Libertyランタイム Tips集

2017/12

日本アイ・ビー・エム システムズ・エンジニアリング株式会社

Disclaimer

- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2017年12月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意下さい。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- IBM、IBMロゴおよびibm.comは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。現時点でのIBMの商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。
- 当資料をコピー等で複製することは、日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の承諾なしではできません。
- 当資料に記載された製品名または会社名はそれぞれの各社の商標または登録商標です。
- JavaおよびすべてのJava関連の商標およびロゴはOracleやその関連会社の米国およびその他の国における商標または登録商標です。
- Microsoft, WindowsおよびWindowsロゴは、Microsoft Corporationの米国およびその他の国における商標です。
- Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。
- UNIXはThe Open Groupの米国およびその他の国における登録商標です。

目次

構成管理

- Libertyの構成管理
- server.xmlにパラメーターをどこまで書くか？
- server.xmlを分割するかどうか、どのように分割するか？
- 環境差異の吸収方法

Libertyの構成管理

- Libertyの構成はWebSphere Developer Tools for Eclipse(以下、WDT)を利用する
  - 自動的な構文チェックやGUIベースの構成が可能
  - Libertyの構成を行うためにEclipseを導入する

- マスターとなるLiberty構成ファイルを管理し、本番環境の設定は変更せずにマスターの構成ファイルで上書きする
  - Immutable Infrastructureに基づく構成管理

    「今、動いているものには手をつけない」というポリシーを元にした基盤構築の考え方。新たな変更が必要になった場合、稼働中の基盤には一切手を触れず、新しく「同じ構成の基盤」を準備して変更を適用する。テストを実施したそのものが本番化されるので、品質が明確に担保でき安全である。

server.xmlにパラメーターをどこまで書くか？

- server.xmlにどこまで詳細にパラメーターを記述するかは基本的に自由

（案１）最低限の内容のみ記載する

- デフォルト値を変更したい場合にのみ記載する
- 書かれているもの＝デフォルトから変更したものなので、環境固有の値が一目でわかる
- 構成ファイルの記述量が少ないため、構成が容易ですぐにアプリケーションを動かすことが可能
- 記載されていないパラメーターのデフォルト値はWDT(WebSphere Developer Tools)やマニュアルで確認する
- 開発環境におけるスタートアップの構成はこちらがおすすめ

（案２）設計の結果、デフォルト値を採用する場合は明示的に記載する

- 設計パラメーターが全て構成ファイル上に記載されているため、実機の構成ファイルで設定確認が可能
- タイムアウトやキューイングネットワークのチューニング対象のパラメーターは事前に記載しておけば、変更も容易になる
- server.xmlがパラメーター一覧になるため、環境定義書は不要にしてもよいという考え方もできる
- どこまで書くのかが難しく、実際に設定できる項目は多岐にわたるため、そのどの部分までを記述するのかの判断が必要
- 詳細な部分まで全部書いていくと可読性が著しく失われる

server.xmlを分割するかどうか、どのように分割するか？

- server.xmlは分割せずに1ファイルで定義した方がバージョン管理がシンプルになる
- 以下の例のように分割することも可能

(1)機能単位で構成ファイルを用意して、server.xml内で組み合わせて使う

例：アプリケーション、DB接続、JMS接続、EJB、ユーザー認証、SSLなど機能単位で標準構成を定義してそれを組み合わせる

(2)複数JVMで構成ファイルを共有する

shared.config.dirディレクトリにXMLファイルを配置すれば、同一ノード上の複数JVMから参照することができる

例）複数JVMから同一LDAPサーバーに接続してユーザー認証を行うため、LDAPサーバーの設定は共有する

(3)チームで分ける

例：アプリケーションチームが構成する定義を別ファイルにする
（アプリケーション、共有ライブラリなど）

(4)可読性を上げる

例：JMSキューなどリソース設定の量が多く、server.xmlの行数が膨大になる場合、リソース設定のみ外だしする

```
<server description="new server">
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>jndi-1.0</feature>
        <feature>jdbc-4.1</feature>
        <feature>appSecurity-2.0</feature>
        <feature>wmqJmsClient-2.0</feature>
    </featureManager>
    ・・(中略)・・
    <include location="${server.config.dir}/db.xml"/>                ・・・(1)
    <include location="${shared.config.dir}/LDAPRegistry.xml"/>      ・・・(2)
    <include location="/applhome/application.xml"/>                  ・・・(3)
    <include location="${server.config.dir}/jmsq.xml"/>              ・・・(4)
    ・・(中略)・・
</server>
```

環境差異の吸収方法(1/2)

- server.envにサーバー・プロセスで使用する環境変数を定義して、環境固有のホスト名やポート番号を記載する

```
DB_SERVERNAME=dbhost01.makuhari.japan.ibm.com
DB_PORT=50000
DB_DBNAME=sample
```

  - server.xmlでは環境変数名を指定することで、server.xmlの環境差異の値を吸収することができる

```
<dataSource id="dataSource01" jndiName="jdbc/sample" type="javax.sql.ConnectionPoolDataSource">
    <properties.db2.jcc
        serverName="${env.DB_SERVERNAME}"
        portNumber="${env.DB_PORT}"
        databaseName="${env.DB_DBNAME}"
    />
    <jdbcDriver libraryRef="db2jcc4lib"/>
</dataSource>
```

- server.xml内でOSの環境変数を使うことも可能であり、以下のように指定する
  - ${env.<OS環境変数名>}
  - 例) serverName="${env.OS_ENV}"

https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_setup_vars.html

環境差異の吸収方法(2/2)

- bootstrap.propertiesに変数を定義して、環境固有の値を記載する

```
HTTP_port=9086
```

  - server.xmlでは以下のように変数名を指定することで、server.xmlの環境差異の値を吸収することができる

```
<httpEndpoint id="defaultHttpEndpoint" httpPort="${HTTP_port}" />
```

  - bootstrap.propertiesでは外部ファイルの読み込みができる
    - 相対パスを記入した場合はbootstrap.propertiesファイルの場所が起点になる

```
bootstrap.include=../../shared/shared_bootsrap.properties
```

    - 外部ファイルに変数名を指定する

shared_bootsrap.properties

```
HTTP_port=9086
```

- 変数を定義したインクルードファイルを作成して、server.xmlから読み込む

server.xml

```
<include location="${shared.config.dir}/variable.xml"/>
```

variable.xml

```
<variable name="HTTP_port" value="8889" />
```

同じ変数が複数の場所に指定されている場合、優先順位は次のようなる

- bootstrap.properties 内の変数が、プロセス環境変数をオーバーライドする
- server.xml 内の変数、または組み込まれている XML ファイルが、bootstrap.properties 内の変数およびプロセス環境変数をオーバーライドする

JVM

- Libertyが利用するJavaの選択 - server.envにJAVA_HOMEを設定する
- jvm.optionsのサンプル

Libertyが利用するJavaの選択 - server.envにJAVA_HOMEを設定する

- Libertyは環境変数"JAVA_HOME"で指定されているJavaを使う
- server.envにJAVA_HOMEを指定する

```
JAVA_HOME=/opt/IBM/wlp/java/8.0
```

– 以下の優先順位で決定する
  1. JAVA_HOME
  2. JRE_HOME
  3. WLP_DEFAULT_JAVA_HOME(Libertyの導入ディレクトリ配下のJava)
  4. パスが通っているjava

– Libertyの導入ディレクトリ配下のJavaを使いたい場合
  - JAVA_HOMEのJavaが優先されるため、Libertyの導入ディレクトリ配下のJavaが常に使われるとは限らない

jvm.optionsのサンプル

- JVM関連の設定を行う
  - ヒープサイズ
  - GC関連（GCポリシー、GCログ）
  - ダンプ設定（-Xdump)
  - システムプロパティー
  - など

  （※）-CLASSPATHオプションは使えない
- jvm.optionsのサンプルは以下のとおり。要件に応じて設定する。

```
-Xms512m                …初期ヒープサイズ
-Xmx1024m               …最大ヒープサイズ
-verbose:gc             …冗長ガーベッジコレクションの有効化
-Xgcpolicy:gencon       …GCポリシー
-Xverbosegclog:logs/verbosegc.%Y%m%d.%H%M%S.%pid.%seq.log,10,7000                    …GCログのファイル名およびローテーション
-Xdump:system:file=/core/system/core.%Y%m%d.%H%M%S.%pid.%seq.dmp                     …coreダンプの出力ファイル名
-Xdump:java:file=/core/javacore/javacore.%Y%m%d.%H%M%S.%pid.%seq.txt                …javacoreの出力ファイル名
-Xdump:heap:file=/core/heapdump/heapdump.%Y%m%d.%H%M%S.%pid.%seq.phd                …heapdumpの出力ファイル名
-Xdump:snap:file=/core/snap/Snap.%Y%m%d.%H%M%S.%pid.%seq.trc                        …Snapトレースの出力ファイル名
-Xdump:tool:events=systhrow,filter=java/lang/OutOfMemoryError,exec=kill -9 %pid     …OutOfMemory発生時にプロセスをkillする
-Dfile.encoding=UTF-8                                                               …ファイルエンコーディング
```

フィーチャー

- フィーチャーの定義 - 明示的ロードと暗黙的ロード
- フィーチャーの定義 - 他のフィーチャーの暗黙的ロードに頼らない

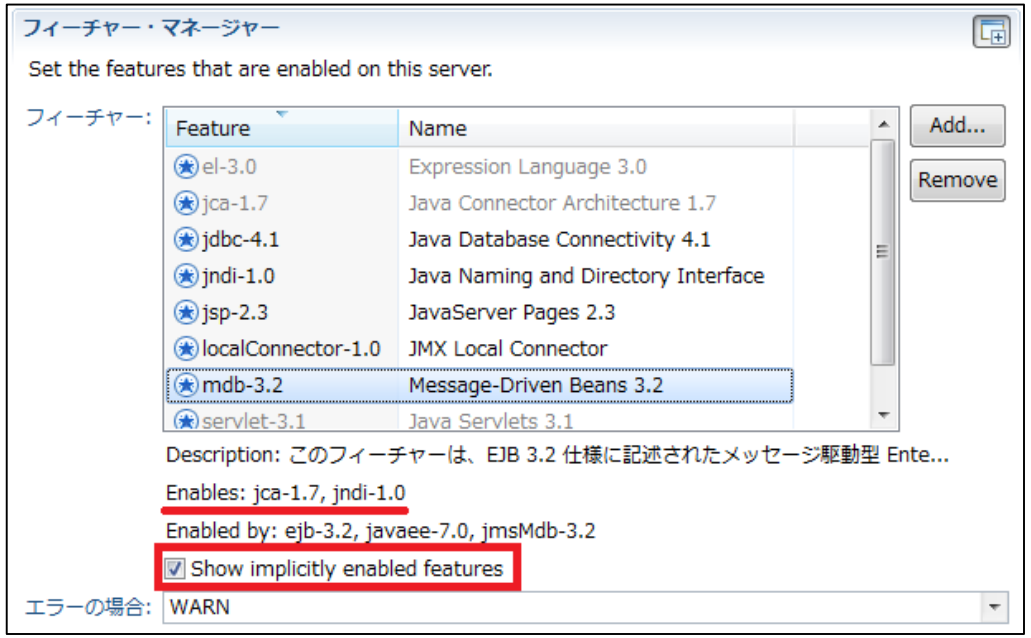
フィーチャーの定義 - 明示的ロードと暗黙的ロード

- server.xmlに明示的に指定するフィーチャーと暗黙的に有効化されるフィーチャーがある
- Eclipse上のWDTで暗黙的フィーチャーを確認することも可能

server.xml

```
<featureManager>
    <feature>jsp-2.3</feature>
    <feature>jdbc-4.1</feature>
    <feature>jndi-1.0</feature>
    <feature>mdb-3.2</feature>
    <feature>localConnector-1.0</feature>
</featureManager>
```

Eclipse上のデザインエディター



- 起動時にロードされたフィーチャーがmessages.logおよびconsole.logに出力される

```
[17/12/17 12:49:42:388 JST] 00000020 com.ibm.ws.kernel.feature.internal.FeatureManager            A CWWKF0012I: サーバーは次のフィーチャーをインストールしました。[jsp-2.3, servlet-3.1, mdb-3.2, jndi-1.0, jca-1.7, localConnector-1.0, jdbc-4.1, el-3.0]。
```

フィーチャーの定義 - 他のフィーチャーの暗黙的ロードに頼らない

- JSP、REST、EJB、JMS接続、DB接続など各機能で必要とするフィーチャーは明示的にfeatureManagerに指定する
- 別機能のために指定したフィーチャーの暗黙的ロードに頼らないフィーチャー定義がおすすめ
  - フィーチャーには暗黙的にロードされるフィーチャーが含まれている場合がある
  - 例)MDB-3.2はjndi-1.0を暗黙的にロードする

例）JSP、MDB、DB接続を行う場合、①と②のいずれのフィーチャー定義でも必要なフィーチャーはロードされているため問題なく稼働するが・・・

①
```
<featureManager>
    <feature>jsp-2.3</feature>
    <feature>jdbc-4.1</feature>
    <feature>mdb-3.2</feature>
    <feature>localConnector-1.0</feature>
</featureManager>
```

jndi-1.0を暗黙的にロードする

②
```
<featureManager>
    <feature>jsp-2.3</feature>
    <feature>jdbc-4.1</feature>
    <feature>jndi-1.0</feature>
    <feature>mdb-3.2</feature>
    <feature>localConnector-1.0</feature>
</featureManager>
```

- 仮にMDBの要件がなくなり、mdb-3.2を削除するとDB接続ができなくなる問題が発生する
- MDBの暗黙的ロード(jndi-1.0)に頼ってDB接続が動いていたため

①
```
<featureManager>
    <feature>jsp-2.3</feature>
    <feature>jdbc-4.1</feature>
    <feature>localConnector-1.0</feature>
</featureManager>
```

DB接続NG

②
```
<featureManager>
    <feature>jsp-2.3</feature>
    <feature>jdbc-4.1</feature>
    <feature>jndi-1.0</feature>
    <feature>localConnector-1.0</feature>
</featureManager>
```

DB接続OK

アプリケーション管理

- アプリケーションのアーカイブ配置の制約
- 共有ライブラリーの指定方法　file、fileset、folderの違いと使い分け
- バインディング情報をserver.xmlで指定する方法

アプリケーションのアーカイブ配置の制約

- WARファイル、EARファイルをそのままディレクトリー上に配置することが可能だが、以下の制約がある

【制約】

– ServletContext#getRealPath()を利用して仮想パスに対応する絶対パスを取得することができず、nullを返す

```
ServletContext context = this.getServletContext();
String path = context.getRealPath("/WEB-INF/sample.props");
```

【対応策】

以下のいずれかの対応をとる

1. ServletContext#getRealPath()を使わないようにアプリケーションを改修する
2. アプリケーションを展開してから配置する
3. server.xmlにautoExpand=trueを設定して、自動展開する

```
<applicationManager autoExpand="true"/>
```

Liberty: ランタイム環境での既知の問題および制約事項

https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_restrict.html

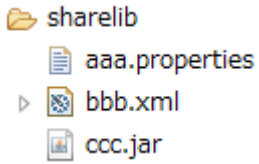
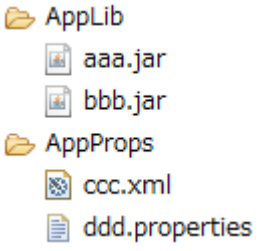
共有ライブラリーの指定方法　file、fileset、folderの違いと使い分け

- 共有ライブラリー(library)の定義方法

子エレメント名	用途
file	ネイティブライブラリーあるいはJARファイル、ZIPファイルが指定できる。
fileset	複数のfile(ネイティブライブラリーあるいはJARファイル、ZIPファイル)を指定する。includesで読み込むファイルを指定。
folder	XMLファイルやプロパティーファイルなどのリソースファイルが格納されたディレクトリを指定する。 fileやfilesetで指定するJARファイルなどは読み込まれないため注意。

https://www.ibm.com/support/knowledgecenter/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_sharedlibrary.html

- 共有ライブラリーの追加・変更・削除があったとしてもserver.xmlの修正が不要になる設定方法がおすすめ
  - server.xmlでディレクトリを指定しておけば、共有ライブラリーを指定ディレクトリに配置するだけでロードできる

ディレクトリおよびファイル配置	library定義サンプル	
	<library id=shareLibrary> <folder dir="/sharelib" /> <fileset dir="/sharelib" includes="*.jar" /> </library>	<library id=shareLibrary> <fileset dir="/sharelib" includes="*.jar" /> </library> <library id=shareProps> <folder dir="/sharelib" /> </library>
	<library id=appLibrary> <fileset dir="/AppLib" includes="*.jar" /> <folder dir="/AppProps" /> </library>	<library id=appLibrary> <fileset dir="/AppLib" includes="*.jar" /> </library> <library id=appProps> <folder dir="/AppProps" /> </library>

バインディング情報をserver.xmlで指定する方法

- WAS traditionalではアプリケーションをデプロイする際にバインディングを行っていたが、Libertyではアプリケーションのデプロイ操作がないため、バインディングファイル（ibm-web-bnd.xml）を用意する必要があった。
- Liberty 17.0.0.1以上では、これらバインディング情報をserver.xmlの中で記述可能となった。
  - server.xml（サンプル）

```
<webApplication location=“test.war” >
    <web-bnd>
        <virtual-host name=“customeHost” />
        <resource-ref name=“jdbc/dataSource”  binding-name=“jdbc/webAppDataSource”  />
    </web-bnd>
</webApplication>
```

※ibm-web-bnd.xmlとserver.xmlの両方に指定された場合、server.xmlが優先される。

Webコンテナー/Servlet/JSP/HTTPセッション

- リクエストパラメーターの数を制限する
- HTTPエンドポイントと仮想ホストを複数定義する方法
- WEB-INF配下のJSPはプリコンパイルできない
- セッションDB障害時のリクエスト遅延緩和策
- Servletの初期化順序およびタイミング

リクエストパラメーターの数を制限する（maxParamPerRequest）

- リクエストパラメーター数の上限は、デフォルトで”10000“に設定されている。
- 以下の設定を行うことにより、上限数を増加させることができる。
  - server.xml（サンプル）

```
<webContainer maxParamPerRequest= “30000” />
```

- アプリケーション要件でパラメーターの上限を変更する必要がある場合には指定する。
- 値に「-1」を設定すると、パラメーター数を無制限にすることができる。ただ、この場合には外部からHashDos攻撃によりサーバーを使用不能にすることができるため、インターネットに公開されているサーバーなどでは無制限に設定することは避ける。

HTTPエンドポイントと仮想ホストを複数定義する方法

- HTTPエンドポイントと仮想ホストを複数定義する場合、仮想ホスト側でHTTPエンドポイントを紐づける
- HTTPエンドポイントで定義したポート番号は仮想ホスト側にも定義する

```
<httpEndpoint
    id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443"
    accessLoggingRef="http_accesslog"
    httpOptionsRef="http_option"
    sslOptionsRef="ssl_option"
    tcpOptionsRef="tcp_option" enabled="true">
</httpEndpoint>

<httpEndpoint
    id="sampleHttpEndpoint"
    host="*"
    httpPort="9081"
    httpsPort="9444"
    accessLoggingRef="http_accesslog"
    httpOptionsRef="http_option"
    sslOptionsRef="ssl_option"
    tcpOptionsRef="tcp_option" enabled="true">
</httpEndpoint>

続き⇒
```

```
⇒続き
<virtualHost id="default_host" allowFromEndpointRef="defaultHttpEndpoint">
    <hostAlias>test01.ise.com:9080</hostAlias>
    <hostAlias>test01.ise.com:9443</hostAlias>
</virtualHost>

<virtualHost id="sample_host" allowFromEndpointRef="sampleHttpEndpoint">
    <hostAlias>test02.ise.com:9081</hostAlias>
    <hostAlias>test02.ise.com:9444</hostAlias>
    <hostAlias>test02.ise.com:80</hostAlias>
    <hostAlias>test02.ise.com:443</hostAlias>
</virtualHost>

<httpAccessLogging
    id="http_accesslog"
    filePath="${server.output.dir}/logs/http_access.log"
    logFormat='%h %u %{t}W "%r" %s %b'
    maxFileSize="20" maxFiles="2" enabled="true" />
<httpOptions id="http_option" keepAliveEnabled="true" />
<sslOptions id="ssl_option" sessionTimeout="1d" sslSessionTimeout="8640ms"/>
<tcpOptions id="tcp_option" inactivityTimeout="60s" />
```

WEB-INF配下のJSPはプリコンパイルできない

- LibertyではjspEngineでprepareJSPsに指定されたバイト数以上のJSPファイルがプリコンパイルされる。
- すべてのJSPをプリコンパイルしたい場合は0を指定する。
  - server.xml（サンプル）

```
<jspEngine prepareJSPs="0"/>
```

- 注意点として、LibertyではWEB-INF配下のJSPファイルはプリコンパイルされない。
  - Libertyではサーバーが JSPに直接リクエストすることでプリコンパイルを行うため、外部からアクセスできる必要がある
  - WAS traditionalではWEB-INF配下のJSPファイルもプリコンパイルされていたが、本来WEB-INF配下のディレクトリは外部に公開されないディレクトリとなるため、仕様が変更された
- また、WAS traditionalで使用可能だったJSPBatchCompilerコマンドはLibertyでは提供されていない。

- LibertyでJSPファイルのプリコンパイルを行う場合、WEB-INF配下にJSPファイルを配置しない設計とするか、暖気運転（起動直後にWEB-INF配下のJSPファイルを参照する機能にアクセスする）といった運用を検討する。
- なお、直接アクセスさせたくないJSPファイル（例えばinclude用のJSPファイルなど）を、外部に公開されないことを目的にWEB-INF配下に配置していた場合は、ディレクトリを変更し前段の負荷分散装置等からアクセス制御するといった変更が必要。

セッションDB障害時のリクエスト遅延緩和策

- セッションDBのノード障害やネットワーク障害が発生するとリクエストの応答遅延が発生して、パフォーマンスに影響を及ぼす可能性がある

【対策】

1. セッション・マネージャーによる接続再試行回数を調整する
   - セッション・マネージャーがセッションDBに対してデフォルトで2回の接続再試行を行うため、getConnection()を3回試行する
   - 接続再試行回数を調整することで遅延時間を短縮することが可能
   - 再試行を行わないことでJavaヒープとセッションDBのHTTPセッション情報の不整合は発生する（ネットワーク瞬断など）
     セッションDB障害時にJVM障害が発生した場合に、セッションDBから古いセッション情報を取得することになる

```
<httpSession ConnectionRetryCount="1" />
```

```
<httpSession ConnectionRetryCount="0" />
```

2. DB接続に関するタイムアウトパラメーターを調整する
   - ノード障害およびネットワーク障害の検知までにかかる時間をTCP/IP関連のパラメーターで調整する

詳細は下記URLを参照

- WASセッションDB障害時のリクエスト遅延緩和策
- https://www-01.ibm.com/support/docview.wss?uid=jpn1J1013323

Servletの初期化順序およびタイミング

- Servlet仕様で定義されているload-on-startupについて
  - アプリケーションデプロイ時に指定された順序でサーブレットを初期化する（アプリ要件）
  - リクエストが来る前にサーブレットを初期化することで、初回リクエスト時の応答時間のオーバーヘッドを解消する（パフォーマンス要件）

web.xmlの場合

```
<servlet>
    <servlet-name>Servlet01</servlet-name>
    <servlet-class>com.ise.Servlet01</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

アノテーションの場合

```
@WebServlet(urlPatterns = "/Servlet01", loadOnStartup = 1)
public class Servlet01 extends HttpServlet {
…
}
```

- V17.0.0.1以降、Webコンテナーの deferServletLoadの設定にかかわらず、load-on-startupが指定されたサーブレットは初回リクエストが来る前にロード・初期化が行われる

```
<webContainer deferServletLoad="true"/>  ・・・デフォルト設定
```

deferServletLoad	load-on-startup	初期化タイミング	初期化順序	補足
true（デフォルト）	指定あり	サーバー起動完了直後	指定順序で初期化	サーバー起動完了後、Webモジュールを起動したスレッドとは別のスレッドでServletの初期化を実施する
	指定なし	初回リクエスト	-	-
false	指定あり	サーバー起動時	指定順序で初期化	サーバー起動完了前にWebモジュールを起動するスレッドがServletの初期化を実施する
	指定なし	初回リクエスト	-	-

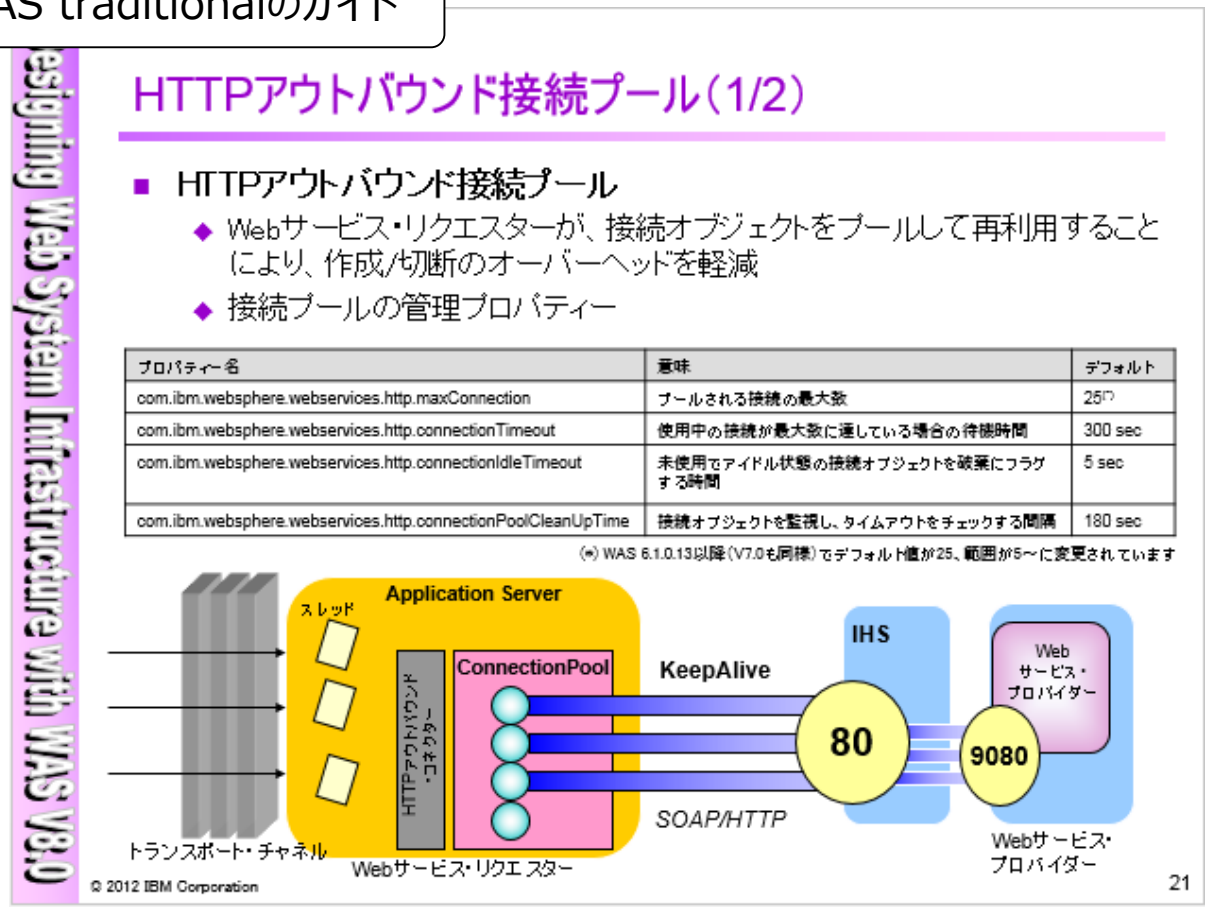
外部連携

- JAX-WS Webサービス(SOAP)　HTTPアウトバウンド接続プールはない
- Libertyにおけるタイムアウト値の"0"と"-1"の意味　　無効?即時?

JAX-WS Webサービス(SOAP)　HTTPアウトバウンド接続プールはない

- WAS traditionalがJAX-WS Webサービスリクエスターになるとき、HTTPアウトバウンド接続プールがあった
- LibertyにはHTTPアウトバウンド接続プールは提供されていない

WAS traditionalのガイド



Libertyにおけるタイムアウト値の"0"と"-1"の意味　無効?即時?

- Libertyにおけるタイムアウト値の"0"と"-1"の意味
  “0” ⇒ **「即時」**タイムアウト
  "-1" ⇒ タイムアウト**「なし（無効）」**
- 特に以下の属性に適用される(connectionManagerエレメント)
  - agedTimeout
  - connectionTimeout
  - maxIdleTime(tWASのunusedTimeoutに相当)
  - reapTime
- WAS traditionalとは逆の意味になるため注意する

WebSphere Application Server traditionalと Liberty での構成の相違点: connectionManager エレメント
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_connpool_diff.html

セキュリティー

- SSL関連
- 【構成サンプル】Active Directoryに接続する

SSL関連

- Libertyでは、証明書作成用にsecurityUtilityコマンドが用意されているが、指定できるオプションが少なく、jks形式の鍵ストアのみ作成可能となっている。
- このTips集では、Global Security Kit（GSKit）を使用し、PKCS12形式の証明書（鍵ストア）を使用する方法を紹介する。
- なお、GSKitはLibertyには付属しないため、IHS付属のものを使用するか別途ダウンロードする必要がある。

- gskcapicmdによる鍵ストア/自己署名証明書の作成（サンプル）

```
# gskcapicmd -keydb -create -db myKeyStore.p12 -type p12 -pw password
# gskcapicmd -cert -create -db myKeyStore.p12 -pw password -label default -dn  “cn=...”
```

- デフォルトSSL構成での指定方法（PKCS12形式を使用する場合は、typeの指定が必要）
  - server.xml（サンプル）

```
<sslDefault sslRef="mySSLSettings" />
<ssl id=" mySSLSettings " keyStoreRef= “myKeyStore" trustStoreRef="defaultTrustStore" />
<keyStore id="myKeyStore" location="myKeyStore.p12" password= “password" type="PKCS12"/>
```

【構成サンプル】Active Directoryに接続する

- 必要最低限の設定

```
<featureManager>
    <feature>appSecurity-2.0</feature>
    <feature>ldapRegistry-3.0</feature>
     ・・・
</featureManager>

<ldapRegistry
   realm="LdapRegistry"
   ldapType="Microsoft Active Directory"
   host="ad01"
   port="389"
   sslEnabled="false"
   bindDN="username"
   bindPassword="{xor}Lz4sLCgwLTs="
   baseDN="cn=Users,dc=ise,dc=ibm,dc=local"
>
</ldapRegistry>
```

- LDAPSの場合はSSLの設定も追加する

- 本番環境を想定した設定（冗長構成やチューニングパラメーターを含む)

```
<featureManager>
    <feature>appSecurity-2.0</feature>
    <feature>ldapRegistry-3.0</feature>
     ・・・・
</featureManager>

<federatedRepository>
    <primaryRealmname="LdapRegistry" allowOpIfRepoDown="false"/>
</federatedRepository>

<ldapRegistry realm="LdapRegistry" ldapType="Microsoft Active Directory"
    host="ad01" port="389" sslEnabled="false" bindDN="username"
    bindPassword="{xor}Lz4sLCgwLTs="
    baseDN="cn=Users,dc=ise,dc=ibm,dc=local"
    returnToPrimaryServer="true" searchTimeout="1m" connectTimeout="1m">
    <failoverServers>
        <server host="ad02"port="389"/>
    </failoverServers>
    <contextPool enabled="true" initialSize="1" maxSize="0" timeout="0s"
                 waitTime="3s" preferredSize="3"/>
    <ldapCache>
        <searchResultsCache enabled="true" size="2000" timeout="1200ms"
                            resultsSizeLimit="2000"/>
        <attributesCache enabled="true" size="2000" timeout="1200ms"
                         sizeLimit="2000"/>
    </ldapCache>
</ldapRegistry>
```

運用

- 製品ログの言語
- java.util.loggingによる業務ログの出力
- ランタイムトレース設定　動的更新、cURLサンプル
- chainQuiesceTimeout / startTimeout / stopTimeout
- server startと打つと、defaultServerが自動的に作成されて起動してしまう
- プロセス稼動確認に.pidファイルを使用しない
- ハングスレッド検知　requestTimingフィーチャーの詳細挙動
- OutOfMemoryError発生時のdumpAgentの指定
- -Xdumpのサンプル

製品ログの言語

- 海外サポートチームにログを連携する場合などで、Libertyが出力するログの言語を変更する場合
  jvm.optionsに以下のシステムプロパティーを指定する

```
-Duser.language=en
```

java.util.loggingによる業務ログの出力

- Libertyのロギング・コンポーネントはSystem.out、System.err、java.util.logging、および OSGi ロギングに書き込まれるメッセージをキャプチャーする
- java.util.loggingを使用して業務ログを出力する場合、ロギング設定のtraceSpecificationで指定したログレベルより詳細を出力することができない
  - デフォルト設定(traceSpecification="*=info")の場合
    - 出力されるレベル　⇒　Logger.info(),Logger.warning(),Logger.severe()
    - 出力されないレベル　⇒　Logger.finest(),Logger.finer(),Logger.fine(),Logger.config()
  - infoより低いログレベルを出力するためには、以下のように設定を行うことで業務ログの出力が可能になる

```
<logging traceSpecification="*=info:myLogger=FINEST" />
```

- ログレベルがinfo以上の業務ログはmessages.logにも出力される
- 業務ログをmessages.logに出力したくない場合は、Libertyのログレベルと業務ログのログレベルに差をつける

  例）業務ログのログレベルは通常のログはFINE、デバッグレベルのログはFINER/FINESTで出力する。infoは利用しない。

https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_logging.html

ランタイムトレース設定とserver.xmlの動的更新およびcURLサンプル

- WAS traditionalではランタイムでトレース設定を変更することが可能だった。
- Libertyではserver.xmlの動的更新を行うことで同様のことが可能。
- 動的更新には、定期的に更新を見に行く方法と、mbeanにより更新する方法の2つがある。
  - 500ミリ秒毎に更新を見にいく設定
    - server.xml（サンプル）

```
<config updateTrigger="polled" monitorInterval="500ms"/>
```

  - mbeanにより更新する設定
    - server.xml（サンプル）

```
<config updateTrigger="mbean"/>
```

    - Mbeanコマンドサンプル

```
curl -X POST -s -d '{"params":[{"value" : [""],"type" :
{"className":"java.util.ArrayList","items":["java.lang.String"]}},{"value" : ["更新対象のファイルパス"],"type" :
{"className":"java.util.ArrayList","items":["java.lang.String","java.lang.String"]}},{"value" : [""],"type" :
{"className":"java.util.ArrayList","items":["java.lang.String"]}}],"signature":["java.util.Collection","java.util
.Collection","java.util.Collection"]}' -H 'Content-Type:application/json' -u restユーザーID:パスワード -k
https://対象サーバー:ポート
/IBMJMXConnectorREST/mbeans/WebSphere%3Aservice%3Dcom.ibm.ws.kernel.filemonitor.FileNotificationMBean/operations/
notifyFileChanges
```

chainQuiesceTimeout / startTimeout / stopTimeout（1/2）

- アプリケーションの起動/停止に関するタイムアウトを以下のパラメータで設定できる。
  - server.xml（サンプル）

```
<channelfw chainQuiesceTimeout="30s" />
<applicationManager startTimeout="30s" stopTimeout="30s" />
```

    - chainQuiesceTimeout　　リクエストの静止を待つ時間。（デフォルト30秒、最大30秒*）
    - startTimeout　　アプリケーションの起動を待つ時間。（デフォルト30秒）
    - stopTimeout　　アプリケーションの停止を待つ時間。（デフォルト30秒、最大60秒*）

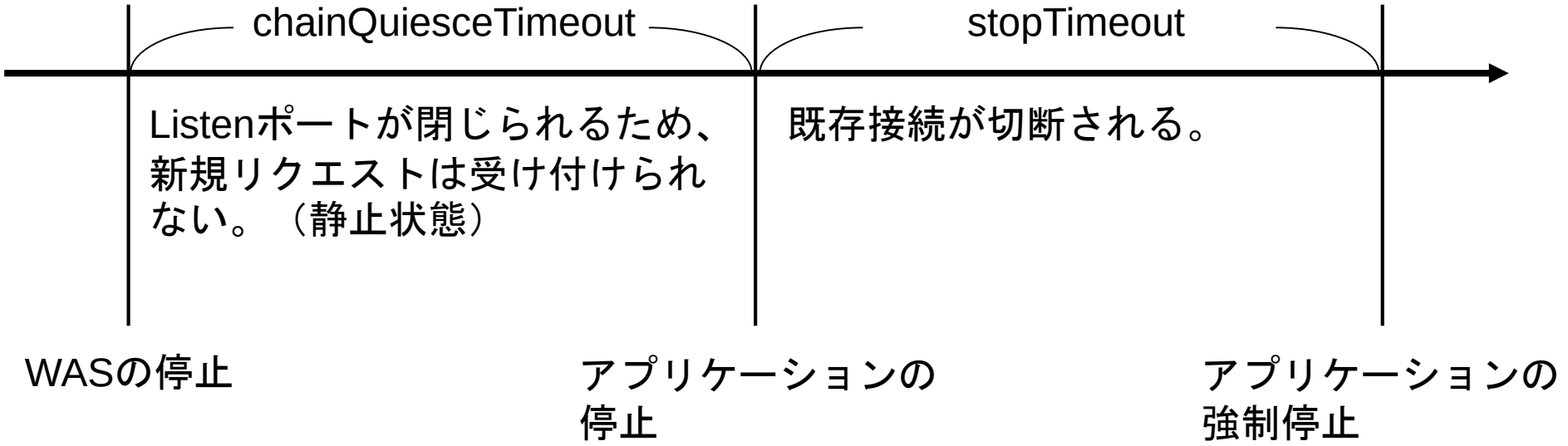
    * タイムアウト値に最大秒数より長いタイムアウト値を設定したとしても、最大秒数でタイムアウトする

  - WASを開始した場合の挙動
    - アプリケーションの起動の完了を待つ。
    - startTimeout経過後、（アプリケーションの起動状態に関わらず）サーバーの起動コマンドは完了となる。

    LibertyはWAS traditionalと比較すると起動時間は短くなっているが、重いアプリケーションを起動するには同じように時間が掛かる。

chainQuiesceTimeout / startTimeout / stopTimeout（2/2）

– WASを停止させた場合の挙動（アプリケーションが複数存在している場合、並行して停止が行われる）
  - 新規リクエストが受け付けられなくなる。（静止状態）
  - chainQuiesceTimeout経過後、アプリケーションを停止する。
  - stopTimeout経過後、アプリケーションを強制停止する。



server startと打つと、defaultServerが自動的に作成されて起動してしまう

- serverコマンドでLibertyを起動させる際に、引数にサーバー名を指定しない場合、“defaultServer”という名前のサーバーが作成、起動される。
  - –コマンド実行結果

```
# server start
サーバー defaultServer を始動中です。
サーバー defaultServer が始動しました。
```

- 設定等による防止方法が無いため、運用スクリプトを用意して起動停止を行うといった回避が必要。

プロセス稼動確認に.pidファイルを使用しない

- Libertyを起動すると、プロセスIDが(WLP_OUTPUT_DIR)/.pid/(サーバー名).pidファイルに記録される。

- .pidファイルは排他制御されていないため、サーバーの起動処理が同時に複数実行された場合に、実際のプロセスIDと.pidファイル内に記載されたプロセスIDが異なることがある。
- Libertyのランタイムは.pidファイルを参照しないためこの状態でも稼動や停止には影響はない。

- 運用スクリプト等でプロセスの稼動確認にこの.pidファイルを使用している場合で問題となるケースがある。
- この場合、psコマンド等でプロセスを確認するといった方法に変更する必要がある。

ハングスレッド検知　requestTimingフィーチャーの詳細挙動(1/3)

- requestTimingフィーチャーはリクエストの処理時間を監視することができる
- slowRequestとhungRequestの2段階で検知が可能で、messages.logへの出力およびjavacoreの生成を行う
  - slowRequest　・・・処理が遅いと判断して、ログを出力する（デフォルト10秒）
  - hungRequest ・・・処理がハングしていると判断して、ログを出力してjavacoreを生成する（デフォルト10分）
- requestTimingフィーチャーの設定サンプルは以下の通り

```
<featureManager>
    <feature>requestTiming-1.0</feature>
</featureManager>

<requestTiming
    slowRequestThreshold="10s"
    hungRequestThreshold="10m"
    sampleRate="1"
    includeContextInfo="true"
    interruptHungRequests="false">
</requestTiming>
```

- slowRequest検知の閾値を設定(デフォルト10秒)
- hungRequest検知の閾値を設定(デフォルト10分)
- トラッキング対象のリクエストをサンプリングするレートを設定。n件ごとに1件のリクエストをサンプリングする場合はnを指定。デフォルトは"1"で全リクエストが対象。
- コンテキスト情報をログに出力するかどうか
- hungRequestを中断するかどうか

ハングスレッド検知　requestTimingフィーチャーの詳細挙動(2/3)

- messages.logへのメッセージ出力

イベント	メッセージID	出力間隔および回数	メッセージ出力単位
slowRequestの検知	TRAS0112W	閾値の間隔で最大3回	サンプリングされたリクエスト単位
hungRequestの検知	TRAS0114W	ハング解消するまで閾値の間隔で毎回	サンプリングされたリクエスト単位
hungRequestの解消	TRAS0115W	処理が完了したタイミングで1回	サンプリングされたリクエスト単位

- javacore生成の挙動

イベント	出力有無	出力間隔および回数	javacore出力単位
slowRequestの検知	無	-	-
hungRequestの検知	有	閾値の間隔で1セット (1分間隔で3回)	ハング検知単位。リクエスト単位ではない。 複数リクエストが並行してハングしていたとしても、javacoreは1セットだけ生成される。
hungRequestの解消	無	-	-

ハングスレッド検知　requestTimingフィーチャーの詳細挙動(3/3)

- messages.logのサンプル　(slowRequestThreshold=“10s“ hungRequestThreshold=“23s”を設定した例)

```
[17/12/14 10:32:45:157 JST] 00000044 com.ibm.ws.request.timing.manager.SlowRequestManager        W TRAS0112W: 要求 AACu4m/OaDI_AAAAAAAAAB は、スレッド 00000041 で 10025.676 ミリ秒以上実行されています。以下のスタック・トレースは、このスレッドが現在実行中の内容を示しています。
    at java.lang.Thread.sleep(Native Method)
    at java.lang.Thread.sleep(Thread.java:941)
  (中略)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)
    at java.lang.Thread.run(Thread.java:811)
次の表は、この要求で実行されたイベントを示しています。
所要時間          Operation
10059.005ms + websphere.servlet.service | HeapTest | HeapServlet
  ・・・
  (中略)
  ・・・
[17/12/14 10:32:58:093 JST] 00000048 com.ibm.ws.request.timing.manager.HungRequestManager        W TRAS0114W: 要求 AACu4m/OaDI_AAAAAAAAAB は、スレッド 00000041 で 23012.745 ミリ秒以上実行されています。次の表は、この要求で実行されたイベントを示しています。
所要時間          Operation
23013.353ms + websphere.servlet.service | HeapTest | HeapServlet
[17/12/14 10:32:58:218 JST] 00000049 com.ibm.ws.kernel.launch.internal.FrameworkManager          A CWWKE0067I: Java dump request received.
[17/12/14 10:33:10:381 JST] 00000041 SystemOut                                     O 30000 ミリ秒 待機
[17/12/14 10:33:10:381 JST] 00000049 com.ibm.ws.kernel.launch.internal.FrameworkManager          A CWWKE0068I: Java dump created:
C:¥IBM¥Liberty¥usr¥servers¥defaultServer¥javacore.20171214.103258.5192.0001.txt
[17/12/14 10:33:10:381 JST] 00000041 com.ibm.ws.request.timing.manager.HungRequestManager        W TRAS0115W: スレッド 00000041 の要求 AACu4m/OaDI_AAAAAAAAAB は、ハングしたと先に検出されましたが、35295.446 ミリ秒後に完了しました。
[17/12/14 10:33:58:118 JST] 0000004e com.ibm.ws.kernel.launch.internal.FrameworkManager          A CWWKE0067I: Java dump request received.
[17/12/14 10:34:00:816 JST] 0000004e com.ibm.ws.kernel.launch.internal.FrameworkManager          A CWWKE0068I: Java dump created:
C:¥IBM¥Liberty¥usr¥servers¥defaultServer¥javacore.20171214.103358.5192.0002.txt
[17/12/14 10:34:58:115 JST] 00000054 com.ibm.ws.kernel.launch.internal.FrameworkManager          A CWWKE0067I: Java dump request received.
[17/12/14 10:35:00:284 JST] 00000054 com.ibm.ws.kernel.launch.internal.FrameworkManager          A CWWKE0068I: Java dump created:
C:¥IBM¥Liberty¥usr¥servers¥defaultServer¥javacore.20171214.103458.5192.0003.txt
```

OutOfMemoryError発生時のdumpAgentの指定（Liberty/tWAS共通）

- Out of Memoryの発生によりJavaプロセスの動作が不安定になることがあるため、OutOfMemoryError発生時にはJavaプロセスを再起動させる
- Libertyでは、jvm.optionsに-Xdumpオプションを設定することで、OutOfMemory発生時の動作を指定することができる。（JAVA_DUMP_OPTS環境変数による指定も可能だが、-Xdumpが優先される）
  - OutOfMemory発生時にkillコマンドでJVMを停止する場合
    - jvm.options（サンプル）

```
-Xdump:tool:events=systhrow,filter=java/lang/OutOfMemoryError,exec=/usr/bin/kill -9 %pid
```

  - execには、コマンドの他に実行させたいシェルスクリプト等を指定することも可能。

-Xdumpのサンプル

- ダンプの設定は複数の箇所で可能となっているが、以下の優先順位となっている。
  （リスト順が後の設定ほど優先される）

```
・デフォルトの JVM ダンプ動作。
・-Xdump:<type>:defaults を指定する -Xdump コマンド行オプション。
・DISABLE_JAVADUMP、IBM_HEAPDUMP、および IBM_HEAP_DUMP 環境変数。
・IBM_JAVADUMP_OUTOFMEMORY および IBM_HEAPDUMP_OUTOFMEMORY 環境変数。
・JAVA_DUMP_OPTS 環境変数。
・その他の -Xdump コマンド行オプション。
```

- このため、Xdumpオプションを使用する方法が確実。
- -Xdumpに設定されている内容は、以下のオプションでJavaを起動させることで確認可能。

```
# java -Xdump:what
```

- 設定サンプル
  - jvm.options（サンプル）

```
-Xdump:system:file=/core/system/core.%Y%m%d.%H%M%S.%pid.%seq.dmp                    …coreダンプの出力ファイル名
-Xdump:java:file=/core/javacore/javacore.%Y%m%d.%H%M%S.%pid.%seq.txt                …javacoreの出力ファイル名
-Xdump:heap:file=/core/heapdump/heapdump.%Y%m%d.%H%M%S.%pid.%seq.phd                …heapdumpの出力ファイル名
-Xdump:snap:file=/core/snap/Snap.%Y%m%d.%H%M%S.%pid.%seq.trc                        …Snapトレースの出力ファイル名
-Xdump:tool:events=systhrow,filter=java/lang/OutOfMemoryError,exec=kill -9 %pid  …OutOfMemory発生時にプロセスを
killする
```

参考リンク

- IBM Knowledge Center
  https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/as_ditamaps/was900_welcome_libertyndmp.html

- WASdev
  https://developer.ibm.com/wasdev/

- developerWorks - WebSphere Application Server Liberty プロファイルのテクニカル情報ページ
  https://www.ibm.com/developerworks/jp/websphere/category/libertylist/

- WebSphere Application Server コミュニティ Japan
  https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W3142f890ea04_4b2b_b746_ac9e833c537e

- Fix list for IBM WebSphere Application Server Liberty - Continuous Delivery
  http://www-01.ibm.com/support/docview.wss?uid=swg27043863