Code Inspection

Mite Ristovski Dushica Stojkoska

January 5, 2016

Software Engineering 2
Politecnico di Milano

version 1.0

Contents

1 Assigned classes and methods

1.1 Classes

`EjbBundleValidator` is the main assigned class. It is in the namespace `org.glassfish.ejb.deployment.util` whose structure is shown on Figure 1. This namespace also contains two other classes: `EjbBundleTracerVisitor` and `InterceptorBindingTranslator`; and one interface - `EjbVisitor`.

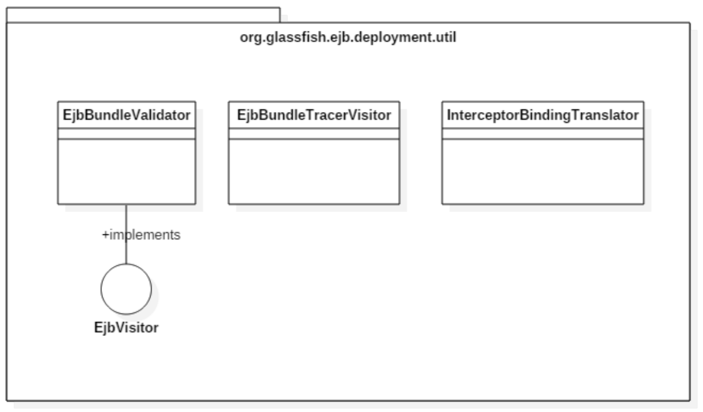
org.glassfish.ejb.deployment.util
EjbBundleValidator
EjbBundleTracerVisitor
InterceptorBindingTranslator
+implements
EjbVisitor

Figure 1: `org.glassfish.ejb.deployment.util` structure

1.2 Methods

The following methods from the `EjbBundleValidator` class were assigned to our group:

Signature: `public void accept( EjbDescriptor ejb )`

Start Line: `285`

Signature: `private void validateConcurrencyMetadata( EjbDescriptor ejb )`

Start Line: `462`

Signature: `private void validatePassivationConfiguration( EjbDescriptor ejb )`

Start Line: `518`

2 Functional roles of assigned classes and methods

2.1 `EjbBundleValidator` class

The `EjbBundleValidator` according to its *javadoc*, validates an EJB Bundle descriptor once loaded from a jar file. An EJB Bundle descriptor contains all the configurable deployment information contained in an EJB jar. The `EjbBundleValidator` class hierarchy is shown on Figure 2.

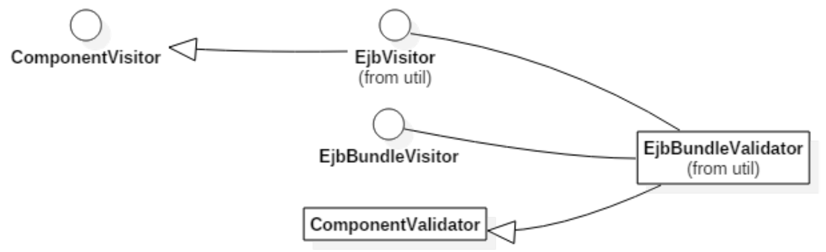

Figure 2: `EjbBundleVisitor` hierarchy

2.2 Methods

public void accept(EjbDescriptor ejb) : The *accept* method visits a descriptor recursively and verifies it. Accordingly there are defined protocols for visiting distinct descriptor classes.

private void validateConcurrencyMetadata(EjbDescriptor ejb) : This method checks whether the given argument is a session bean descriptor, and if this is true whether the described *beans* have defined *access timeout* methods and methods with *locks* so that they can run safely in concurrent setup.

private void validatePassivationConfiguration(EjbDescriptor ejb) : This method checks whether the given argument is a session bean descriptor and if it is the case gives warning, if the bean is session bean and is not passivation-capable, that PrePassivate and PostActivate configurations are not recommended.

3 Detected Issues

The issues described here are in compliance with the checklist that the professor gave us with the Assignment 3 document.

3.1 Issues in the *EjbBundleValidator* class

Comments The *javadoc* are present, but they are very simple as shown on Code 1. Maybe a little more explanation can be helpful and will provide better understanding of the code.

Code 1: `EjbBundleValidator` javadoc

```
86  /**
87   * This class validates a EJB Bundle descriptor once loaded from an .jar file
88   *
89   * @author Jerome Dochez
90   */
```

3.2 Issues in the *accept* method

Comments There is a very short *javadoc* comment and says little about the functionality of this method. Shown on Code 2.

Code 2: `accept` javadoc

```
280 /**
281  * visits an ejb descriptor
282  * @param ejb descriptor
283  */
```

Line break (line: 291) Line break not occurring after an operator or comma.

Code 3: Original

```
290 if (ejb instanceof DummyEjbDescriptor) {
291     throw new IllegalArgumentException(localStrings.getLocalString(
292     "enterprise.deployment.exceptionbeanbundle",
293     "Referencing error: this bundle has no bean of name: {0}",
294     new Object[] {ejb.getName()}));
295 }
```

Code 4: Recommended

```
290 if (ejb instanceof DummyEjbDescriptor) {
291     throw new IllegalArgumentException(localStrings.
292         getLocalString("enterprise.deployment.exceptionbeanbundle",
293             "Referencing error: this bundle has no bean of name: {0}",
294             new Object[] {ejb.getName()}));
295 }
```

Line width and spacing (line: 317) Line width is 112 chars which is not necessarily needed. Also there is no space left between the variable declaration and the if clause.

Code 5: Original

```
317 AnnotationTypesProvider provider = Globals.getDefaultHabitat().getService...
318 if (provider == null) {
```

Code 6: Recommended

```
317 AnnotationTypesProvider provider = Globals.getDefaultHabitat().
318         getService(AnnotationTypesProvider.class, "EJB");
319
320 if (provider == null) {
```

Indentation (lines: 326-328) Expressions are not aligned consistently after line break.

Code 7: Original

```
324 MethodDescriptor timedObjectMethod =
325         new MethodDescriptor("ejbTimeout",
326                                  "TimedObject timeout method",
327                                  new String[] {"javax.ejb.Timer"},
328                                  MethodDescriptor.TIMER_METHOD);
```

Code 8: Recommended

```
324 MethodDescriptor timedObjectMethod =
325         new MethodDescriptor("ejbTimeout",
326                              "TimedObject timeout method",
327                              new String[] {"javax.ejb.Timer"},
328                              MethodDescriptor.TIMER_METHOD);
```

Line break (line: 365) Line break not occurring after an operator or comma.

Code 9: Original

```
363 } catch(Exception e) {
364     RuntimeException re = new RuntimeException
365         ("Error processing EjbDescriptor");
366     re.initCause(e);
367     throw re;
368 }
```

Code 10: Recommended

```
363 } catch(Exception e) {
364     RuntimeException re =
365             new RuntimeException("Error processing EjbDescriptor");
366     re.initCause(e);
367     throw re;
368 }
```

Line break (lines: 400-429) There are several **for** loops in this scope and the author was not consistent with line breaking.

Code 11: Original

```
400 // Visit all injectables first.  In some cases, basic type information
401 // has to be derived from target inject method or inject field.
402 for (InjectionCapable injectable :
403         ejb.getEjbBundleDescriptor().getInjectableResources(ejb)) {
404     accept(injectable);
405 }
406 
407 for (Iterator itr = ejb.getEjbReferenceDescriptors().iterator(); itr.hasNext();) {
408     EjbReference aRef = (EjbReference) itr.next();
409     accept(aRef);
410 }
411 
412 for (Iterator it = ejb.getResourceReferenceDescriptors().iterator();
413      it.hasNext();) {
414     ResourceReferenceDescriptor next =
415             (ResourceReferenceDescriptor) it.next();
416     accept(next);
417 }
418 
419 for (Iterator it = ejb.getResourceEnvReferenceDescriptors().iterator(); it.hasNext();) {
```

```
420      ResourceEnvReferenceDescriptor next =
421              (ResourceEnvReferenceDescriptor) it.next();
422      accept(next);
423  }
424
425  for (Iterator it = ejb.getMessageDestinationReferenceDescriptors().iterator(); it.hasNext();) {
426      MessageDestinationReferencer next =
427              (MessageDestinationReferencer) it.next();
428      accept(next);
429  }
```

Code 12: Recommended

```
400  // Visit all injectables first.  In some cases, basic type information
401  // has to be derived from target inject method or inject field.
402  for (InjectionCapable injectable :
403          ejb.getEjbBundleDescriptor().getInjectableResources(ejb)) {
404
405      accept(injectable);
406  }
407
408  for (Iterator itr = ejb.getEjbReferenceDescriptors().iterator(); itr.hasNext();) {
409
410      EjbReference aRef = (EjbReference) itr.next();
411      accept(aRef);
412  }
413
414  for (Iterator it =
415          ejb.getResourceReferenceDescriptors().iterator(); it.hasNext();) {
416
417      ResourceReferenceDescriptor next =
418              (ResourceReferenceDescriptor) it.next();
419      accept(next);
420  }
421
422  for (Iterator it =
423          ejb.getResourceEnvReferenceDescriptors().iterator(); it.hasNext();) {
424
425      ResourceEnvReferenceDescriptor next =
426              (ResourceEnvReferenceDescriptor) it.next();
427      accept(next);
428  }
429
430  for (Iterator it =
431          ejb.getMessageDestinationReferenceDescriptors().iterator(); it.hasNext();) {
432
433      MessageDestinationReferencer next =
434              (MessageDestinationReferencer) it.next();
435      accept(next);
436  }
```

Indentation (line: 436) Author not consistent with his style of indentation after line breaking. Elsewhere he used 8 spaces to indent the expression after a line break.

Code 13: Original

```
435  MessageDestinationReferencer msgDestReferencer =
436      (MessageDestinationReferencer) ejb;
```

Code 14: Recommended

```
435  MessageDestinationReferencer msgDestReferencer =
436          (MessageDestinationReferencer) ejb;
```

For loop style (line: 451) Author not consistent with previous styles of writing **for** loops.

Code 15: Original

```
451 for (Iterator e=persistenceDesc.getCMPFields().iterator();e.hasNext();) {
```

Code 16: Recommended

```
451 for (Iterator e = persistenceDesc.getCMPFields().iterator(); e.hasNext();) {
```

3.3 Issues in the *validateConcurrencyMetadata* method

Not commented There are no comments describing what the method does. Its name is to a point self-explanatory but can't be understood correctly without reading the code. This issue was considered because there are also other `private` methods, in the same class, whose names are self-explanatory but they have comments.

Line break(line: 475) and spacing style (lines: 464, 471, 474, 482, 484) Line break occurrence before an operator. There is no space left between **for** and (, or **if** and (, where in all previous cases the author was consistent with that style.

Code 17: Original

```
464 if( ejb instanceof EjbSessionDescriptor ) {
465
466     EjbSessionDescriptor sessionDesc = (EjbSessionDescriptor) ejb;
467
468     List<EjbSessionDescriptor.AccessTimeoutHolder> accessTimeoutInfo =
469             sessionDesc.getAccessTimeoutInfo();
470
471     for(EjbSessionDescriptor.AccessTimeoutHolder accessTimeoutHolder : accessTimeoutInfo) {
472         MethodDescriptor accessTimeoutMethodDesc = accessTimeoutHolder.method;
473         Method accessTimeoutMethod = accessTimeoutMethodDesc.getMethod(ejb);
474         if(accessTimeoutMethod == null) {
475             throw new RuntimeException("Invalid AccessTimeout method signature "
476                     + accessTimeoutMethodDesc +
477                     " . Method could not be resolved to a bean class method for bean " +
478                     ejb.getName());
479         }
480     }
481
482     for(MethodDescriptor lockMethodDesc : sessionDesc.getReadAndWriteLockMethods()) {
483         Method readLockMethod = lockMethodDesc.getMethod(sessionDesc);
484         if( readLockMethod == null ) {
```

Code 18: Recommended

```
464 if ( ejb instanceof EjbSessionDescriptor ) {
465
466     EjbSessionDescriptor sessionDesc = (EjbSessionDescriptor) ejb;
467
468     List<EjbSessionDescriptor.AccessTimeoutHolder> accessTimeoutInfo =
469             sessionDesc.getAccessTimeoutInfo();
470
471     for (EjbSessionDescriptor.AccessTimeoutHolder accessTimeoutHolder : accessTimeoutInfo) {
472         MethodDescriptor accessTimeoutMethodDesc = accessTimeoutHolder.method;
473         Method accessTimeoutMethod = accessTimeoutMethodDesc.getMethod(ejb);
```

```
474         if (accessTimeoutMethod == null) {
475             throw new RuntimeException("Invalid AccessTimeout method signature " +
476                     accessTimeoutMethodDesc +
477                     " . Method could not be resolved to a bean class method for bean " +
478                     ejb.getName());
479         }
480     }
481 
482     for (MethodDescriptor lockMethodDesc : sessionDesc.getReadAndWriteLockMethods()) {
483         Method readLockMethod = lockMethodDesc.getMethod(sessionDesc);
484         if ( readLockMethod == null ) {
```

3.4 Issues in the *validatePassivationConfiguration* method

This method consists of only 12 lines of code. The only possible issue against the checklist that can be noticed is on **line 527** where the line non necessarily exceeds 80 characters and might not be well readable in some text editors or IDEs. Code listings 19 and 20 shows the original code and the recommended version.

Code 19: Original

```
514 /**
515  * Check when passivation-capable of sfsb is false, PrePassivate and PostActivate configurations
516  * are not recommended.
517  */
518 private void validatePassivationConfiguration(EjbDescriptor ejb) {
519     if (ejb instanceof EjbSessionDescriptor) {
520         EjbSessionDescriptor sessionDesc = (EjbSessionDescriptor) ejb;
521         if (!sessionDesc.isStateful() || sessionDesc.isPassivationCapable()) {
522             return;
523         }
524 
525         String callbackInfo = getAllPrePassivatePostActivateCallbackInfo(sessionDesc);
526         if (callbackInfo.length() > 0) {
527             _logger.log(Level.WARNING, REDUNDANT_PASSIVATION_CALLBACK_METADATA, new Object...
528         }
529     }
530 }
```

Code 20: Recommended

```
514 /**
515  * Check when passivation-capable of sfsb is false, PrePassivate and PostActivate configurations
516  * are not recommended.
517  */
518 private void validatePassivationConfiguration(EjbDescriptor ejb) {
519     if (ejb instanceof EjbSessionDescriptor) {
520         EjbSessionDescriptor sessionDesc = (EjbSessionDescriptor) ejb;
521         if (!sessionDesc.isStateful() || sessionDesc.isPassivationCapable()) {
522             return;
523         }
524 
525         String callbackInfo = getAllPrePassivatePostActivateCallbackInfo(sessionDesc);
526         if (callbackInfo.length() > 0) {
527             _logger.log(Level.WARNING, REDUNDANT_PASSIVATION_CALLBACK_METADATA,
528                     new Object[]{ejb.getName(), callbackInfo});
529         }
530     }
531 }
```

4 Appendix

4.1 Hours of work

Mite Ristovski: ≈16h.

Dushica Stojkoska: ≈12h.

4.2 Tools

TeXnicCenter(LaTeX): For writing this file.

StarUML: For building UML from the source code.

`http://grepcode.com`**:** For exploring the source code more easily.