IBM MQ V9.3 機能と構成

5. MQアプリケーション

2023年09月

日本アイ・ビー・エム　システムズ・エンジニアリング（株）

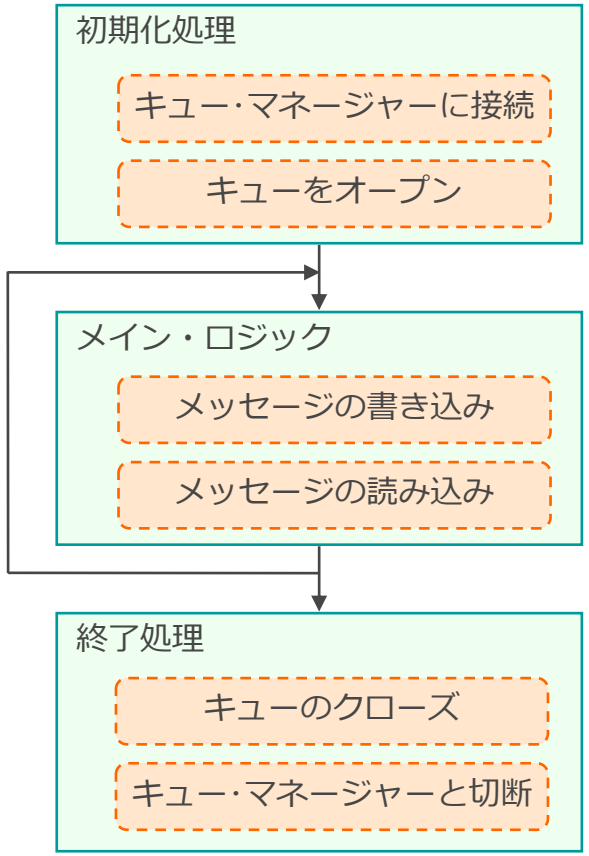
トピック

- MQアプリケーションの構造
- メッセージ
- MQI
  - API一覧
  - MQIを用いたアプリケーションの流れ
  - サンプル
- MQのJavaインターフェース
  - JMS 2.0/Jakarta Messaging 3.0
    - JMS 2.0/Jakarta Messaging 3.0について
    - 主要なAPIの説明
    - JMS 2.0/Jakarta Messaging 3.0を用いたアプリケーションの流れ
    - サンプル
  - MQ Base Java
    - MQ Base javaについて
    - 主要なAPIの説明
    - MQ Base javaを用いたアプリケーションの流れ
    - サンプル
- messaging REST API
  - messaging REST APIについて
  - messaging REST APIの仕様
  - 実行例

MQアプリケーションの構造

MQアプリケーションの構造

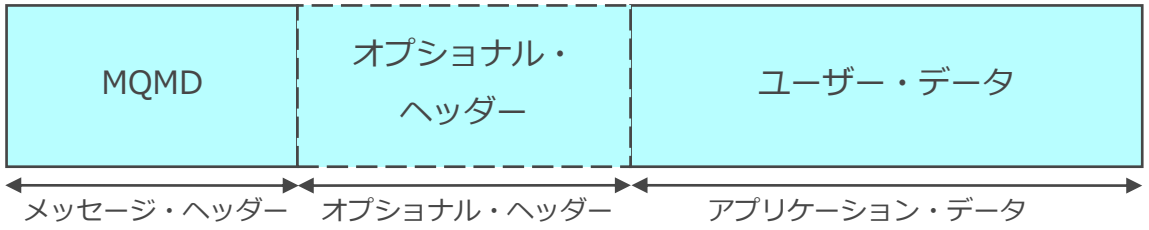
- キューに対してPUT/GETを行う代表的なMQアプリケーションの流れ
  - ◆ 初期化処理
    - キュー・マネージャーに接続
    - キューをオープン
  - ◆ メイン・ロジック
    - キューにメッセージを書き込み
    - キューからメッセージを読み込み
  - ◆ 終了処理
    - キューをクローズ
    - キュー･マネージャーとの接続を切断
- MQが提供するAPI
  - ◆ MQI(Message Queue Interface)
    - MQ標準のAPI
    - C、C++、COBOL、MQ Base javaをはじめ多数の言語をサポート
  - ◆ Java Message Service(JMS) 2.0/Jakarta Messaging 3.0
    - Java環境での標準メッセージング・インターフェース



メッセージ

メッセージ

- キューを介してプログラム間で送受信されるデータ
- メッセージの構成
  - ◆ メッセージ・ヘッダー(MQMD)
  - ◆ オプショナル・ヘッダー(RFH2やIIHなど)
  - ◆ ユーザー・データ
    - キュー･マネージャーは最大100MBまでのユーザー・データを処理可能



- MQMD
  - ◆ メッセージの識別情報やメッセージの制御情報などを含むメッセージ・ヘッダー
    - MQMDの主なフィールド

フィールド	説明
Report	レポート・メッセージを受け取るための指定
MsgType	扱うメッセージのタイプ
Expiry	メッセージの有効期限
CodedCharSetID	ユーザー・データのCCSID
Format	ユーザー・データのフォーマット
Priority	メッセージの優先順位
Persistence	メッセージの永続性
MsgID/CorrelID	メッセージの識別ID
ReplyToQ/ReplyToQMgr	サーバ応答先やレポート・メッセージの送信先

MQI

MQI

- MQI
  - ◆ MQの標準API
  - ◆ 多数の言語から同一のインターフェースでプログラミング可能
  - ◆ 稼動環境(プラットフォーム、ネットワーク・プロトコル)の違いを吸収
  - ◆ メッセージの読み書き、同期点制御を始めとする様々なロジックを提供
- MQIが使用できる開発言語のサポート状況

	AIX	Linux	Windows	z/OS	IBM i
C	○	○	○	○	○
C++	○	○	○	○	○
COBOL	○	–	○	○	○
PL/I	– (v5.3まで○)	–	– (v6.0まで○)	○	–
VB	–	–	○	–	–
ActiveX	–	–	– (v9.1まで○)	–	–
.NET	–	○	○	–	–
MQ Base Java(注1)	○	○	○	○	○
JMS 2.0(注2)	○	○	○	○	○
Jakarta Messaging 3.0(注3)	○	○	○	○	○
RPG	–	–	–	–	○
Assembler	–	–	–	○	–

注1：MQ Base Javaは、v8.0の機能レベルまでをサポートし、今後の新機能の追加や機能拡張は提供されない
注2：JMS 2.0は、JMS 2.0 standardをサポートするが、既存のJMS 2.0アプリケーションの保守および拡張のみで、新規の機能拡張は後継のJakarta Messaging 3.0で行われる
注3：Jakarta Messaging 3.0はJMS 2.0と機能的に同等であり、新規にJavaでメッセージング・アプリケーションを開発する際はJakarta Messaging 3.0の使用が推奨される

MQI一覧

■ MQIの一覧と役割ごとの分類

キュー・マネージャーとの接続/切断	
MQCONN	キューマネージャーに接続
MQCONNX	キューマネージャーに接続(拡張)
MQDISC	キューマネージャーから接続
キューのオープン/クローズ	
MQOPEN	キューをオープン
MQCLOSE	キューをクローズ
メッセージの読み/書き	
MQPUT	メッセージを書き込む
MQGET	メッセージを読み込む
MQPUT1	キューをオープンしてメッセージを１つ書き込む
オブジェクト定義情報の紹介/更新	
MQSET	属性を設定
MQINQ	属性を参照
作業単位(UOW)の制御	
MQCMIT	UOWをコミット
MQBACK	UOWをバックアウト
MQBEGIN	グローバルトランザクションを開始

Pub/Sub	
MQSUB	サブスクリプション登録
MQSUBRQ	リテインメッセージ受信をリクエスト
メッセージ・プロパティ	
MQSETMP	メッセージ・プロパティを設定
MQINQMP	メッセージ・プロパティを参照
MQDLTMP	メッセージ・プロパティを削除
MQCRTMH	メッセージ・ハンドルを作成
MQDLTMH	メッセージ・ハンドルを削除
MQBUFMH	バッファーをメッセージ・ハンドルに変換
MQMHBUF	メッセージ・ハンドルをバッファーに変換
非同期メッセージ受信	
MQCB	コールバック関数の登録
MQCTL	コールバック関数の制御(開始/停止)
MQCB_FUNCTION	コールバック関数の登録(CICS用)
非同期メッセージ送信	
MQSTAT	非同期メッセージ受信状況の確認

MQIが提供する主な構造体とデータ・タイプ

主な構造体

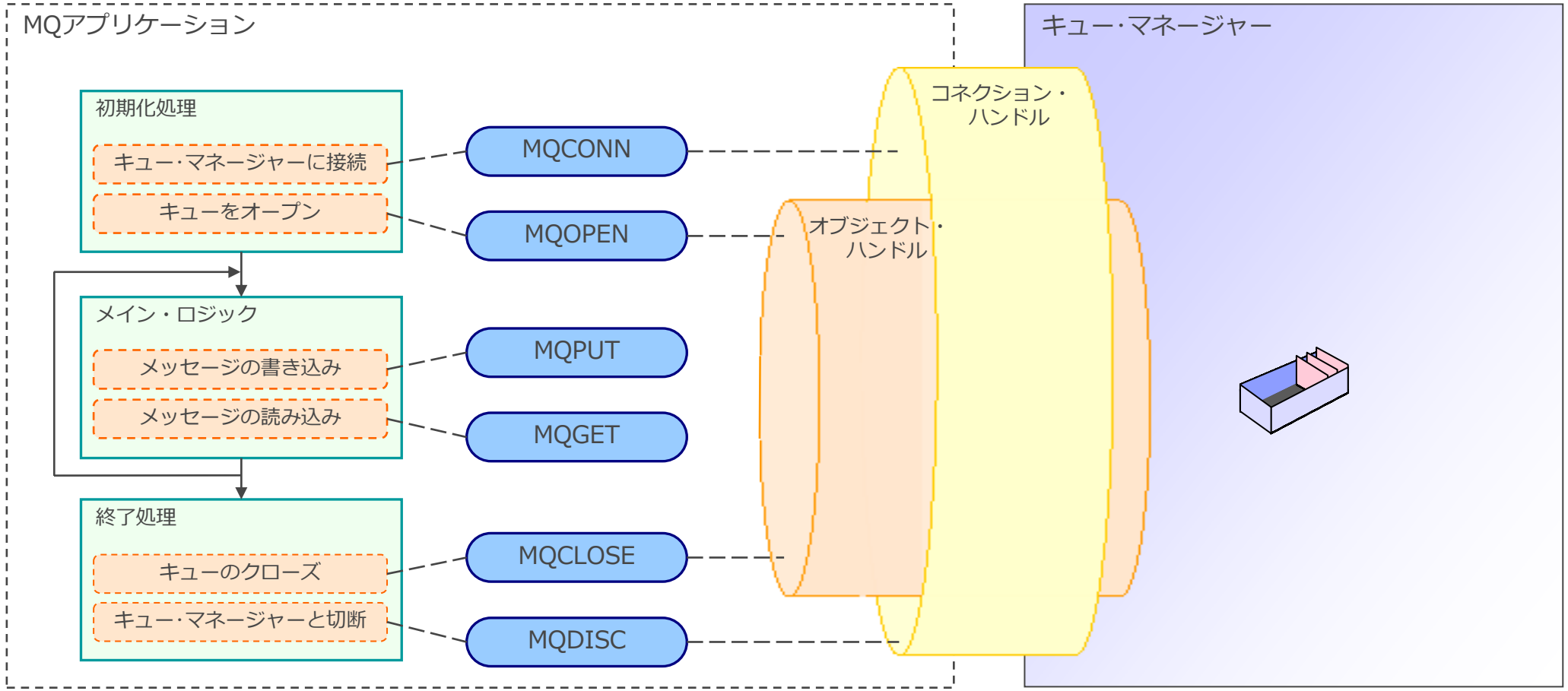
構造体	使用目的
MQOD	オープンするキュー名の指定
MQMD	メッセージヘッダー、メッセージの属性を指定(永続性、メッセージID、プライオリティ、存続時間など)
MQPMO	メッセージの送信方法に関するオプション(同期点処理の要否など)
MQGMO	メッセージの受信方法に関するオプション(同期点処理の要否、受信待機の要否、待機時間など)

主なデータ・タイプ

データ・タイプ	説明
MQBYTE	1バイトのバイナリ・データ
MQBYTEn	nバイトのバイナリ・データ
MQCHAR	1文字の文字列ストリング
MQCHARn	n文字の文字列ストリング
MQHCONN	コネクション・ハンドル(実際のデータ・タイプはMQLONG)
MQHOBJ	オブジェクト・ハンドル(実際のデータ・タイプはMQLONG)
MQLONG	フル・ワード整数

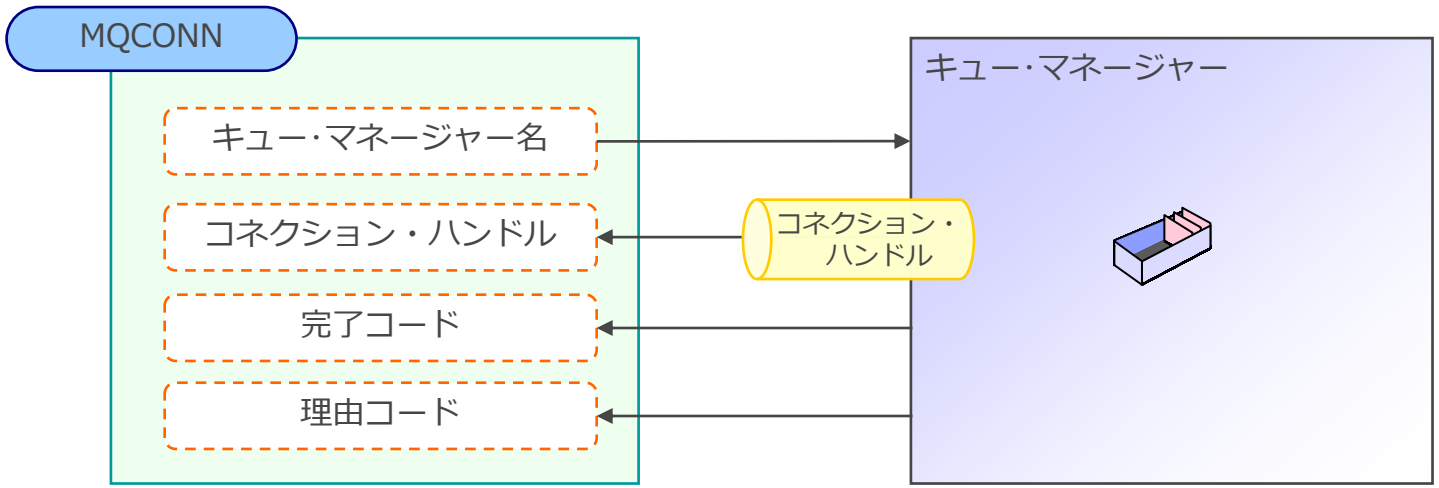
MQIを用いたプログラミングの流れ

- MQIを用いた、キューに対してPUT/GETを行う代表的なMQアプリケーションの流れ
  - キュー･マネージャー、キューに対するそれぞれの処理に対応するMQIを使用
  - キュー･マネージャー、キューとの接続には、コネクション・ハンドル、オブジェクト・ハンドルが必要



MQCONN

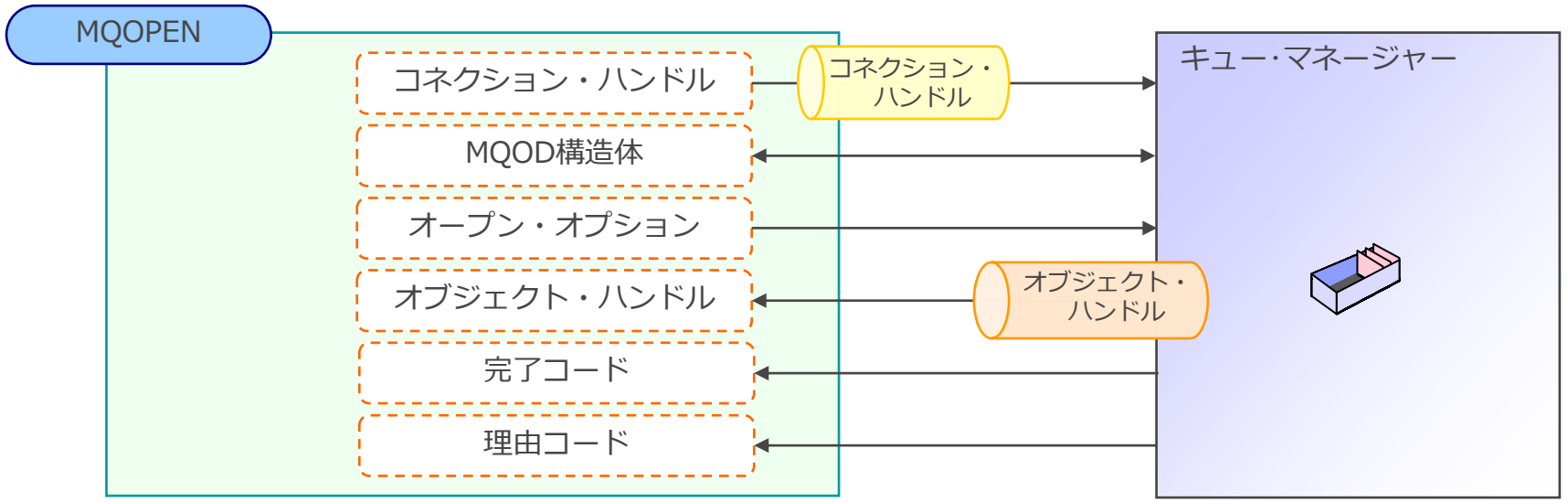
- キュー･マネージャーとの接続に用いる関数
- キュー･マネージャーとの接続に成功すると、コネクション・ハンドルが返る
  - ◆ アプリケーションとキュー･マネージャーとの接続を識別するためのハンドル
  - ◆ 後続のAPIを呼び出す際には、コネクション･ハンドルが必須
- 指定するパラメーター
  - ◆ 接続先キュー・マネージャー名



※ 完了コード: 実行結果(正常終了/異常終了/エラー)
※ 理由コード: エラーや警告の理由を表す値

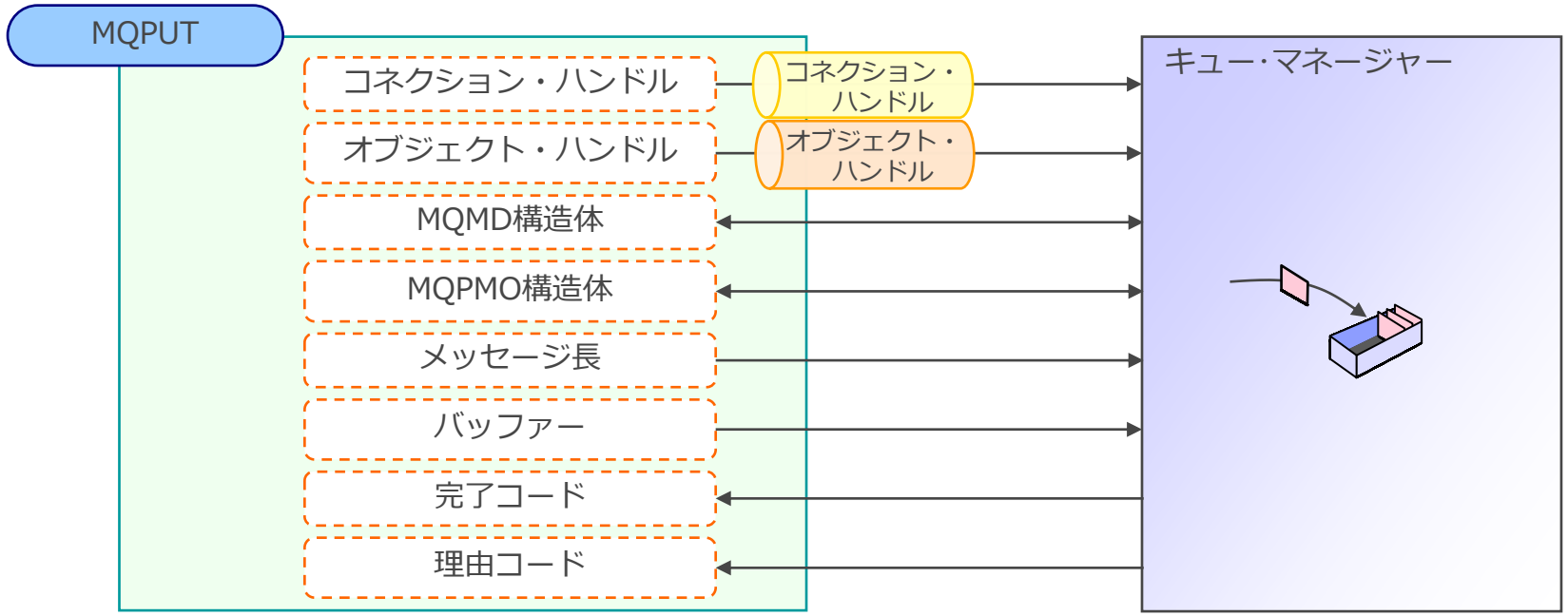
MQOPEN

- キューをはじめとするMQオブジェクトをオープンするため関数
- MQオブジェクトのオープンに成功すると、オブジェクト・ハンドルが返る
  - ◆ MQオブジェクトとの接続を識別するためのハンドル
  - ◆ オブジェクトに対する操作には、オブジェクト・ハンドルが必須
- オブジェクト毎にMQOPENを呼び出すことが必要
  - ◆ 送信用、受信用キューを使用する場合は2回MQOPENをする必要がある
- 指定するパラメーター
  - ◆ MQOD構造体: オブジェクト名、オブジェクトのタイプを指定
  - ◆ オープン・オプション: オブジェクトの使用目的(書き込み用、読み込み用)などを指定



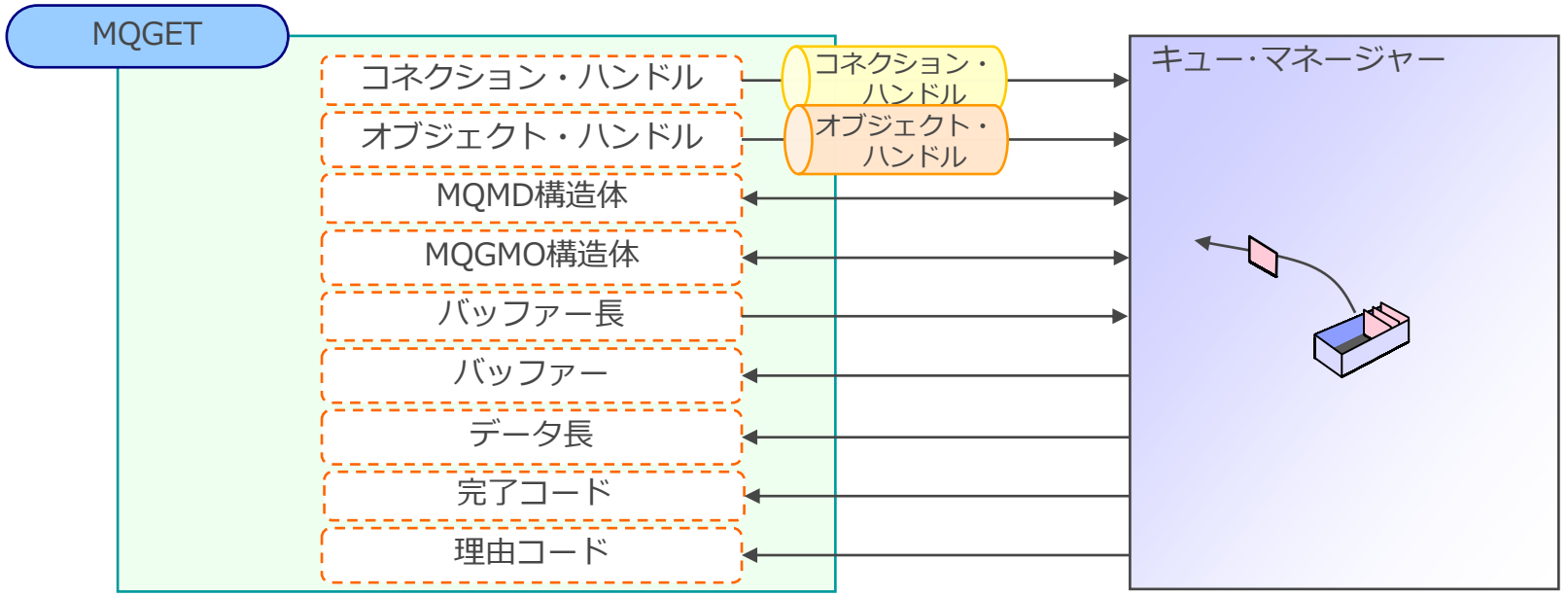
MQPUT

- キューにメッセージを書き込みむための関数
- 指定するパラメーター
  - ◆ MQMD構造体: メッセージ・ヘッダー(MQMD)に相当する構造体
    - 優先度/メッセージ・タイプ/永続性などを指定
  - ◆ MQPMO構造体: MQPUTを制御するための各種オプションを設定
    - メッセージを同期点処理の対象にするか
    - メッセージIDを新規に割り当てるか など
  - ◆ PUTするメッセージを格納するバッファーとメッセージ長



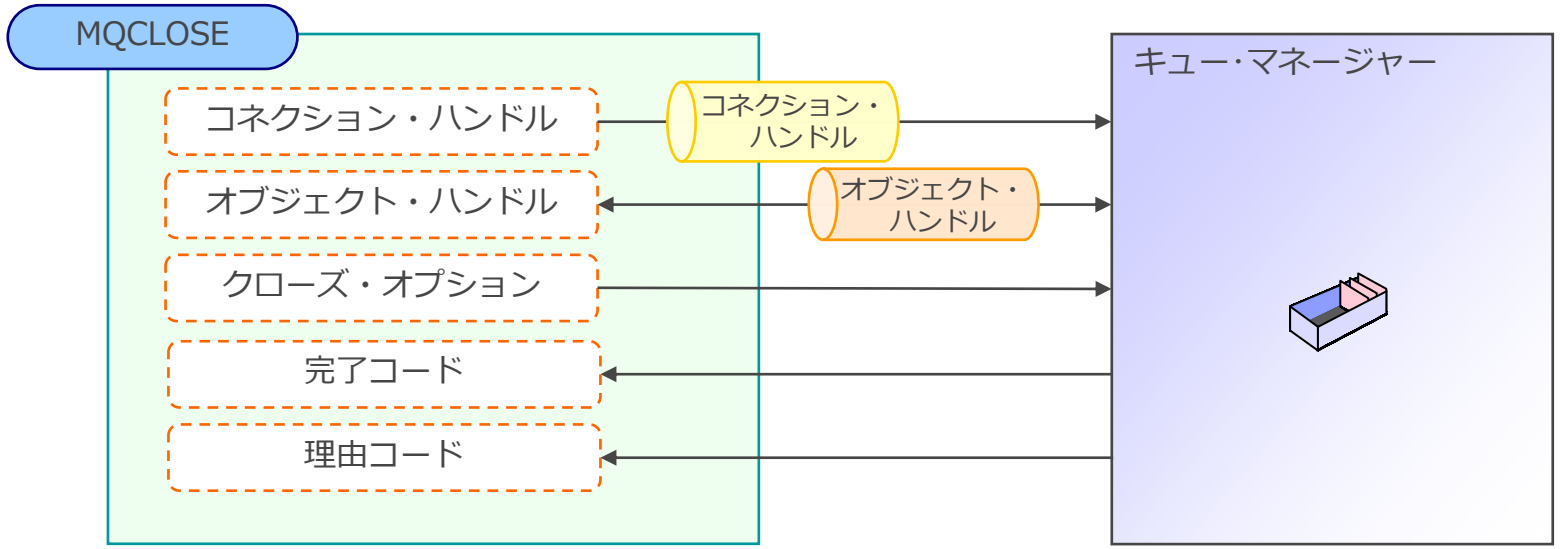
MQGET

- キューからメッセージを読込む関数
  - ◆ 指定したバッファーにキューから読み込んだメッセージのユーザー・データが格納される
- MQMD構造体には読み込んだメッセージのヘッダ部分がコピーされる
- 指定するパラメーター
  - ◆ MQMD構造体: キューから特定のメッセージを読み込む場合に指定
  - ◆ MQGMO構造体: MQGETを制御するための各種オプションを設定
    - メッセージを同期点処理の対象にするか
    - メッセージのコード変換
    - GET Waitの指定 など



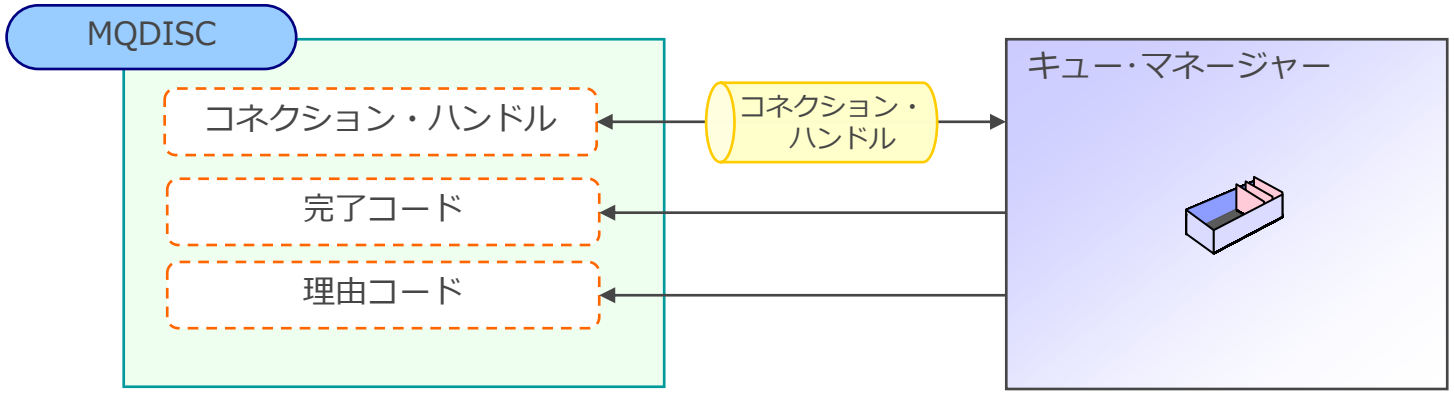
MQCLOSE

- オープンしているMQオブジェクトをクローズするための関数
  - ◆ MQOPENで入手したオブジェクト・ハンドルを解放
- 一旦クローズしたMQオブジェクトを利用するためには、再度MQOPENが必要
- 指定するパラメーター
  - ◆ クローズ・オプション: 動的キューの削除設定などを指定



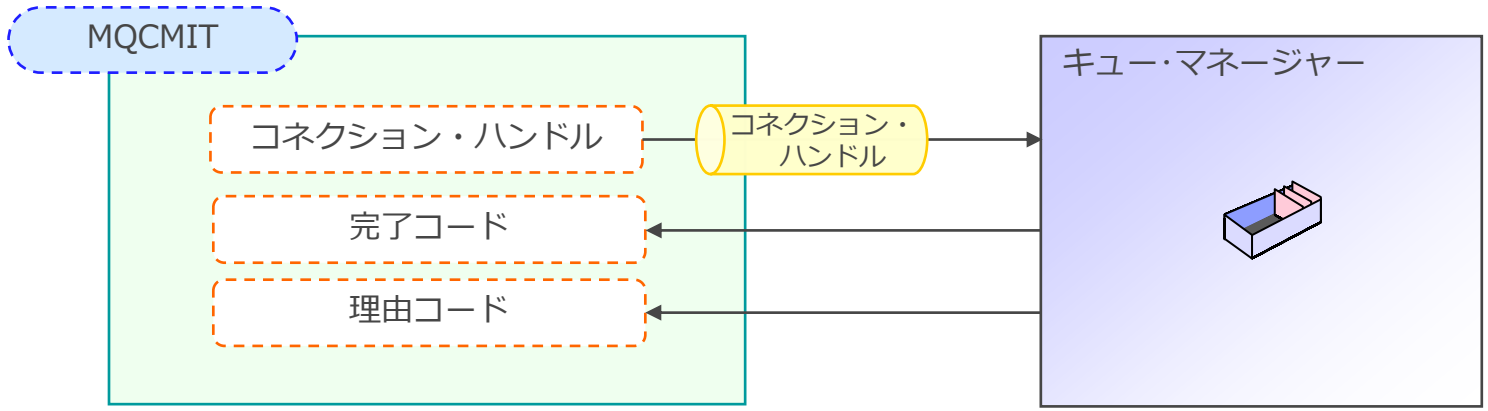
MQDISC

- アプリケーションとキュー･マネージャーとの接続を切断する関数
  - ◆ MQCONNで入手したコネクション・ハンドルの開放
- 一旦切断したキュー･マネージャーへの接続には、再度MQCONNが必要



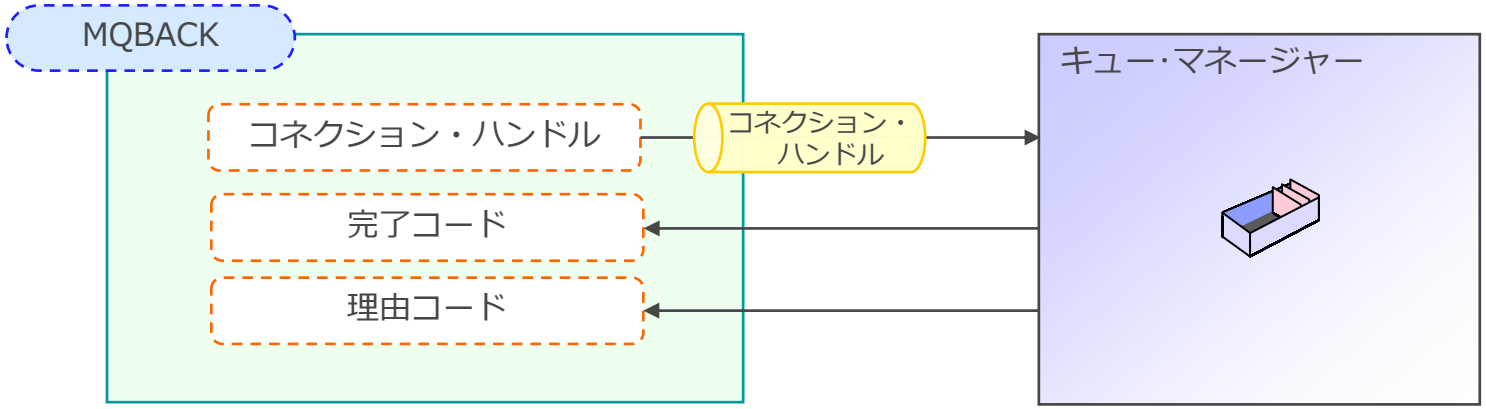
＜参考＞MQCMIT

- 作業単位(UOW)内のPUT/GET処理を確定(COMMIT)するための関数
- 動作
  - ◆ PUT処理の場合
    - COMMITされるまで、同期点処理の対象メッセージは、他のアプリケーションからは読み込み不可
    - COMMITされると、他のアプリーションが読み取ることが可能になる
  - ◆ GET処理の場合
    - COMMITされるまで、同期点処理の対象メッセージは、キューに残る
      - 他のアプリケーションからはアクセス不可
    - COMMITされると、キューからメッセージが消去される
- 指定するパラメーター
  - ◆ コネクション・ハンドル

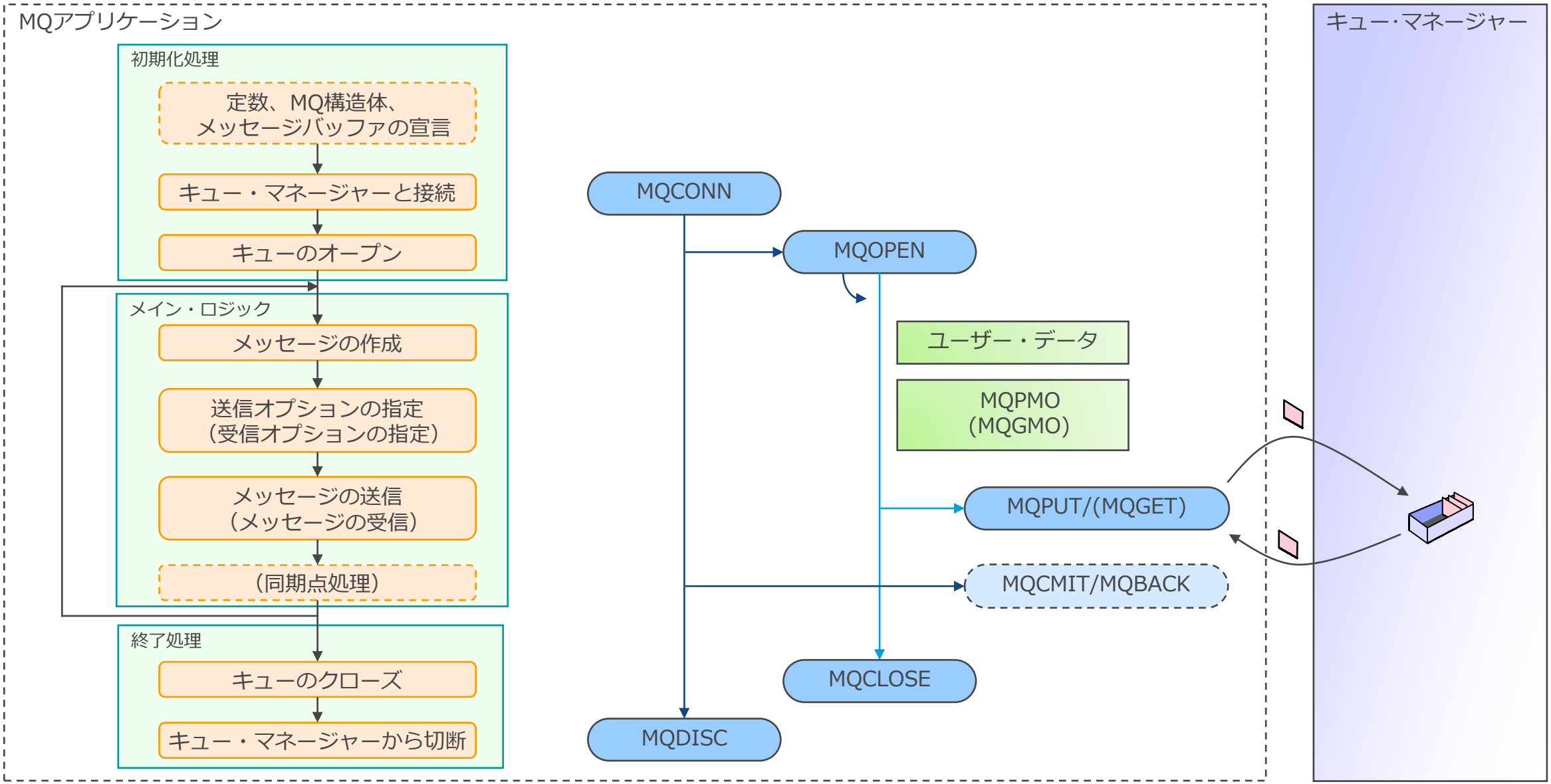


＜参考＞MQBACK

- 作業単位(UOW)内のPUT/GET処理を取り消す(ROLLBACK)するための関数
- 動作
  - ◆ PUT処理の場合
    - COMMITされるまで、同期点処理の対象メッセージは、他のアプリケーションからは読み込み不可
    - ROLLBACKされると、PUTされたメッセージはすべてキューから削除される
  - ◆ GET処理の場合
    - COMMITされるまで、同期点処理の対象メッセージは、キューに残る
    - ROLLBACKされると、GETされたメッセージはすべてキューに復元される
- 指定するパラメーター
  - ◆ コネクション・ハンドル



MQIを用いたプログラミングの手順のまとめ



サンプル・コード(C言語)

- キュー・マネージャー: TESTQMに接続し、キュー: QLにテストメッセージをPUT/GET

```
#include <cmqc.h>                                                    ·····1
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
MQOD od = {MQOD_DEFAULT};                                            ·····2
MQMD md_put = {MQMD_DEFAULT};
MQMD md_get = {MQMD_DEFAULT};
MQPMO pmo = {MQPMO_DEFAULT};
MQGMO gmo = {MQGMO_DEFAULT};

MQHCONN Hcon;                                                        ·····3
MQHOBJ Hobj;
MQLONG O_options,C_options;
MQLONG CompCode, Reason;
MQLONG messlen,buflen;
char buffer_put[1024];
char buffer_get[1024];
char QMName[50];

strcpy(QMName, "TESTQM");                                            ·····4
MQCONN(QMName, &Hcon, &CompCode, &Reason);

strncpy(od.ObjectName, "QL", (size_t)MQ_Q_NAME_LENGTH);
O_options = MQOO_OUTPUT | MQOO_INPUT_SHARED| MQOO_FAIL_IF_QUIESCING ;  ·····5
MQOPEN(Hcon, &od, O_options, &Hobj, &CompCode, &Reason);              ·····6
```

サンプル・コード(C言語)

■ キュー・マネージャー: TESTQMに接続し、キュー: QLにテストメッセージをPUT/GET(続き)

```
memcpy(md_put.Format, MQFMT_STRING, (size_t)MQ_FORMAT_LENGTH);
pmo.Options = MQPMO_NO_SYNCPOINT | MQPMO_FAIL_IF_QUIESCING;                    ·····7
strcpy(buffer_put,"test message");
messlen = strlen(buffer_put);
memcpy(md_put.MsgId, MQMI_NONE, sizeof(md_put.MsgId) );
memcpy(md_put.CorrelId, MQCI_NONE, sizeof(md_put.CorrelId) );

MQPUT(Hcon, Hobj, &md_put, &pmo, messlen, buffer_put, &CompCode, &Reason);      ·····8

gmo.Options = MQGMO_WAIT + MQGMO_ACCEPT_TRUNCATED_MSG;                         ·····9
gmo.WaitInterval = 15000;                                                      ·····10
buflen = sizeof(buffer_get) - 1;

memcpy(md_get.MsgId, MQMI_NONE, sizeof(md_get.MsgId));
memcpy(md_get.CorrelId, MQCI_NONE, sizeof(md_get.CorrelId));
md_get.Encoding = MQENC_NATIVE;
md_get.CodedCharSetId = MQCCSI_Q_MGR;

MQGET(Hcon, Hobj, &md_get, &gmo, buflen, buffer_get, &messlen, &CompCode, &Reason);  ·····11
buffer_get[messlen] = '¥0';
printf("message <%s>¥n", buffer_get);

MQCLOSE(Hcon, &Hobj, C_options, &CompCode, &Reason);                           ·····12
MQDISC(&Hcon, &CompCode, &Reason);                                             ·····13

return(0);
}
```

※エラー・ハンドリングは含まれていません

サンプル・コード(C言語)

1. ヘッダー・ファイル: cmqc.hのインクルード
2. 各種構造体の宣言
3. 各種変数の宣言
4. MQCONNよりキュー･マネージャー: TESTQMとの接続を開始
5. オープン・オプションの設定
6. MQOPENよりキュー: QLのオープン
7. MQPMOのPUTオプションを設定(ノン･パーシステント)
8. MQPUTよりメッセージをキュー: QLに書き込み
9. MQGMOのGETオプションを設定(待機受信設定)
10. GETの待機時間を15秒に設定
11. MQGETよりメッセージをキュー: QLから読み込み
12. キューをクローズ
13. キュー・マネージャーから切断

主要なサンプル･プログラム(C言語)

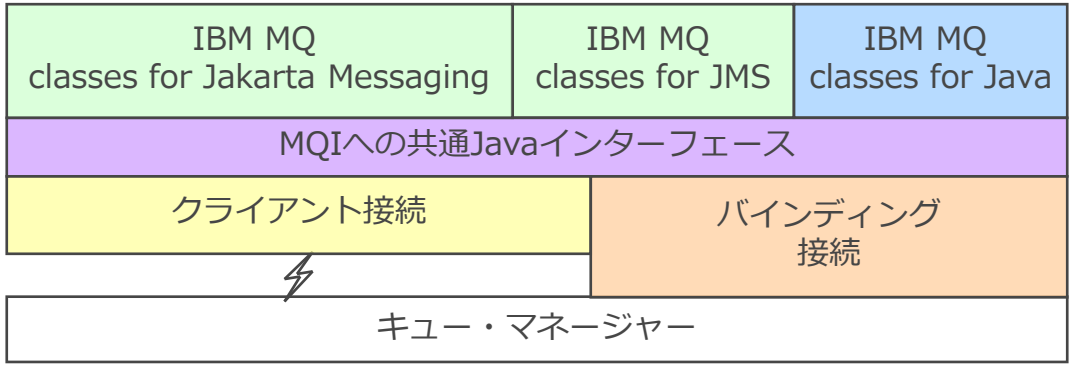
- MQ提供のサンプル・プログラム
  - ◆ 配置ディレクトリー
    - UNIX系：　<導入ディレクトリ>/mqm/samp
    - Windows：　<導入ディレクトリ>¥tools¥c¥Samples

	ソースファイル	実行モジュール	実行モジュール（MQクライアント）
メッセージの書き込み	amqsput0.c	amqsput	amqsputc
メッセージの読み取り	amqsget0.c	amqsget	amqsgetc
メッセージのブラウズ	amqsgbr0.c	amqsgbr	amqsgbrc
メッセージのブラウズ	amqsbcg0.c	amqsbcg	amqsbcgc

MQのJavaインターフェース

MQのJavaインターフェース

- JMS 2.0
  - ◆ Java EE標準のメッセージング・インターフェース
    - JMS 1.1のAPI(クラシックAPI)を保持しながら、クラシックAPIと同等の機能を持つ簡易APIを提供
  - ◆ v8.0からJMS 2.0をサポートするIBM MQ classes for JMSが提供
    - v9.3ではサポートは提供されるが、今後の新機能の追加や機能拡張は提供されない
      - 既存のアプリケーションの保守/拡張、もしくは、Jakarta Messaging 3.0をサポートしない環境での使用に推奨
- Jakarta Messaging 3.0
  - ◆ Jakarta EE標準のメッセージング・インターフェース
    - JMS 2.0の後継であり、JMS 2.0とJakarta Messaging 3.0の間の新機能はなし
  - ◆ v9.3からJakarta Messaging 3.0をサポートするIBM MQ classes for Jakarta Messagingが提供
    - IBM MQ classes for JMSと機能的に同等
      - 今後の機能拡張は、IBM MQ classes for Jakarta Messagingにて行われる
      - 新規アプリケーションを開発する際の使用に推奨
- MQ Base Java(IBM MQ classes for Java)
  - ◆ IBM MQ(MQ)専用のメッセージング・インターフェース
  - ◆ MQIそのままのイメージでコーディング可能
  - ◆ v8.0の機能レベルまでサポート
    - 今後の新機能の追加や機能拡張は提供されない

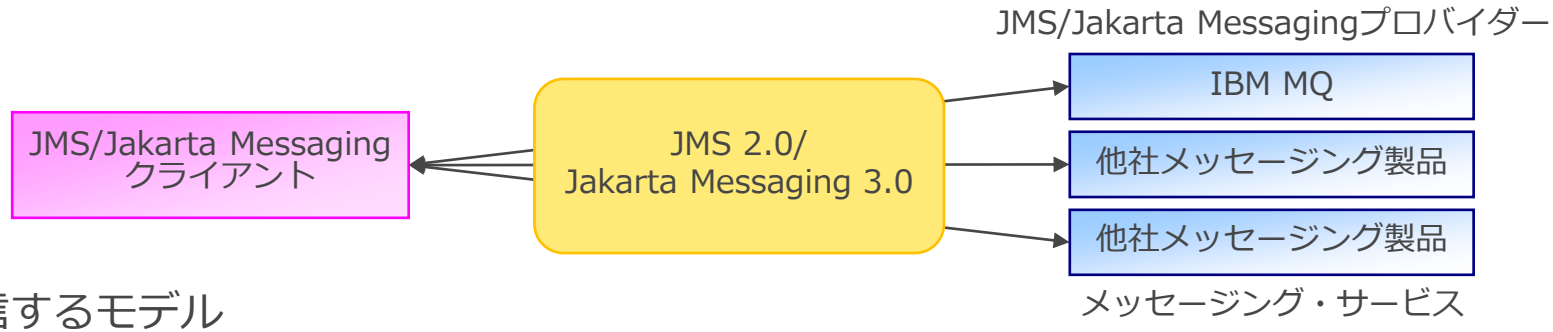


MQのJavaインターフェース

- MQリソース・アダプター
  - ◆ JMS 2.0用のMQリソース・アダプター
    - IBM MQ classes for JMSを含むJCA 1.7インターフェースを実装
    - Java EE対応のアプリケーション・サーバーでMQへのJMS 2.0を使用した接続をサポート
  - ◆ Jakarta Messaging 3.0用のMQリソース・アダプター
    - IBM MQ classes for Jakarta Messagingを含むJakarta Connectors 2.0.0インターフェースを実装
    - Jakarta EE対応のアプリケーション・サーバーでMQへのJakarta Messaging 3.0を使用した接続をサポート
  - ◆ WebSphere Application Server traditionalでのMQリソース・アダプターの使用
    - MQリソース・アダプターはWebSphere Application Server traditional 9.0以降でプリインストール
    - 現時点(2023/09)では、Jakarta EEは未サポート
  - ◆ WebSphere Libertyでのリソース・アダプターの使用
    - WebSphere LibertyからMQへの接続には、MQリソース・アダプターが必要
    - 使用するリソース・アダプターのバージョンは、デプロイ先のLibertyのバージョンに依存
- Message Driven Bean(MDB)
  - ◆ Java EE/Jakarta EE標準のメッセージ駆動型処理を行うEJB(Enterprise Java Bean)
  - ◆ JMS 2.0/Jakarta Messaging 3.0の機能を使用して実装

JMS 2.0/Jakarta Messaging 3.0

- Java環境での標準メッセージング・インターフェース
  - ◆ 製品に依存しないアプリケーション構築が可能(アプリケーション・ポータビリティー)
    - プロバイダー間の相互接続については保障されない
  - ◆ 各ベンダーが提供するメッセージング製品にて実装
  - ◆ JMS/Jakarta Messagingプロバイダー
    - JMS/Jakarta Messagingクライアントの実行環境を提供
    - メッセージング製品のJMS 2.0/Jakarta Messaging 3.0の実装クラスを提供
  - ◆ JMS/Jakarta Messagingクライアント
    - JMS/Jakarta Messagingインターフェースにより、メッセージ・サービスを使用するアプリケーション



- 2つのドメイン
  - ◆Point to point(PTP)
    - キュー宛先に対してメッセージを送受信するモデル
  - ◆Publish/Subscribe(Pub/Sub)
    - トピック宛先に対してメッセージを送受信するモデル

JMS 2.0/Jakarta Messaging 3.0

- MQはJMS/Jakarta Messagingプロバイダーとしての機能をもつ
  - MQにアクセス可能な、JMS/Jakarta Messagingインターフェースを実装したJavaクラスを提供
    - IBM MQ classes for JMS(JMS 2.0)
    - IBM MQ classes for Jakarta Messaging(Jakarta Messaging 3.0)
  - MQとのローカル接続/クライアント接続が可能
  - PTPとPub/Subモデルをサポート
    - Pub/Subブローカーエンジンも MQが提供
- JMS特有の機能を提供
  - メッセージ・リスナー、メッセージ・セレクター、豊富なメッセージ・クラスの提供など
  - アプリケーション・サーバー(WAS、WebLogicなど)をコーディネーターとした 2 フェーズ・コミットにMQはリソース・マネージャーとして参加可能
    - MQをコーディネーターとした2フェーズ・コミットは不可

JMS 2.0/Jakarta Messaging 3.0

- MQがサポートする機能は、ほぼ使用可能
  - 使用できない機能
    - グループ・メッセージ、キューの属性照会(MQINQ)、コンテキスト情報の利用、PCFコマンドなど
  - プロバイダー固有のクラス、インターフェースを提供
    - アプリケーションのポータビリティーは、損なわれる
- JMSに準拠したプログラミング手順、デザインが必要
  - MQIと同様なプログラミング手順、デザインは不可

JMS 2.0/Jakarta Messaging 3.0

- JMS 2.0/Jakarta Messaging 3.0の差異
  - Jakarta Messaging 3.0での新機能はなく、命名のみ異なる

    ex) パッケージ名の変更

JMS 2.0パッケージ名	Jakarta Messaging 3.0パッケージ名
javax.*	jakarta.*

- IBM MQ classes for JMS/IBM MQ classes for Jakarta Messagingの差異
  - 機能的に同等、命名のみ異なる

    ex) パッケージ名の変更

IBM MQ classes for JMSパッケージ名	IBM MQ classes for Jakarta Messagingパッケージ名
com.ibm.mq.jms[.*]	com.ibm.mq.jakarta.jms[.*]
com.ibm.jms	com.ibm.jakarta.jms
com.ibm.msg.client.jms.*	com.ibm.msg.client.jakarta.jms.*
com.ibm.msg.client.wmq.*	com.ibm.msg.client.jakarta.wmq.*

    他にも、 MQ拡張機能を有効にするプロパティ名(com.ibm.mq.jakarta.jms.SupportMQExtensions)などが変更

  - 両者は相互接続が可能
    - IBM MQ classes for JMSがプロデューサー、IBM MQ classes for Jakarta Messagingがコンシューマーの構成は可能(逆も可能)
  - 同一アプリケーション内で両クラスの混在は不可

<参考> JMS 1.0/JMS 1.1/JMS 2.0

JMS 1.0/JMS 1.1/JMS 2.0の差異

◆ JMS 1.1の変更点

- JMS 1.0のAPIを保持しながら、2つのドメイン(PTP、Pub/Sub)で共通したAPI(クラシックAPI)を提供
- その他の変更点は以下リンク先を参照
  - Java(TM) Message Service Specification Final Release 1.1

◆ JMS 2.0の変更点

- JMS 1.0/JMS 1.1のAPIを保持しながら、クラシックAPIと同等の機能を持つ簡易APIを提供
  - クラシックAPIよりインターフェースの種類を削減、冗長なオプションを廃止しAPIを簡素化
- その他の変更点は以下リンク先を参照
  - JMS 2.0 Final Release

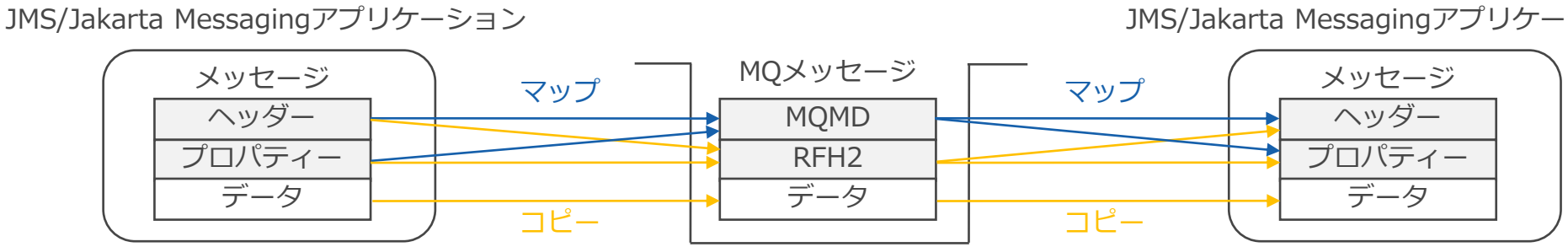
MQ v5.2 -		**MQ v6.0 -**	**MQ v8.0 -**
JMS 1.0		**JMS 1.1**	**JMS 2.0**
PTPドメイン・モデル	**Pub/Subドメイン・モデル**	**クラシックAPI**	**簡易API**
QueueConnectionFactory	TopicConnectionFactory	ConnectionFactory	ConnectionFactory
QueueConnection	TopicConnection	Connection	JMSContext
QueueSession	TopicSession	Session	
QueueSender	TopicPublisher	MessageProducer	JMSProducer
QueueReceiver	TopicSubscriber	MessageConsumer	JMSConsumer
QueueBrowser			

JMS 2.0/Jakarta Messaging 3.0 - メッセージ

- JMS/Jakarta Messagingクライアント、プロバイダー間でやり取りされるメッセージ
  - JMS 2.0/Jakarta Messaging 3.0で扱うメッセージの構成は、MQメッセージと若干異なる
- メッセージの構成

ヘッダー	プロパティー	ボディー

  - ヘッダー：メッセージの識別情報やメッセージの制御情報など
  - プロパティー：ヘッダーに加え、付加的な情報を保持するフィールド
  - ボディー：ユーザーデータ
    - BytesMessage/TextMessage/MapMessage/StreamMessage/ObjectMessageの5タイプをサポート
- JMS/Jakarta MessagingクライアントがMQに書き込んだメッセージは、MQメッセージにマップ、および、コピー
  - MQMDにヘッダー、および、プロパティーと対応するフィールドが含まれている場合、ヘッダー、および、プロパティーはMQMDフィールドにマッピング
  - MQMDフィールドに存在しないヘッダー、または、プロパティーはMQRFH2フィールドにコピー

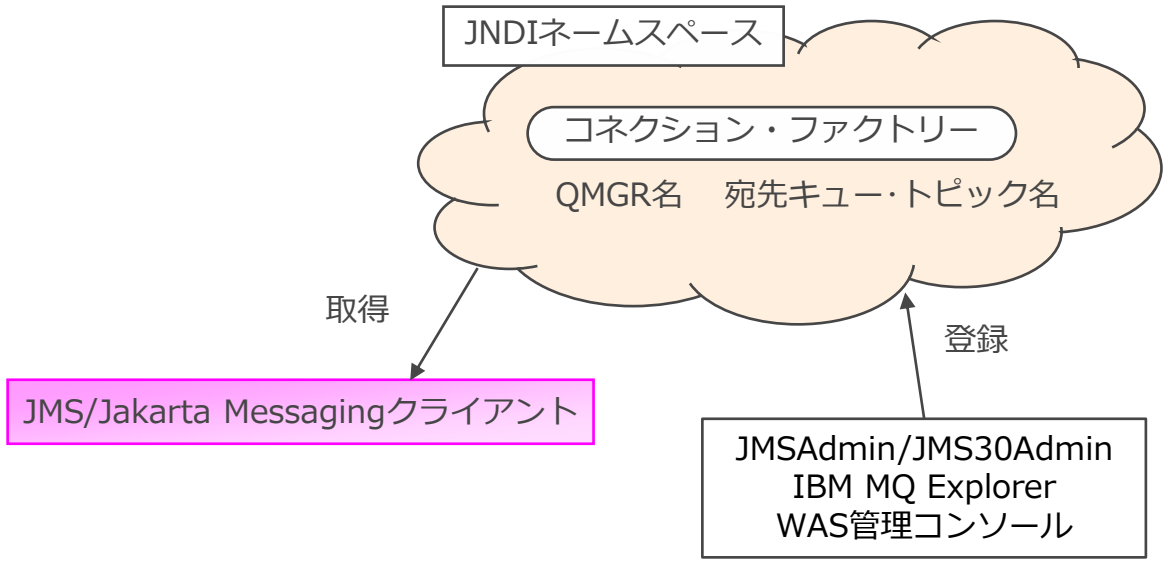


JMS 2.0/Jakarta Messaging 3.0 - 管理オブジェクト

- JMS/Jakarta Messagingプロバイダーに接続するための情報を持つ、Javaオブジェクト
  - 各社のメッセージング製品に関する情報を、JMS/Jakarta Messagingクライアントから切り離し、アプリケーションのポータビリティーを高めることを目的
  - JNDIネーム･スペースに格納
    - ファイル、LDAP、WASネーミング・サービス
  - JMS/Jakarta Messagingクライアントは実行時にネーム･スペースにアクセスし、オブジェクトを取得
- JMS/Jakarta Messaging管理オブジェクト
  - コネクション・ファクトリー
    - ConnectionFactory
  - 宛先
    - MessageProducer/MessageConsumer

JMS 2.0/Jakarta Messaging 3.0 - 管理オブジェクト

- プロバイダーが用意する管理ツールで登録、および属性の設定、変更
  - JNDI管理ツール(MQが提供)
    - JMSAdminコマンド(JMS 2.0)
    - JMS30Adminコマンド(Jakarta Messaging 3.0)
  - IBM MQ Explorer
  - WAS管理コンソール
  - WebSphere Libertyのserver.xmlファイル



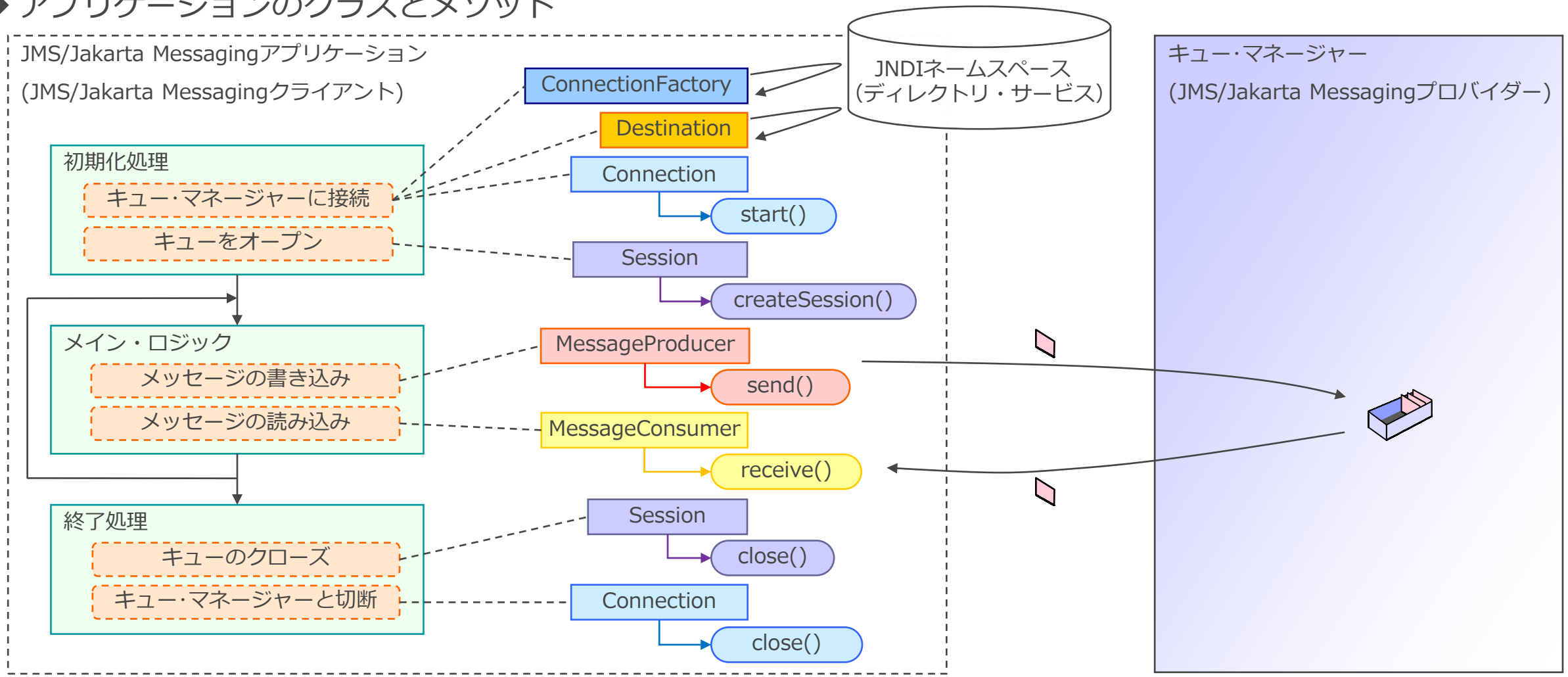
JMS 2.0/Jakarta Messaging 3.0 - 主要API

JMS 2.0/Jakarta Messaging 3.0が提供する主なクラスとメソッド

クラス	用途とメソッド
ConnectionFactory	・JMS/Jakarta Messagingクライアント(JMS/Jakarta Messagingアプリケーション)がJMS/Jakarta Messagingプロバイダーと接続するために必要な情報を設定 ・主なメソッド - createConnection() - createSession()
Destination	メッセージの送信先/受信先のキューやトピックを設定
Connection	JMS/Jakarta MessagingクライアントとJMS/Jakarta Messagingプロバイダーとの接続を管理
Session	・送信されるメッセージの順序、トランザクションを管理 ・主なメソッド - createProducer() - createConsumer()
MessageProducer	・メッセージを送信 ・主なメソッド - send()
MessageConsumer	・メッセージを受信 ・主なメソッド - receive() - receiveNoWait()
Message	BytesMessage/TextMessage/MapMessage/StreamMessage/ObjectMessageの5タイプのメッセージの親クラス

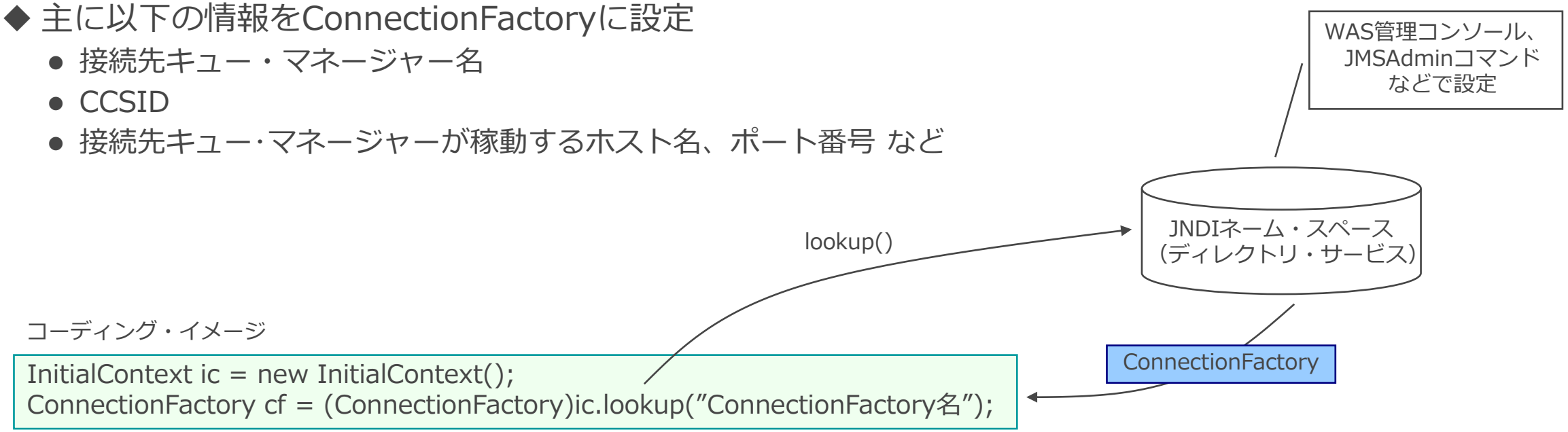
JMS 2.0/Jakarta Messaging 3.0のプログラムの流れ

- MQIとは処理手順が異なる
  - アプリケーションのクラスとメソッド



ConnectionFactory

- JMS/Jakarta Messagingクライアント(JMS/Jakarta Messagingアプリケーション)がJMS/Jakarta Messagingプロバイダーと接続するために必要な情報を設定
  - ◆ 後続のクラスでJMS/Jakarta Messagingプロバイダーとのコネクションを生成する際に必要
  - ◆ JNDIネーム･スペースにアクセスし、ConnectionFactoryオブジェクトを取得
    - InitialContextクラスのlookup()メソッドを使用
    - アプリケーション内でConnectionFactoryオブジェクトを作成、設定することも可能
  - ◆ 主に以下の情報をConnectionFactoryに設定
    - 接続先キュー・マネージャー名
    - CCSID
    - 接続先キュー･マネージャーが稼動するホスト名、ポート番号 など

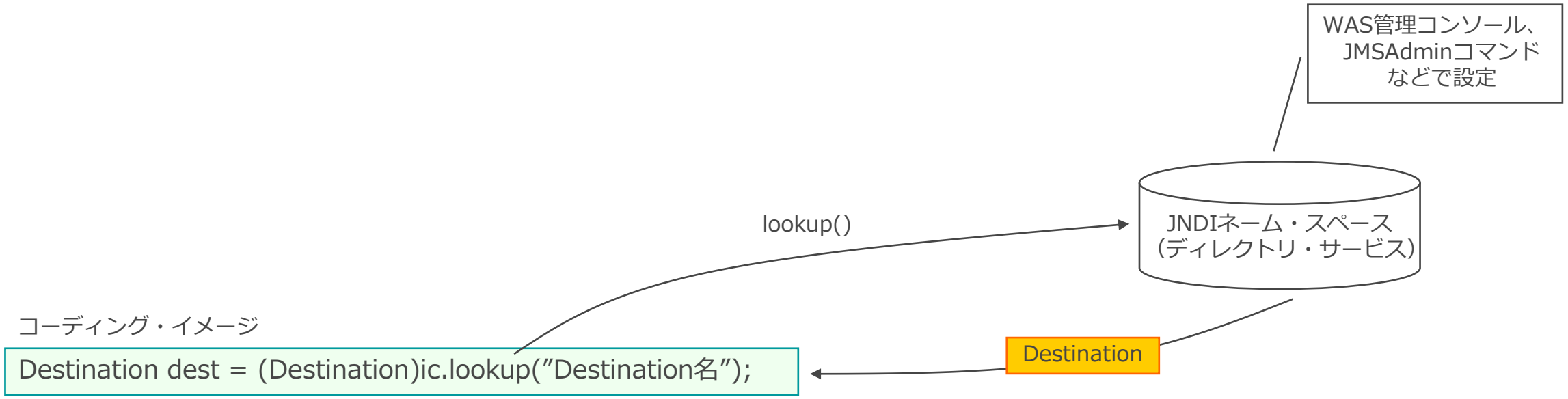


コーディング・イメージ

```
InitialContext ic = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)ic.lookup(”ConnectionFactory名”);
```

Destination

- メッセージの送信先/受信先のキューやトピックを設定
  - ◆ 後続の処理で、MessageProducer、MessageConsumerを生成する際に必要
  - ◆ JNDIネーム･スペースにアクセスし、Destinationオブジェクトを取得
    - InitialContextクラスのlookup()メソッドを使用
    - アプリケーション内でDestinationオブジェクトを作成、設定することも可能



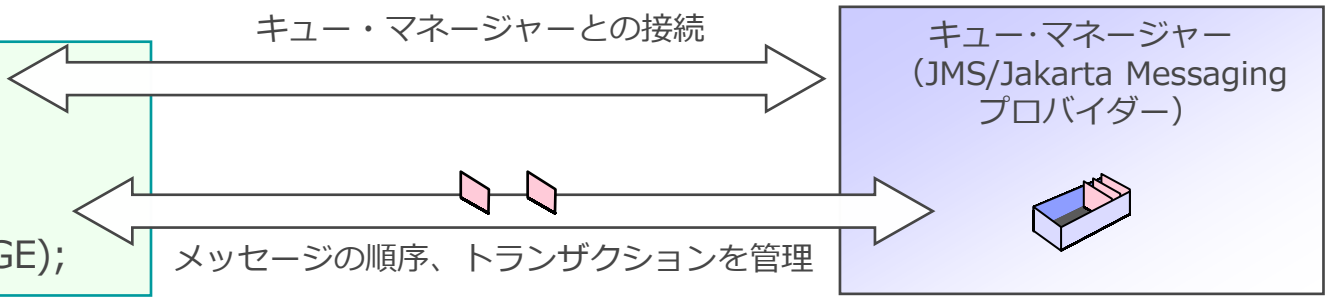
Connection/Session

- Connection
  - JMS/Jakarta MessagingクライアントとJMS/Jakarta Messagingプロバイダーとの接続を管理
    - ConnectionFactoryのcreateConnection()メソッドより取得
    - Sessionオブジェクトの生成に必要
    - start()メソッドから明示的に開始
    - close()メソッドで停止
- Session
  - 送信されるメッセージの順序、トランザクションを管理
    - ConnectionのcreateSession()メソッドより取得
    - 1つのコネクションから複数のSessionオブジェクトを生成することが可能
    - 後続の処理で、MessageProducer、MessageConsumerを生成する際に必要
    - close()メソッドで停止

コーディング・イメージ

```
Connection con = (Connection)cf.createConnection();
con.start();

Session session =
  con.createSession(false,Session.AUTO_ACKNOWLEDGE);
```



MessageProducer

- メッセージを送信
  - ◆ Sessionオブジェクトの createProducer()メソッドより生成
  - ◆ close()メソッドで停止

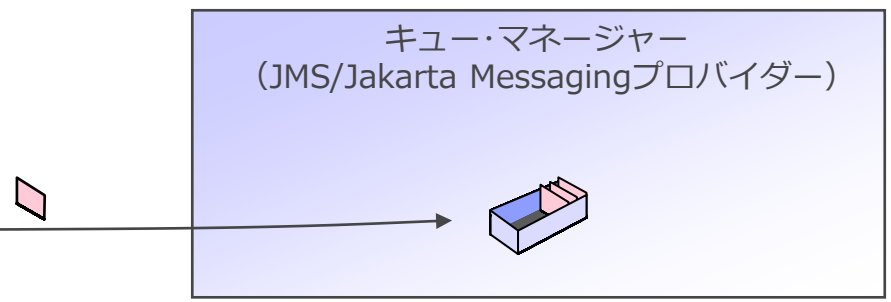
コーディング・イメージ

```
MessageProducer prd = session.createProducer(dest);
```

  - ◆ メッセージの送信処理
    - Messageオブジェクトを、Sessionオブジェクトから生成
      - TextMessageなどのメッセージ･タイプごとにメソッドが異なる
    - MessageProducerのsend()メソッドより、メッセージを送信

コーディング・イメージ: TextMessageを送信する例

```
String msgContent = ”This is a TextMessage”;
TextMessage msg  = session.createTextMessage( );
msg.setText( msgContent );
prd.send(msg);
```



MessageConsumer

- メッセージを受信
  - ◆ Sessionオブジェクトの createConsumer()メソッドより生成
  - ◆ close()メソッドで停止

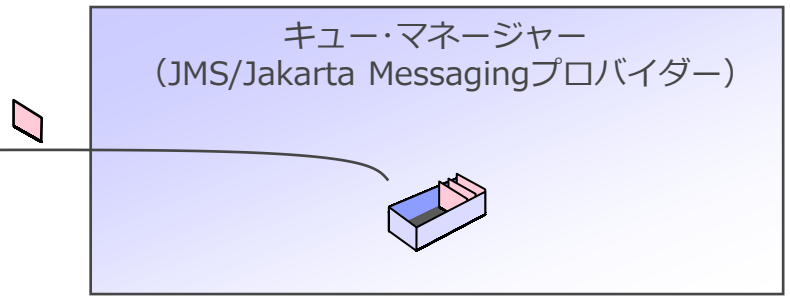
コーディング・イメージ

```
MessageConsumer mc = session.createConsumer(dest);
```

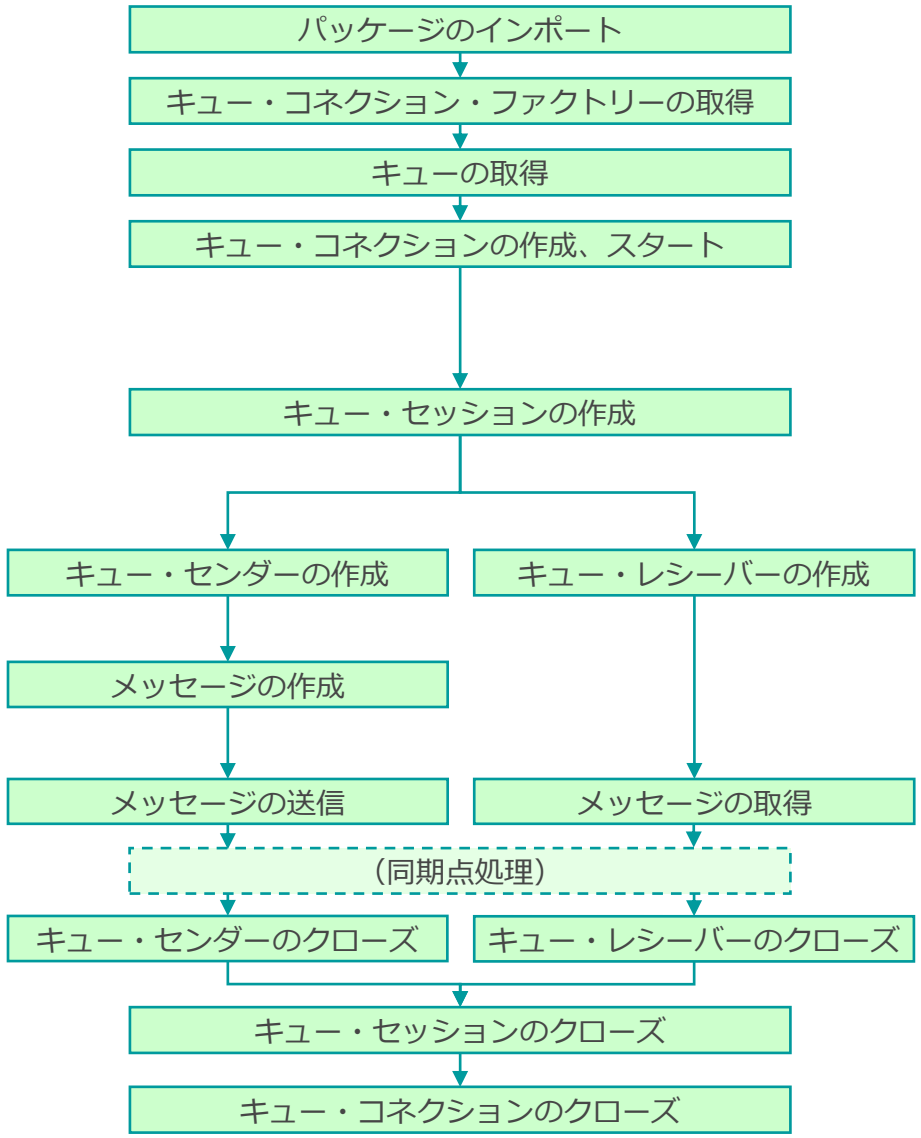
  - ◆ メッセージの受信処理
    - MessageConsumerオブジェクトのメソッドを使用
      - receive()メソッド: GET Waitに相当するメソッド
      - receiveNoWait()メソッド: WaitなしのGETに相当するメソッド
    - Messageオブジェクトが返される

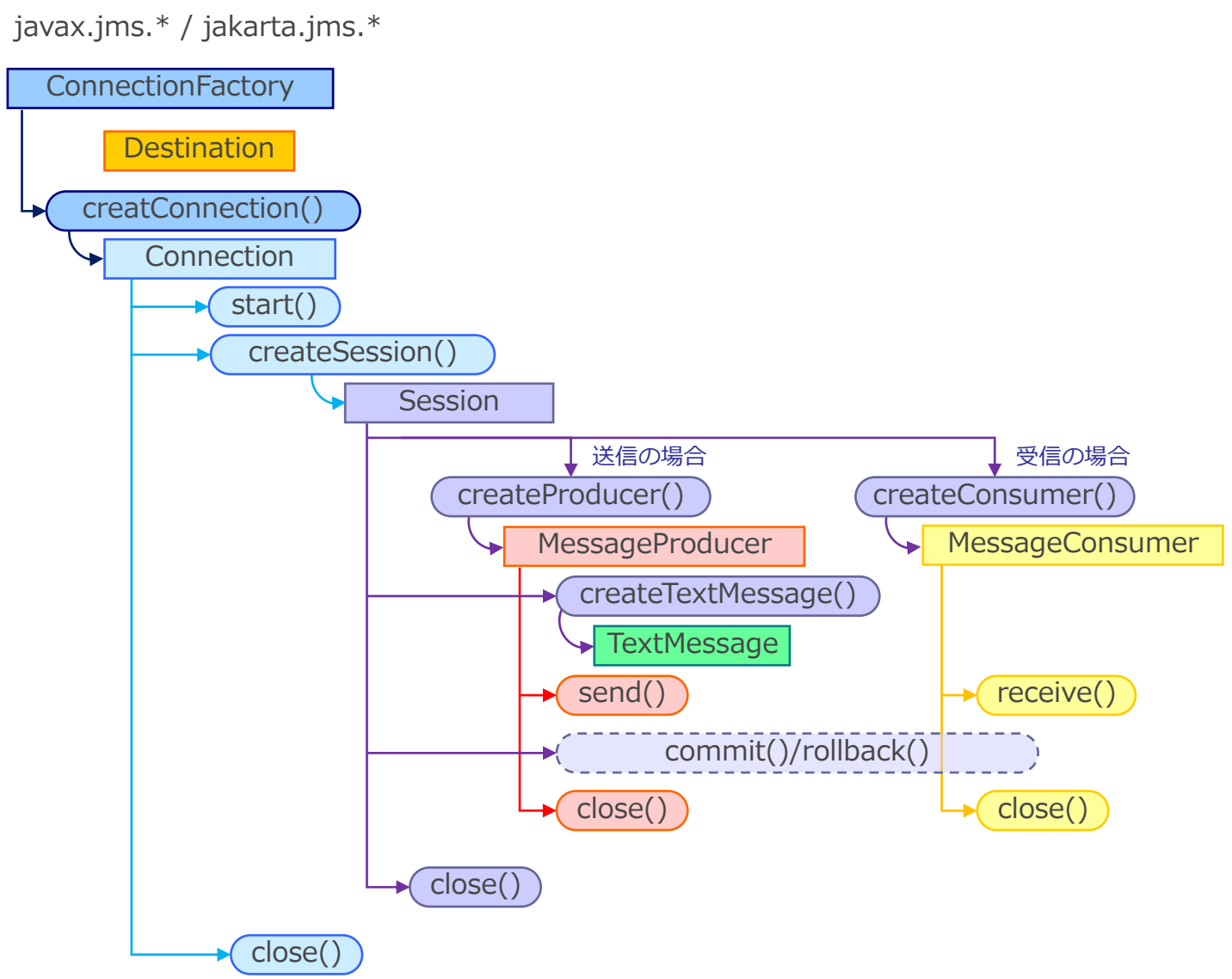
コーディング・イメージ

```
long waitInteral = 3000;          // 待ち時間(ms)
Message receivedMsg = msgReceiver.receive(waitInterval);
String receivedContent = ((TextMessage) receivedMsg).getText();
```



JMS 2.0/Jakarta Messaging 3.0を用いたプログラミング手順





サンプル・コード(JMS 2.0/Jakarta Messaging 3.0)

- JMS 2.0 + クラシックAPI
  - Jakarta Messaging 3.0の場合、javax.jms.* の代わりに jakarta.jms.* を使用

```
import javax.jms.*;                                                          ·····1
import javax.naming.*;

public static void main(String[] args) {
    try{
        InitialContext ic = new InitialContext();
        ConnectionFactory cf = (ConnectionFactory)ic.lookup("QcfName");      ·····2
        Destination dest = (Destination)ic.lookup("QName");                  ·····3

        Connection connection = cf.createConnection();                       ·····4
        connection.start();

        Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);  ·····5

        MessageProducer producer = session.createProducer(dest);             ·····6

        TextMessage outMessage = session.createTextMessage();                ·····7
        outMessage.setText("テキスト・メッセージ");
        producer.send(outMessage);                                           ·····8

        MessageConsumer consumer = session.createConsumer(dest);             ·····9
        Message  inMessage = consumer.receive(10000);                        ·····10
        String message = ((TextMessage)inMessage).getText();                 ·····11

        producer.close();                                                    ·····12
        consumer.close();                                                    ·····13
        session.close();                                                     ·····14
        connection.close();                                                  ·····15
    } catch(javax.jms.JMSException ex){
        System.out.println("JMSException occurred." + ex);
    }
}
```

サンプル・コード(JMS 2.0/Jakarta Messaging 3.0)

1. パッケージのインポート : javax.jms.* (Jakarta Messaging 3.0の場合、jakarta.jms.*)
   - IBM提供クラスを使用する場合は、com.ibm.mq.*、com.ibm.mq.jms.* (Jakarta Messaging 3.0の場合、com.ibm.mq.jakarta.jms.*)もインポート
2. JNDIのcontext.lookupで、QcfNameという名前のコネクション・ファクトリーを入手
3. JNDIのcontext.lookupで、QNameという名前のキュー宛先を入手
4. コネクションの作成と開始: createConnection
5. キュー・セッションの作成: createSession
   - 同期点処理を行うかどうかを指定（サンプルでは「同期点なし」に設定）
6. プロデューサーを作成: createProducer
7. テキスト・メッセージを作成: createTextMessage
8. プロデューサーのsend()メソッドでメッセージを送信
9. コンシューマーを作成: createConsumer
10. コンシューマーのreceive()メソッドでメッセージを受信
    - メッセージの待ち時間をミリ秒単位で指定
11. 受信したメッセージをテキスト・メッセージとして処理
12. プロデューサーのクローズ
13. コンシューマーのクローズ
14. セッションのクローズ
15. コネクションのクローズ

サンプル・コード(JMS 2.0/Jakarta Messaging 3.0)

MQ提供のサンプル・プログラム

JMS 2.0

- 配置ディレクトリー
  - UNIX系： <導入ディレクトリ>/samp/jms/samples
  - Windows： <導入ディレクトリ>¥tools¥jms¥samples

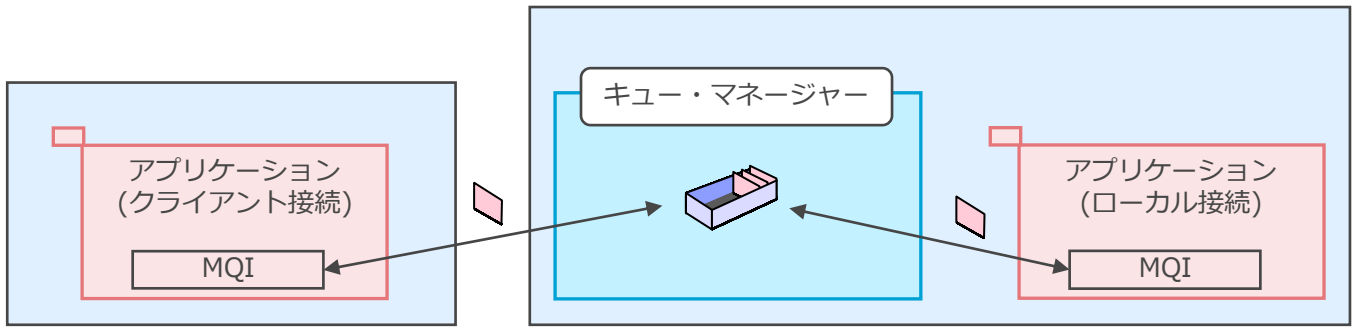
	ソースファイル	コンパイル済み
メッセージの書き込み	JmsProducer.java	JmsProducer.class
メッセージの読み取り	JmsConsumer.java	JmsConsumer.class
メッセージのブラウズ	JmsBrowser.java	JmsBrowser.class

Jakarta Messaging 3.0

- 現時点(2023/09)で、サンプル・プログラムは準備中

MQ Base Java

- MQが提供するIBM独自のJavaインターフェース
  - 従来のMQIと同じプログラミング手順、デザインが可能
  - v8.0の機能レベルまでサポート
    - 今後の新機能の追加や機能拡張は提供されない
- 標準的なMQI機能のサポート
  - MQIがサポートする機能はすべて使用可能
    - 同期点制御、メッセージID / 相関IDの利用、レポート・オプション、MQクラスターなど
  - MQをコーディネーターとした2フェーズ･コミットが可能
    - アプリケーション・サーバー(WAS、WebLogicなど)をコーディネーターとした２フェーズ・コミットへの参加不可
  - メッセージ取り出し/書き込みのためのメソッドを提供
    - readString、readLine、writeString、writeIntなど
  - MQとのローカル接続/クライアント接続が可能



主要API

MQ Base Javaが提供する主なクラスとメソッド

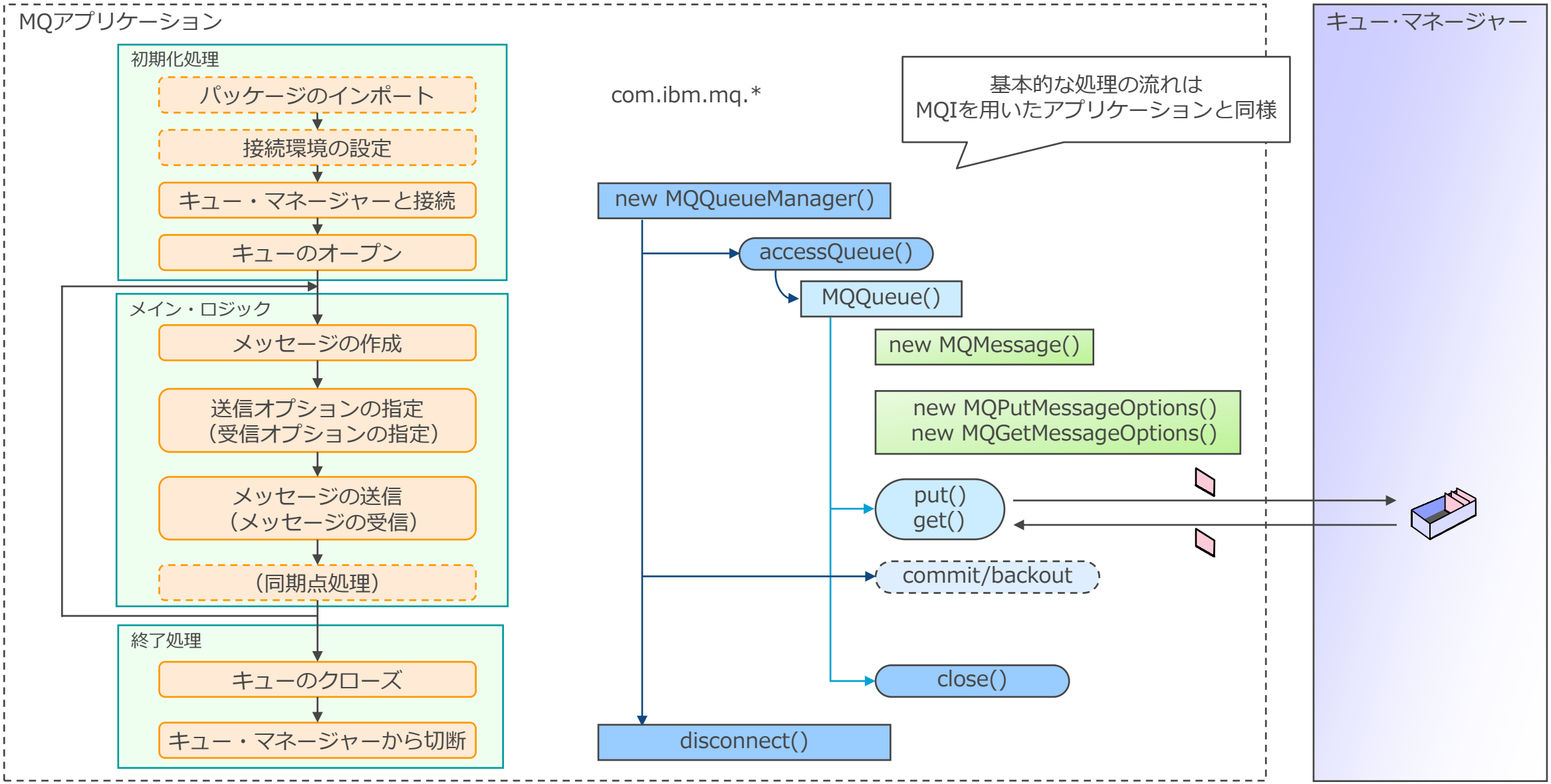
クラス	用途とメソッド
MQEnvironment	CCSIDやクライアント接続情報などのMQ環境を設定

MQQueueManager	- キュー･マネージャーへの接続
	- 主なメソッド
	  - accessQueue()
	  - disconnect()
	  - begin()
	  - commit()
	  - backout()

MQQueue	- キューへの接続
	- 主なメソッド
	  - put()、get()
	  - close()

MQMessage	- メッセージ - MQMDに相当するフィールドを持つ - 主なメソッド - readString()、writeStrig()
MQPutMessageOptions	MQPUT時のオプション(MQPMO)
MQGetMessageOptions	MQGET時のオプション(MQGMO)
MQConstants	各種オプションを表す定数
MQExceptoin	MQ関連処理中の例外処理

MQ Base Javaのプログラミング手順



サンプル・コード(MQ Base Java)

- キュー・マネージャー: TESTQMに接続し、キュー: QLにテストメッセージをPUT/GET

```
import com.ibm.mq.*;                                                    ·····1
import com.ibm.mq.constants.MQConstants;

public class Sample {
  public static void main(String args[]) {
    try {
      MQQueueManager qMgr = new MQQueueManager("TESTQM");               ·····2

      int openOptions = MQConstants.MQOO_INPUT_AS_Q_DEF | MQConstants.MQOO_OUTPUT;
      MQQueue queue = qMgr.accessQueue("QL", openOptions);              ·····3

      MQMessage putMessage = new MQMessage();
      putMessage.priority = 5;                                          ·····4
      putMessage.characterSet = 943;
      putMessage.writeString("This is a message");

      MQPutMessageOptions pmo = new MQPutMessageOptions();              ·····5
      pmo.options = MQC.MQPMO_NO_SYNCPOINT;

      queue.put(putMessage, pmo);                                       ·····6
```

サンプル・コード(MQ Base Java)

- キュー・マネージャー: TESTQMに接続し、キュー: QLにテストメッセージをPUT/GET(続き)

```
      MQMessage getMessage = new MQMessage();                          ·····7
      MQGetMessageOptions gmo = new MQGetMessageOptions();             ·····8

      queue.get(getMessage, gmo);                                      ·····9
      String strMessage = getMessage.readString(17);                   ·····10
      System.out.println(strMessage);

      queue.close();                                                   ·····11
      qMgr.disconnect();                                               ·····12
    }
    catch(MQException ex) {
      System.out.println("MQException occurred¥ncc:" +
      ex.completionCode +
      "¥nrc:" + ex.reasonCode);
    }
    catch(java.io.IOException ex){
      System.out.println("IOException occurred");
    }
  }
}
```

サンプル・コード(MQ Base Java)

1. パッケージ・ファイルのインポート
2. キュー・マネージャー・オブジェクトを作成し、キュー・マネージャー: TESTQMに接続
3. キュー・オブジェクトを取得することで、キュー: QL1のオープン
4. MQMD作成、priority=5、CCSID=943(S-JIS)に指定してメッセージ・オブジェクトを作成
5. MQPMO作成、同期点処理無しに設定
6. メッセージPUT
7. MQMDをデフォルトの設定でメッセージ・オブジェクトを作成
8. MQGMOをデフォルトの設定で作成
9. 同一のキューからメッセージGET
10. メッセージ・オブジェクトからメッセージを取得
11. キューをクローズ
12. キュー・マネージャーから切断

主要なサンプル･プログラム(MQ Base Java)

- MQ提供のサンプル・プログラム
  - メッセージの書き込み + 読み込み
    - ソース: MQSample.java
    - コンパイル済みファイル: MQSample.class
  - 配置ディレクトリー
    - UNIX系: <導入ディレクトリ>/samp/wmqjava/samples
    - Windows: <導入ディレクトリ>¥tools¥wmqjava¥samples

messaging REST API

messaging REST API

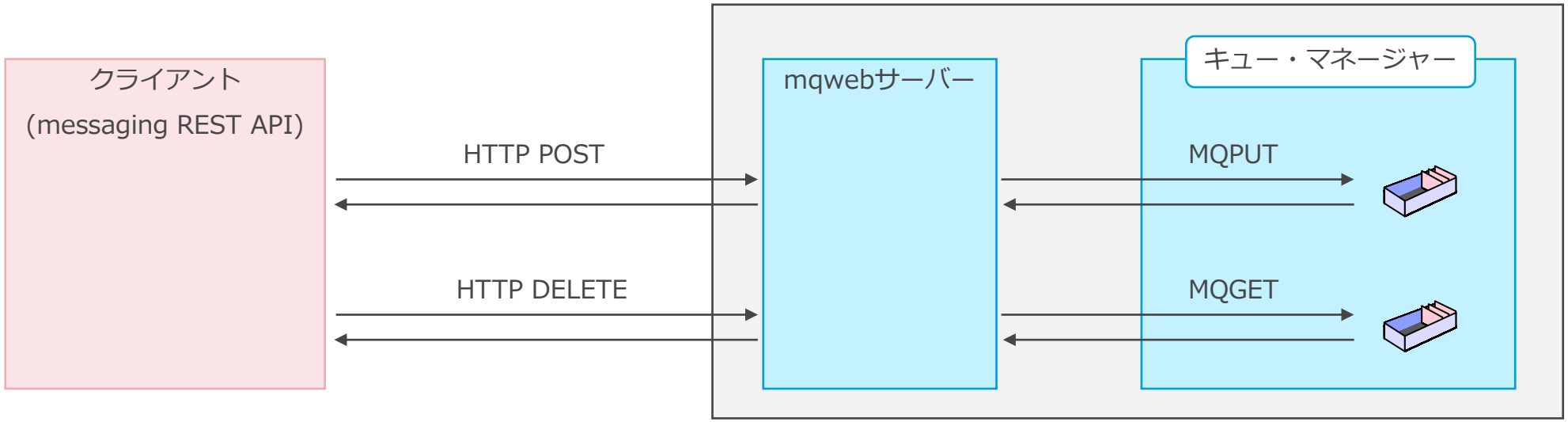
- メッセージを送信、受信、および、ブラウズするためのAPI
  - ◆ Point-to-Pointメッセージング、および、パブリッシュ・メッセージングを実行可能

リソースURL	メソッド	説明
/messaging/qmgr/{qmgrName}/queue/{queueName}/message	POST	キューにメッセージを送信
	GET	キューのメッセージをブラウズ
	DELETE	キューからメッセージを受信
/messaging/qmgr/{qmgrName}/queue/{queueName}/messagelist	GET	キューから使用可能なメッセージのリストを取得
/messaging/qmgr/{qmgrName}/topic/{topicString}/message	POST	トピックにメッセージをパブリッシュ

- 一部のMQMD属性(メッセージの有効期限/永続性/優先順位など)やGET Waitの指定が可能
- メッセージは、プレーン・テキスト形式で送受信
- トピックに対するサブスクライブは未サポート

messaging REST API

- messaging REST APIからのリクエストはmqwebサーバーが処理
  - ◆ messaging REST APIからのリクエストを受信し、キュー・マネージャーに対して処理を実施
  - ◆ mqwebサーバーはインストレーション毎に構成、z/OSではバージョン毎に構成
    - mqwebサーバーは同一インストレーション内のキュー・マネージャーに接続可能、z/OSは同一バージョンのみ可能



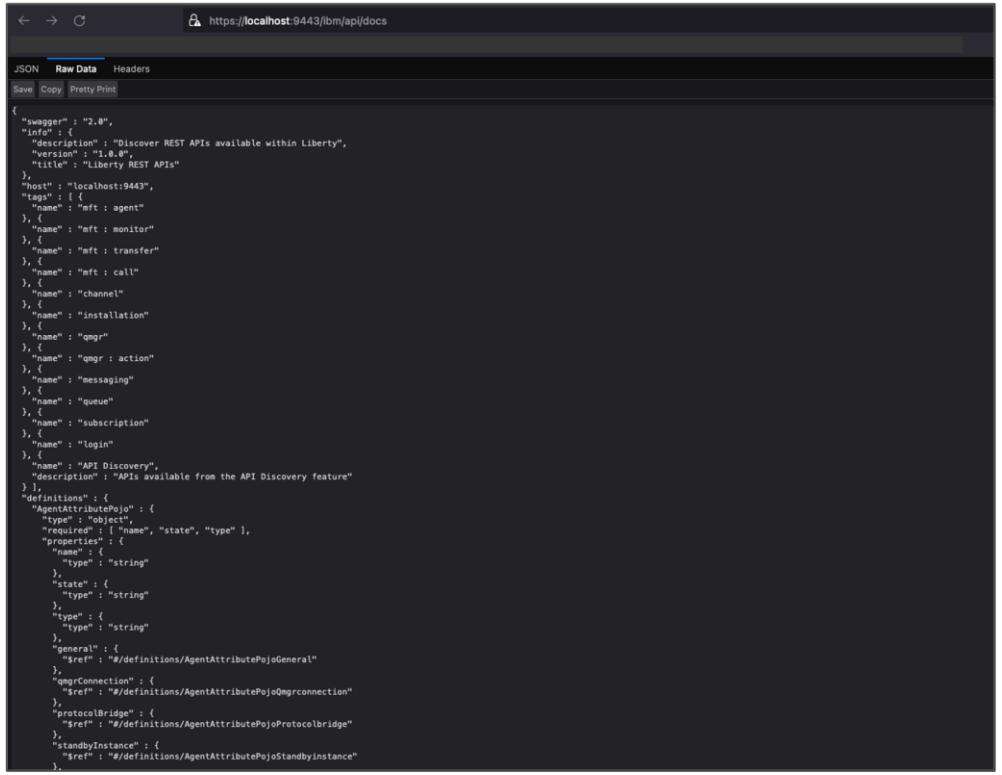
    - v9.3.3より、messaging REST APIはmqwebサーバーを介して任意のキュー・マネージャーに接続可能
      - 同一インストレーション内のキュー・マネージャーだけでなく、異なるインストレーション内のキュー・マネージャーに接続可能

messaging REST APIの仕様

- Swagger文書による仕様の確認
  - ◆ mqwebサーバーでAPIディスカバリー機能を有効にすることで表示可能
    - mqwebサーバーの構成ファイルである`mqwebuser.xml`の<featureManager>タグに以下の一行を追加
      - <feature>apiDiscovery-1.0</feature>

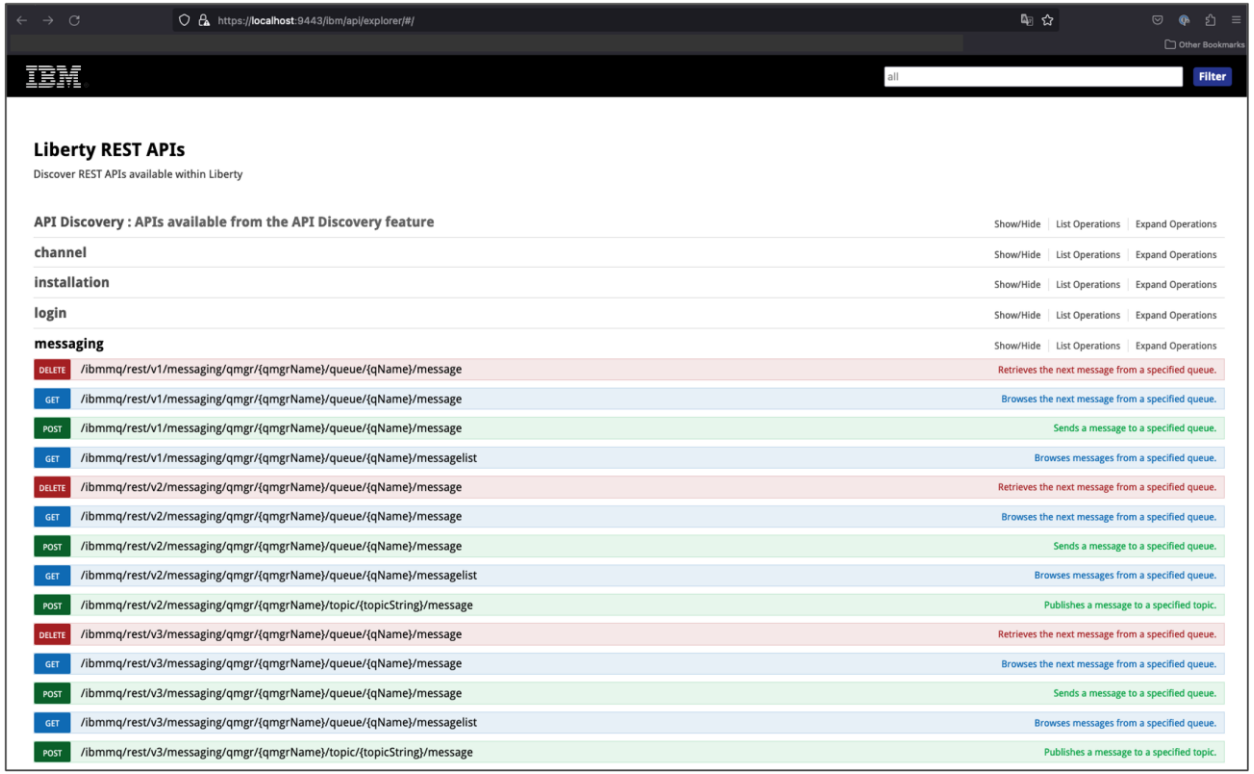
Swagger文書の仕様書を取得可能

https://<host>:<port>/ibm/api/docs



Webブラウザから Swagger文書を参照・APIを実行可能

https://<host>:<port>/ibm/api/explorer



キューへのメッセージ送信処理の実行例

- メッセージ送信処理
  - リソースURL: /messaging/qmgr/{qmgrName}/queue/{queueName}/message
  - メソッド: POST
  - リクエスト:
    - HTTPリクエスト・ヘッダーで一部のMQMD属性を指定可能
    - テキストベースのデータのみ送信可能
      - 文字コードはUTF-8
      - 送信されるメッセージのフォーマットは、MQSTR、もしくは、JMS TextMessage
  - レスポンス:
    - メッセージが正常に送信された場合はBodyはBlank、エラーの場合はBodyにエラー・メッセージが返る
    - 送信したメッセージのメッセージIDはHTTPのレスポンス・ヘッダーで確認可能

実行例) キュー・マネージャー: QM1のキュー: DEV.QUEUE.1にメッセージを送信

```
curl -i -k https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/QM1/queue/DEV.QUEUE.1/message -X POST -u mqm:mqm
-H "ibm-mq-rest-csrf-token: blank" -H "Content-Type: text/plain;charset=utf-8" -H "ibm-mq-md-persistence: persistent"
-H "ibm-mq-md-expiry: 1800000" -H "ibm-mq-md-priority: 5" -d "Hello World!"

HTTP/2 201
content-type: text/plain; charset=utf-8
ibm-mq-md-messageid: ID:414d5120514d312020202020202020200f10016501390040
ibm-mq-resolved-qmgr: QM1
content-length: 0
date: Wed, 13 Sep 2023 02:43:44 GMT
content-language: en-US
```

キューのメッセージのブラウズ処理の実行例

- メッセージのブラウズ処理
  - リソースURL: /messaging/qmgr/{qmgrName}/queue/{queueName}/message
  - メソッド: GET
  - リクエスト:
    - メッセージIDや相関IDを指定して、特定のメッセージをブラウズ可能
    - GET Waitは指定不可
  - レスポンス:
    - メッセージはレスポンスのBodyにセット
    - メッセージのメッセージIDや有効期限、永続性、優先順位などをHTTPのレスポンス・ヘッダーで確認可能

実行例) キュー・マネージャー: QM1のキュー: DEV.QUEUE.1のメッセージをブラウズ

```
curl -i -k
https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/QM1/queue/DEV.QUEUE.1/message?messageId=ID:414d5120514d312020202020202020200f10016501390040
-X GET -u mqm:mqm -H "ibm-mq-rest-csrf-token: blank"

HTTP/2 200
content-type: text/plain;charset=utf-8
ibm-mq-md-messageid: ID:414d5120514d312020202020202020200f10016501390040
ibm-mq-md-expiry: 1247100
ibm-mq-md-priority: 5
ibm-mq-md-persistence: persistent
ibm-mq-resolved-qmgr: QM1
date: Wed, 13 Sep 2023 02:52:57 GMT
content-language: en-US
content-length: 18

Hello World!
```

キューへのメッセージ受信処理の実行例

- メッセージの受信処理
  - リソースURL: /messaging/qmgr/{qmgrName}/queue/{queueName}/message
  - メソッド: DELETE
  - リクエスト:
    - メッセージIDや相関IDを指定して、特定のメッセージを受信可能
    - GET Waitを指定可能
  - レスポンス:
    - メッセージはレスポンスのBodyにセット
    - 受信したメッセージのメッセージIDや有効期限、永続性、優先順位などをHTTPのレスポンス・ヘッダーで確認可能

実行例) キュー・マネージャー: QM1のキュー: DEV.QUEUE.1からメッセージを受信

```
$ curl -i -k https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/QM1/queue/DEV.QUEUE.1/message?wait=1800000 -X DELETE -u mqm:mqm
-H "ibm-mq-rest-csrf-token: blank"

HTTP/2 200
content-type: text/plain;charset=utf-8
ibm-mq-md-messageid: ID:414d5120514d312020202020202020200f10016501480040
ibm-mq-md-expiry: 1800000
ibm-mq-md-priority: 5
ibm-mq-md-persistence: persistent
ibm-mq-resolved-qmgr: QM1
date: Wed, 13 Sep 2023 03:06:27 GMT
content-language: en-US
content-length: 18

Hello World!
```

メッセージのリストの取得処理の実行例

- メッセージのリストの取得処理
  - ◆ リソースURL: /messaging/qmgr/{qmgrName}/queue/{queueName}/messagelist
  - ◆ メソッド: GET
  - ◆ リクエスト:
    - メッセージIDや相関IDを指定して、特定のメッセージをリスト可能
    - リストするメッセージ数を制限可能
  - ◆ レスポンス:
    - messagesという単一のJSON配列を含むJSONオブジェクトがBodyにセット
      - メッセージ毎のフォーマット、メッセージID、相関IDなどが含まれる

実行例) キュー・マネージャー: QM1のキュー: DEV.QUEUE.1のメッセージのリストを取得

```
$ curl -i -k https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/QM1/queue/DEV.QUEUE.1/messagelist?limit=3 -X GET -u mqm:mqm
-H "ibm-mq-rest-csrf-token: blank"

HTTP/2 200
content-type: application/json;charset=utf-8
ibm-mq-resolved-qmgr: QM1
date: Wed, 13 Sep 2023 03:28:25 GMT
content-language: en-US
content-length: 269

{"messages":[{"format":"MQSTR","messageId":"ID:414d5120514d312020202020202020200f100165014e0040"},{"format":"MQSTR","messageId":"ID:414d
5120514d312020202020202020200f100165014b0040"},{"format":"MQSTR","messageId":"ID:414d5120514d312020202020202020200f100165024b0040"}]}
```

トピックへのメッセージのパブリッシュ処理の実行例

- トピックへのメッセージのパブリッシュ処理
  - リソースURL: /messaging/qmgr/{qmgrName}/topic/{topicString}/message
  - メソッド: POST
  - リクエスト:
    - HTTPリクエスト・ヘッダーで一部のMQMD属性を指定可能
    - テキストベースのデータのみ送信可能
      - 文字コードはUTF-8
      - パブリッシュされるメッセージのフォーマットは、MQSTR、もしくは、JMS TextMessage
  - レスポンス:
    - メッセージが正常にパブリッシュされた場合はBodyはBlank、エラーの場合はBodyにエラー・メッセージが返る

実行例) キュー・マネージャー: QM1のトピック: DEV.BASE.TOPICにメッセージをパブリッシュ

```
$ curl -i -k https://localhost:9443/ibmmq/rest/v3/messaging/qmgr/QM1/topic/DEV.BASE.TOPIC/message -X POST -u mqm:mqm
-H "ibm-mq-rest-csrf-token: blank" -H "Content-Type: text/plain;charset=utf-8" -H "ibm-mq-md-persistence: persistent" -d "Hello World!"

HTTP/2 201
content-type: text/plain; charset=utf-8
ibm-mq-resolved-qmgr: QM1
content-length: 0
date: Wed, 30 Aug 2023 13:28:50 GMT
content-language: en-US
```

<参考> messaging REST APIで指定可能なMQMD/パラメーター一覧

HTTPリクエスト・ヘッダーとして指定可能なMQMD

ヘッダー名	用途	デフォルト値	指定範囲値	指定可能なAPI
ibm-mq-md-correlationId	メッセージの相関IDを指定	-	-	①
ibm-mq-md-expiry	メッセージの有効期限を指定	unlimited	[0 - 99999999900]	①,⑤
ibm-mq-md-persistence	メッセージの永続性を指定	nonPersistent	[nonPersistent, persistent]	①,⑤
ibm-mq-md-priority	メッセージの優先順位を指定	asDestination	[0 – 9]	①,⑤
ibm-mq-md-replyTo	メッセージの応答先のキューとキュー・マネージャーを指定	-	-	①,⑤

リソースURLのオプションとして指定可能なパラメーター

パラメーター名	用途	デフォルト値	指定範囲値	指定可能なAPI
correlationId	特定のメッセージの相関IDを指定	-	-	②,③,④
messageId	特定のメッセージのメッセージIDを指定	-	-	②,③,④
wait	メッセージ到着までの待機時間[ms]を指定	-	[- 2147483647]	③
limit	リストするメッセージ数を指定	10	[- 2147483647]	④

① POST /messaging/qmgr/{qmgrName}/queue/{queueName}/message
② GET /messaging/qmgr/{qmgrName}/queue/{queueName}/message
③ DELETE /messaging/qmgr/{qmgrName}/queue/{queueName}/message
④ GET /messaging/qmgr/{qmgrName}/queue/{queueName}/messagelist
⑤ POST /messaging/qmgr/{qmgrName}/topic/{topicString}/message