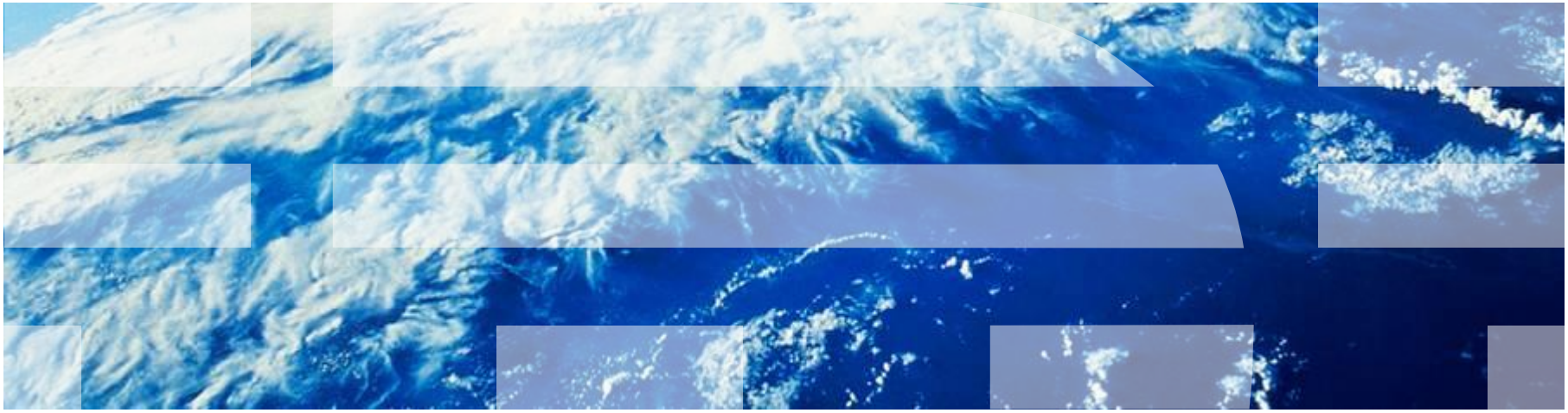
2015年3月
日本アイ･ビー･エム株式会社

Java EE 7 アプリケーション開発ガイド ー Servlet 3.1 編



前提事項

- 当ガイドは、Servlet 3.1 が提供する機能を利用して、Webアプリケーションを開発する際に必要となる情報を提供するものです。
- Servlet 3.1 が提供する全ての機能や API を網羅するものではありませんので、実際の開発に当たっては、巻末の参考資料をご参照ください。
- また、当ガイドでは各業務アプリケーションで必要となる業務要件についてはカバーしませんので、それぞれの案件における要件から実装を検討する必要があります。

目次

- Servlet の歴史
- Servlet 3.1 新機能

1. ノンブロッキング I/O
   - 新 I/O APIの追加でノンブロッキング I/O 処理が可能に

2. HTTP以外のプロトコルの利用
   - HTTPからWebSocketにプロトコルをアップグレード

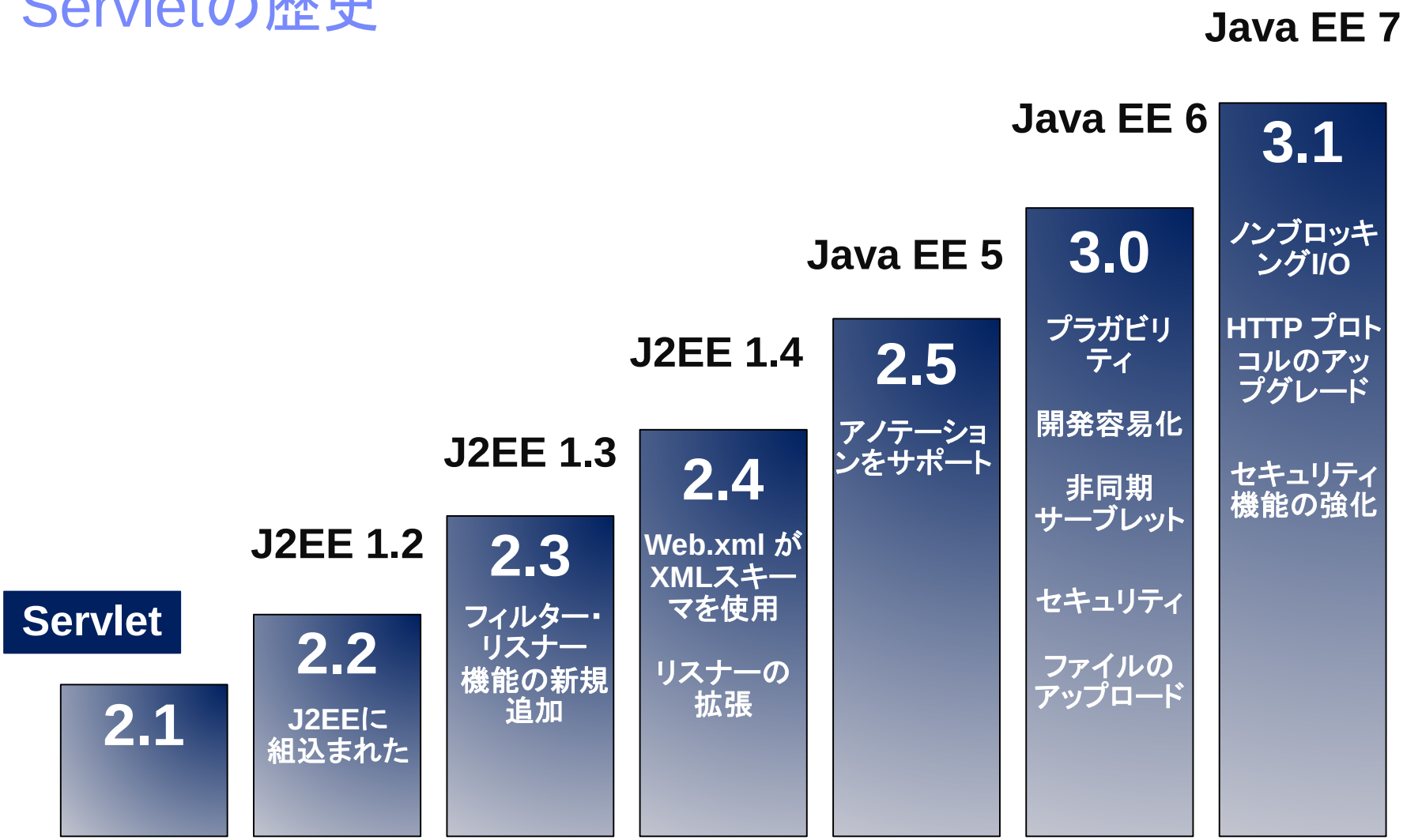
3. セキュリティ機能の強化
   - SessionIDの変更
   - HTTPメソッドに対するアクセス制限

- 参考資料

JSR 340 (Servlet 3.1)

参考： https://jcp.org/en/jsr/detail?id=340

Servletの歴史



※ (参考) http://en.wikipedia.org/wiki/Java_Servlet

Servlet 3.1 新機能

ノンブロッキング I/O - API追加の背景

- Java EE 6 Servlet 3.0で追加された非同期処理

  – 非同期リクエスト処理用のモデルを提供
    - サーブレットの実行スレッドを早期解放できる
      >JDBCやJMSなど、時間のかかる処理をリスナー・スレッドに委譲

  – 課題
    - Traditional I/Oしか使えないためアプリのスケーラビリティに制限がある

    例) サーブレットの実行スレッドがデータを待つために起こるブロッキング
    インバウンドデータがブロックされた時
    データがサーバーが読取可能なスピードよりも遅く流れた時

ノンブロッキング I/O - 概要

- Java EE 7 Servlet 3.1の新機能による対応

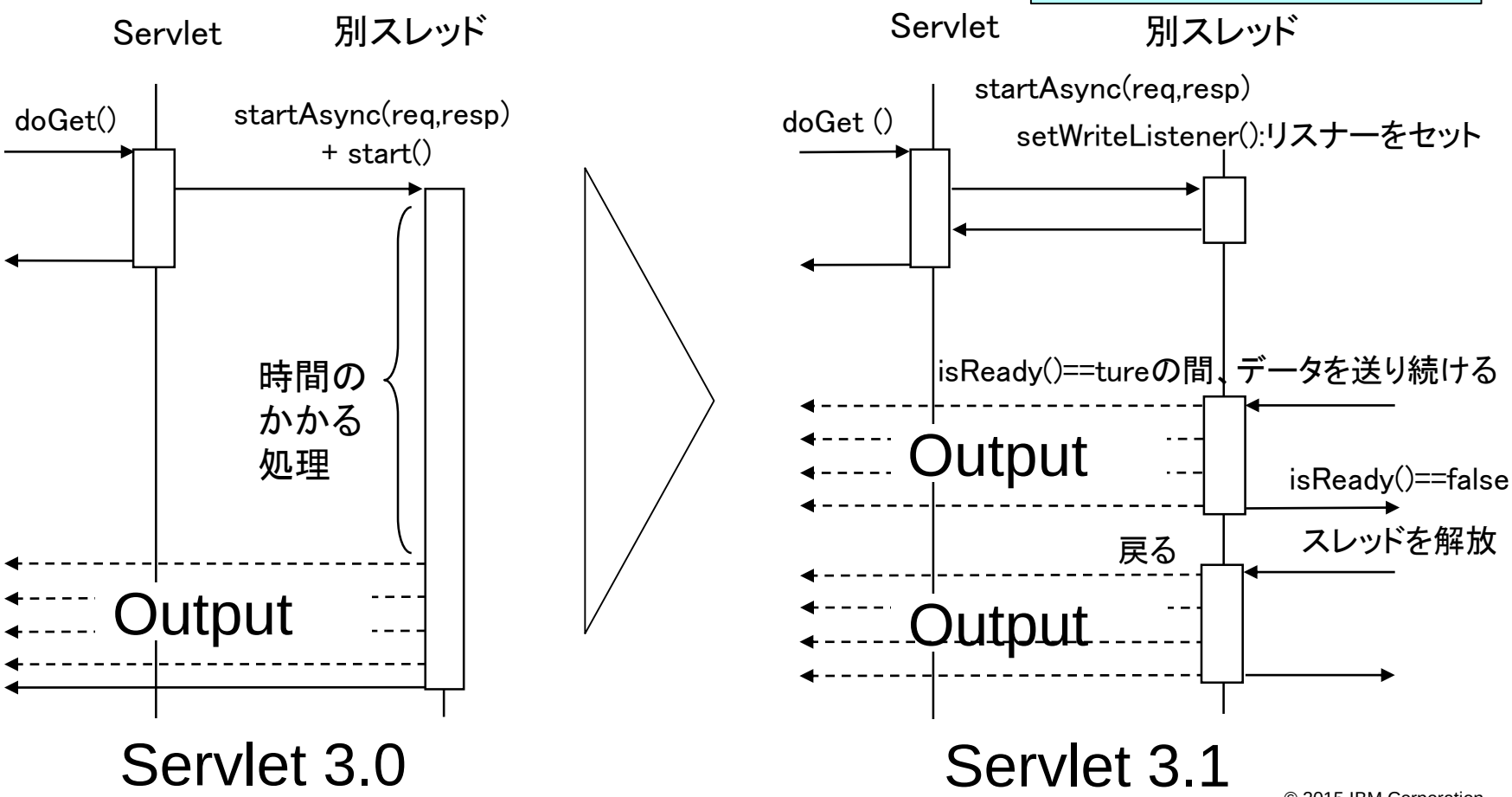
  – I/Oの処理を非同期に行えるAPIを追加
    - 2つのインタフェース を追加
      >ReadListener、WriteListener インタフェース
    - ServletInputStream、ServletOutputStream クラスにメソッドを追加

  – 実行スレッドをブロッキングすることなく、コンテンツが読取可能 or 書出可能になった時点で起動させることが可能

ノンブロッキング I/O - 処理の流れ

- サーブレットはリスナーにI/O機能を委譲
  - リスナーがServletInputStream, ServletOutputStreamにデータの読書きを行う
  - サーブレットの実行スレッドを早期解放 / インバウンド(アウトバウンド)のブロッキングを回避
  - Servlet 3.0の非同期サーブレットと比較して、I/Oに特化

WriteListenerを使用した例



ノンブロッキング I/O - 追加されたメソッド

■ クラス: ServletInputStream
- public boolean isFinished();
- public boolean isReady();
- public void setReadListener(ReadListener readListener);

■ クラス: ServletOutputStream
- public boolean isReady();
- public void setWriteListener(WriteListener writeListener);

ServletInputStream

- isFinished()
  true: 全てのデータ読込が完了した時
  false: それ以外
- isReady()
  true: ブロッキング無しで読込可能な時
  false: それ以外
- setReadListener()
  読込可能な時に引数のReadListenerを呼出す

ServletOutputStream

- isReady()
  true: ブロッキング無しで書込可能な時
  false: それ以外
- setWriteListener()
  書込可能な時に引数のWriteListenerを呼出す

http://docs.oracle.com/javaee/7/api/javax/servlet/ServletInputStream.html
http://docs.oracle.com/javaee/7/api/javax/servlet/ServletOutputStream.html

ノンブロッキング I/O - 追加されたインタフェース

■ インタフェース: ReadListener
- public void onAllDataRead();
- public void onDataAvailable();
- public void onError();

■ インタフェース: WriteListener
- public void onWritePossible();
- public void onError();

- onAllDataRead()
  現在のリクエストのデータが全て読み込まれた際に呼び出される
- onDataAvailable()
  データが読み込み可能な時に登録したServletInputStreamから呼び出される
- onError()
  リクエスト処理中にエラーが発生した時に呼び出される

- onWritePossible()
  データが書き込み可能な時に登録したServletOutoutStreamから呼び出される
- onError()
  リクエスト処理中にエラーが発生した時に呼び出される

http://docs.oracle.com/javaee/7/api/javax/servlet/ReadListener.html
http://docs.oracle.com/javaee/7/api/javax/servlet/WriteListener.html

ノンブロッキング I/O - コーディング例

- サーブレット: NIOServlet (指定された巨大なファイルを送信する)
  - 非同期処理の有効化 asyncSupported = true
  - サーブレットの実行スレッドで、リスナーをセット setWriteListener(listener)

```
@WebServlet(urlPatterns = "/NIOServlet", asyncSupported = true)
public class NIOServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static final String FILE_NAME = "任意のファイル";

    public NIOServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        File file = new File(FILE_NAME);
        response.setContentType("application/octet-stream"); //出力ファイルに合わせて型設定
        response.setContentLengthLong(file.length());

        AsyncContext ac = request.startAsync(); //非同期処理の開始
        ServletOutputStream out = response.getOutputStream();
        WriteListener listener = new AsyncFileWriter(ac, out, file);
        out.setWriteListener(listener); //書込み用WriteListenerをスレッドにセット
    }
}
```

ノンブロッキング I/O - コーディング例

- リスナー： SampleAsyncWriteListener

```
public class AsyncFileWriter implements WriteListener {
   private AsyncContext ac;
   private ServletOutputStream out;
   private FileInputStream file;

   public AsyncFileWriter(AsyncContext ac, ServletOutputStream out, File file) throws IOException {
      this.ac = ac;
      this.out = out;
      this.file = new FileInputStream(file);
   }

   @Override
   public void onError(Throwable t) {
      System.error.println("Exception in AsyncFileWriter“ + t);
      ac.complete();
   }

   @Override
   public void onWritePossible() throws IOException {
      byte[] buf = new byte[1024];
      while(out.isReady()) {
         int len = file.read(buf);
         if (len < 0) {
            ac.complete();
            file.close();
            return;
         }
         out.write(buf, 0, len);
      }
   }
}}
```

- onWritePossible()コールバックメソッドはWriteListenerで書込み可能時に呼ばれる
- オーバーライドし、書込み処理を記述

- isReady()==trueの時、つまりServletOutputStreamで書込み可能時にデータを読込み、out.writeで出力

- ac.complete()で非同期処理を完了し、AsyncContextの非同期処理を終了する

HTTP以外のプロトコルが利用可能に

- HTTP接続をWebSocketにアップグレードすることが可能
  - アップグレード方法
    - サーブレットから、upgrade()メソッドを呼び出す
  - メリット
    - 単一HTTPポートに複数のプロトコルを多重化できる
    - これによりファイアウォールなどの設定が楽になる
  - 備考
    - 現時点でブラウザで対応しているプロトコルは、HTTPとWebSocketであるため利用シーンは限られる
    - WebSocketを使う場合はAPIがあるため、ユーザーコードでアップグレードを使う機会は少ないと考えられる

セキュリティ機能の強化 - SessionIDの変更

- どのような強化がされたか
  - XSSやXSRFなどによるセッションハイジャックを防止するため、アプリケーションが任意にSessionIDを変更する機能が追加された

    - HttpServletRequest に changeSessionId() メソッドが追加
    - 効果:

      ログイン成功画面でセッションIDを変更することにより、ログイン前後のセキュリティを安全に分断することが可能

    - 適用方法:

SessionIDの変更

ログイン前のセッションID : ySnd8OQ7GMKw8qWX0mgAms5

```
String sessionIdBeforeLogin = request.getSession().getId();

request.login("user1", “user1pw");

String sessionIdAfterLogin = request.changeSessionId();
```

ログイン後のセッションID : BkWCoyKm8VFBaSCgIa1dCAd

セキュリティ機能の強化 - HTTPメソッドのアクセス制限

- どのような強化がされたか
  - HTTPメソッドに対するアクセス制限が可能になり、想定していたアクセス方法以外の接続を拒否することでXSS、XSRFなどの対応を強化

  - <deny-uncovered-http-methods> が追加
  - 効果:
    <http-method>で指定されていないメソッドは全て接続を拒否
  - 適用方法:

HTTPメソッドに
対するアクセス制限

POST以外の
アクセスは拒否

```
<web-app version="3.1">
    <deny-uncovered-http-methods/>
    <web-resource-collection>
      <url-pattern>/sample/*</url-pattern>
      <http-method>POST</http-method>
    </web-resource-collection>
</web-app>
```

(参考) XSSとは

- クロス・サイト・スクリプティングとは
  - Cross Site Scripting (XSS) 脆弱性
    - 定義:
      クライアントから送られてきたデータに含まれるスクリプトをサーバーが
      そのままコンテンツの一部としてクライアントに送り返してしまうこと

    - XSS脆弱性による問題:
      Origin(1つのサイト)を超えてスクリプトによる操作が行われてしまう
      例:
      ユーザー入力の読取り、Cookieの読取りや上書き、
      画面表示の変更、第三者への情報の転送、など
    - 解決策:
      scriptタグを無効化する
      < > & “ ‘ をエスケープさせる

(参考) XSRFとは

- クロス・サイト・リクエスト・フォージェリ とは
  - Cross Site Request Forgery (XSRF) 脆弱性
    - 定義:
      Webサイト閲覧者が意図しない別のWebサイト上の操作を行わせる攻撃
    - XSRF脆弱性による問題:
      > 被害例:
      あるWebサイトのボタンをクリックすると、勝手に全く無関係のSNSにメッセージを投稿されてしまう
    - 解決策:
      ページのhiddenパラメータに秘密情報を埋め込んでおくなどの方法で画面遷移をアプリケーションで管理し、意図しない外部からの不正リクエストを排除する

XSS, XSRF脆弱性への対応に関して

- Servlet 3.1 では、下記の機能によってセキュリティ機能の強化がなされました
  「SessionIDの変更」、「HTTPメソッドのアクセス制限」

- 上記の他、通常のXSS、XSRF脆弱性の対策は必要です

(参考) 安全なウェブサイトの作り方 # IPA発信
http://www.ipa.go.jp/security/vuln/websecurity.html

参考資料

- Java™ EE 7 Specification APIs
  - http://docs.oracle.com/javaee/7/api/
- JSR 340 : Servlet 3.1
  - https://jcp.org/en/jsr/detail?id=340
- JSR 315 : Servlet 3.0
  - https://jcp.org/en/jsr/detail?id=315