「Javaアプリケーション脆弱性事例調査資料」について

- この資料は、Javaプログラマである皆様に、脆弱性を身近な問題として感じてもらい、セキュアコーディングの重要性を認識していただくことを目指して作成しています。

- 「Javaセキュアコーディングスタンダード CERT/Oracle版」と合わせて、セキュアコーディングに関する理解を深めるためにご利用ください。

JPCERTコーディネーションセンター
セキュアコーディングプロジェクト
secure-coding@jpcert.or.jp

Apache ActiveMQにおける認証処理不備の脆弱性

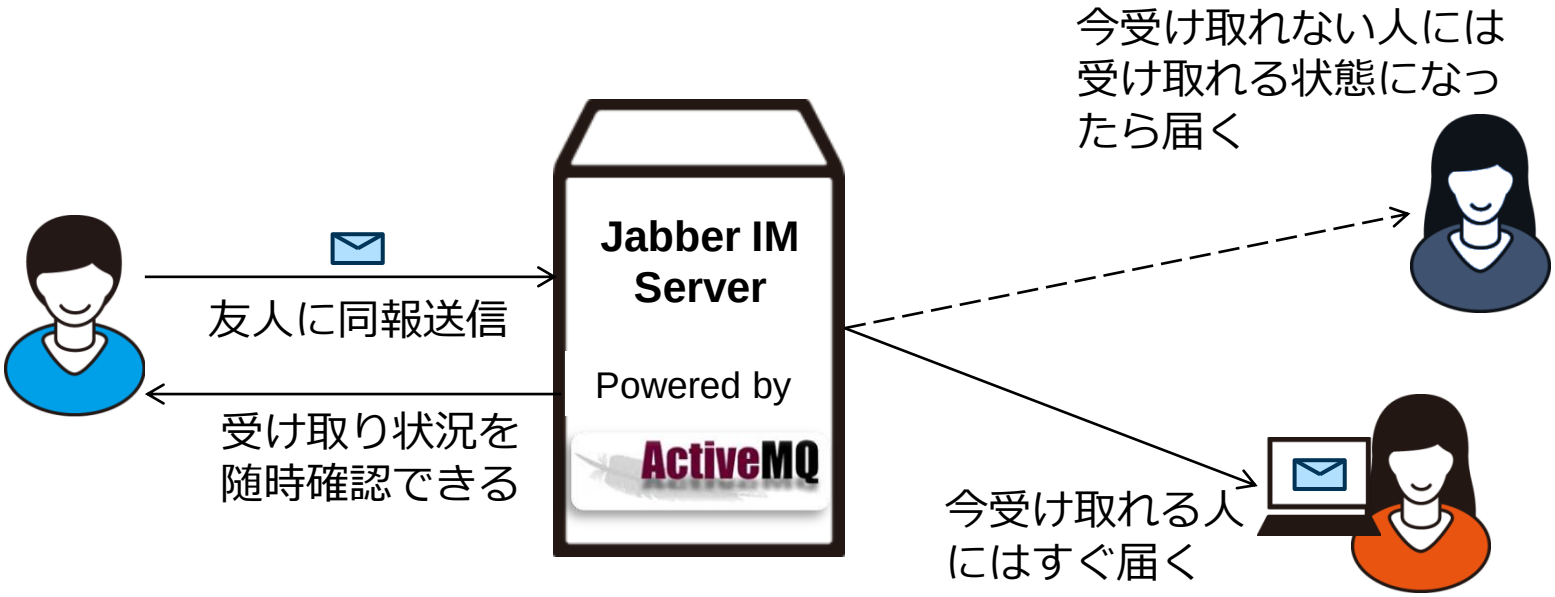
AMQ-1272

ActiveMQ とは

- サーバへの処理要求を非同期処理するためのミドルウェア
- APIとしてJMS(JavaMessageService) を実装しており、Javaとの親和性が高い
- OpenWireやStomp等9種類以上のプロトコルが使用可能

ActiveMQ の使用例: IM 中継サーバ

- 送信者は、受信者の状態に関わらずActiveMQにメッセージを預ければよい
- 受信者は好きなタイミングでメッセージを受け取ることが可能



脆弱性の概要

- ActiveMQサーバに接続する際、IDとパスワードによる認証を行うが、この認証処理が正しく行われず、誰もがログインできる状態だった。
- 存在するユーザになりすますことも、存在しないユーザで操作することも可能になってしまっていた。

※本ドキュメントは、脆弱性を内包する ActiveMQ 5.0 を対象に記載している。

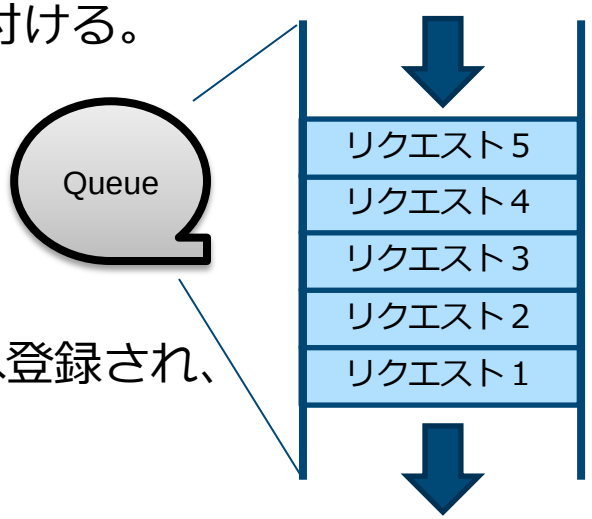
脆弱性が悪用された場合のリスク

- 権限のないユーザに操作を許可してしまう
  - 利用できないはずのユーザ(登録されていないユーザ)がキューへアクセス
  - 既に登録されているユーザに成りすましてキューへアクセス
  - 情報が漏えいしたり改竄される恐れがある。

キューを介したメッセージ処理

- ActiveMQはキュー構造を用いて、要求側が指示したい処理を蓄積する。
  —これを、「メッセージング」と呼ぶ。

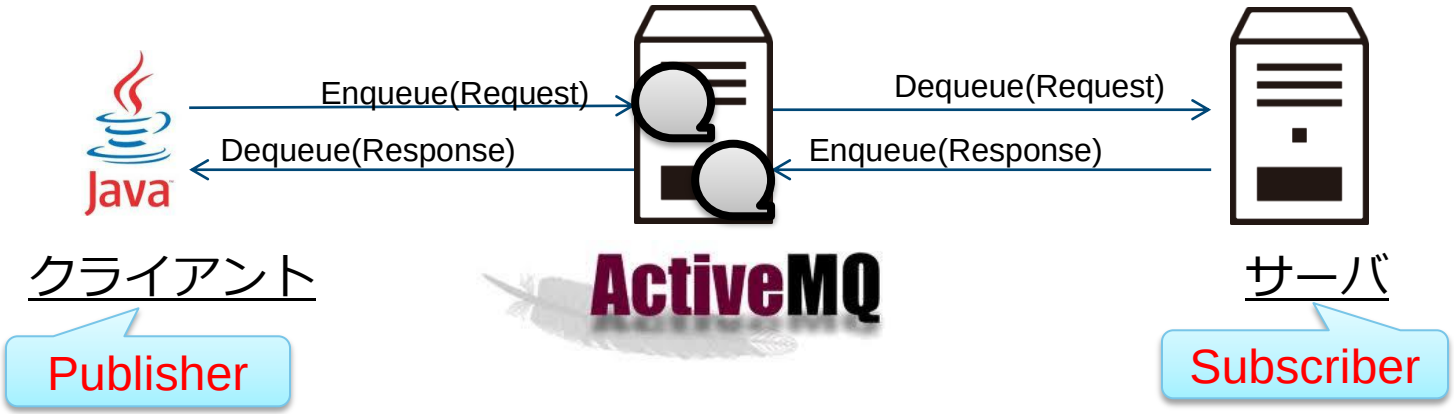
各種端末からの接続を受け付ける。



指示したい処理=リクエストは順番にキューへ登録され、
先に登録されたものから処理される。

Publisher と Subscriber

- クライアントはサーバへの要求(メッセージ)をActiveMQに登録し、サーバはその要求を受け取り処理する。また、逆の流れで結果を返却する。



ActiveMQ を用いた非同期のクライアント／サーバ構成例

Point!

ActiveMQを介して通信するアプリケーションを、その機能に応じて以下のように呼び分ける。

- Publisher: 処理を要求するアプリケーション(一般的なクライアント機能)
- Subscriber: 要求を処理し結果を返却するアプリケーション(一般的なサーバ機能)

ActiveMQ を介したメッセージング処理フロー

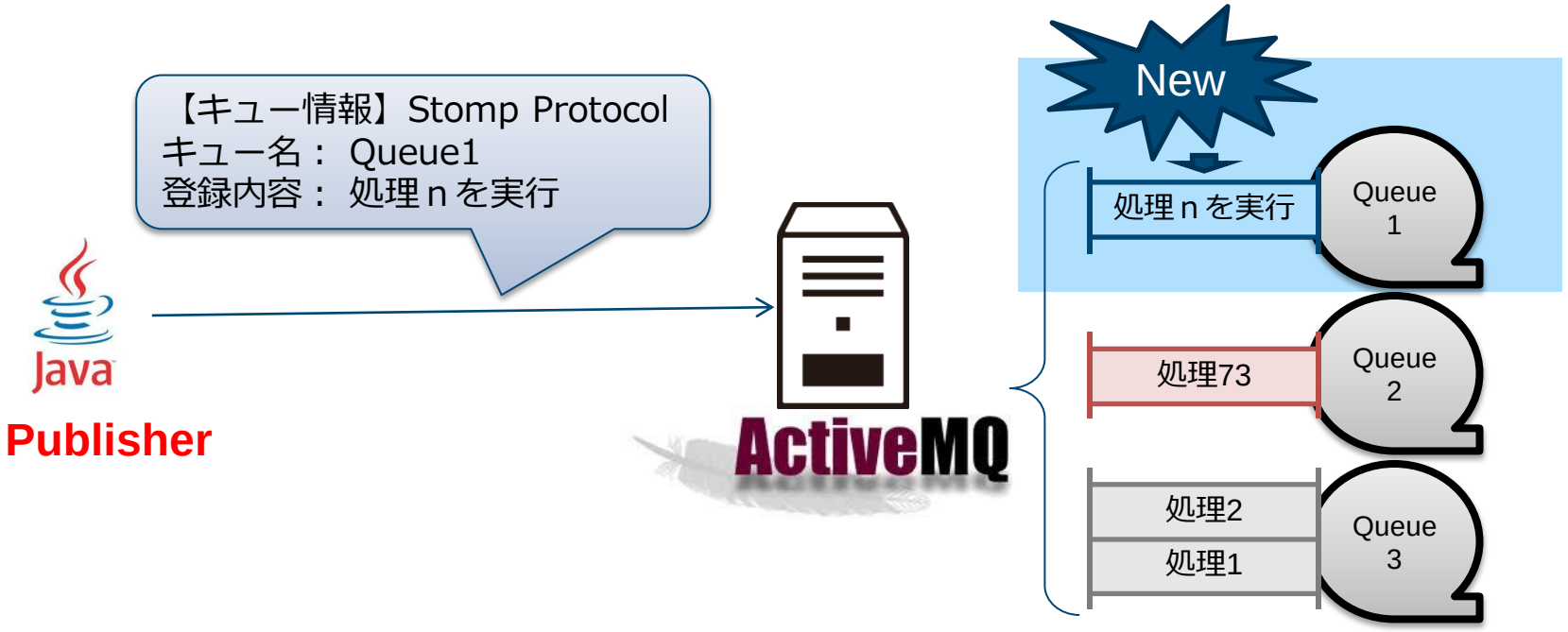
ActiveMQを介したPublisher/Subscriber間の処理フロー

① [Publisher] ActiveMQへリクエスト登録

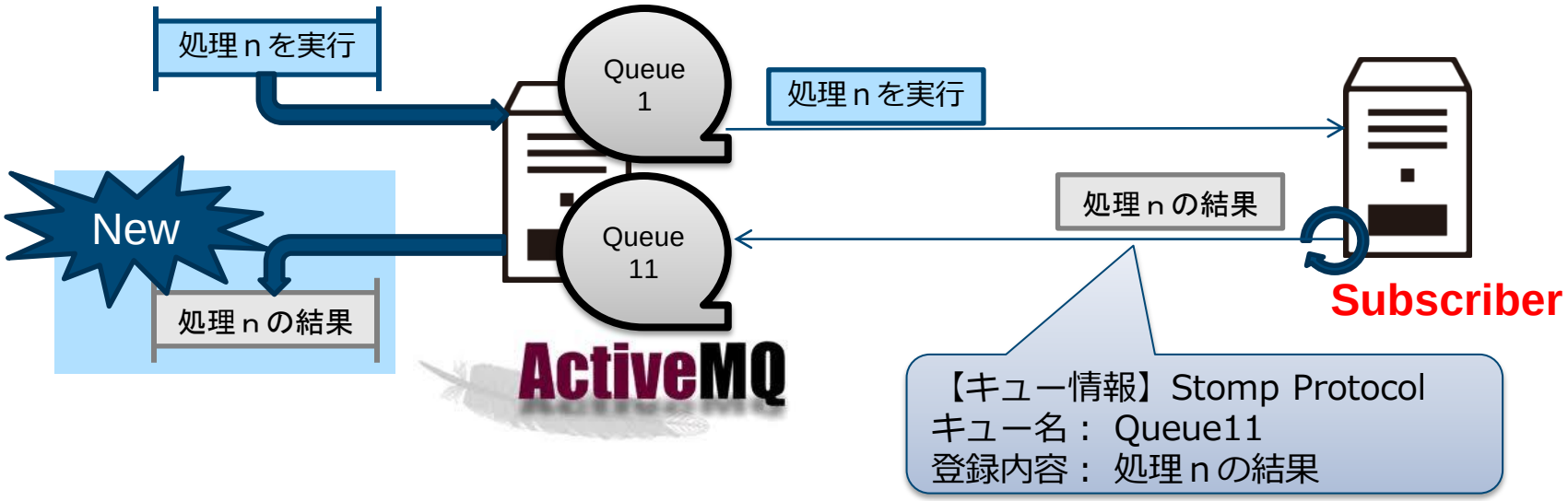
② [Subscriber] ActiveMQからリクエストメッセージを取得・処理、
ActiveMQへレスポンス登録

③ [Publisher] ActiveMQからレスポンスメッセージを取得・処理

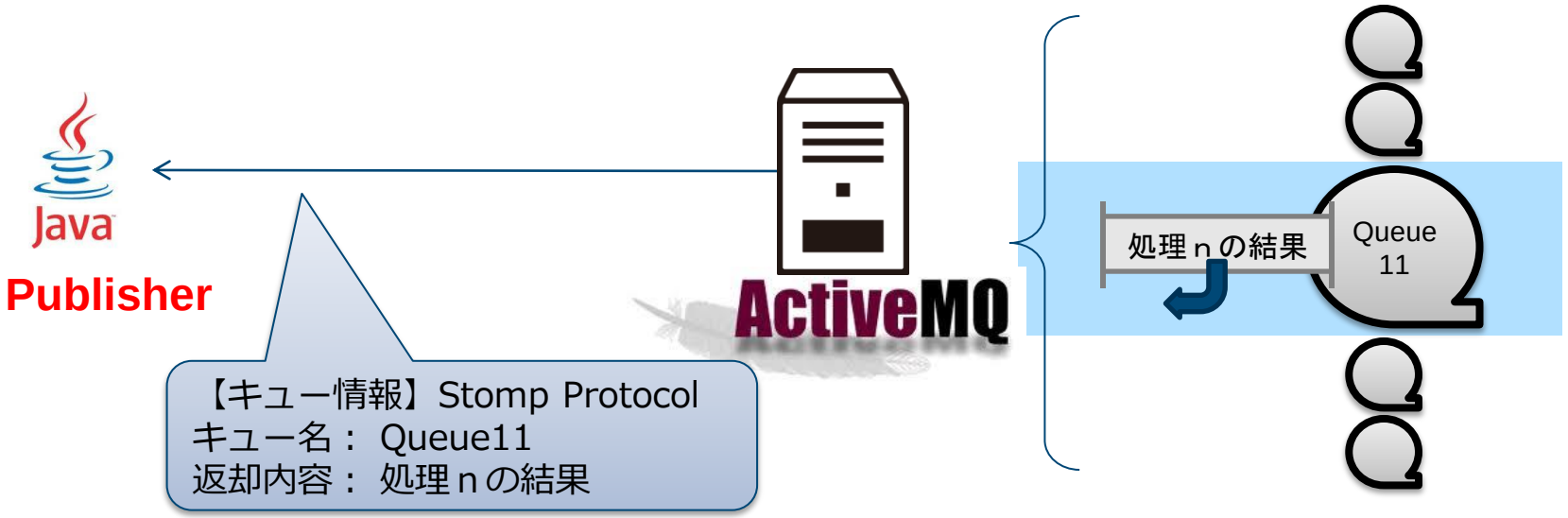
① [Publisher] ActiveMQへリクエストメッセージ登録



② [Subscriber] リクエストメッセージを取得・処理、レスポンス登録

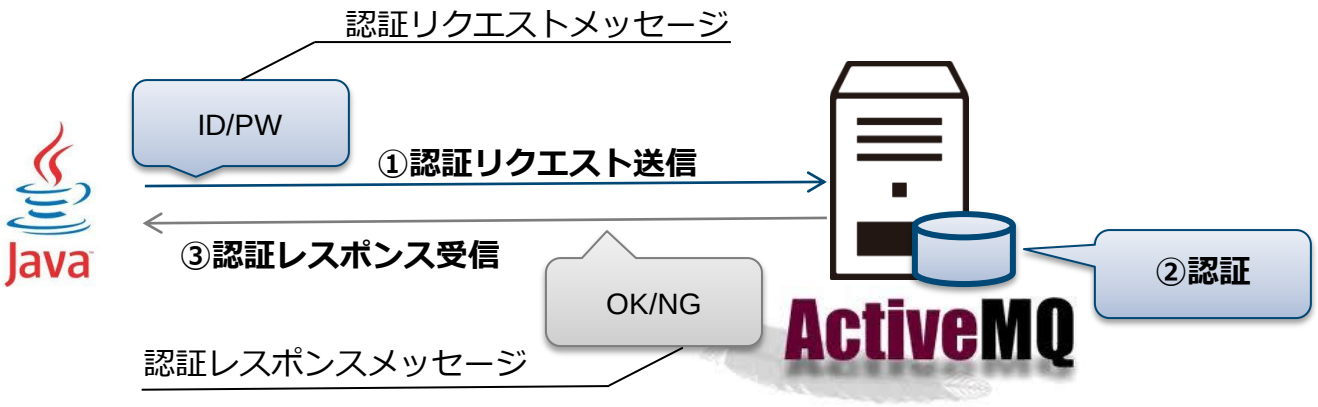


③ [Publisher] レスポンスメッセージを取得・処理



ActiveMQの認証(概要)

- SimpleAuthenticationPluginという認証モジュールで
  ID/PW認証が簡単に実装できる

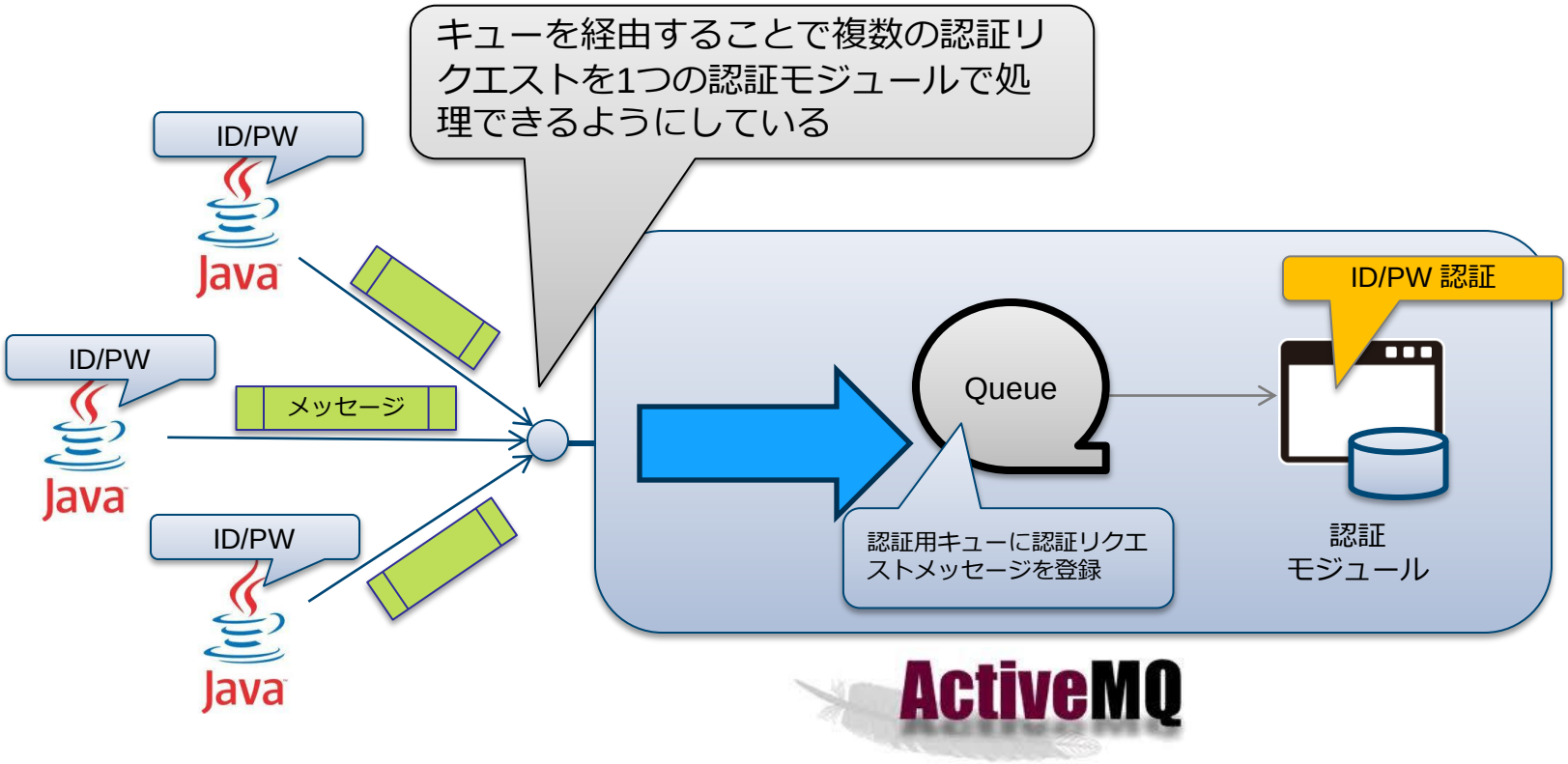


Publisher 側の認証フロー

※PublisherもSubscriberも同様に、接続時に認証が必要となる。
以降、Publisher側を例に認証フローを説明する。

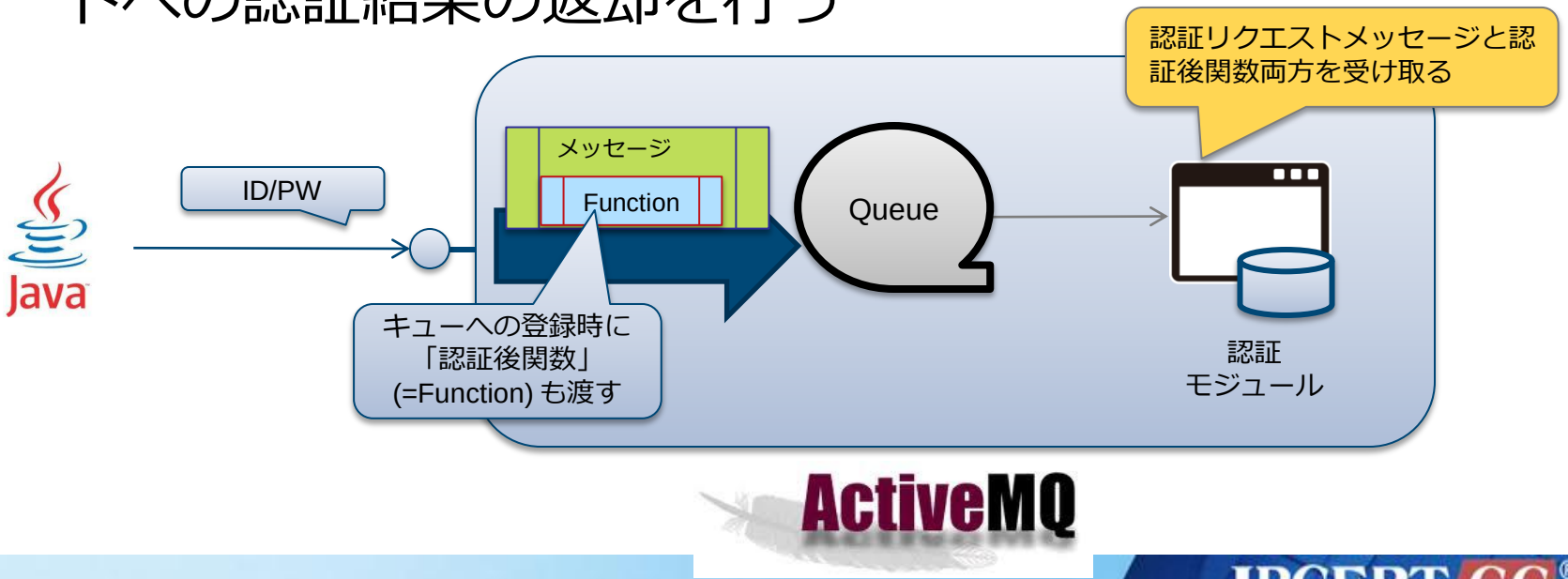
認証メッセージの処理

- ActiveMQが認証リクエストメッセージを受信すると、キューを経由して認証モジュールに認証リクエストメッセージを送信する仕組みになっている。



「認証後関数」

- ActiveMQが認証リクエストメッセージをキューに登録する際、認証後処理を定めた「認証後関数」も一緒に渡している。
  - 認証モジュールは認証処理後に認証後関数を呼び出し、認証結果を渡す
  - 認証後処理では、接続セッションの確立とクライアントへの認証結果の返却を行う



認証成功時の処理

① 認証モジュールにて認証を行う⇒OK

② 認証後関数を実行する

- ③ 認証後処理(接続を確立する等)を行う
- ④ 結果をクライアントに返却する

認証失敗時の処理

① 認証モジュールにて認証を行う⇒NG

② 認証後関数を実行する

- ③ NG時処理を行い認証後関数を異常終了する
- ④ 結果(NG)をクライアントに返却する

ソースコード解説

認証モジュールにて認証する際の処理フローに沿って解説する。

ユーザID/パスワード認証を実行する処理フロー

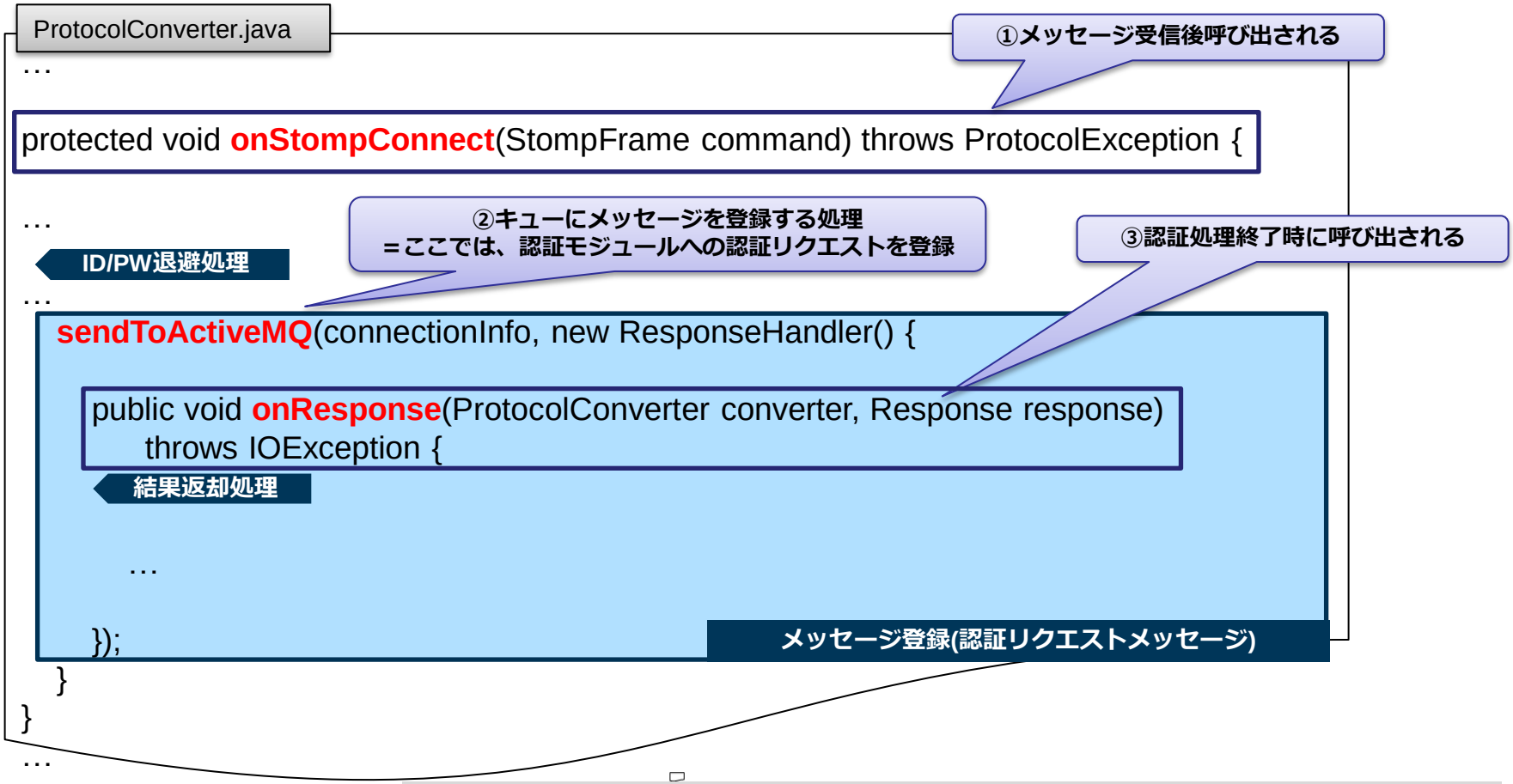
① クライアントから送信された認証要求をActiveMQが受け取り解析し、
ID/PW等を認証情報(connectionInfo)に格納する。

② 認証モジュールが持つキューに対し認証リクエストを登録する。
⇒メッセージは認証モジュールで処理される。

③ 認証後処理関数が呼び出され、結果をクライアントに返却する。

コードの全体構成

①～③の処理は、以下のコードで構成されている。



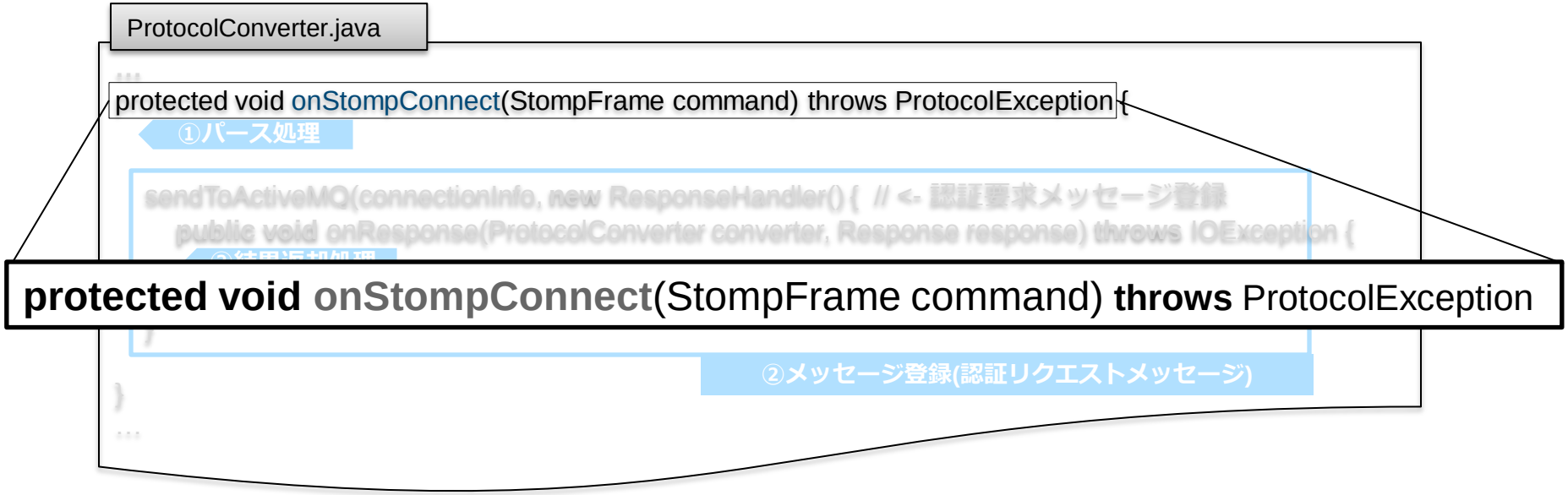
Point!

無名クラスを用いて認証後の処理を定義している。

(new ResponseHandler(){ … })

コードの全体構成(詳細)

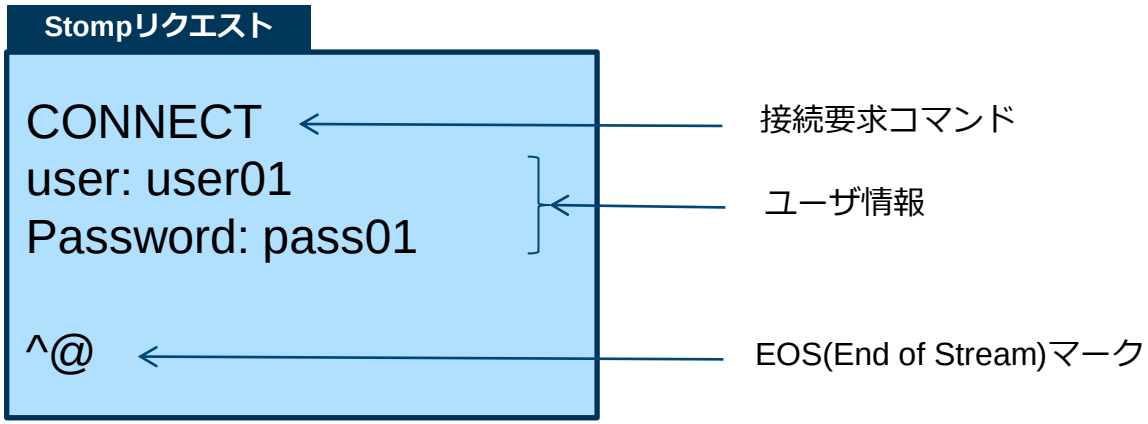
ActiveMQが認証要求を受けると onStompConnect() が呼び出される。



本処理内で、認証モジュールに対し認証要求を行っている。

①クライアントから送信された内容を解析し、認証情報に格納する

接続要求データを解析する。(例: Stompプロトコルでのリクエスト)



ProtocolConverter.java

```
protected void onStompConnect(StompFrame command) throws ProtocolException {
…
    String login = headers.get(Stomp.Headers.Connect.LOGIN);
    String passcode = headers.get(Stomp.Headers.Connect.PASSCODE);
    String clientId = headers.get(Stomp.Headers.Connect.CLIENT_ID);
```

解析処理

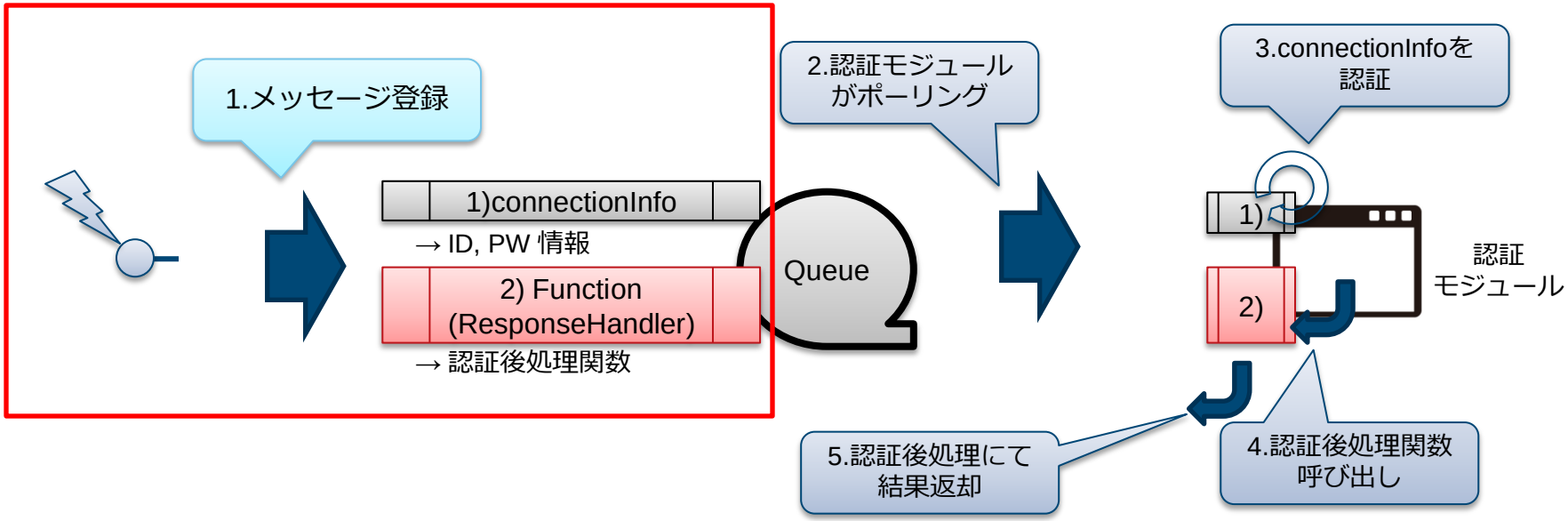
ユーザ名(user01)
パスワード(pass01)

```
    final ConnectionInfo connectionInfo = new ConnectionInfo();
    …
    connectionInfo.setUserName(login);
    connectionInfo.setPasseword(passcode);
…
```

connectionInfo に格納する。

② 認証モジュールが持つキューに対し認証を要求するメッセージを登録する

認証リクエストを受け取ったActiveMQは、認証モジュール向けのキューに認証リクエストを登録する。（1.）



② 認証モジュールが持つキューに対し認証を要求するメッセージを登録する

メッセージを登録するメソッド(sendToActiveMQ)を呼び出す。

1) connectionInfo
パースした情報を格納したクラス。
この情報を基に認証を行う。

② 認証モジュールが持つキューに対し認証を要求するメッセージを登録する

認証後の処理は onResponse() で定義されている。

ProtocolConverter.java

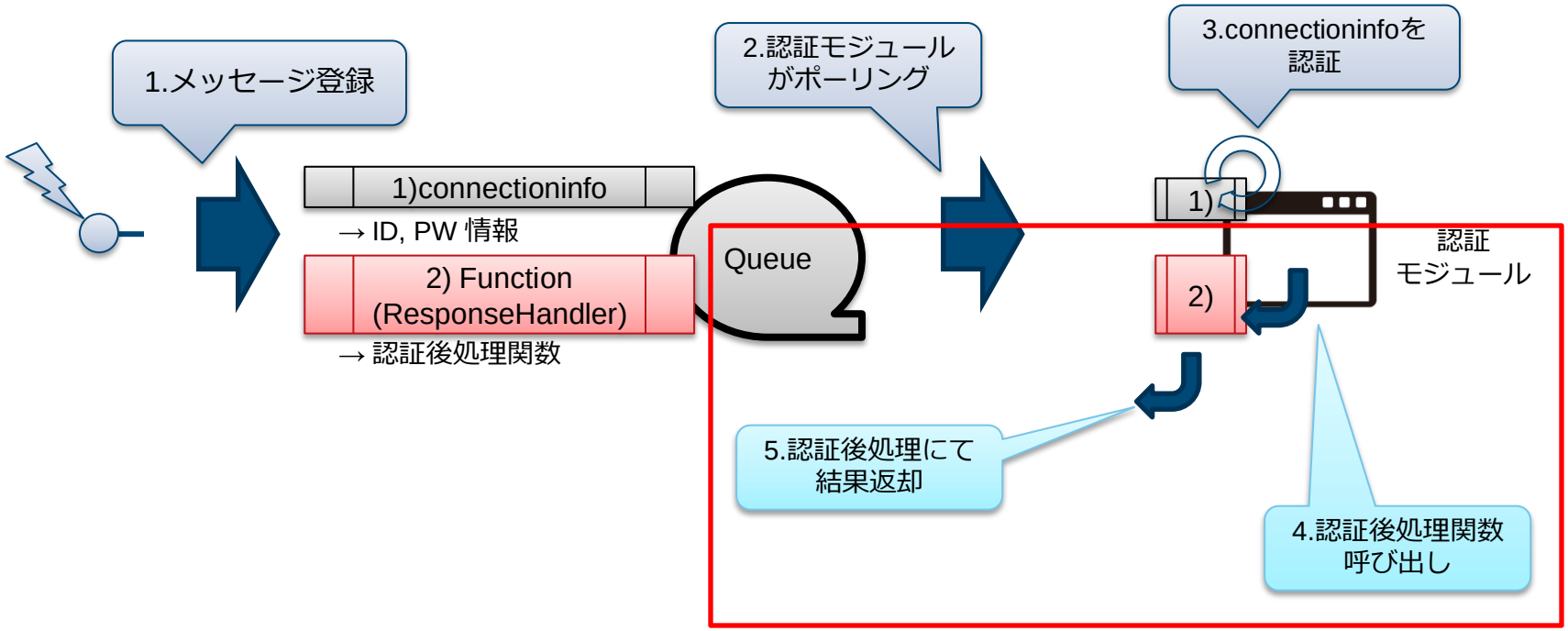
```
...
protected void onStompConnect(StompFrame command) throws ProtocolException {
    ①パース処理
    sendToActiveMQ(connectionInfo, new ResponseHandler() {  // <- 認証要求メッセージ登録
        public void onResponse(ProtocolConverter converter, Response response) throws IOException {
            ...
            ③結果返却処理
        }
    });
```

```
sendToActiveMQ(connectionInfo, new ResponseHandler() { … });
```

2) … ResponseHandler() { … }
認証処理後関数の本体(無名クラスによる定義)。
認証モジュールは処理完了後に onResponse() を呼び出す。

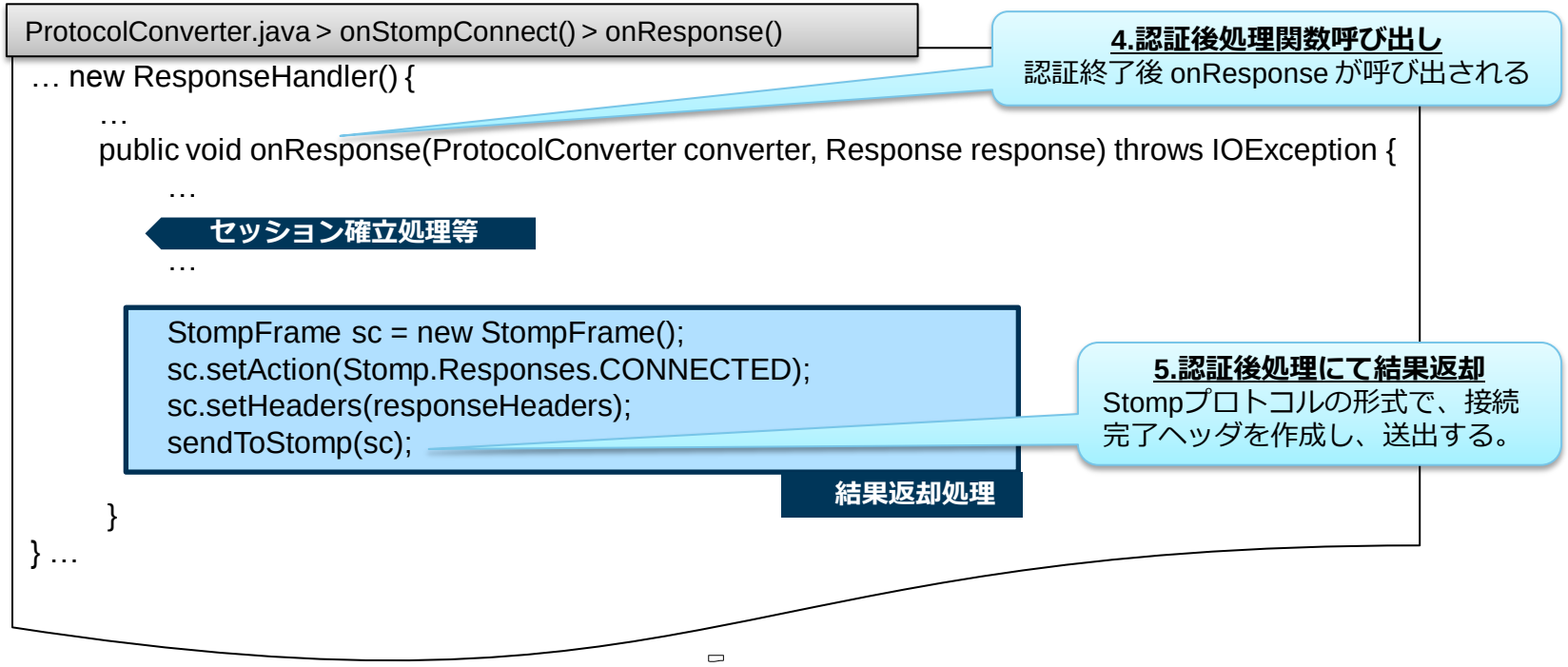
③セッション情報キューと送信オブジェクトの生成指示メッセージを登録する。

認証モジュールが認証を行い、認証後処理関数を呼び出して
結果を返却する。 (4～5)



③セッション情報キューと送信オブジェクトの生成指示メッセージを登録する。

認証後処理関数(ResponseHandler.onResponse)は、以下のコードで構成される。

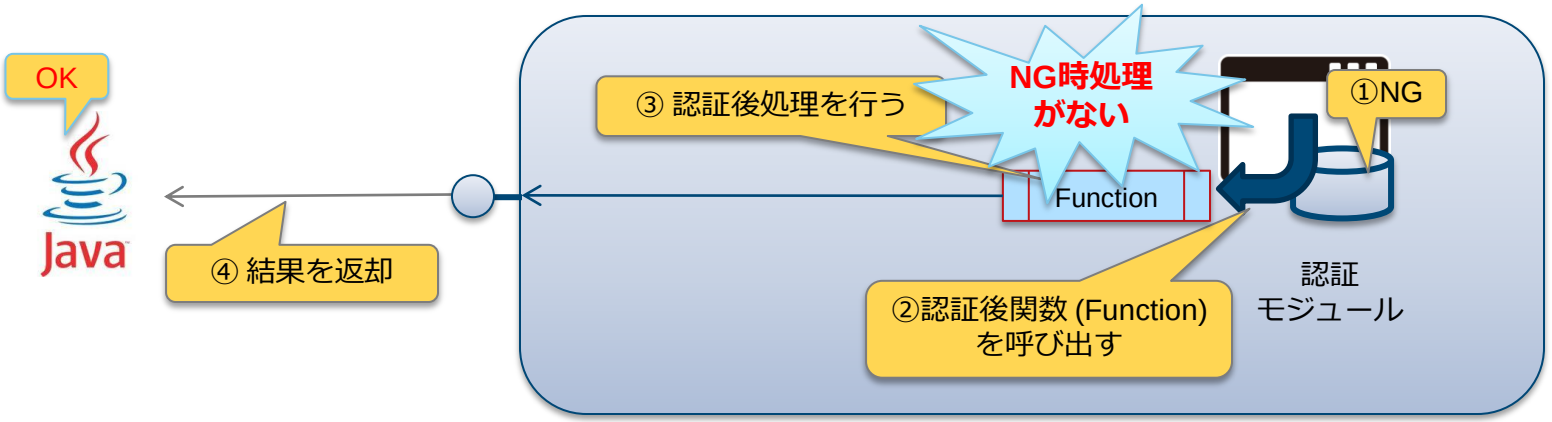


Point!

呼び出された認証後処理関数(onResponse)は、無条件にセッションを確立し、結果「成功」を返却する。

問題点

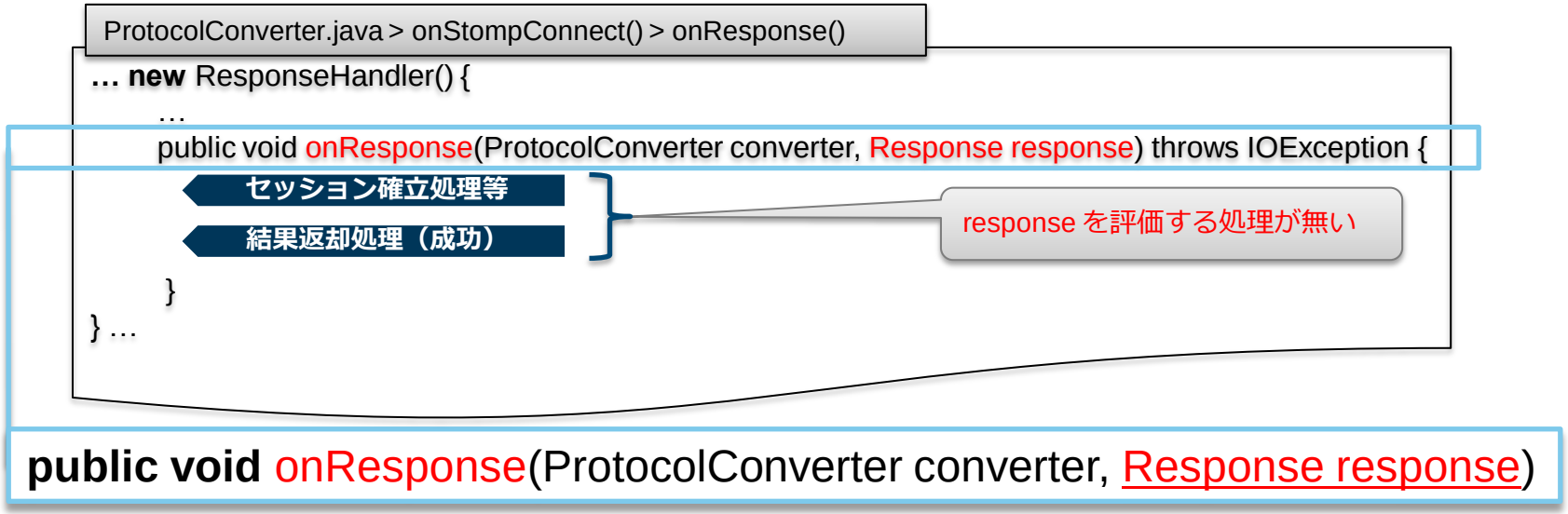
- 認証失敗時、認証後関数内で異常終了すべきところを、NG時処理が存在しなかった。
  - その結果、認証結果がNGであってもOK時処理を行っており、どんなID/PWでも接続できてしまった。



Point! NGを返却すべきところ、認証後関数にて認証OK時の処理を実施していたため、接続が不正に確立されてしまった。

問題点

認証後処理関数は引数を 2 つ持ち、第二引数の response には認証結果が設定されている。



Point!

エラー情報が格納されている response の値を全く評価していなかったため、接続の成功/失敗にかかわらず接続処理が完了してしまった。

問題点

- **今回のアプリケーションにおける具体的な問題点**

  認証後処理関数(onResponse)が認証結果をチェックしておらず、認証成功時の処理だけを行っていた。

- **問題点に対してどうすべきだったか。**

  認証結果 OK, NG それぞれに対する処理をきちんと実装すべきだった。

問題となったProtocolConverterクラスはSTOMPプロトコル用にActiveMQ 4.1で新設されたもの。それ以前のSTOMP用クラス群を統合し大幅にリニューアルされている。
このリニューアルの際に実装内容の検討と動作チェックが不十分だったものと考えられる。

修正版コード

認証モジュールの処理フロー③に認証失敗時の処理を追加した。

ユーザID/パスワード認証を実行する際の認証モジュールの処理フロー

① クライアントから送信された内容を解析しID/PW等を認証情報に格納する。

② 認証モジュールが持つキューに対し認証リクエストを登録する。

⇒メッセージは認証モジュールで処理される。

③ 認証終了後処理が呼び出され、結果をクライアントに返却する。

⇒[認証失敗時処理]

認証に失敗した場合は例外オブジェクトを設定し処理を中断する。

※ActiveMQ 5.1.0 Release で修正が行われている。

修正版コード

③認証終了後処理が呼び出され、結果をクライアントに返却する。

ProtocolConverter.java

```
…
protected void onStompConnect(StompFrame command) throws ProtocolException {
    ①パース処理
    sendToActiveMQ(connectionInfo, new ResponseHandler() {
        public void onResponse(ProtocolConverter converter, Response response) throws IOException {
            …
            if (response.isException()) {
                // If the connection attempt fails we close the socket.
                Throwable exception = ((ExceptionResponse)response).getException();
                handleException(exception, command);
                getTransportFilter().onException(IOExceptionSupport.create(exception));
                return;
            }
            セッション確立処理等
            結果返却処理（成功）
        }
    });
}
…
```

認証処理後に呼び出される

③response 引数評価処理

認証失敗時処理
例外オブジェクトを呼び出し元に通知する処理

②メッセージ登録(認証リクエストメッセージ)

認証モジュールで認証エラー(Security Exception)が発生したことを isException() で判別し、後続の処理に遷移しないように修正。

まとめ

■認証機能の実装

認証結果 OK, NG それぞれに対する処理を正しく実装すること

■実装のテスト

認証結果 OK, NG それぞれに対する実装が正しく行われているか動作テストを行う

著作権・引用や二次利用について

■本資料の著作権はJPCERT/CCに帰属します。

■本資料あるいはその一部を引用・転載・再配布する際は、引用元名、資料名および URL の明示をお願いします。

記載例

引用元：一般社団法人JPCERTコーディネーションセンター
Java アプリケーション脆弱性事例解説資料
Apache ActiveMQ における認証処理不備の脆弱性
https://www.jpcert.or.jp/securecoding/2012/No.06_Apache_ActiveMQ.pdf

■本資料を引用・転載・再配布をする際は、引用先文書、時期、内容等の情報を、JPCERT コーディネーションセンター広報(office@jpcert.or.jp)までメールにてお知らせください。なお、この連絡により取得した個人情報は、別途定めるJPCERT コーディネーションセンターの「プライバシーポリシー」に則って取り扱います。

本資料の利用方法等に関するお問い合わせ
JPCERTコーディネーションセンター
広報担当
E-mail：office@jpcert.or.jp

本資料の技術的な内容に関するお問い合わせ
JPCERTコーディネーションセンター
セキュアコーディング担当
E-mail：secure-coding@jpcert.or.jp