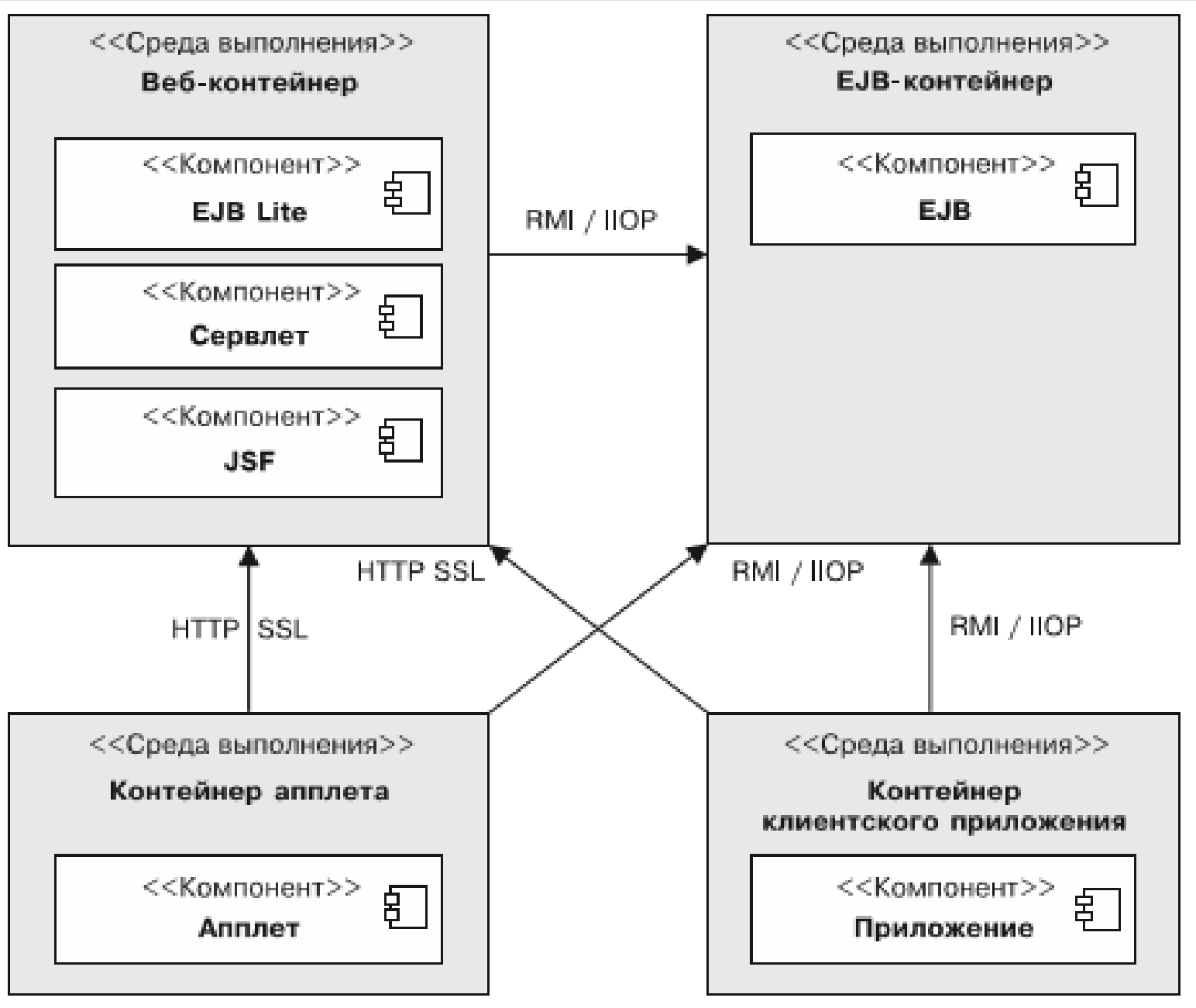
Java-2

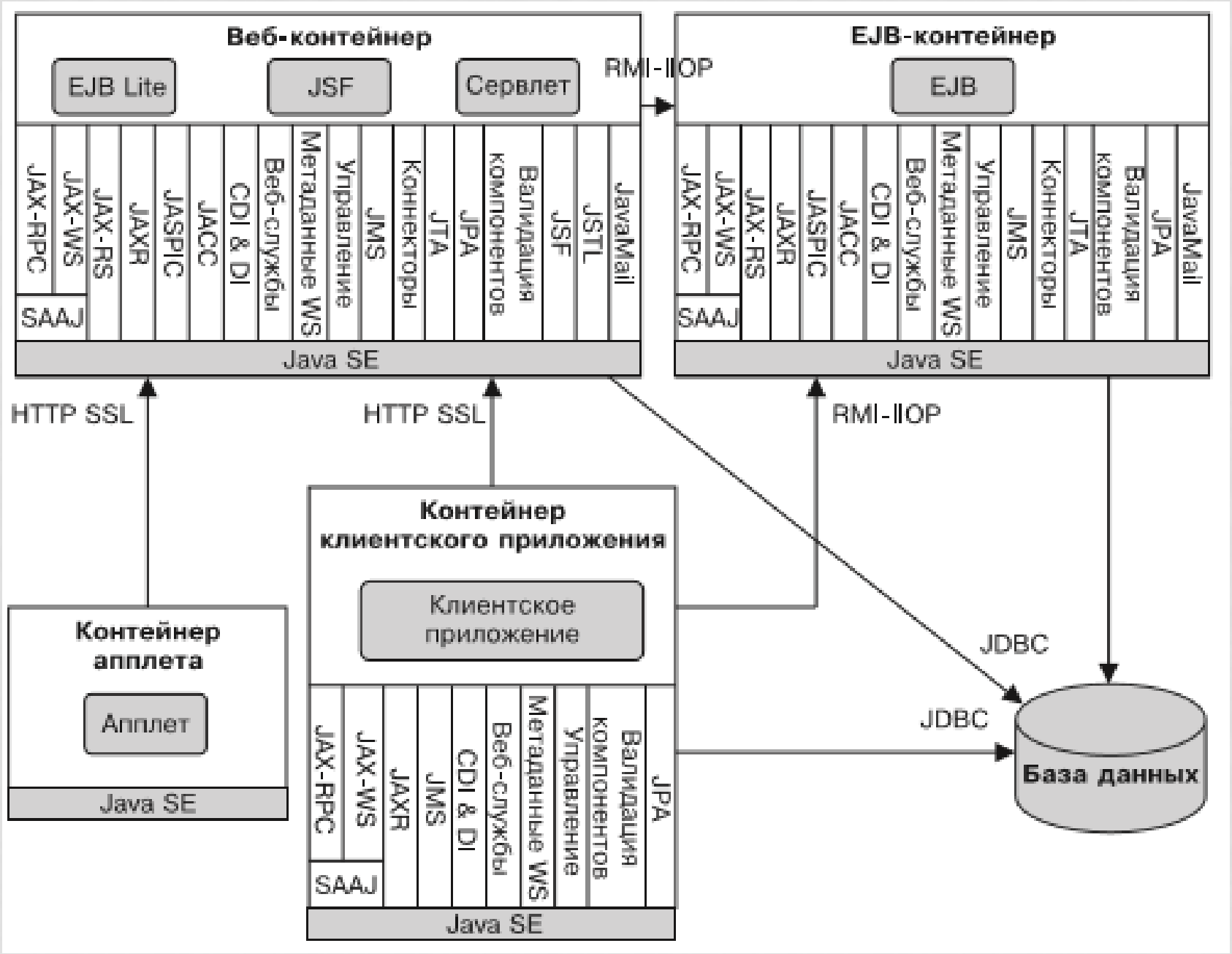
Java EE

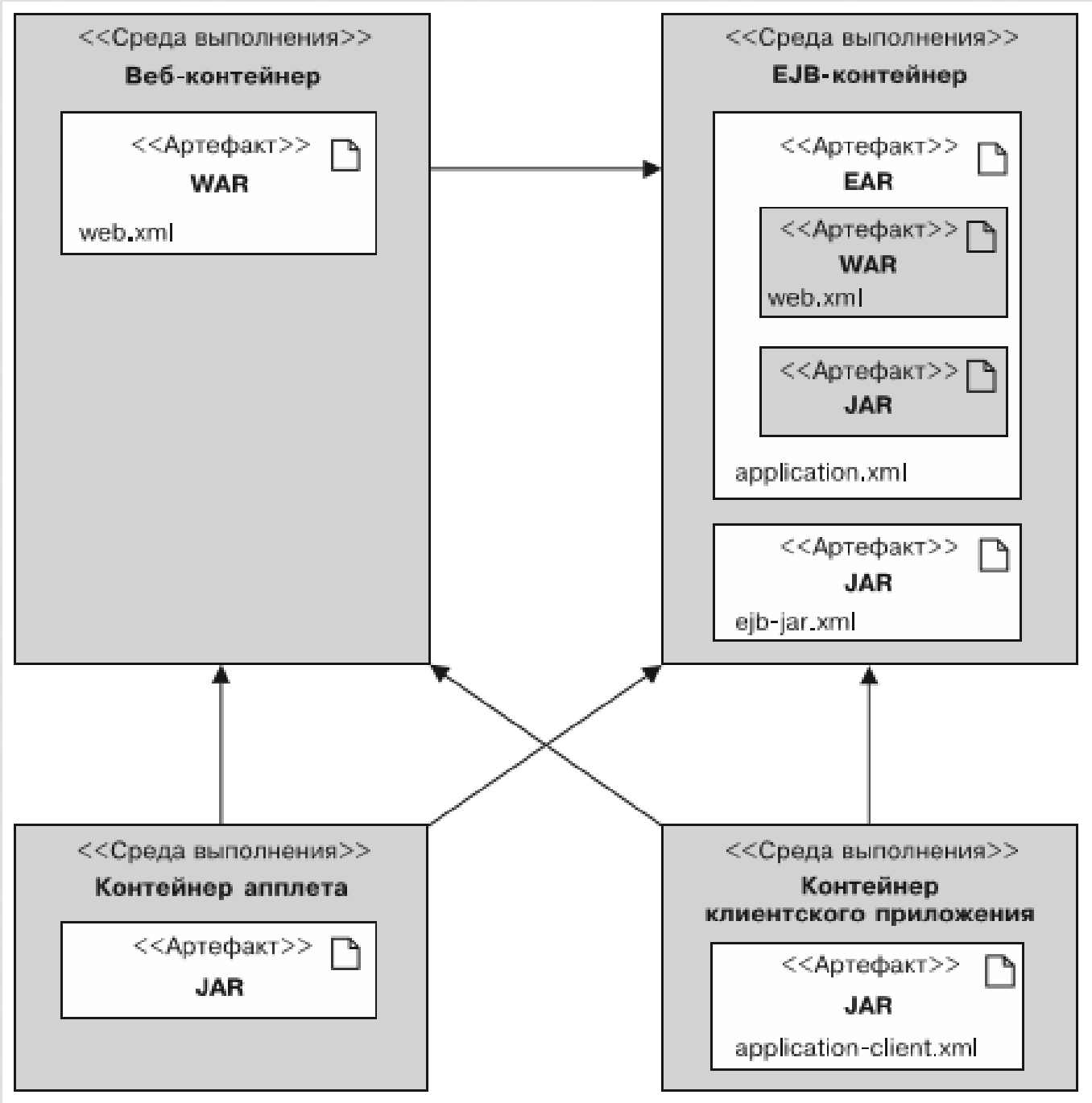
- Java EE – Java Enterprise Edition.

Набор спецификаций, реализуемых различными контейнерами.

- Контейнер – средство среды выполнения JEE, предоставляющее размещенными в ней компонентам определенные службы (управление жизненным циклом разработки, внедрение зависимостей, параллельный доступ…)


<<Среда выполнения>>
Веб-контейнер
<<Компонент>>
EJB Lite
<<Компонент>>
Сервлет
<<Компонент>>
JSF
<<Среда выполнения>>
EJB-контейнер
<<Компонент>>
EJB
RMI / IIOP
HTTP SSL
HTTP SSL
RMI / IIOP
RMI / IIOP
<<Среда выполнения>>
Контейнер апплета
<<Компонент>>
Апплет
<<Среда выполнения>>
Контейнер клиентского приложения
<<Компонент>>
Приложение


Веб-контейнер
EJB Lite
JSF
Сервлет
RMI-IIOP
EJB-контейнер
EJB
JAX-RPC
JAX-WS
JAX-RS
JAXR
JASPIC
JACC
CDI & DI
Веб-службы
Метаданные WS
Управление
JMS
Коннекторы
JTA
JPA
Валидация компонентов
JSF
JSTL
JavaMail
SAAJ
Java SE
HTTP SSL
HTTP SSL
RMI-IIOP
Контейнер клиентского приложения
Клиентское приложение
Контейнер апплета
Апплет
JDBC
JDBC
База данных


<<Среда выполнения>>
Веб-контейнер
<<Артефакт>>
WAR
web.xml
<<Среда выполнения>>
EJB-контейнер
<<Артефакт>>
EAR
<<Артефакт>>
WAR
web.xml
<<Артефакт>>
JAR
application.xml
<<Артефакт>>
JAR
ejb-jar.xml
<<Среда выполнения>>
Контейнер апплета
<<Артефакт>>
JAR
<<Среда выполнения>>
Контейнер клиентского приложения
<<Артефакт>>
JAR
application-client.xml

```java
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {
  @PersistenceContext(unitName = "chapter01PU")
  private EntityManager em;
  public Book findBookById(Long id) {
    return em.find(Book.class, id);
  }
}
```

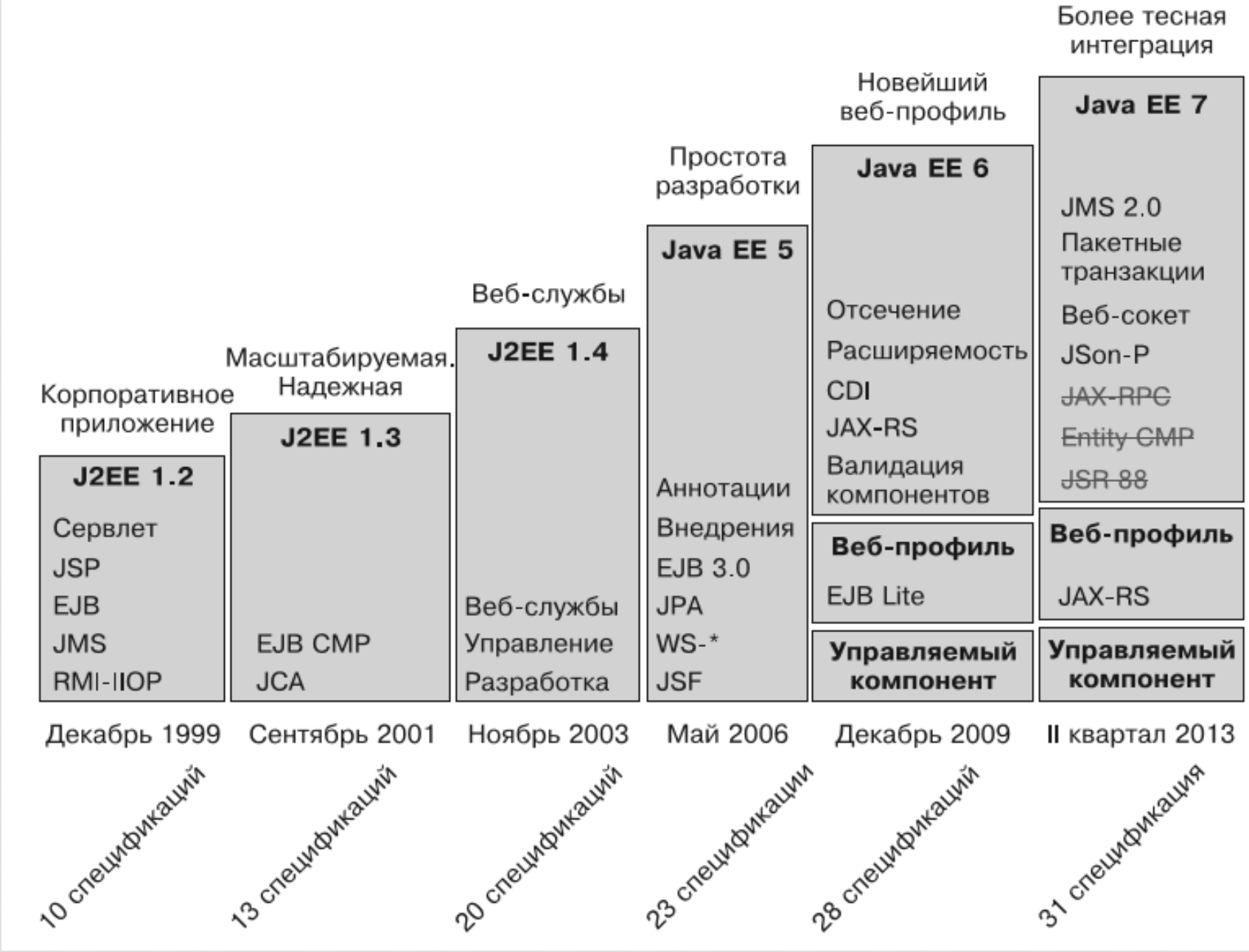
```xml
<ejb-jar xmlns="http://xmlns.jcp.org/xml/ns/javaee" ➥
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ➥
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee ➥
                     http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd" ➥
         version="3.2" >

  <enterprise-beans>
    <session>
      <ejb-name>ItemEJB</ejb-name>
      <remote>org.agoncal.book.javaee7.ItemRemote</remote>
      <local>org.agoncal.book.javaee7.ItemLocal</local>
      <local-bean/>
      <ejb-class>org.agoncal.book.javaee7.ItemEJB</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

JCP

- JCP – Java Communication Process, организация созданная Sun в 1998.
- Одно из направлений – определение будущих версий и функционала.

- Запрос на спецификацию (JSR) -> спецификация -> базовая реализация + TCK (набор тестов для сторонних реализаций)


Более тесная интеграция
Новейший веб-профиль
Простота разработки
Веб-службы
Масштабируемая. Надежная
Корпоративное приложение
Java EE 7
JMS 2.0
Пакетные транзакции
Веб-сокет
JSon-P
~~JAX-RPC~~
~~Entity CMP~~
~~JSR 88~~
Java EE 6
Отсечение
Расширяемость
CDI
JAX-RS
Валидация компонентов
Java EE 5
Аннотации
Внедрения
EJB 3.0
JPA
WS-*
JSF
J2EE 1.4
Веб-службы
Управление
Разработка
J2EE 1.3
EJB CMP
JCA
J2EE 1.2
Сервлет
JSP
EJB
JMS
RMI-IIOP
Веб-профиль
JAX-RS
Веб-профиль
EJB Lite
Управляемый компонент
Управляемый компонент
Декабрь 1999
Сентябрь 2001
Ноябрь 2003
Май 2006
Декабрь 2009
II квартал 2013
10 спецификаций
13 спецификаций
20 спецификаций
23 спецификации
28 спецификаций
31 спецификация

DI & CDI

Внедрение зависимостей

- Dependecy Injection – шаблон разработки для разделения зависимых компонентов
- Это называется «инверсия управления» - контейнер обеспечивает управление бизнес-кодом, а не наоборот.
- Объект не ищет другие объекты – контейнер внедряет зависимые сущности без вашего участия.
- Принцип Голливуда: «Не звоните нам, мы сами вам позвоним»

Как мы работаем обычно?

- POJO – Plain Old Java Object

- Пишем new для создания экземпляра
- Что-то делаем
- Ждем пока сборщик мусора освободит память

CDI

- Для запуска CDI внутри контейнера нельзя использовать new
- Контейнер сам создает, передает нам ссылку после создания объекта и перед его уничтожением

- Текущая версия 1.1
- Базовая реализация Weld (JBoss)

CDI

- Для запуска CD
  использовать n
- Контейнер сам
  создания объе

Область видимости и контекст

- Компоненты CDI – сохраняют состояние и являются контекстуальными.
- Существуют в пределах области видимости – запроса, сеанса, приложения или диалога

Дескриптор развертывания

- Почти все спецификации JEE содержат опциональный дескриптор развертывания – XML-файл, который описывает, как модуль/компонент/приложение должны быть сконфигурированы.
- Для CDI – beans.xml
- Во время развертывания CDI проверяет все WAR и JAR файлы на наличие beans.xml. Исходя из них преобразует объекты POJO в объекты CDI.

Компоненты CDI

- Компонент – любой класс, если
  - Не относится в не статическим внутренним классам
  - Это конкретный класс или класс с аннотацией @Decorator
  - Имеет конструктор по умолчанию без параметров или объявляет конструктор с аннотацией @Inject

```
public class BookService {
@Inject
private NumberGenerator numberGenerator;
  @Inject
  private EntityManager em;
  private Date instanciationDate;
  @PostConstruct
  private void initDate() {
    instanciationDate = new Date();
  }
  @Transactional
  public Book createBook(String title, Float price, String description) {
    Book book = new Book(title, price, description);
    book.setIsbn(numberGenerator.generateNumber());
    book.setInstanciationDate(instanciationDate);
    em.persist(book);
    return book;
  }
}
```

```
public class IsbnGenerator implements NumberGenerator {
  public String generateNumber() {
    return "13-84356-" + Math.abs(new Random().nextInt());
  }
}
```

Точки внедрения

Существует три варианта внедрения зависимости:

```
@Inject
private NumberGenerator numberGenerator;
```

```
@Inject
public BookService (NumberGenerator numberGenerator) {
  this.numberGenerator = numberGenerator;
}
```

```
@Inject
public void setNumberGenerator(NumberGenerator numberGenerator) {
  this.numberGenerator = numberGenerator;
}
```

Внедрение по умолчанию

- @Default – это встроенный квалификатор, сообщающий CDI, когда нужно внедрить реализацию компонента по умолчанию. Если вы определите компонент без квалификатора, ему автоматически присвоится квалификатор @Default

```
@Inject @Default
private NumberGenerator numberGenerator;
```

```
@Default
public class IsbnGenerator implements NumberGenerator {
  public String generateNumber() {
    return "13-84356-" + Math.abs(new Random().nextInt());
  }
}
```

Квалификаторы

Квалификаторы

```
@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD})
public @interface ThirteenDigits { }
```

```
@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD})
public @interface EightDigits { }
```

```
@ThirteenDigits
public class IsbnGenerator implements NumberGenerator {
  public String generateNumber() {
    return "13-84356-" + Math.abs(new Random().nextInt());
  }
}
```

```
@EightDigits
public class IssnGenerator implements NumberGenerator {
  public String generateNumber() {
    return "8-" + Math.abs(new Random().nextInt());
  }
}
```

Квалификаторы

- Чтобы это работало не нужна внешняя конфигурация.
- Поэтому говорят, что **CDI использует строгую типизацию**. Можно как угодно переименовать ваши реализации или квалификатор — точка внедрения не изменится (так называемая слабая связанность).

```
public class LegacyBookService {
  @Inject @EightDigits
  private NumberGenerator numberGenerator;
  public Book createBook(String title, Float price, String description) {
    Book book = new Book(title, price, description);
    book.setIsbn(numberGenerator.generateNumber());
    return book;
  }
}
```

Квалификаторы с членами

```
@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD})
public @interface NumberOfDigits {
  Digits value();
  boolean odd();
}
public enum Digits {
  TWO,
  EIGHT,
  TEN,
  THIRTEEN
}
```

```
@Inject @NumberOfDigits(value = Digits.THIRTEEN, odd = false)
private NumberGenerator numberGenerator;
```

```
@NumberOfDigits(value = Digits.THIRTEEN, odd = false)
public class IsbnEvenGenerator implements NumberGenerator {...}
```

Множественные квалификаторы

```
@ThirteenDigits @Even
public class IsbnEvenGenerator implements NumberGenerator {...}
```

```
@Inject @ThirteenDigits @Even
private NumberGenerator numberGenerator;
```

Альтернативы

- Квалификаторы позволяют выбирать между множественными реализациями интерфейса во время развертывания.
- Иногда бывает целесообразно внедрить реализацию, зависящую от конкретного сценария развертывания.

- Альтернативы — это компоненты, аннотированные специальным квалификатором javax.enterprise.inject.Alternative.
- По умолчанию альтернативы отключены, и чтобы сделать их доступными для инстанцирования и внедрения, необходимо активизировать их в дескрипторе beans.xml.

```
@Alternative
public class MockGenerator implements NumberGenerator {
  public String generateNumber() {
    return "MOCK";
  }
}
@Alternative @Default
public class MockGenerator implements NumberGenerator {...}

@Alternative @ThirteenDigits
public class MockGenerator implements NumberGenerator {...}
```

```
<beans xmlns=" http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee ➥
                           http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
       version="1.1" bean-discovery-mode="all">
  <alternatives>
    <class>org.agoncal.book.javaee7.chapter02.MockGenerator</class>
  </alternatives>
</beans>
```

Производители данных

Производители данных

- По умолчанию нельзя внедрять такие классы, как java.util.Date или java.lang.String.
- Так происходит потому, что все эти классы упакованы в файл rt.jar (классы среды исполнения Java), а этот архив не содержит дескриптор развертывания beans.xml.
- Единственный способ внедрения POJO состоит в использовании полей и методов производителей данных

Производители данных

- Метод производителя данных random() — это метод, выступающий в качестве фабрики экземпляров компонентов. Он позволяет внедряться возвращаемому значению
- Поле производителя данных (prefix13digits и editorNumber) — более простая альтернатива методу производителя данных. Это свойство, которое становится внедряемым.

```
public class NumberProducer {
  @Produces @ThirteenDigits
  private String prefix13digits = "13-";
  @Produces @ThirteenDigits
  private int editorNumber = 84356;
  @Produces @Random
  public double random() {
    return Math.abs(new Random().nextInt());
  }
}
```

```
@ThirteenDigits
public class IsbnGenerator implements NumberGenerator {
  @Inject @ThirteenDigits
  private String prefix;
  @Inject @ThirteenDigits
  private int editorNumber;
  @Inject @Random
  private double postfix;
  public String generateNumber() {
    returnprefix + editorNumber + postfix;
  }
}
```

InjectionPoint API

- Как получить объект, которому что-то необходимо знать о точке внедрения (имя класса, например)?

```
public class LoggingProducer {
  @Produces
  private Logger createLogger(InjectionPoint injectionPoint) {
    return Logger.getLogger(injectionPoint.getMember().getDeclaringClass().
getName());
  }
}
```

```
@Inject Logger log;
```

Утилизаторы

- Каждый метод утилизатора (с аннотацией @Dispose) должен иметь строго один параметр такого же типа и квалификаторы, как соответствующий тип возврата метода производителя данных (с аннотацией @Produces)
- Метод утилизатора вызывается автоматически по окончании контекста клиента и параметр получает объект, порожденный производителем данных

```
public class JDBCConnectionProducer {
  @Produces
  private Connection createConnection() {
    Connection conn = null;
    try {
      Class.forName("org.apache.derby.jdbc.EmbeddedDriver").newInstance();
      conn = DriverManager.getConnection("jdbc:derby:memory", ➥
"APP", "APP");
    } catch (InstantiationException | IllegalAccessException |
ClassNotFoundException) {
      e.printStackTrace();
    }
    return conn;
  }
  private void closeConnection(@Disposes Connection conn) throws SQLException
{
    conn.close();
  }
}
```

Области видимости

Области видимости

В CDI есть 5 встроенных областей видимости:

- ***Область видимости приложения* (@ApplicationScoped)** — действует на протяжении всей работы приложения.
- ***Область видимости сеанса* (@SessionScoped)** — действует на протяжении нескольких запросов HTTP или нескольких вызовов метода для одного пользовательского сеанса.
- ***Область видимости запроса* (@RequestScoped)** — соответствует единственному HTTP-запросу или вызову метода.

Области видимости

- ***Область видимости диалога* (@ConverationScoped)** — действительна между множественными вызовами в рамках одной сессии, ее начальная и конечная точка определяются приложением. Диалоги используются среди множественных страниц как часть многоступенчатого рабочего потока.
- ***Зависимая псевдообласть видимости* (@Dependent)** — ее жизненный цикл совпадает с жизненным циклом клиента. Зависимый компонент создается каждый раз при внедрении, а ссылка удаляется одновременно с удалением целевой точки внедрения. Эта область видимости по умолчанию предназначена для CDI.

Перехватчики

Перехватчики

- Перехватчики позволяют добавлять к вашим компонентам сквозную функциональность.
- Когда клиент вызывает метод на управляемом компоненте (а значит, и на компоненте CDI, EJB либо веб-службе RESTful и т. д.) контейнер может перехватить вызов и обработать бизнес-логику перед тем, как будет вызван метод компонента

Перехватчики

Перехватчики делятся на четыре типа:

- ..*перехватчики, действующие на уровне конструктора,* — перехватчик, ассоциированный с конструктором целевого класса (@AroundConstruct);
- ..*перехватчики, действующие на уровне метода,* — перехватчик, ассоциированный со специальным бизнес-методом (@AroundInvoke);
- ..*перехватчики методов задержки* — перехватчик, помеченный аннотацией @AroundTimeout, вмешивается в работу методов задержки;
- ..*перехватчики обратного вызова жизненного цикла* — перехватчик, который вмешивается в работу обратных вызовов событий жизненного цикла целевого экземпляра (@PostConstruct и @PreDestroy).

```
@Transactional
public class CustomerService {
  @Inject
  private EntityManager em;
  @Inject
  private Logger logger;
  public void createCustomer(Customer customer) {
    em.persist(customer);
  }
  public Customer findCustomerById(Long id) {
    return em.find(Customer.class, id);
  }
  @AroundInvoke
  private Object logMethod(InvocationContext ic) throws Exception {
    logger.entering(ic.getTarget().toString(), ic.getMethod().getName());
    try {
      return ic.proceed();
    } finally {
      logger.exiting(ic.getTarget().toString(), ic.getMethod().getName());
    }
  }
}
```

Класс-перехватчик

```
public class LoggingInterceptor {
  @Inject
  private Logger logger;
  @AroundConstruct
  private void init(InvocationContext ic) throws Exception {
    logger.fine("Entering constructor");
    try {
      ic.proceed();
    } finally {
      logger.fine("Exiting constructor");
    }
  }
  @AroundInvoke
  public Object logMethod(InvocationContext ic) throws Exception {
    logger.entering(ic.getTarget().toString(), ic.getMethod().getName());
    try {
      return ic.proceed();
    } finally {
      logger.exiting(ic.getTarget().toString(), ic.getMethod().getName());
    }
  }
}
```

Класс-перехватчик

```
@Transactional
public class CustomerService {
  @Inject
  private EntityManager em;
  @Interceptors(LoggingInterceptor.class)
  public void createCustomer(Customer customer) {
    em.persist(customer);
  }
  public Customer findCustomerById(Long id) {
    return em.find(Customer.class, id);
  }
}
```

```
@Transactional
@Interceptors(LoggingInterceptor.class)
public class CustomerService {
  public void createCustomer(Customer customer) {...}
  public Customer findCustomerById(Long id) {...}
  @ExcludeClassInterceptors
  public Customer updateCustomer(Customer customer) { ... }
}
```

Связывание с перехватчиком

```
@InterceptorBinding
@Target({METHOD, TYPE})
@Retention(RUNTIME)
public @interface Loggable { }
```

```
@Transactional
@Loggable
public class CustomerService {
  @Inject
  private EntityManager em;
  public void createCustomer(Customer customer) {
    em.persist(customer);
  }
  public Customer findCustomerById(Long id) {
    return em.find(Customer.class, id);
  }
}
```

```
@Interceptor
@Loggable
public class LoggingInterceptor {
  @Inject
  private Logger logger;
  @AroundInvoke
  public Object logMethod(InvocationContext ic) throws Exception {
    logger.entering(ic .getTarget().toString(), ic .getMethod().getName());
    try {
      return ic.proceed() ;
    } finally {
      logger.exiting(ic .getTarget().toString(), ic .getMethod().getName());
    }
  }
}
```

Перехватчики

- Перехватчики специфичны для развертывания и отключены по умолчанию. Как и альтернативы, перехватчики необходимо активизировать, используя дескриптор развертывания beans.xml

```
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee ➥
                           http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
       version="1.1" bean-discovery-mode="all">
  <interceptors>
    <class>javaee7.LoggingInterceptor</class>
  </interceptors>
</beans>
```

Наблюдатель

```
public class BookService {
  @Inject
  private NumberGenerator numberGenerator;
  @Inject @Added
  private Event<Book> bookAddedEvent;
  @Inject @Removed
  private Event<Book> bookRemovedEvent;
  public Book createBook(String title, Float price, String description) {
    Book book = new Book(title, price, description);
    book.setIsbn(numberGenerator.generateNumber());
    bookAddedEvent.fire(book);
    return book;
  }
  public void deleteBook(Book book) {
    bookRemovedEvent.fire(book);
  }
}
```

```
public class InventoryService {
  @Inject
  private Logger logger;
  List<Book> inventory = new ArrayList<>();
  public void addBook(@Observes @Added Book book) {
    logger.warning("Книга " + book.getTitle() + " добавлена в список");
    inventory.add(book);
  }
  public void removeBook(@Observes @Removed Book book) {
    logger.warning("Книга " + book.getTitle() + " удалена из списка");
    inventory.remove(book);
  }
}
```

Декораторы

Декораторы

- Декораторы должны иметь точку внедрения делегата (аннотированную @Delegate) такого же типа, как и компоненты, которые они декорируют (здесь интерфейс NumberGenerator). Это позволяет объекту вызывать объект-делегат (например, целевой компонент IssnNumberGenerator), а затем, в свою очередь, вызывать на него любой бизнес-метод

```
@Decorator
public class FromEightToThirteenDigitsDecorator implements NumberGenerator {
  @Inject @Delegate
  private NumberGenerator numberGenerator;
  public String generateNumber() {
    String issn = numberGenerator.generateNumber();
    String isbn = "13-84356" + issn.substring(1);
    returnisbn;
  }
}
```

Включение декораторов

- По умолчанию все декораторы отключены, как и альтернативы с перехватчиками. Декораторы необходимо активизировать в файле beans.xml

```
<beansxmlns="http://xmlns.jcp.org/xml/ns/javaee"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee ➥
                               http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
            version="1.1" bean-discovery-mode="all">
  <decorators>
    <class> javaee7.chapter02.FromEightToThirteenDigitsDeco-
rator</class>
  </decorators>
</beans>
```