最適化に有効*
- ToolRetryMiddleware ⭐
  - *ツール呼び出し失敗時に自動リトライ（指数バックオフなどを設定可能）*
  - *ネットワーク一時障害などに強いエージェントを構築*
- ModelCallLimit / ToolCallLimit ⭐
  - *コスト上限や API 呼び出し回数の SLO を守るための制御*
- （Anthropic）PromptCachingMiddleware
  - *長いシステムプロンプトやツール定義をキャッシュし、レイテンシとコストを削減*

開発者生産性・ツール連携系 ⭐

- LLMToolSelectorMiddleware
  - *サブLLM で「どのツールを使うべきか」を選別し、本体モデルには絞り込んだツールだけ渡す*
  - *トークン削減・精度向上に寄与*
- LLMToolEmulator
  - *実ツールを呼ばずに、LLM が「ツール結果っぽいレスポンス」を生成（モック）*
  - *外部API 未実装・コスト高ツールの代替として開発初期に便利*
- ShellToolMiddleware
  - *永続シェルセッションをエージェントから操作可能（実行ポリシーで権限制御）*
- FilesystemFileSearchMiddleware ⭐
  - *ファイルシステムに対するGlob / Grep をツールとして提供し、コードベースやドキュメント検索に有効*
- （Anthropic）Text Editor / StateFileSearch
  - *Claude のテキスト編集と仮想ファイル検索で大規模リファクタリングを支援*

Provider 別 Middleware 一覧

- Provider-agnostic（どの LLM でも利用可能）
  - *Summarization / Human-in-the-loop*
  - *Model call limit / Tool call limit / Model fallback*
  - *PII detection / To-do list / LLM tool selector / Tool retry / LLM tool emulator*
  - *Context editing / Shell tool / File search*
- Anthropic 専用
  - *Prompt caching / Bash tool / Text editor / Memory / File search*
- OpenAI 専用
  - *Content moderation（OpenAIModerationMiddleware）*

自社エージェントでの活用パターン例

- 社内 Q&A / 社内ポータルボット
  - *Summarization + ContextEditing：長時間利用でも履歴維持*
  - *PIIMiddleware / OpenAIModeration：コンプライアンス対応*
- 業務自動化エージェント（メール送信、チケット登録など）
  - *HumanInTheLoop：高リスク操作の事前確認*
  - *ToolCallLimit / ToolRetry：外部 API 安全利用*
- 開発者向けコーディングエージェント
  - *FilesystemFileSearch + ShellTool：コード検索・テスト実行*
  - *（Anthropic）Text Editor / Memory：リファクタリングと継続的作業*

カスタム middleware

- [@before_model]などのデコレーターが用意されていて呼び出したいところの値を設定した関数を作成するだけ
- クラスで実装する場合はAgentMiddlewareを継承してbefore_modeなどを実装

機能のフェードアウト（Prompt template / Output parser / LCEL）

- 言いたいこと（結論）
  - これらは「廃止」ではなく、公式ドキュメントの“入口の作例”で主役ではなくなったため、フェードアウトして見えます。いまの入口は *Agent*（`*create_agent*`）中心です。
- 何が変わったか
  - 以前の「*PromptTemplate* → *LCEL*（`|`合成）→ *OutputParser*」型の例より、*v1* 系は `*create_agent*` + *middleware* + *structured output* の導線が前面に出ます。
  - 構造化出力は、プロンプトで*JSON*を書かせてパースする方式（*prompted output*）が非推奨になり、`*response_format*` での *prompted output* はサポート対象外と明記されています。

古い記事やサンプルで PromptTemplate/LCEL/OutputParser を見てもまだりようできるので焦らなくてOK。ただし、公式の推奨は Agent（create_agent）+ middleware + structured output に移っている。

LangGraphでLocalServerにできる

- 何ができる？
  - `langgraph dev` で *Agent Server* をローカル起動できます（開発・テスト用の *in-memory* モード）。 *([LangChain Docs][6])*
- 何が嬉しい？
  - ローカルに *API* サーバが立つので、外部ツールや*UI*から叩ける形で動作確認・デモがしやすい（*/docs* が提供される想定）。
- - 注意点
  - `langgraph dev` は *in-memory*で、開発・テスト向け。本番は永続ストレージ等を伴うデプロイが前提

LangChainの脆弱性

- 対象の脆弱性（2025年12月公開）
  - *Python（langchain-core）: CVE-2025-68664 / GHSA-c67j-w6g6-q2cm (GitHub)*
  - *JavaScript（@langchain/core / langchain）: CVE-2025-68665 / GHSA-r399-636x-v7f6 (GitHub)*
- 脆弱性の種類（要点）
  - *Serialization Injection*（シリアライズ注入）：シリアライズ時に、ユーザー由来の自由形式データ内の *'lc'* キー を適切にエスケープできず、後段の *load() / loads()*（*JS*は *load()*）で *LangChain*オブジェクトとして誤解釈され得る。 *(GitHub)*
- 攻撃が成立する典型パターン
  - *LLM*出力やユーザー入力に混入したデータ（例：*metadata, additional_kwargs, response_metadata*）がシリアライズされ、後でデシリアライズされる経路で悪用される。 *(GitHub)*

LangChainの脆弱性

- 環境変数シークレットの漏えい：secrets_from_env（Python）や secretsFromEnv（JS）が有効だと、細工された構造により 環境変数から秘密情報を読み出しされ得る。(GitHub)
- （条件付きで）クラスのインスタンス化による副作用：
  - Python：許可された信頼名前空間（langchain_core 等）内の Serializable サブクラスが生成され、__init__ の副作用（ネットワークアクセス等）に繋がり得る。 (GitHub)
- 影響を受ける条件（代表例）
  - *Python*：*astream_events(version="v1")* や *Runnable.astream_log()* が内部で該当シリアライズを使う（*v2*は非該当と明記）。 *(GitHub)*
  - *Python*：*dumps() / dumpd()* を 非信頼データに対して使い、その結果を *load() / loads()* で復元する。 *(GitHub)*

LangChainの脆弱性

- 影響バージョンと修正版
  - ***Python（pip: langchain-core）***
    - 影響あり：>= 1.0.0, < 1.2.5
    - 影響あり：< 0.3.81
    - 修正済み：1.2.5 / 0.3.81
  - ***JavaScript（npm: @langchain/core）***
    - 影響あり：>= 1.0.0, < 1.1.8
    - 影響あり：< 0.3.80
    - 修正済み：1.1.8 / 0.3.80
  - ***JavaScript（npm: langchain）***
    - 影響あり：>= 1.0.0, < 1.2.3
    - 影響あり：< 0.3.37