

SDK 使い方ガイド

Infini-Brain A101/B, A101/BH

目次

本書をお読みになる前に	3
安全にお使いいただくために	3
本書の表記	3
Windows の操作	3
商標および著作権	4
第 1 章 お使いになる前に	
1 SDK Distributed Manager の概要	6
2 ソフトウェア一覧	6
SDK Distributed Manager 提供機能について	7
SDK Support Tool 提供機能について	7
FUJITSU Hardware Monitor 提供機能について	7
Model Management 提供機能について	7
3 本製品における推論処理構造について	8
第 2 章 推論コンテナを開発する	
1 Docker イメージを準備する	10
2 Docker コンテナに SDK Distributed Manager をインストールする	10
3 Docker コンテナで SDK Distributed Manager のインストールを確認する	10
4 AI アプリを開発する	11
第 3 章 トラブルシューティング	
1 トラブルー覧	17
運用時	17
2 推論 Config 情報	21
第 4 章 付録	
1 開発環境	23
AI アプリ開発環境	23
2 サンプルアプリの起動	23

本書をお読みになる前に

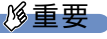


安全にお使いいただくために

本製品を安全に正しくお使いいただくための重要な情報が『取扱説明書』に記載されています。特に、「安全上のご注意」をよくお読みになり、理解されたうえで本製品をお使いください。

本書の表記

■ 本書の記号

本書に記載されている記号には、次のような意味があります。

	お使いになるときの注意点や、してはいけないことを記述しています。 必ずお読みください。
	操作に関連することを記述しています。必要に応じてお読みください。
	参照ページを示しています。

■ キーの表記と操作方法

本書中のキーの表記は、キーボードに書かれているマークを記述するのではなく、説明に必要な文字を使い、次のように記述しています。

例：【Ctrl】キー、【Enter】キー、【→】キーなど
また、複数のキーを同時に押す場合には、次のように「+」でつないで表記しています。
例：【Ctrl】+【F3】キー、【Shift】+【↑】キーなど

■ 連続する操作の表記方法

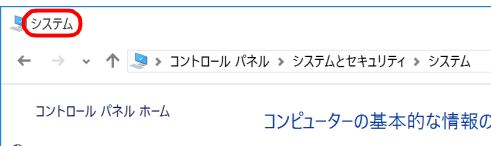
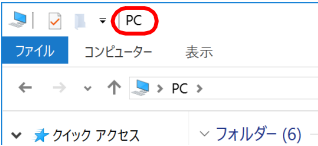
本書中の操作手順において、連続する操作手順を、「→」でつなげて記述しています。

例：コントロールパネルの「システムとセキュリティ」をクリックし、「システム」をクリックし、「デバイスマネージャー」をクリックする操作
↓
「システムとセキュリティ」→「システム」→「デバイスマネージャー」の順にクリックします。

■ ウィンドウ名の表記

本文中のウィンドウ名は、アドレスバーの最後に表示されている名称を表記しています。

例：



↓

「PC」ウィンドウ 「システム」ウィンドウ

■ 画面例およびイラストについて

本文中の画面およびイラストは一例です。お使いの機種やモデルによって、実際に表示される画面やイラスト、およびファイル名などが異なることがあります。また、イラストは説明の都合上、本来接続されているケーブル類を省略したり形状を簡略化したりしていることがあります。

■ 製品の呼び方


本書では、製品名称を次のように略して表記します。

製品名称	本書の表記		
Windows 10 IoT Enterprise 2019 LTSC	Windows 10 IoT Enterprise	Windows 10	Windows

Windows の操作

■ アクションセンター

アプリからの通知を表示するほか、クリックすることで画面の明るさ設定や通信機能の状態などを設定できるアイコンが表示されます。

- 1 画面右下の通知領域にある  をクリックします。
画面右側に「アクションセンター」が表示されます。

■ 「コントロールパネル」ウィンドウ

次の手順で「コントロールパネル」ウィンドウを表示させてください。

- 1 「スタート」ボタン→「Windows システム ツール」→「コントロールパネル」の順にクリックします。

■ 「コマンドプロンプト」ウィンドウ


次の手順で「コマンドプロンプト」ウィンドウを表示させてください。

- 1 「スタート」ボタン→「Windows システム ツール」の順にクリックします。
- 2 「コマンドプロンプト」を右クリックし、「その他」→「管理者として実行」をクリックします。

■ ユーザーアカウント制御

本書で説明している Windows の操作の途中で、「ユーザーアカウント制御」ウィンドウが表示される場合があります。これは、重要な操作や管理者の権限が必要な操作の前に Windows が表示しているものです。表示されるメッセージに従って操作してください。

■ 通知領域のアイコン

デスクトップ画面右下の通知領域にすべてのアイコンが表示されていない場合があります。
表示されていないアイコンを一時的に表示するには、通知領域の  をクリックします。

商標および著作権

NVIDIA、CUDA、Jetson、Jetpack、NVIDIA Jetpack、TensorRT は、アメリカ合衆国および / またはその他の国における NVIDIA Corporation の商標または登録商標です。
本製品には、Apache License V2.0 に基づきライセンスされるソフトウェアに当社が必要な改変を施して使用しております。
本製品には、BSD、GNU General Public License (GPL)、MIT、その他のライセンスに基づくオープンソースソフトウェアが含まれています。
オープンソースソフトウェアのライセンスに関する詳細およびソフトウェアのソースコードについては本製品のマニュアルをご覧ください。
FUJITSU Hardware Monitor、Model Management、SDK Distributed Manager、SDK Support Tool、バーチャル LAN ドライバー、ブリッジコントローラードライバー
は、富士通クライアントコンピューティング株式会社の製品です。著作権は富士通クライアントコンピューティング株式会社にあります。
その他の各製品名は、各社の商標、または登録商標です。
その他の各製品は、各社の著作物です。
その他のすべての商標は、それぞれの所有者に帰属します。

Copyright FUJITSU LIMITED 2020

1

第 1 章 お使いになる前に

ここでは、SDK の概要について、説明します。

1. SDK Distributed Manager の概要	6
2. ソフトウェア一覧	6
3. 本製品における推論処理構造について	8

1. SDK Distributed Manager の概要

本製品添付の SDK Distributed Manager では、AI 拡張ボードの性能をフルに活用するアプリをお客様が容易に開発できるよう強力にサポートする API ライブラリを提供します。SDK Distributed Manager を使うことで次のことを実現できます。

- 複数の AI 拡張ボードを搭載する場合、次のメリットがあります（最大 6 台）。
 - ・推論処理を複数の AI 拡張ボードで並列に実行できるようになるため、処理性能が向上します。
 - ・AI 拡張ボードごとに異なる推論処理を実行できるようになるため、異なる推論処理をスムーズに同時実行できます。
- Windows での カメラの映像や静止画像 の入力に対して、AI 拡張ボード上で 推論処理を実行できます。
- Windows の豊富な資産を活用し、お客様のニーズに合わせ カメラなどのデバイス、各種ライブラリを利用できます。
- これらの機能は Windows 上で Python 言語 により活用することができます。
- NVIDIA Jetson 向けに開発した推論プログラムに対して同時／並列に実行するアプリに移植できます。

2. ソフトウェア一覧

本製品で提供される 主なソフトウェアは次のとおりです。

■メインボード側

種別	提供ソフトウェア	概要	備考
OS	Windows 10 IoT Enterprise 2019 LTSC		
ドライバー	ブリッジコントローラードライバー バーチャル LAN ドライバー	推論演算機能を提供する AI 拡張ボード を、通常のネットワーク接続先と同様に利用可能とする機能を提供します。	Ethernet 準拠 [MTU=64K, 最大 16Gbps (理論値)]
SDK	SDK Distributed Manager SDK Support Tool	Docker 配置、AI 推論のロードバランシング API の提供します (→ P.7)。 次の機能を提供します。 ・メインボードと AI 拡張ボードの通信を可能にし、AI 拡張ボードのセキュリティを守ります。また、保守運用に活用できる機能 (→ P.7)。 ・AI 拡張ボードのログを収集し、リアルタイムにメインボードに転送する機能。	
暗号化機能	Model Management	SDK Distributed Manager を利用している、お客様の AI モデルを暗号化しセキュアに管理する機能を提供します。本機能を使用する場合は、インストールする必要があります。詳しくは、『SDK 使い方ガイド補足情報 (Model Management の使い方)』をご覧ください。	
ハード監視	FUJITSU Hardware Monitor	次の機能を提供します。 ・AI 拡張ボードを含め、ハードウェアが正常に稼働しているかを確認できる機能を提供します。また、各種監視アプリと連携するための、コマンドラインによるハードウェアのモニターコマンド (→ P.7)。 ・ブリッジコントローラーのエラーを検出し、ログに記録する機能。	

■AI 拡張ボード側

種別	提供ソフトウェア	概要	備考
OS	Ubuntu 18.04		
ドライバー	ブリッジコントローラードライバー バーチャル LAN ドライバー	推論演算機能を提供する AI 拡張ボード を、通常のネットワーク接続先と同様に利用可能とする機能を提供します。	Ethernet 準拠 [MTU=64K, 最大 16Gbps (理論値)]
JetPack	NVIDIA® JetPack™ 4.2.2		
プログラミング言語	Python 3.6 Python 2.7.12-1		本 SDK では Python3 系のみサポートしています。

SDK Distributed Manager 提供機能について

SDK Distributed Manager で提供される機能は、次のとおりです。

種別	機能	概要	備考
推論実行	静止画の推論処理の実行（同期／非同期）	指定の推論処理を最も効率良く実行できる AI 拡張ボードを自動的に識別／選択し、推論処理を実行させる機能	動画を静止画に切り出すことで利用可能
推論制御	推論コンテナ 読み / 削除	Windows 上に用意された推論コンテナを、AI 拡張ボードに 転送／起動、終了／削除 などを制御する機能	
	推論コンテナ 起動 / 停止	初回起動に時間のかかる推論コンテナを、事前に AI 拡張ボードで起動／待機させる機能	
	推論コンテナ の情報取得	本製品で利用可能な推論コンテナの情報を取得する機能	
	AI 拡張ボード稼働状態取得	AI 拡張ボード上で起動している 推論コンテナの稼働状況を取得する機能	
	推論キュー状態管理	多数の静止画を効率的に推論処理させる場合に、アプリでの開発を不要とするキュー管理を提供する機能	キュー上限／タイムアウト設定可
推論コンテナ	推論コンテナ の接続コネクタ	AI 拡張ボード上で開発した推論コンテナを動作させたい場合に、この推論キューに接続するためのコネクタ機能	
	障害ログの収集	AI 拡張ボードでのログを、自動的に Windows 上に収集し確認できるようにする機能	

提供する API の詳細は、『SDK API リファレンスガイド』をご覧ください。

SDK Support Tool 提供機能について

SDK Support Tool で提供される機能は、次のとおりです。

機能	概要	備考
初期設定実行機能	AI 拡張ボードの構成を自動判別し、メインボード・AI 拡張ボード間の通信設定など SDK 利用に必要な設定を行う機能	
設定の修復機能	増設や交換などで AI 拡張ボードの構成が変更になった場合に、SDK 利用に必要な設定を修復する機能	SDK Support Tool を再セットアップすることで設定を修復
AI 拡張ボードの構成変更検出機能	・ AI 拡張ボードの構成変更を検出し、構成変更をお知らせするコードを STATUS 表示部に表示する機能 ・ 本体起動時に AI 拡張ボードの構成変更などで SSH 通信エラーを検出し、お知らせするコードを STATUS 表示部に表示する機能	
認証鍵更新／初期化ツール	メインボードと AI 拡張ボード間でセキュアな通信を行うための認証鍵設定機能	
AI 拡張ボードの詳細情報収集	AI 拡張ボードの詳細情報（製造番号、シリアル番号など）を取得する機能	

SDK Support Tool の再セットアップについては、『管理者ガイド』をご覧ください。

FUJITSU Hardware Monitor 提供機能について

FUJITSU Hardware Monitor で提供される機能は、次のとおりです。

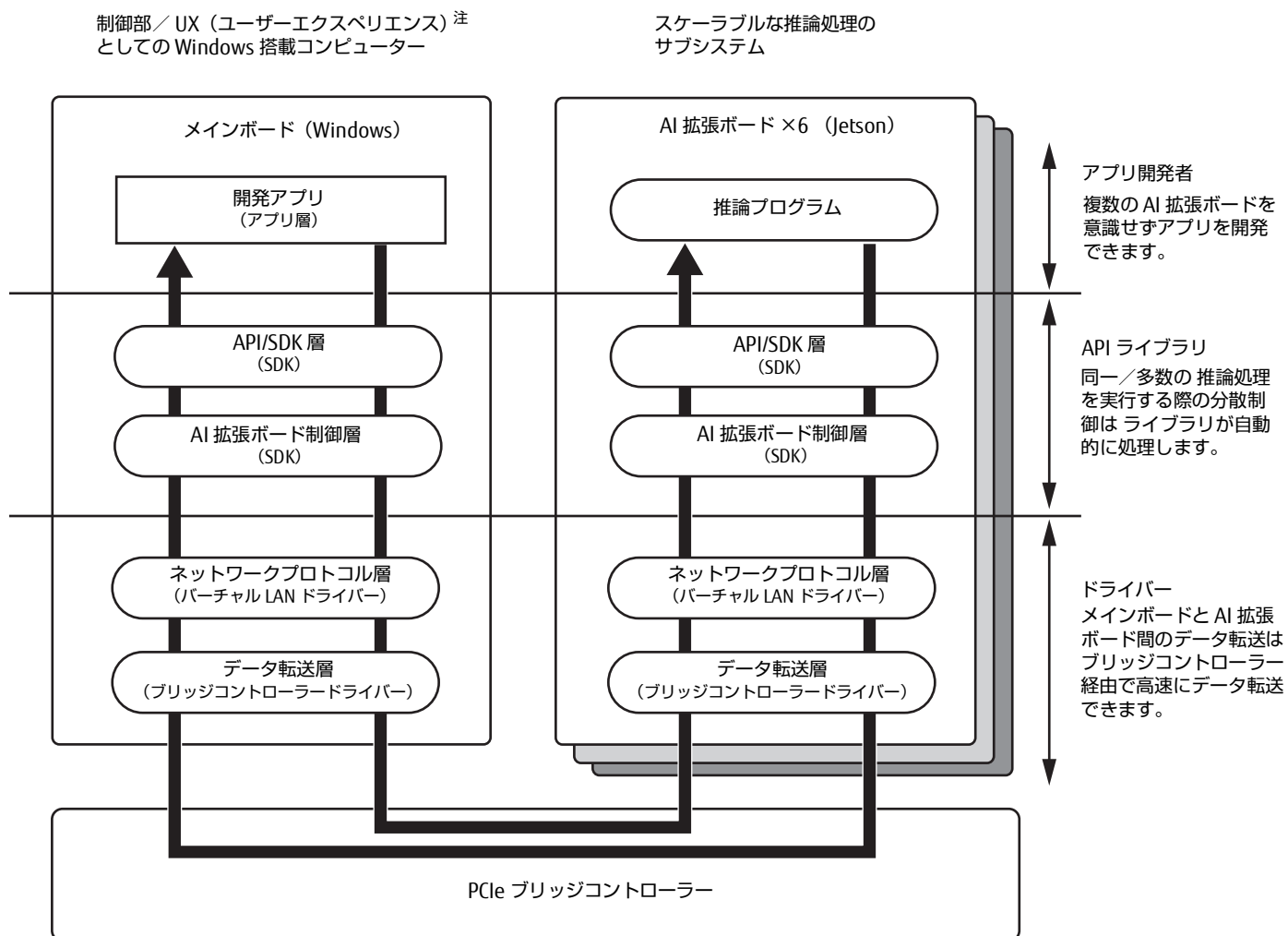
機能	概要	備考
AI 拡張ボード稼働状況の監視機能	AI 拡張ボードの稼働状態をメインボードで監視し、エラー情報をイベントログに記録する機能	
AI 拡張ボード制御コマンド機能	AI 拡張ボードのステータス取得や電源の ON／OFF をメインボードのコマンドで制御する機能	
メインボードの温度・ファン定期監視機能	メインボードの温度センサーとファン回転数を監視し、異常を STATUS 表示部に通知する機能	
定期清掃のお知らせ通知機能	システムの累積稼働時間を監視し、防塵フィルタの定期清掃を促す機能	
AI 拡張ボード構成変更監視機能	AI 拡張ボード構成変更を STATUS 表示部に通知し、SDK Support Tool の再セットアップを促す機能	SDK Support Tool の再セットアップについては、『管理者ガイド』をご覧ください。

Model Management 提供機能について

Model Management 提供機能については、『SDK 使い方ガイド補足情報（Model Management の使い方）』をご覧ください。

3. 本製品における推論処理構造について

推論処理構造は、下図のようになります。



注：本製品が提供するサービスや UI を通じて実現するユーザー体験

2

第 2 章 推論コンテナを開発する

1. Docker イメージを準備する.....	10
2. Docker コンテナに SDK Distributed Manager をインストールする	10
3. Docker コンテナで SDK Distributed Manager のインストールを確認する	10
4. AI アプリを開発する	11

1. Docker イメージを準備する

お客様のアプリケーション環境に合わせた Docker 推論コンテナ環境を構築します。

- Docker イメージの作成
- Docker コンテナの作成
- Docker コンテナに Python および Python パッケージをインストール
- Docker コンテナ上で推論環境の構築（AI フレームワークのインストール／モデルの配置）
- Docker コンテナの起動コマンドで推論プログラムを指定しておく

2. Docker コンテナに SDK Distributed Manager をインストールする

次の手順で Docker コンテナに SDK Distributed Manager をインストールします。

- 1 Windows 上で次のフォルダーにある SDK Distributed Manager のパッケージ一式を任意の場所にコピーします。

`C:\Program Files\FUJITSU CLIENT COMPUTING LIMITED\SDK\SDKBasic\SDK Distributed Manager\IBSDK\Libs\pyscript\infinibrain`

- 2 手順 1 でコピーしたパッケージ一式を、AI 拡張ボードに転送します。

- 3 AI 拡張ボードに SSH でログインします。

例) コマンドプロンプトを起動して、次のコマンドを入力します。

```
ssh fujitsu@192.168.1.102
```

```
User: fujitsu
```

```
Pass (default) : fujitsu
```

```
IP アドレス : 192.168.1.102~107 (バーチャル LAN ドライバー x 6)
```

- 4 手順 2 で AI 拡張ボードに転送したパッケージ一式を、次のコマンドで Docker コンテナ上へコピーします。

```
$ sudo docker cp infinibrain {コンテナ名}:/opt/
```

- 5 Docker コンテナ上で次のコマンドを実行し、環境変数を設定します。

```
$ export PYTHONPATH="/opt/"
```

3. Docker コンテナで SDK Distributed Manager のインストールを確認する

- 1 Docker コンテナ上で次のコマンドを実行し、【Enter】キーを押します。

```
$ python3
```

python インタプリタが起動します。

- 2 SDK パッケージをインポートするため、次のコマンドを入力し【Enter】キーを押します。

```
from infinibrain.inference import AICluster
```

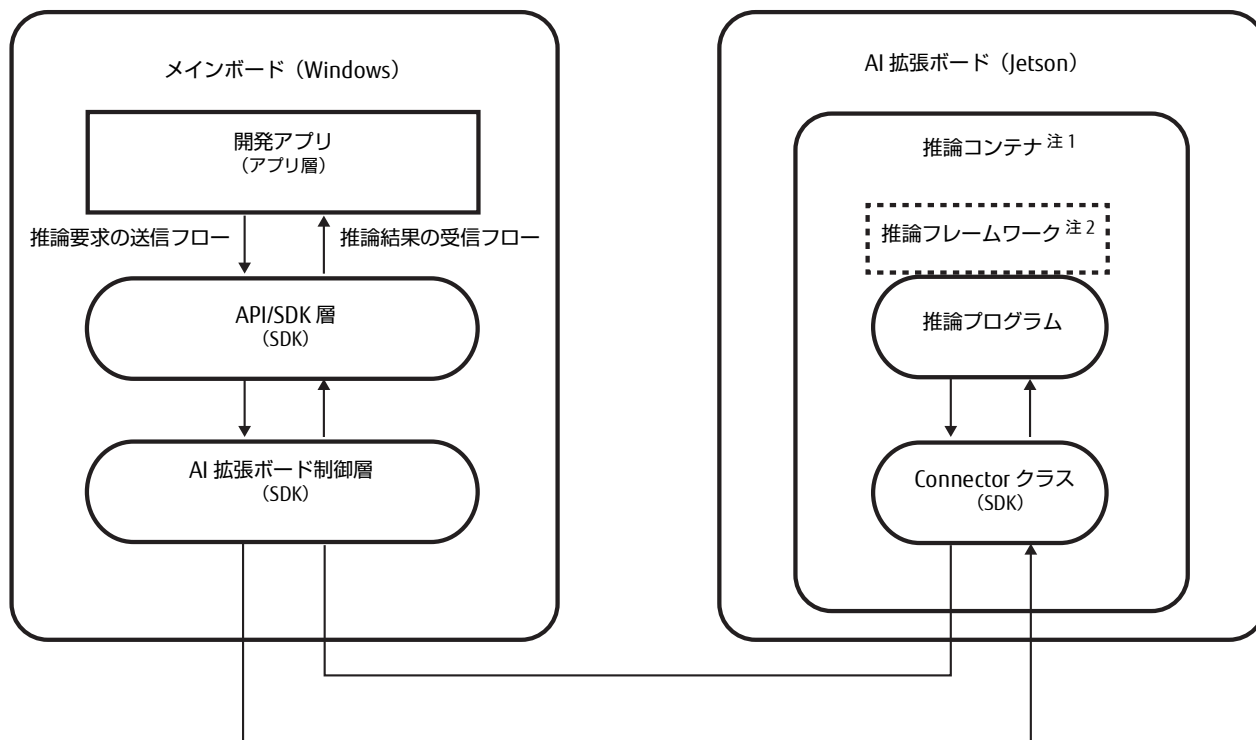
- 3 インポートでエラーが発生しないことを確認した後、次のコマンドを入力し、【Enter】キーを押します。

```
exit()
```

python インタプリタが終了します。

4. AI アプリを開発する

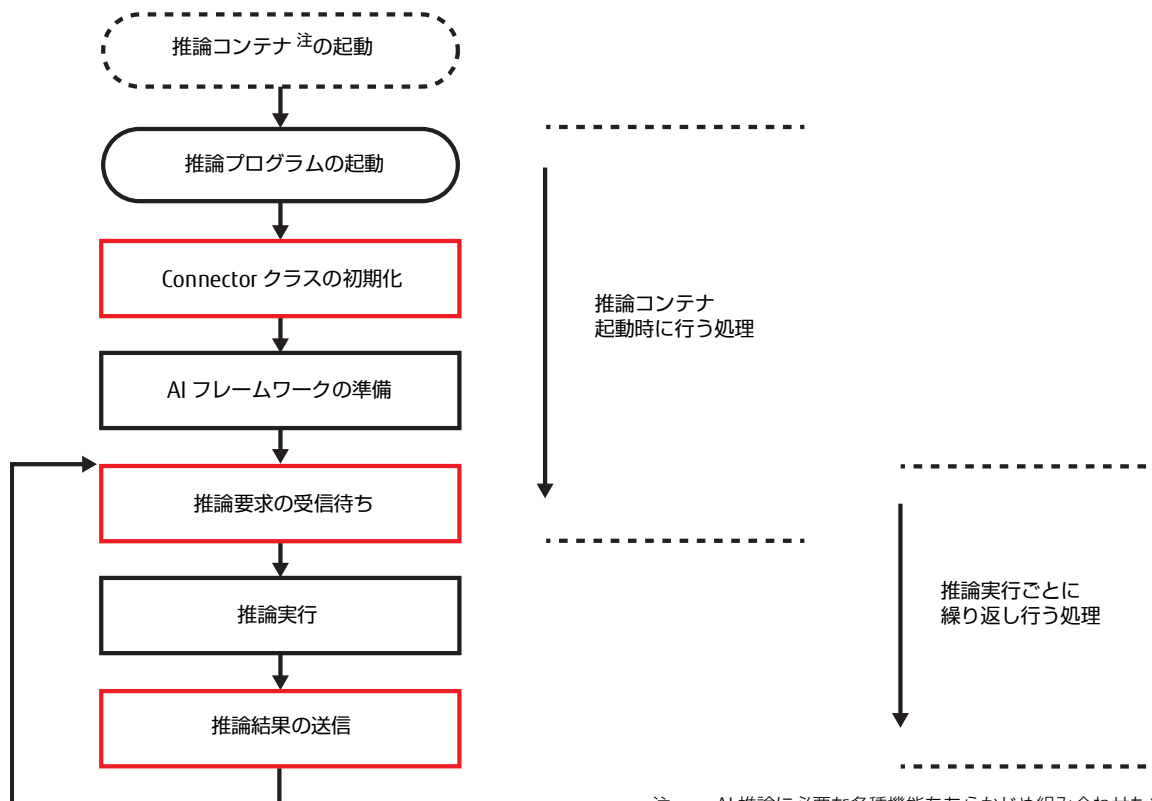
推論コンテナ上で動作する推論プログラムを開発するにあたっては、SDK Distributed Manager のパッケージー式に含まれる Connector クラスを使用します。Connector クラスとは、メインボードと推論コンテナが通信するための API を備えたクラスです。推論実行時のメインボードと推論コンテナのデータフローは下図のようになります。



注1: Docker イメージを元に作成される仮想環境で、推論プログラムの実行環境です。

注2: AI 推論に必要な各種機能をあらかじめ組み合わせたソフトウェア群です。

Connector クラスの API を使用するためには、推論プログラムで Connector クラスをインポートする必要があります。Connector クラスを使用した推論プログラムの基本的な処理フローおよび、使用する API の概要は下図のようになります。図のうち赤枠の処理が Connector クラスで提供している API で実行可能な処理です。



注: AI 推論に必要な各種機能をあらかじめ組み合わせたソフトウェア群です。

SDK にて提供される機能は、次のとおりです。

機能名	API 名	説明
Connector クラスの初期化	__init__()	Connector クラスの初期化処理を行います。
推論要求受信	receive_request_image()	画像データの推論要求を待ち受けます。 本 API を使用することで、メインボードからの推論要求の受信を待つ状態になります。
推論結果の送信	send_response()	推論結果を返却します。 本 API を使用することで、推論を行った結果をメインボードのアプリケーションに送信します。

以降の項では、各処理の説明を行います。

1 Connector クラスの初期化を行う

推論プログラムのメイン処理では、まず Connector クラスの初期化処理を行い、Connector クラスのインスタンスを生成してください。初期化処理では、メインボードとの通信の準備を行います。メインボードとの通信に使用する各パラメータは初期化処理の引数で指定することもできますが、指定しない場合は、SDK にて自動で設定を行います（推奨）。

初期化処理のプログラム例を以下に示します。

```
#####
# 初期化処理
# Connector クラスの初期化を行い、インスタンス生成してください。
# インスタンスを生成する際に、引数で与えられたパラメータでメインボードとの通信ソケットを準備します。
# 引数を指定しない場合には、パラメータを自動判別し、通信ソケットを準備します。（推奨）
# 引数には以下があります。
#
# sub_name：データ受信する際のトピック文字列
# host_ip：mainPC 側の IP
# pub_port：送信ポート
# sub_port：受信ポート
#####
DEFAULT_WINDOWS_IP = "192.168.1.101"
DEFAULT_ZMQ_PUB_PORT = 15001
DEFAULT_ZMQ_SUB_PORT = 15002
if __name__ == '__main__':

    # 通信パラメータを指定する場合
    #connector = Connector(sub_name='XX', host_ip=DEFAULT_WINDOWS_IP, \
    # pub_port=str(DEFAULT_ZMQ_PUB_PORT), sub_port=str(DEFAULT_ZMQ_SUB_PORT))

    # 通信パラメータを自動設定する場合（推奨）
    connector = Connector()
```

2 AI フレームワークの準備を行う

使用する AI モデルのロードなど、AI のフレームワークにあわせた推論実行の準備を行ってください。

3 推論要求の受信を待ち受ける

推論の実行環境の準備ができれば、Connector クラスの API を使用して推論要求の受信を待ち受けます。推論要求受信の API は同期型の API となり、メインボードからの推論要求を受信すると、API の戻り値で受信データを返却します。

また、推論要求を受信後は推論を実行してメインボードへ推論結果を送信しますが、送信後は、再び本 API を使用して推論要求受信待ち受けを行ってください。

4 推論を実行する

推論要求を受信したら、使用する AI フレームワークに合わせた推論処理を行ってください。

5 推論結果を送信する

推論処理が終わったら、Connector クラスの API を使用して推論結果をメインボードに送信します。送信した推論結果は、メインボードで推論要求を行ったアプリケーションに返却されます。推論結果を送信したら、必ず 3. 推論要求の受信を待ち受ける 以降の処理を繰り返し実行してください。
手順 3「推論要求の受信を待ち受ける」から本項までのプログラム例を以下に示します。

```
#####
# 受信ループ
# Connector クラスの API を使用して、メインボードからの推論要求の受信および推論結果の送信を行います。
#
# ・推論要求の受信
# API:
# image, data, res = connector.receive_request_image()
#
# 本 API を実行することで、メインボードからの推論要求待ち状態になります。
# 各戻り値を以下で説明します。
# image(numpy.ndarray) : メインボードからデータ送信時、infinibrain.inference.infer_image() の引数で与えられた image_data に相当します。
# data(dict) : メインボードからデータ送信時、infinibrain.inference.infer_image() の引数で与えられた param に相当します。
# res(InferenceResponse) : データ送信のタイムスタンプ情報や、各メタ情報が格納されたデータクラスです (InferenceResponse クラス参照)
#
# ・推論結果の送信
# API:
#
# connector.send_response(result, res)
#
# 本 API を実行することで、mainPC へデータ送信処理を行います。
# 各引数を以下で説明します。
# result : メインボードへ返却するデータを設定します。AI の推論結果を想定しています。Json シリアライズできるデータ型である必要があります。
# res : メインボードへ返却するデータクラスを設定します。受信した InferenceResponse クラスのインスタンスをそのまま格納しても構いません
#####
def run_inference_loop(connector, param):

    debug('run_inference_loop start param : {}'.format(param))

    while True:
        debug('receive wait...')

        # 推論要求の受信を待ち受ける
        image, data, res = connector.receive_request_image()

        debug('received data : {}'.format(data))

        # 推論を行う
        result = 'sample success'

        debug('send_response result : {}'.format(result))

        # 推論要求結果を送信する
        connector.send_response(result, res)
```

Connector を使用した推論プログラムの実装サンプル全体を以下に示します。

```
#####
# Connector 実装サンプル
#####
# 実行環境
# ・API 一式をコンテナ内に配置してください
# 例：
# /opt/infinibrain/...
# ・API 一式までのパスを推論コンテナの環境変数に設定してください
# $ export PYTHONPATH="/opt/"
#####

#####
# 必要パッケージのインポート
# 以下、infinibrain で始まるパッケージをインポートしてください
# 必須は Connector クラスです。
# 他、InferenceResponse クラスや container_log_util にはユーティリティ機能がありますが、使用は任意です。
#####
import numpy as np
from infinibrain.inference import InferenceResponse
from infinibrain.containers.connector import Connector
from infinibrain.containers.container_log_util import log_setting, debug
import json
from time import sleep
import logging

#####
# AI フレームワークの準備
# AI モデルのロード処理やその他の必要な初期化処理を想定して本関数を規定しています（任意実装）
#####
def init_inference(connector, model_path, param):

    log_setting(__name__, logging.DEBUG)

    debug('initialize')
    debug('init_inference param : {}'.format(param))

    # モデルのロード処理の実装
    #param['model'] = model_path

#####
# 受信ループ
# Connector クラスの API を使用して、メインボードからの推論要求の受信および推論結果の送信を行います。
#
# ・推論要求の受信
# API:
# image, data, res = connector.receive_request_image()
#
# 本 API を実行することで、メインボードからの推論要求待ち状態になります。
# 各戻り値を以下で説明します。
# image(numpy.ndarray) : メインボードからデータ送信時、infinibrain.inference.infer_image() の引数で与えられた image_data に相当します。
# data(dict) : メインボードからデータ送信時、infinibrain.inference.infer_image() の引数で与えられた param に相当します。
# res(InferenceResponse) : データ送信のタイムスタンプ情報や、各メタ情報が格納されたデータクラスです（InferenceResponse クラス参照）
#
#
# ・推論結果の送信
# API:
#
# connector.send_response(result, res)
#
# 本 API を実行することで、mainPC へデータ送信処理を行います。
# 各引数を以下で説明します。
# result : メインボードへ返却するデータを設定します。AI の推論結果を想定しています。Json シリアライズできるデータ型である必要があります。
# res : メインボードへ返却するデータクラスを設定します。受信した InferenceResponse クラスのインスタンスをそのまま格納しても構いません
#####
def run_inference_loop(connector, param):

    debug('run_inference_loop start param : {}'.format(param))

    while True:
        debug('receive wait...')
```

```
# 推論要求の受信を待ち受ける
image, data, res = connector.receive_request_image()

debug('received data : {}'.format(data))

# 推論を行う
result = 'sample success'

debug('send_response result : {}'.format(result))

# 推論要求結果を送信する
connector.send_response(result, res)

#####
# 初期化处理
# Connector クラスの初期化を行い、インスタンス生成してください。
# インスタンスを生成する際に、引数で与えられたパラメタでメインボードとの通信ソケットを準備します。
# 引数を指定しない場合には、パラメタを自動判別し、通信ソケットを準備します。(推奨)
# 引数には以下があります。
#
# sub_name : データ受信する際のトピック文字列
# host_ip : mainPC 側の IP
# pub_port : 送信ポート
# sub_port : 受信ポート
#####
DEFAULT_WINDOWS_IP = "192.168.1.101"
DEFAULT_ZMQ_PUB_PORT = 15001
DEFAULT_ZMQ_SUB_PORT = 15002
if __name__ == '__main__':

    # 通信パラメータを指定する場合
    #connector = Connector(sub_name='XX', host_ip=DEFAULT_WINDOWS_IP, \
    # pub_port=str(DEFAULT_ZMQ_PUB_PORT), sub_port=str(DEFAULT_ZMQ_SUB_PORT))

    # 通信パラメータを自動設定する場合
    connector = Connector()

    param = {}

    init_inference(connector, model_path, param)

    run_inference_loop(connector, param)
```

3

第 3 章 トラブルシューティング

おかしいなと思ったときや、わからないことがあったときの対処方法について説明しています。

1. トラブル一覧	17
2. 推論 Config 情報	21

1. トラブル一覧

□ 運用時

- 推論実行時に発生するエラーコード (→ P.17)
- 推論コンテナ 読み / 削除時のエラーコード (→ P.19)
- 推論コンテナ 起動 / 停止時のエラー対処 (→ P.20)
- 推論コンテナ の情報取得時のエラー対処 (→ P.20)
- AI 拡張ボード稼動状態取得時のエラー対処 (→ P.20)
- 推論キュー状態管理時のエラー対処 (→ P.20)
- その他のエラー対処 (→ P.20)

運用時

■ 推論実行時に発生するエラーコード

次の表は、推論実行時に発生するエラーコード一覧と対処方法です。

コード	メッセージ	状態	対処
0	Success	API は正常に実行されました。	特になし。
305	Send Wait Queue Full	送信待ちキューがフル状態です。	<p>データを正しく転送し、応答を受信しているか確認してください (以下3点)。</p> <ol style="list-style-type: none"> 1.メインボードから、AI 拡張ボードにデータを転送しているかの確認 aicluster.log で、以下のようなログが出力されていれば、データ転送は実行されています (ログレベル: debug)。 Send :-> AI (Req No. { 要求 ID }) Message to : { 推論名 } 該当のログが出力されていない場合、データ転送できる推論コンテナが存在しない可能性があります。 AI 拡張ボード稼動状態取得機能を使い、該当の推論コンテナが起動状態にあるか確認してください。 応答の status パラメータが "Exit" になっている場合は、推論コンテナ 起動機能で推論コンテナを起動してから再度実行してください。 推論コンテナ自体が存在しない場合は、推論コンテナの読み / 実行をしてから再度実施してください。 2.推論コンテナでデータを受信し、推論実行しているかの確認 AI 拡張ボードに ssh でログインし、推論コンテナのログを取得してください。 ユーザー処理の中で、設定されているログが出力されていれば推論コンテナで処理実行されています。 AI 拡張ボードでデータを受信できていない場合、ファイアウォール設定でデータが遮断されている可能性があります。 メインボード上でポート「15001」「15002」の送信を許可するように設定し、再度実施してください。 データ受信後にエラーが発生している場合、推論プログラムを修正し、再度推論コンテナの読みから実行してください。 3.メインボードで、AI 拡張ボードからの応答を受信しているかの確認 aicluster.log で、以下のようなログが出力されていれば、応答データを受信できています (ログレベル: debug)。 Receive :<- AI (Req No. { 要求 ID }) Message from : { 推論名 } メインボードでデータを受信できていない場合、ファイアウォール設定でデータが遮断されている可能性があります。 メインボード上でポート「15001」「15002」の受信を許可するように設定し、再度実施してください。 <p>正しく転送され、応答が返却されている場合、推論プログラムの処理速度に対してアプリからのデータ投入間隔が短い可能性があります。 推論コンテナに対するデータの最大転送設定 (キューサイズ: max_send_num パラメータ) を大きくしてください。 もしくは、推論アプリ内で、データの投入間隔を長くして再実行してください (応答データを持ってから新しいデータを投入するなど)。</p>

コード	メッセージ	状態	対処
306	Send Wait Queue TimeOut	送信待ち中のキューがタイムアウトしました。	<p>データを正しく転送し、応答を受信しているか確認してください（以下3点）。</p> <ol style="list-style-type: none"> 1. メインボードから、AI 拡張ボードにデータを転送しているかの確認 aicluster.log で、以下のようなログが出力されていれば、データ転送は実行されています（ログレベル：debug）。 Send :-> AI (Req No. { 要求 ID }) Message to : { 推論名 } 該当のログが出力されていない場合、データ転送できる推論コンテナが存在しない可能性があります。 AI 拡張ボード稼動状態取得機能を使い、該当の推論コンテナが起動状態にあるか確認してください。 応答の status パラメータが "Exit" になっている場合は、推論コンテナ 起動機能で推論コンテナを起動してから再度実行してください。 推論コンテナ自体が存在しない場合は、推論コンテナの読み込み / 実行をしてから再度実施してください。 2. 推論コンテナでデータを受信し、推論実行しているかの確認 AI 拡張ボードに ssh でログインし、推論コンテナのログを取得してください。 ユーザー処理の中で、設定されているログが出力されていれば推論コンテナで処理実行されています。 AI 拡張ボードでデータを受信できていない場合、ファイアウォール設定でデータが遮断されている可能性があります。 メインボード上でポート「15001」「15002」の送信を許可するように設定し、再度実施してください。 データ受信後にエラーが発生している場合、推論プログラムを修正し、再度推論コンテナの読み込みから実行してください。 3. メインボードで、AI 拡張ボードからの応答を受信しているかの確認 aicluster.log で、以下のようなログが出力されていれば、応答データを受信できています（ログレベル：debug）。 Receive : <- AI (Req No. { 要求 ID }) Message from : { 推論名 } メインボードでデータを受信できていない場合、ファイアウォール設定でデータが遮断されている可能性があります。 メインボード上でポート「15001」「15002」の受信を許可するように設定し、再度実施してください。 <p>正しく転送され、応答が返却されている場合、推論プログラムの処理速度に対してアプリからのデータ投入間隔が短い可能性があります。 推論コンテナに対するデータの最大転送設定（キューサイズ：max_send_num パラメータ）を大きくするか、タイムアウトの時間設定（wait_queue_timeout パラメータ）を長くしてください。 もしくは、推論アプリ内で、データの投入間隔を長くして再実行してください（応答データを待ってから新しいデータを投入するなど）。</p>
400	Unregistered InferenceName	データ転送先の推論が登録されていません。	<p>推論コンテナの読み込みを実施した上で推論を実行してください。 推論が読み込まれているかどうかは、推論コンテナの情報取得機能で確認可能です。</p>
401	Response TimeOut Error	データ送信後の応答データがタイムアウトしました。	<p>データが AI 拡張ボード上で受信され、応答を返却しているか確認してください（以下2点）。</p> <ol style="list-style-type: none"> 1. 推論コンテナでデータを受信し、推論実行しているかの確認 AI 拡張ボードに ssh でログインし、推論コンテナのログを取得してください。 ユーザー処理の中で、設定されているログが出力されていれば推論コンテナで処理実行されています。 AI 拡張ボードでデータを受信できていない場合、ファイアウォール設定でデータが遮断されている可能性があります。 メインボード上でポート「15001」「15002」の送信を許可するように設定し、再度実施してください。 AI 拡張ボードでデータ受信後にエラーが発生している場合、推論プログラムを修正し、再度推論コンテナの読み込みから実行してください。 2. メインボードで、AI 拡張ボードからの応答を受信しているかの確認 aicluster.log で、以下のようなログが出力されていれば、応答データを受信できています（ログレベル：debug）。 Receive : <- AI (Req No. { 要求 ID }) Message from : { 推論名 } メインボードでデータを受信できていない場合、ファイアウォール設定でデータが遮断されている可能性があります。 メインボード上でポート「15001」「15002」の受信を許可するように設定し、再度実施してください。 <p>正しく応答が返却されている場合、推論の処理時間に対してタイムアウト時間が短い可能性があります。 推論コンテナ読み込みで設定するタイムアウト時間（send_timeout パラメータ）を長くし、再実行してください。</p>
402	Response Disconnect Error	データ送信後にコンテナとの接続が切れました。	<p>推論プログラムが起動状態にあるか確認してください。 AI 拡張ボードに ssh でログインするか、AI 拡張ボード稼動状態取得機能で確認できます。 応答の status パラメータが "Exit" になっている場合は推論コンテナが停止しているので、推論コンテナ起動を実施した上で推論を実行してください。</p>
403	Does not accept data.	データを転送できません。	<p>推論コンテナの設定パラメータ "input" に対して、"None" が設定されている場合はデータ転送できません。 "input" の設定を "Main" に変更して推論コンテナ 読み込みを行うか、推論アプリからデータ転送しないようにしてください。</p>

■ 推論コンテナ 読み込み / 削除時のエラーコード

次の表は、推論コンテナ 読み込み / 削除時に発生するエラーコードと、その対応一覧と対処方法です。

コード	メッセージ	状態	対処
0	Success	API は正常に実行されました。	特になし。
201	No Required Key	API の実行に必要なパラメータがありません。	『SDK API リファレンスガイド』をご覧ください。API の実行に必要なパラメータを設定して再度、実行してください。
203	Tag version has changed from {old} into {new}	推論の docker イメージが更新されています。	イメージを含めて推論プログラムを更新する場合は、以下の 2 方法で実施してください。 ・ 強制オプションを有効にして推論コンテナ読み込みを実行する。 ・ 推論コンテナ削除を実行後、推論コンテナ読み込みを実行する。
204	File Not Found	推論の docker イメージファイルがありません。	推論コンテナのイメージファイルが、推論コンテナ 読み込み実行時に指定したディレクトリに格納されていることを確認してください。 また、設定しているディレクトリ path が、実在しているディレクトリと相違ないことを確認して再度実行してください。
205	Inference config file not found	推論コンフィグファイルがありません。	推論コンテナ読み込みを実施し、推論を登録してください。推論コンフィグファイルは、SDK が自動で作成します。 または、推論コンフィグファイルを作成して推論コンテナ内の所定の場所に格納してください。 (『推論 Config 情報』 (→ P.21) を参照)
206	Inference config file JSON format error	推論コンフィグファイルが JSON 形式になっていません。	推論コンテナ読み込みを実施し、推論を登録してください。推論コンテナと推論コンフィグファイルを再生成します。 または、推論コンテナ内の推論コンフィグファイルを正しい JSON 形式に書き換えてください。 (『推論 Config 情報』 (→ P.21) を参照)
207	Inference config file mismatch	推論コンフィグファイルの情報が不一致です。	推論コンテナ読み込みを実施し、推論を登録してください。推論コンテナと推論コンフィグファイルを再生成します。 または、各推論コンテナ内の推論コンフィグファイルに記載されている内容を一致させるように書き換えてください。 (『推論 Config 情報』 (→ P.21) を参照)
300	Specified InferenceName does not exist	推論名が未登録です。	推論コンテナ読み込みを実施し、推論を登録してください。 推論が読み込まれているかどうかは、推論コンテナの情報取得機能で確認可能です。 推論コンテナの情報取得機能で取得された推論が分散対象になりますので、推論名を指定して再度実行してください。
301	Can not open configure file	指定のファイルがありません。	ファイルを指定の場所に格納し、設定している path に間違いがないことを確認して再度実行してください。
302	Specified Param Error	指定のパラメータに異常があります。	すでに読み込み済みの推論コンテナに対しては、同じ設定で読み込まないと本エラーが発生します。 以下の 2 方法で回避ください。 ・ 強制オプションを有効にして推論コンテナ読み込みを実行する (再生成する)。 ・ 推論コンテナの情報取得機能を実施して設定情報を取得し、その設定で推論コンテナ読み込みを実施する。
303	There are still processes in use	推論を使用中のプロセスがあります。	使用中のプロセスがあるので、API 実行できません。時間を置いて再度実行してください。 どうしても本エラーが続く場合は、以下を実施してください。 ・ SDK Distributed Manager のサービスを停止 ・ cluster_tbl.json ファイルを削除 C:\Program Files\FUJITSU CLIENT COMPUTING LIMITED\SDK\Basic\SDK Distributed Manager\AI\Cluster\config ・ SDK Distributed Manager のサービスを再起動 ・ 推論コンテナ 読み込み < 注意 > 上記を実行すると、読み込み済みの推論コンテナ情報が削除されます。 他のユーザー (アプリ) に影響がないかを確認して対応してください。
304	Specified CoprocessorName does not exist	AI 拡張ボード名が登録されていません。	推論アプリで指定する AI 拡張ボード名と、コンフィグ (cluster_config.json) で指定している AI 拡張ボード名が一致する必要があります。 どちらかを修正し、再度実行してください。
309	{names} is used by other process. Please retry a little later.	推論に対して処理をしている別のプロセスがあります。	使用中のプロセスがあるので、API 実行できません。時間を置いて再度実行してください。 どうしても本エラーが続く場合は、以下を実施してください。 ・ SDK Distributed Manager のサービスを停止 ・ cluster_tbl.json ファイルを削除 C:\Program Files\FUJITSU CLIENT COMPUTING LIMITED\SDK\Basic\SDK Distributed Manager\AI\Cluster\config ・ SDK Distributed Manager のサービスを再起動 ・ 推論コンテナ 読み込み < 注意 > 上記を実行すると、読み込み済みの推論コンテナ情報が削除されます。 他のユーザー (アプリ) に影響がないかを確認して対応してください。
500	SSH error	SSH エラーが発生しました。	本エラーでは、以下の可能性があります。 ・ ssh connect error メインボードと AI 拡張ボード間のネットワークが切断されている。 メインボードから AI 拡張ボードに ssh で接続できることを確認してから再度実行してください。 ・ ssh error : putFile ・ ssh command error : docker load AI 拡張ボードに十分なストレージ容量がない。 イメージファイルを格納し、展開できるだけの十分な空き容量を確保してから再度実行してください。 ・ start up time out ・ ファイアウォールで通信が遮断されている。 メインボード上で「15001」と「15002」の port について通信が許可されていることを確認してから再度実行してください。 ・ 推論コンテナ内でエラーが発生している。 推論コンテナの状態を確認し、起動状態になっていることを確認してから再度実行してください。

■ 推論コンテナ 起動 / 停止時のエラー対処

推論コンテナ 起動 / 停止時に発生するエラーコードと、その対応一覧

コード	メッセージ	状態	対処
0	Success	API は正常に実行されました。	特になし。
304	Specified CoprocessorName does not exist	AI 拡張ボード名が登録されていません。	推論アプリで指定する AI 拡張ボード名と、コンフィグ (cluster_config.json) で指定している AI 拡張ボード名が一致している必要があります。 どちらかを修正し、再実行してください。
307	Specified Container does not exist	コンテナ名が登録されていません。	推論コンテナの情報取得機能を実施し、コンテナ名を確認してください。 確認したコンテナ名を指定し、再度実行してください。 所望の推論コンテナが存在しない場合は、推論コンテナの読込を実施してください。
500	SSH error	SSH エラーが発生しました。	本エラーでは、以下の可能性があります。 ・ssh connect error ・メインボードと AI 拡張ボード間のネットワークが切断されている。 メインボードから AI 拡張ボードに ssh で接続できることを確認してから再度実行してください。 ・start up time out ・ファイアウォールで通信が遮断されている。 メインボード上で「15001」と「15002」の port について通信が許可されていることを確認してから再度実行してください。 ・推論コンテナ内でエラーが発生している。 推論コンテナの状態を確認し、起動状態になっていることを確認してから再度実行してください。

■ 推論コンテナ の情報取得時のエラー対処

推論コンテナ の情報取得時に発生するエラーコードと、その対応一覧

コード	メッセージ	状態	対処
0	Success	API は正常に実行されました。	特になし。
300	Specified InferenceName does not exist	推論名が未登録です。	推論コンテナ読込を実施し、推論を登録してください。 推論が読み込まれているかどうかは、推論コンテナの情報取得機能で確認可能です。 取得された情報に含まれる推論名を指定して再度実行してください。

■ AI 拡張ボード稼動状態取得時のエラー対処

AI 拡張ボード稼動状態取得時に発生するエラーコードと、その対応一覧

コード	メッセージ	状態	対処
0	Success	API は正常に実行されました。	特になし。
300	Specified InferenceName does not exist	推論名が未登録です。	推論コンテナ読込を実施し、推論を登録してください。 推論が読み込まれているかどうかは、推論コンテナの情報取得機能で確認可能です。 取得された情報に含まれる推論名を指定して再度実行してください。 または、推論名を指定せずに実行することも可能です。
500	SSH error	SSH エラーが発生しました。	本エラーでは、次の可能性があります。 ・ssh connect error ・メインボードと AI 拡張ボード間のネットワークが切断されている。 メインボードから AI 拡張ボードに ssh で接続できることを確認してから再度実行してください。

■ 推論キュー状態管理時のエラー対処

推論キュー状態管理時に発生するエラーコードと、その対応一覧

コード	メッセージ	状態	対処
0	Success	API は正常に実行されました。	特になし。
300	Specified InferenceName does not exist	推論名が未登録です。	推論コンテナ読込を実施し、推論を登録してください。 推論が読み込まれているかどうかは、推論コンテナの情報取得機能で確認可能です。 取得された情報に含まれる推論名を指定して再度実行してください。 または、推論名を指定せずに実行することも可能です。
308	Can not delete queue (permission error)	キューを削除できません。	別のプロセスで使っているキューデータは削除できません。

■ その他のエラー対処

その他のエラーコードと、その対応一覧

コード	メッセージ	状態	対処
114	Inference {name} became unsubscribe state	推論コンテナの subscribe が切断されました。	意図してコンテナ停止していない場合は、コンテナのログをご確認ください。
202	Unknown Command	未対応の API がコールされました。	API の名前を確認し、用意されている API 名になっていることをご確認ください。

2. 推論 Config 情報

名称	型	通称	備考
name	str	推論名	
implementation_type	str	推論実装種別	推論の実装種別 "docker": Docker コンテナとしてコプロに推論を展開します。(SSH 通信を行います) "nvidia": NvidiaDocker コンテナとしてコプロに推論を展開します。(SSH 通信を行います) "Loop": AI に送信せず、デバッグ用メッセージをアプリに返信します。 "other": コンテナを展開しません。(SSH 通信なし。テーブル情報としてコプロ指定は可能です)
input	str	推論のデータ受信	推論のデータ受信を誰から行うかを指定します。 "Main": 並列分散処理からデータを受信します。 "None": だれからでもデータを受信しません。
output	str	推論のデータ応答動作	推論のデータ応答動作を指定します。 "Main": 応答を並列分散処理へ送信します。 "None": 応答を誰にも送信しません。
send_timeout	int	送信待ち時間超過の時間設定 (ms)	
wait_queue_timeout	int	応答待ち時間超過の時間設定 (ms)	
force_update	bool	設定の上書き指定	
max_send_num	int	一つのコンテナへの最大送信数	
script_path	str	スクリプトパス	転送する推論のスクリプトやモデルを格納している MeinPC のディレクトリパス
inference_modules	InferenceModulesConfig	スクリプト・モデルの設定	推論要求で実行するスクリプト及びモデルの設定
Image_path	str	イメージファイルパス	コンテナのイメージファイル格納場所
Image_filename	str	イメージファイル名	
Image_tag	str	イメージの DockerTAG 情報	
runCommand	str	run コマンド	Docker 起動時に実行させるコマンド
runOption	str	run オプション	ユーザーが Docker 起動の際に任意に割り当てたオプション
copy	dict or list	コピーファイル	推論コンテナにコピーするファイルの設定

■ 推論 Config 情報ファイルの配置 PATH

コンテナ上 /home/infinibrain/

■ 推論 Config 情報ファイルの名称

cluster_推論名.json

4

第4章 付録

1. 開発環境	23
2. サンプルアプリの起動	23

1. 開発環境

ここでは、アプリの開発環境と開発環境について説明します。

AI アプリ開発環境

SDK で使用する開発環境について、動作を確認しているバージョンは次のとおりです。

■ メインボード側

開発言語	動作環境
Python	バージョン 3.7

■ AI 拡張ボード側

開発言語	動作環境
Python	バージョン 3.6
CUDA	10.0.326-1
cuDNN	7.50.56-1+cuda10.0
OpenCV	3.3.1

2. サンプルアプリの起動

SDK のサンプルアプリを起動して、SDK の環境の動作確認を行います。

■ サンプルアプリの起動

1 管理者権限でコマンドプロンプトを起動します (→ P.3)。

2 次のコマンドを入力し、【Enter】キーを押します。

```
cd C:\Program Files\FUJITSU CLIENT COMPUTING LIMITED\SDK\Basic\SDK Distributed Manager\IBSDK\Sample\SDK_Operation_Verification
```

SDK のサンプルアプリが格納されたフォルダーに移動します。

3 次のコマンドを入力し、【Enter】キーを押します。

```
python SDK_Operation_Verification.py
```

python インタプリタが起動し、サンプルアプリが実行されます。

```
管理者: コマンド プロンプト
Microsoft Windows [Version 10.0.17763.864]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Program Files\FUJITSU CLIENT COMPUTING LIMITED\SDK\Basic\SDK Distributed Manager\IBSDK\Sample\SDK_Operation_Verification>python
ation.py
2020-01-21 20:51:20.528 DEBUG _main #####load_inference#####
2020-01-21 20:52:14.878 DEBUG _main container0 -> name : sample, allocated : subsys2
2020-01-21 20:52:14.878 DEBUG _main #####infer_image#####
2020-01-21 20:52:14.878 DEBUG _main in_dir res\images\640x480.jpg
2020-01-21 20:52:15.046 DEBUG _main result -> sample success
2020-01-21 20:52:15.047 DEBUG _main process : MainPC
2020-01-21 20:52:15.047 DEBUG _main APIRcvReq:1579607534.905455
2020-01-21 20:52:15.047 DEBUG _main APISndReq:1579607534.905455
2020-01-21 20:52:15.047 DEBUG _main MngRcvReq:1579607534.908
2020-01-21 20:52:15.047 DEBUG _main MngQueIn:1579607534.908
2020-01-21 20:52:15.048 DEBUG _main MngQueOut:1579607534.908
2020-01-21 20:52:15.048 DEBUG _main MngSndReq:1579607534.913
2020-01-21 20:52:15.048 DEBUG _main MngRcvRes:1579607535.027
2020-01-21 20:52:15.048 DEBUG _main MngSndRes:1579607535.027
2020-01-21 20:52:15.048 DEBUG _main APIRcvRes:1579607535.046815
2020-01-21 20:52:15.048 DEBUG _main APISndRes:1579607535.046815
2020-01-21 20:52:15.049 DEBUG _main process : sample
2020-01-21 20:52:15.049 DEBUG _main coprocessor : subsys2, container : sample
2020-01-21 20:52:15.049 DEBUG _main CoproRcvReq:1579038297.606501
2020-01-21 20:52:15.049 DEBUG _main InferenceStart:1579038297.606501
2020-01-21 20:52:15.049 DEBUG _main infinibrain sample start:1579038297.606501
2020-01-21 20:52:15.049 DEBUG _main infinibrain sample end:1579038297.606501
2020-01-21 20:52:15.050 DEBUG _main InferenceEnd:1579038297.606501
2020-01-21 20:52:15.050 DEBUG _main CoproSndRes:1579038297.606501
2020-01-21 20:52:15.050 DEBUG _main #####success#####

C:\Program Files\FUJITSU CLIENT COMPUTING LIMITED\SDK\Basic\SDK Distributed Manager\IBSDK\Sample\SDK_Operation_Verification>
```

Infini-Brain A101/B, A101/BH
SDK 使い方ガイド

B6FY-4951-01 Z0-00

発行日 2020 年 6 月
発行責任 富士通株式会社

〒105-7123 東京都港区東新橋 1-5-2 汐留シティセンター

- このマニュアルの内容は、改善のため事前連絡なしに変更することがあります。
- このマニュアルに記載されたデータの使用に起因する第三者の特許権およびその他の権利の侵害については、当社はその責を負いません。
- 無断転載を禁じます。