OpenClaw チュートリアル

オープンソース パーソナル AI アシスタント

導入から実践まで

河合勝彦

2026 年 2 月 22 日

概要

OpenClaw は、ローカル環境で動作するオープンソースのパーソナル AI アシスタントである。WhatsApp・Telegram・Discord・Slack をはじめとする多数のチャットプラットフォームと統合し、日常のタスク自動化を実現する。本チュートリアルでは、インストールから基本設定、チャンネル連携、スキルの活用、そして高度な自動化までを段階的に解説する。

目次

1 OpenClaw とは 4
1.1 アーキテクチャ概要 4
1.2 Pi エージェントランタイム 4

2 動作環境の準備 5
2.1 前提条件 5
2.2 Node.js のインストール（未導入の場合） 6

3 インストール 6
3.1 方法 1：ワンライナー（推奨） 6
3.2 方法 2：npm によるインストール 6
3.3 方法 3：ソースからビルド（開発者向け） 6

4 初期セットアップ 7
4.1 オンボーディングウィザードの実行 7
4.2 Gateway の起動確認 7
4.3 Dashboard（管理画面）へのアクセス 7

5 設定ファイル 8
5.1 基本設定の例 8
5.2 主要な設定項目 8
5.3 環境変数 8

6 チャンネルの設定 9
6.1 対応チャンネル一覧 9
6.2 Telegram の設定例 9
6.3 WhatsApp の設定例 10
6.4 DM セキュリティ（ペアリングポリシー） 10

7 基本的な使い方 10
7.1 チャットでの対話 10
7.2 CLI からの操作 11
7.3 Dashboard からの操作 11

8 スキルの管理 11
8.1 スキルの種類と読込順序 11
8.2 ClawHub からのインストール 12
8.3 スキルの設定 12
8.4 カスタムスキルの作成 13

9 ワークスペースとメモリ 14
9.1 ワークスペース構造 14
9.2 SOUL.md によるパーソナリティ設定 14
9.3 メモリシステム 14

10 自動化機能 14
10.1 Cron ジョブ 15
10.2 Webhook 15
10.3 Gmail 連携 15

11 ブラウザ操作 15

12 リモートアクセス 15
12.1 Tailscale によるアクセス 16
12.2 SSH トンネル 16

13 セキュリティ 16
13.1 基本方針 16
13.2 サンドボックスモード 16
13.3 昇格モード 17

14 トラブルシューティング 17
14.1 診断コマンド 17
14.2 よくある問題と対処 17

15 CLI コマンドリファレンス 17

16 実践例：日常タスクの自動化 18
16.1 例 1：朝のブリーフィング自動化 18
16.2 例 2：GitHub Issue の自動トリアージ 19
16.3 例 3：ファイル管理 19

17 ケーススタディ：Telegram を使った株式トレーディング戦略 19
17.1 概要と目的 19
17.2 インフラ構成 19
17.3 ワークスペースの構成 20
17.4 元記事のアーキテクチャ解説に関する補足 22
17.5 この事例から学べること 22

18 モバイルアプリ・デスクトップアプリ 23
18.1 音声機能 23

19 まとめ 23

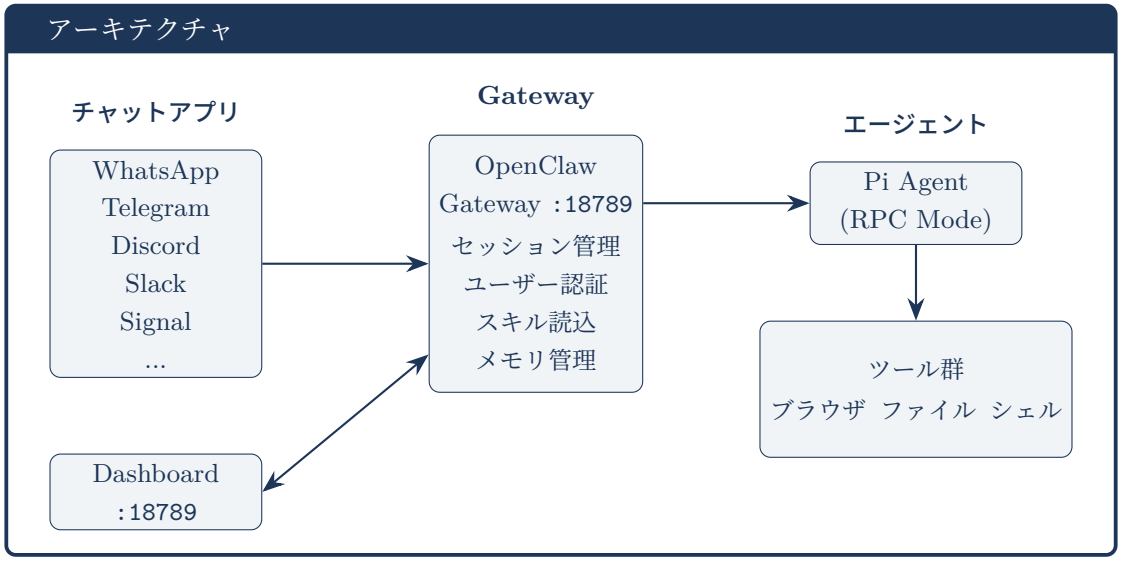
1　OpenClaw とは

OpenClaw[*1]は、MIT ライセンスで公開されているオープンソースのパーソナル AI アシスタントである。以下の特徴を持つ。

- **ローカル実行**：自分のマシン上で動作し、データのプライバシーを確保
- **マルチチャンネル対応**：WhatsApp、Telegram、Discord、Slack、Signal、iMessage、Google Chat、Microsoft Teams、Matrix、LINE など 15 以上のプラットフォームに対応
- **永続メモリ**：会話の文脈やユーザーの好みを学習し、24 時間稼働
- **システムアクセス**：ブラウザ操作、ファイル読み書き、シェルコマンド実行が可能
- **拡張性**：ClawHub のスキルマーケットプレイスを通じてコミュニティ製スキルを導入可能
- **マルチモデル対応**：Anthropic Claude、OpenAI GPT、ローカルモデルを選択可能

1.1　アーキテクチャ概要

OpenClaw の中核は **Gateway** と呼ばれるプロセスである。Gateway は WebSocket ベース（`ws://127.0.0.1:18789`）で動作し、セッション管理、チャンネル接続、ツール呼び出し、イベント処理を一元的に制御する。



1.2　Pi エージェントランタイム

図中の「Pi Agent」は、OpenClaw の頭脳にあたる**エージェントランタイム**である。OpenClaw は `@mariozechner/pi-agent-core`（通称 **pi-mono**）ライブラリを基盤として、LLM への問い合わせ、ツール呼び出し、応答生成といったエージェントループを実行する。

1.2.1　pi-mono とは

pi-mono は Mario Zechner 氏（`@mariozechner`）が開発した AI エージェント基盤ライブラリである。npm パッケージ名は `@mariozechner/pi-agent-core` で、以下の機能を提供する。

[*1] https://openclaw.ai/

- **モデル抽象化**：Anthropic Claude、OpenAI GPT、ローカルモデルなど複数の LLM プロバイダを統一的に呼び出すインターフェース
- **ツール実行フレームワーク**：LLM が生成するツール呼び出しリクエストを受け取り、対応するツールを実行して結果を返す仕組み
- **ストリーミング**：LLM 応答をバッチではなく逐次的にストリーミングし、ツール呼び出しもストリーム中に処理するパイプライン
- **エージェントループ**：「入力→モデル推論→ツール実行→出力」のサイクルを自動的に繰り返すランタイム

1.2.2　OpenClaw と pi-mono の関係

OpenClaw は pi-mono をそのまま使うのではなく、エージェントループとツール実行の中核部分だけを取り出し、独自のセッション管理・チャンネル接続・スキル機構で包み込んでいる。

統合のポイントは以下のとおりである。

- **埋め込み実行**：Pi を外部プロセスとして起動するのではなく、`createAgentSession()` API を呼び出して Gateway プロセス内にエージェントを埋め込む。これにより、セッションのライフサイクルとイベント処理を OpenClaw 側で完全に制御できる。
- **独自管理層**：セッション管理、サービス発見、ツール接続のルーティングは OpenClaw が担当する。pi-mono はあくまで「推論エンジン」としてのみ利用される。
- **マルチチャンネル対応**：Gateway が WhatsApp・Telegram・Discord 等の複数チャンネルを束ね、単一の pi-mono エージェントランタイムに接続するアーキテクチャになっている。

1.2.3　エージェントループの 4 ステップ

具体的には、エージェントランタイムは各ターンで以下の 4 ステップを実行する。

1. **セッション解決**：対話相手とコンテキストの特定
2. **コンテキスト組立**：メモリ、スキル、ツール情報の統合
3. **モデル応答のストリーミング**：LLM への問い合わせとツール呼び出しの逐次実行
4. **状態の永続化**：更新された会話履歴やメモリのディスク書き込み

NOTE

pi-mono が提供するのはステップ 2〜3 のモデル呼び出し・ツール実行の部分であり、ステップ 1 のセッション解決とステップ 4 の永続化は OpenClaw 独自の実装が担っている。

2　動作環境の準備

2.1　前提条件

- **OS**：macOS、Linux、または Windows（WSL2 経由）
- **Node.js**：バージョン 22 以上が必須
- **AI モデルの API キー**：Anthropic（Claude）または OpenAI（GPT）のいずれか

NOTE

Node.js のバージョンは `node --version` で確認できる。v22 未満の場合は https://nodejs.org/ から最新 LTS をインストールすること。

2.2　Node.js のインストール（未導入の場合）

nvm を使った Node.js のインストール

```
# nvm のインストール
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash

# シェルの再読み込み
source ~/.bashrc    # bash の場合
# source ~/.zshrc   # zsh の場合

# Node.js 22 のインストールと有効化
nvm install 22
nvm use 22
node --version      # v22.x.x と表示されれば OK
```

3　インストール

インストール方法は 3 つ用意されている。

3.1　方法 1：ワンライナー（推奨）

macOS / Linux

```
curl -fsSL https://openclaw.ai/install.sh | bash
```

Windows (PowerShell)

```
iwr -useb https://openclaw.ai/install.ps1 | iex
```

3.2　方法 2：npm によるインストール

```
npm install -g openclaw@latest
```

3.3　方法 3：ソースからビルド（開発者向け）

```
git clone https://github.com/openclaw/openclaw.git
cd openclaw
pnpm install
pnpm build
pnpm openclaw onboard --install-daemon
```

TIP

ソースビルドでは pnpm が必要である。npm install -g pnpm で導入できる。

4　初期セットアップ

4.1　オンボーディングウィザードの実行

インストール後、以下のコマンドで対話型セットアップウィザードを起動する。

```
openclaw onboard --install-daemon
```

ウィザードでは以下の項目を順に設定する。

1. **Gateway の設定**：ポート番号（デフォルト 18789）とセキュリティ
2. **ワークスペースの構成**：作業ディレクトリの場所
3. **AI モデルの選択**：Claude / GPT / ローカルモデル
4. **認証設定**：API キーまたは OAuth の設定
5. **チャンネルの接続**：最初のメッセージングプラットフォームの追加
6. **スキルのインストール**：初期スキルの選択

NOTE

オンボーディングは通常 10 分以内に完了する。各ステップは後からでも openclaw configure で変更可能である。

4.2　Gateway の起動確認

```
# Gateway のステータス確認
openclaw gateway status

# フォアグラウンドで起動（デバッグ時に便利）
openclaw gateway --port 18789 --verbose
```

4.3　Dashboard（管理画面）へのアクセス

```
openclaw dashboard
```

ブラウザで http://127.0.0.1:18789/ にアクセスすると、管理画面（Dashboard）が表示される。ここから以下の操作が可能である。

- セッションの監視と管理
- チャンネルの追加・設定
- スキルの有効化・無効化
- メモリの確認
- ログの閲覧

5　設定ファイル

OpenClaw の主要な設定は `~/.openclaw/openclaw.json` に記述する。

5.1　基本設定の例

~/.openclaw/openclaw.json

```
{
  "model": {
    "provider": "anthropic",
    "name": "claude-sonnet-4-6"
  },
  "agents": {
    "defaults": {
      "workspace": "~/.openclaw/workspace"
    }
  },
  "gateway": {
    "port": 18789
  },
  "skills": {
    "entries": {}
  }
}
```

5.2　主要な設定項目

キー	説明
`model.provider`	モデルプロバイダ（`anthropic`, `openai` など）
`model.name`	使用するモデル名
`gateway.port`	Gateway のポート番号
`agents.defaults.workspace`	ワークスペースのパス
`skills.entries`	スキルごとの設定（API キー等）
`skills.load.extraDirs`	追加のスキル読込ディレクトリ
`skills.load.watch`	スキルファイルの変更監視（デフォルト `true`）

5.3　環境変数

設定ファイルの代わりに環境変数でも制御できる。

変数名	説明
`OPENCLAW_HOME`	ホームディレクトリの上書き

変数名	説明
OPENCLAW_STATE_DIR	ステートディレクトリの上書き
OPENCLAW_CONFIG_PATH	設定ファイルパスの上書き

6　チャンネルの設定

OpenClaw の最大の特長は、日常的に使うチャットアプリを通じて AI アシスタントと対話できる点にある。本章では主要なチャンネルの設定方法を解説する。

6.1　対応チャンネル一覧

チャンネル	接続方式	備考
WhatsApp	WhatsApp Web	QR コード認証
Telegram	Bot API	BotFather でトークン取得
Discord	Bot API	Developer Portal で設定
Slack	Bot API	Slack App を作成
Signal	Signal CLI	電話番号認証
iMessage	BlueBubbles	macOS のみ
Google Chat	Workspace API	Google Cloud 設定
Microsoft Teams	Bot Framework	Azure 設定
Matrix	Bot SDK	ホームサーバー接続
LINE	Messaging API	LINE Developers で設定
IRC	IRC プロトコル	サーバー接続

6.2　Telegram の設定例

Telegram は最も簡単に設定できるチャンネルの一つである。

1. Telegram で `@BotFather` にメッセージを送信
2. `/newbot` コマンドでボットを作成
3. ボット名とユーザー名を設定
4. 発行された API トークンをコピー
5. OpenClaw に設定を追加：

CLI でチャンネルを追加

```
openclaw channels add telegram
# 対話型プロンプトに従い、API トークンを入力
```

または、設定ファイルに直接記述する。

openclaw.json にチャンネルを追加

```
{
  "channels": {
    "telegram": {
      "enabled": true,
      "token": "YOUR_BOT_TOKEN"
    }
  }
}
```

WARNING

API トークンは秘密情報である。Git リポジトリにコミットしないよう注意すること。環境変数での管理を推奨する。

6.3　WhatsApp の設定例

1. `openclaw channels add whatsapp` を実行
2. ターミナルに表示される QR コードを WhatsApp アプリでスキャン
3. 接続が確立されると、OpenClaw がメッセージを受信開始

NOTE

WhatsApp は WhatsApp Web プロトコルを使用するため、スマートフォン側の WhatsApp が起動している必要がある。

6.4　DM セキュリティ（ペアリングポリシー）

デフォルトでは、未知の送信者からの DM にはペアリングコードが要求される。これにより、不正なメッセージの処理を防止する。

ペアリングの管理

```
# ペアリングコードの生成
openclaw pairing create

# 特定のユーザーを許可リストに追加
openclaw pairing allow +819012345678
```

7　基本的な使い方

7.1　チャットでの対話

設定済みのチャンネル（Telegram、WhatsApp 等）からボットにメッセージを送信するだけで AI アシスタントと対話できる。

7.1.1　基本コマンド

チャット内で使えるスラッシュコマンドを以下に示す。

コマンド	説明
`/status`	セッションの状態表示
`/new` または `/reset`	セッションのリセット（コンテキストクリア）
`/think <level>`	推論の深さを設定（`low`, `medium`, `high`）
`/compact`	会話を要約して圧縮
`/restart`	Gateway の再起動

7.2　CLI からの操作

チャットアプリを使わず、CLI から直接操作することも可能である。

```
# メッセージの送信
openclaw message send --to +819012345678 --message "こんにちは"

# エージェントに直接質問
openclaw agent --message "明日の天気を教えて" --thinking high

# TUI（ターミナル UI）の起動
openclaw tui
```

7.3　Dashboard からの操作

`openclaw dashboard` で起動するウェブ UI からも操作できる。Dashboard では以下が可能である。

- リアルタイムのセッション監視
- WebChat を使った直接対話
- スキルの管理
- ログの閲覧とフィルタリング
- チャンネルの状態確認

8　スキルの管理

スキルは OpenClaw の機能を拡張するモジュールである。スキルは「ツールの組み合わせ方をエージェントに教えるテキストブック」であり、新しい権限を付与するものではない。

8.1　スキルの種類と読込順序

スキルは以下の 3 箇所から読み込まれる（優先度順）。

1. **ワークスペーススキル**：`<workspace>/skills/`（エージェント固有）

2. **マネージドスキル**：`~/.openclaw/skills/`（全エージェント共有）
3. **バンドルスキル**：OpenClaw 本体に同梱

8.2　ClawHub からのインストール

ClawHub[*2] はコミュニティ製スキルの公開レジストリである。

```
# スキルのインストール
clawhub install <skill-slug>

# インストール済みスキルの一覧
clawhub list

# 全スキルの更新
clawhub update --all

# スキルのアンインストール
clawhub uninstall <skill-slug>
```

CLI でスキルを管理

```
# 利用可能なスキルの確認
openclaw skills list --eligible

# スキルの詳細情報
openclaw skills info <skill-name>

# スキルの動作確認
openclaw skills check
```

8.3　スキルの設定

スキルごとの設定は `openclaw.json` の `skills.entries` に記述する。

スキル設定の例

```
{
  "skills": {
    "entries": {
      "spotify": {
        "enabled": true,
        "apiKey": "YOUR_SPOTIFY_API_KEY",
        "env": {
          "SPOTIFY_CLIENT_ID": "abc123",
          "SPOTIFY_CLIENT_SECRET": "secret456"
        }
      },
      "gmail": {
```

[*2] https://clawhub.com

```
        "enabled": true
      },
      "github": {
        "enabled": true,
        "apiKey": "ghp_xxxxxxxxxxxx"
      }
    }
  }
}
```

8.4　カスタムスキルの作成

独自のスキルを作成するには、ワークスペースの `skills/` ディレクトリに `SKILL.md` ファイルを配置する。

~/.openclaw/workspace/skills/my-skill/SKILL.md

```
---
name: my-custom-skill
description: 独自のカスタムスキル
user-invocable: true
metadata: {"openclaw":{"emoji":" ","requires":{"bins":["curl"]}}}
---

# My Custom Skill

このスキルは以下のことができます：

1. 特定の API からデータを取得する
2. 取得したデータを整形してユーザーに返す

## 使い方

ユーザーが「データを取得して」と言ったら、
以下の手順で処理してください：

1. curl を使って API にリクエストを送信
2. レスポンスを JSON としてパース
3. 必要な情報を抽出してユーザーに提示
```

NOTE

スキルのファイル変更は自動検知される（`skills.load.watch: true`）。新しいセッションで変更が反映される。

WARNING

ClawHub のサードパーティスキルは、インストール前に内容を確認すること。2026 年 2 月時点で悪意のあるスキルの報告例がある。

9　ワークスペースとメモリ

9.1　ワークスペース構造

デフォルトのワークスペースは `~/.openclaw/workspace/` に配置され、以下のファイルでエージェントの挙動をカスタマイズできる。

ファイル	役割
`AGENTS.md`	エージェントの定義と設定
`SOUL.md`	エージェントの性格・話し方の設定
`TOOLS.md`	利用可能なツールのドキュメント
`skills/`	カスタムスキルの格納ディレクトリ

9.2　SOUL.md によるパーソナリティ設定

`SOUL.md` を編集することで、アシスタントの応答スタイルを自由にカスタマイズできる。

SOUL.md の例

```
あなたは親切で丁寧な日本語アシスタントです。

- 敬語を使って応答してください
- 技術的な質問には具体的なコード例を添えてください
- 分からないことは正直に「分かりません」と答えてください
- ユーモアを交えつつ、正確な情報を提供してください
```

9.3　メモリシステム

OpenClaw は永続メモリを備えており、ユーザーの好み、過去の会話、よく使うパターンなどを自動的に記憶する。

```
# メモリの確認
openclaw memory

# メモリの管理（Dashboard からも可能）
openclaw dashboard
```

10　自動化機能

OpenClaw は高度な自動化機能を備えている。

10.1 Cron ジョブ

定期的なタスクを cron 式で設定できる。

```
# cron ジョブの管理
openclaw cron list
openclaw cron add "0 9 * * *" "今日の予定を教えて"
openclaw cron remove <job-id>
```

チャット内から設定することも可能である。

> 「毎朝 9 時にニュースの要約を送って」

10.2 Webhook

外部サービスからのイベントを受信して処理できる。

```
# Webhook の管理
openclaw hooks list
openclaw hooks add --url /my-webhook --action "通知を処理"
```

10.3 Gmail 連携

Gmail Pub/Sub を設定することで、メールの受信をトリガーにした自動処理が可能になる。

11 ブラウザ操作

OpenClaw は Chrome/Chromium を制御し、ウェブブラウジングやフォーム入力を自動化できる。

```
# ブラウザの起動
openclaw browser
```

チャットで以下のような依頼が可能である。

- 「`example.com` を開いてスクリーンショットを撮って」
- 「GitHub の Issue #42 の内容を確認して」
- 「このフォームに以下の内容を入力して送信して」

WARNING

ブラウザ操作は強力な機能であるため、サンドボックスモードでの実行を推奨する。

12 リモートアクセス

自宅や VPS 上の Gateway に外出先からアクセスする方法を説明する。

12.1 Tailscale によるアクセス

Tailscale[*3] を使えば、安全なリモートアクセスが簡単に実現できる。

```
# Tailscale Serve（tailnet 内のみ）
tailscale serve --bg 18789

# Tailscale Funnel（公開アクセス、パスワード認証必須）
tailscale funnel --bg 18789
```

12.2 SSH トンネル

SSH トンネルを使ったリモートアクセスも可能である。

```
# リモートサーバーの Gateway にトンネル接続
ssh -L 18789:localhost:18789 user@remote-server
```

13 セキュリティ

13.1 基本方針

- インバウンドの DM は常に「信頼できない入力」として扱う
- デフォルトのペアリングポリシー（dmPolicy="pairing"）を維持する
- サードパーティスキルのインストール前に内容を確認する
- API キーやトークンは環境変数で管理し、設定ファイルに直接記述しない

13.2 サンドボックスモード

信頼度の低いツールやスキルを実行する際は、サンドボックス（Docker コンテナ）内で実行できる。

サンドボックスの設定例

```
{
  "agents": {
    "defaults": {
      "sandbox": {
        "docker": {
          "enabled": true,
          "setupCommand": "apt-get update && apt-get install -y curl"
        }
      }
    }
  }
}
```

[*3] https://tailscale.com/

13.3 昇格モード

特定のセッションで管理者権限が必要な場合、昇格モードを有効にできる（許可リストに登録済みのユーザーのみ）。

チャット内での昇格モード切替

```
/elevated on    # 昇格モード有効化
/elevated off   # 昇格モード無効化
```

14 トラブルシューティング

14.1 診断コマンド

```
# システム診断
openclaw doctor

# ヘルスチェック
openclaw health

# ログの確認
openclaw logs

# 詳細ログ付きで Gateway を起動
openclaw gateway --port 18789 --verbose
```

14.2 よくある問題と対処

症状	対処法
Gateway が起動しない	`openclaw doctor` で診断。ポート競合を確認
チャンネルが接続できない	API トークンの有効性を確認。`openclaw channels` で状態を確認
スキルが読み込まれない	`openclaw skills list --eligible` で確認。`SKILL.md` の `requires` 条件を確認
応答が遅い	`/think low` で推論レベルを下げる。モデルのフェイルオーバー設定を確認
メモリが反映されない	`/new` でセッションをリセット

15 CLI コマンドリファレンス

主要な CLI コマンドの一覧を示す。

コマンド	説明
`openclaw onboard`	オンボーディングウィザードの起動
`openclaw gateway`	Gateway の起動・管理
`openclaw gateway status`	Gateway の状態確認
`openclaw dashboard`	Dashboard（Web UI）の起動
`openclaw tui`	ターミナル UI の起動
`openclaw agent --message`	エージェントへの直接メッセージ
`openclaw message send`	メッセージの送信
`openclaw channels`	チャンネルの管理
`openclaw skills`	スキルの管理
`openclaw configure`	設定の変更
`openclaw doctor`	システム診断
`openclaw health`	ヘルスチェック
`openclaw logs`	ログの表示
`openclaw memory`	メモリの管理
`openclaw cron`	Cron ジョブの管理
`openclaw hooks`	Webhook の管理
`openclaw browser`	ブラウザの管理
`openclaw update`	OpenClaw の更新
`openclaw update --channel beta`	ベータ版への切替
`openclaw uninstall`	アンインストール
`openclaw reset`	システムのリセット

16　実践例：日常タスクの自動化

ここでは OpenClaw を使った具体的なユースケースを紹介する。

16.1　例 1：朝のブリーフィング自動化

毎朝 7 時に、天気予報・本日の予定・未読メールの要約を Telegram に送信する設定を行う。

1. Telegram チャンネルを設定（第**??**節を参照……前述の通り）
2. 必要なスキル（天気、カレンダー、Gmail）をインストール
3. Cron ジョブを設定

```
clawhub install weather
clawhub install google-calendar
clawhub install gmail
openclaw cron add "0 7 * * *" \
  "おはようございます。今日の天気、予定、未読メールを要約してください"
```

16.2　例 2：GitHub Issue の自動トリアージ

Webhook を使って新規 Issue を自動的に分類する。

```
clawhub install github
openclaw hooks add --url /github-webhook \
  --action "新しい Issue をラベル付けして優先度を判定"
```

16.3　例 3：ファイル管理

チャットから直接ファイルの操作を指示する。

「*Downloads* フォルダの *PDF* ファイルを日付別に整理して」
「このスプレッドシートのデータを集計して結果を教えて」
「プロジェクトの *README.md* を更新して」

17　ケーススタディ：Telegram を使った株式トレーディング戦略

本節では、GMO インターネットグループの K.S. 氏（AI 研究開発室）がブログ記事[*4] で紹介した、OpenClaw と Telegram を組み合わせたペーパートレード運用の事例を取り上げる。

WARNING

本ケーススタディはペーパートレード（模擬取引）のみを扱っており、実際の金融取引を推奨するものではない。投資判断は自己責任で行うこと。

17.1　概要と目的

この事例では、OpenClaw エージェント「Trady」を VPS 上に構築し、Telegram 経由で米国株のトレーディング戦略の提案・分析・PDCA 改善をチャットベースで行う環境を実現している。
主なワークフロー：

1. 毎日の定時タスク（HEARTBEAT）で有望銘柄をスキャンし Telegram に送信
2. ユーザーがチャットで戦略を相談し、エージェントがリアルタイムに助言
3. 取引結果をメモリに記録し、次回以降の戦略改善に活用

17.2　インフラ構成

記事では、セキュリティ上の理由から普段使いの PC ではなく VPS（仮想専用サーバー）上に OpenClaw を構築することが強く推奨されている。

[*4] https://recruit.group.gmo/engineer/jisedai/blog/openclaw_stock_trading/

項目	内容
VPS プロバイダ	Hostinger（KVM 2 プラン）
スペック	2 vCPU / 8GB RAM / 100GB NVMe / 8TB 帯域幅
デプロイ方法	Docker 経由
チャンネル	Telegram（BotFather で Bot Token を取得）

セットアップの流れ：

VPS 上での OpenClaw セットアップ

```
# Docker コンテナの確認
docker ps

# コンテナ内に入る
docker exec -it <container_id> bash

# 初期設定
openclaw onboard

# Telegram チャンネルの追加
openclaw channel add

# Telegram ペアリングの承認
openclaw pairing approve telegram <Telegram_ID>
```

NOTE

元記事では `openclaw channel add`（単数形）が使われているが、公式 CLI では `openclaw channels add`（複数形）が正しいコマンドである。

17.3　ワークスペースの構成

トレーディング用途にカスタマイズされたワークスペース構成は以下のとおりである。

ワークスペースのディレクトリ構成

```
~/.openclaw/workspace/
├── AGENTS.md       # 行動規範（トレーディングルール）
├── SOUL.md         # 性格定義（分析スタイル）
├── USER.md         # ユーザー情報
├── IDENTITY.md     # エージェント名（"Trady"）
├── MEMORY.md       # 長期記憶（取引履歴・学習記録）
├── HEARTBEAT.md    # 定期チェックの設定
├── memory/         # 日次ログ
└── trades/         # 取引記録
```

17.3.1　HEARTBEAT.md による自動スキャン

HEARTBEAT.md を設定することで、指定した時間帯に自動的にタスクを実行させることができる。この事例では、平日 22:00 JST（米国市場のプレマーケット時間帯）に「カタリストスキャン」を自動実行している。

スキャン対象：

- S&P 500 銘柄の決算発表スケジュール
- アナリストによる格上げ・格下げ
- 主要ニュース
- プレマーケットで 3% 以上上昇している銘柄

HEARTBEAT.md の設定例

```
## Catalyst Scan

- Schedule: weekdays 22:00 JST
- Action: S&P 500銘柄から翌日の注目銘柄を抽出し、
  Telegramに推奨銘柄リストを送信する
- Criteria: 決算発表、格上げ、ニュース、
  プレマーケット上昇率3%以上
```

NOTE

元記事では HEARTBEAT の実行時間帯について「21:30–22:30 JST」と「22:00 JST」の二つの記述がある。HEARTBEAT.md では時間帯の範囲を指定し、実際の Cron ジョブは特定時刻（22:00）に発火する、という二段階の仕組みである。

17.3.2　SOUL.md による性格設定

エージェント「Trady」の SOUL.md では、以下のようなコア値が設定されている。

SOUL.md（トレーディングエージェント用）の抜粋

```
# Core Values
- 本当に役立つことを実行すること
- 意見を持つこと（曖昧な回答をしない）
- 見かけの親切ではなく実質的な支援を行う

# Trading Scope
- 米国株・日本株のリサーチとペーパートレード
- 財務助言は一切提供しない
```

17.3.3　MEMORY.md による学習の蓄積

MEMORY.md にはトレーディングの結果と学習事項が自動的に記録される。記事で公開されている第 1 週の結果は以下のとおりである。

指標	値
勝率	0%（全敗）
実現損失	–$450
保有ポジション	META（$675 → $639.77, –5.2%）
	TSLA（$427 → $417.44, –2.2%）
学習事項	「規律 > 確信」「寄付きエントリー禁止」「損切り必須」

このように、失敗からの学習をメモリに蓄積し、次週以降の戦略に反映させるサイクルが MEMORY.md を通じて実現されている。

17.4　元記事のアーキテクチャ解説に関する補足

元記事では OpenClaw のアーキテクチャを「5 層構造」として以下のように説明している。

1. Gateway（メッセージの受付・振り分け）
2. Input Events（メッセージ、Webhook、Cron、Heartbeat）
3. Agent Loop（メッセージをアクションと返答に変換）
4. Tools & Actions（ブラウザ、ファイル、API 呼び出し）
5. Persistent Memory（ローカル Markdown ファイルに保存）

NOTE

この「5 層」分類はブログ著者独自の整理であり、OpenClaw 公式の用語ではない。本チュートリアルの第 1 節で説明したとおり、公式アーキテクチャは Gateway・Pi Agent・ツール群の 3 コンポーネント構成である。また、元記事では Agent Loop を「6 ステップ」としているが、本チュートリアル 1.2.3 節で述べたとおり公式には 4 ステップ（セッション解決→コンテキスト組立→モデル応答のストリーミング→状態の永続化）である。記事の「6 ステップ」は内部処理をより細かく分解したものと考えられる。

17.5　この事例から学べること

- **VPS + Docker による分離運用**：個人 PC に直接インストールせず、VPS 上のコンテナで運用することでセキュリティリスクを低減できる
- **HEARTBEAT による定期自動化**：Cron ジョブとワークスペース設定の組み合わせで、完全自動のデータ収集・分析パイプラインを構築できる
- **MEMORY.md による継続的学習**：取引結果と学習事項をメモリに蓄積し、エージェントの判断品質を対話を通じて段階的に向上させることができる
- **Telegram によるモバイル運用**：外出先からもチャットベースで戦略の確認・修正が可能である

18　モバイルアプリ・デスクトップアプリ

OpenClaw は以下のコンパニオンアプリを提供している。

プラットフォーム	特徴
macOS	メニューバーアプリ、Canvas（ビジュアルワークスペース）、Voice Wake（音声起動）
iOS	カメラ、位置情報などデバイス機能へのアクセス
Android	カメラ、位置情報、通知など
Linux	デスクトップアプリ

18.1　音声機能

Voice Wake と Talk Mode を有効にすると、音声による常時待ち受けと対話が可能になる（ElevenLabs 連携）。

19　まとめ

本チュートリアルでは、OpenClaw の導入から実践的な活用までを解説した。主要なポイントを以下にまとめる。

1. **インストール**は `curl` または `npm` で簡単に行える
2. `openclaw onboard` で対話型セットアップが完了する
3. 好みの**チャットアプリ**をチャンネルとして接続し、自然言語で対話できる
4. **ClawHub** のスキルで機能を拡張できる
5. **Cron**、**Webhook**、**ブラウザ操作**で高度な自動化が実現できる
6. **セキュリティ**ではペアリングポリシーとサンドボックスの活用が重要である

より詳しい情報は以下を参照されたい。

- 公式ドキュメント：https://docs.openclaw.ai/
- GitHub リポジトリ：https://github.com/openclaw/openclaw
- ClawHub（スキルレジストリ）：https://clawhub.com
- コミュニティスキル集：https://github.com/VoltAgent/awesome-openclaw-skills