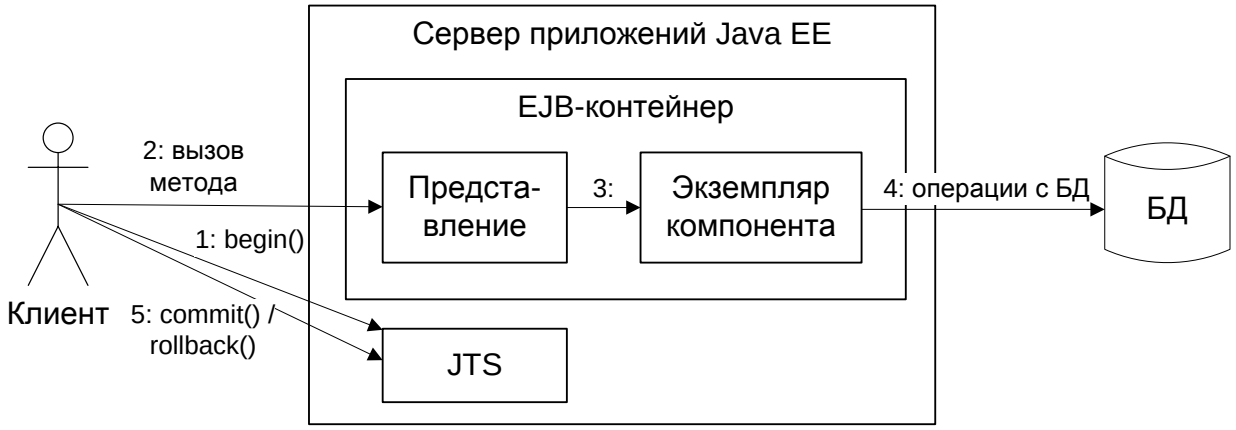
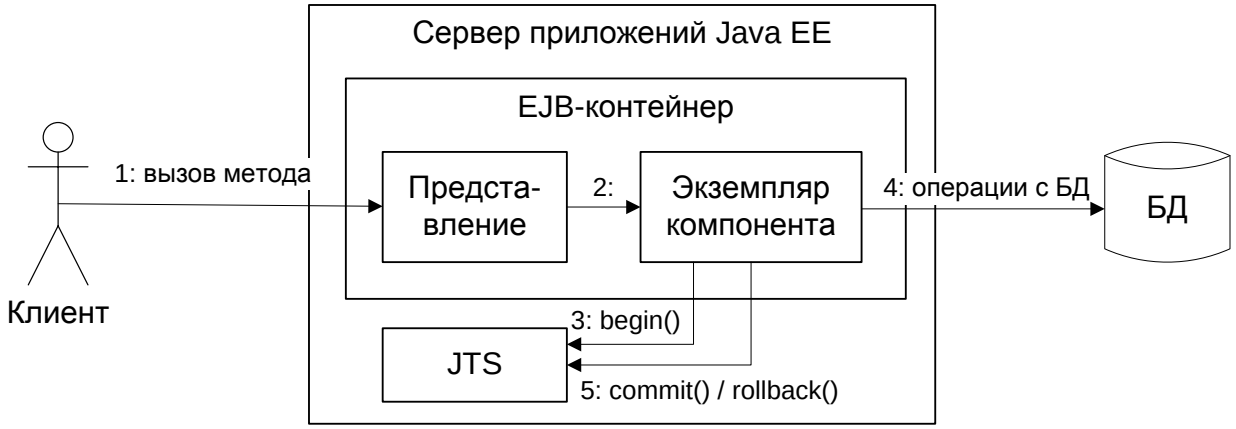
Виды транзакций на платформе Java EE

На платформе Java EE существует 3 вида транзакций:

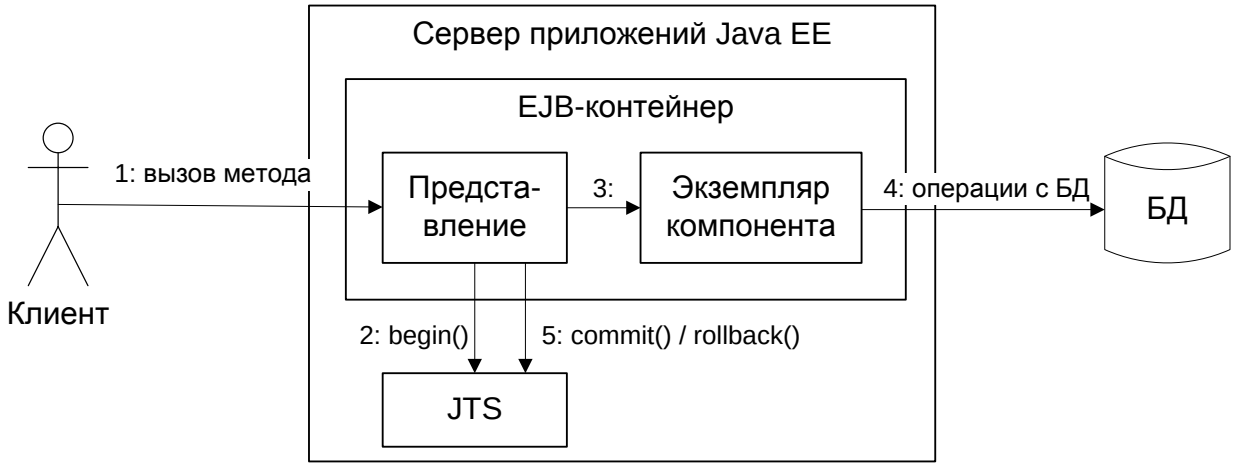
1. инициируемые клиентом (client initiated) – обращение к службе управления транзакциями выполняется еще до того, как клиент делает вызов метода компонента. Последовательность действий в этом случае приведена на рис. 1.



2. управляемые программно (programmatic) – транзакции инициируются экземпляром компонента в рамках выполнения его бизнес-логики. Также данный вид транзакций называется bean managed transactions (транзакции, управляемые компонентом). Последовательность действий приведена на рис. 2.



3. управляемые декларативно (declarative) – транзакция инициируется представлением EJB–компонента. Также данный вид транзакций называется container managed transactions (транзакции, управляемые контейнером). Последовательность действий приведена на рис. 3.



Выбор вида управления транзакциями для сессионного компонента осуществляется при помощи аннотации `@TransactionManagement` либо через XML-дескриптор. В аннотации указывается значение `BEAN` либо `CONTAINER`, последнее принято по умолчанию.

Декларативное управление транзакциями

Если транзакциями управляет контейнер, то разработчик компонента не может непосредственно управлять началом и завершением транзакции, но может повлиять на этот процесс с помощью транзакционного атрибута (transaction attribute). Для каждого метода компонента можно установить свое значение транзакционного атрибута с помощью аннотации `@TransactionAttribute` либо через XML-дескриптор. По умолчанию используется значение `REQUIRED`.

Всего для транзакционного атрибута предусмотрено 6 различных значений. Значение транзакционного атрибута определяет, выполняется ли вызванный клиентом бизнес-метод компонента в контексте какой-либо транзакции или нет, и если да, то в контексте какой транзакции: начатой автоматически при вызове этого метода или той, в рамках которой клиент выполнил вызов.

Поведение контейнера при вызова бизнес-метода компонента в зависимости от 1) значения транзакционного атрибута и 2) наличия связанной с клиентом транзакции определено в табл. 1.

Значение транзакционного атрибута	Клиентская транзакция	Транзакция, связанная с методом компонента
NOT_SUPPORTED	Нет	нет
	T1	нет
REQUIRED	Нет	T1
	T1	T1
SUPPORTS	Нет	нет
	T1	T1
REQUIRES_NEW	Нет	T1
	T1	T2
MANDATORY	Нет	ошибка
	T1	T1
NEVER	Нет	нет
	T1	ошибка

Атрибут NOT_SUPPORTED указывает, что бизнес-метод всегда будет выполняться вне контекста какой-либо транзакции вне зависимости от того, была ли начата транзакция клиентом или нет.

Атрибут REQUIRED указывает, что бизнес-метод компонента всегда будет выполняться в контексте транзакции: клиентской (если транзакция была начата клиентом), либо автоматически запущенной новой транзакции (если клиент сделал вызов вне контекста какой-либо транзакции).

Атрибут SUPPORTS указывает, что метод выполняется либо в контексте клиентской транзакции (если она была начата клиентом), либо вне контекста транзакции (если клиент сделал вызов вне контекста транзакции).

Атрибут REQUIRES_NEW означает, что метод компонента всегда будет выполняться в контексте автоматически запущенной новой транзакции независимо от того, был ли сделан вызов в контексте какой-либо транзакции или нет.

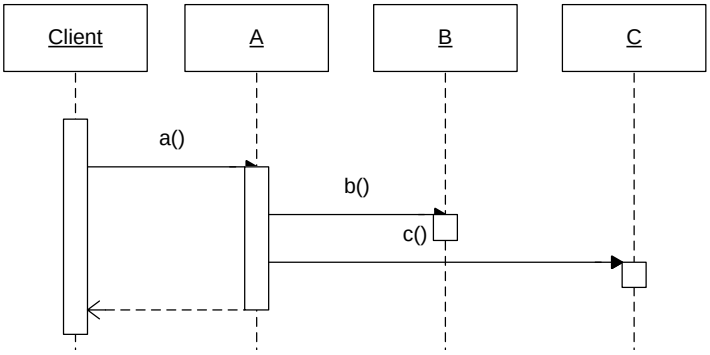
Атрибут MANDATORY указывает, что вызов метода компонента клиент должен выполнять обязательно в контексте транзакции, в противном случае возникает исключение `javax.transaction.TransactionRequiredException`.
Атрибут NEVER указывает, что вызов метода компонента клиент должен выполнять обязательно вне контекста какой-либо транзакции, в противном случае выбрасывается одно из 2-х исключений:

- `java.rmi.RemoteException` (если вызов был сделан через удаленное представление)
- `javax.ejb.EJBException` (если вызов был сделан через локальное представление).

Возможные области применения значений транзакционного атрибута:
REQUIRES_NEW – изменения в источнике данных должны быть произведены независимо от других действий с источниками данных в рамках цепочки вызовов. Это может использоваться для записи в журнал действий пользователя или аналогичных операций.
NOT_SUPPORTED – аналогично REQUIRES_NEW, но при этом контейнер не обеспечивает выполнение транзакционных свойств для этих изменений. Если данные только добавляются, то конфликты при подобных действиях с БД не возникают, и нет необходимости нести накладные расходы, связанные с началом и завершением транзакции.
SUPPORTS – используется в первую очередь для методов, не выполняющих операции с источником данных (например, выполняющих в сессионных компонентах с состоянием вычисления с полями экземпляра компонента). В этом случае отсутствуют накладные расходы, вызываемые как приостановлением транзакции в режиме NOT_SUPPORTED, так и ее началом в режиме REQUIRED.
Режимы NOT_SUPPORTED, SUPPORTS и NEVER используются также для работы с источниками данных, не поддерживающими транзакции.
Если транзакция была автоматически создана при вызове метода, то при успешном возврате из метода она будет подтверждена. При возникновении исключения поведение контейнера зависит от типа исключения, что будет подробно рассмотрено далее. Если исключение вызвано сбоем в работе СУБД или самого контейнера, то происходит откат транзакции.
Экземпляр компонента может пометить транзакцию к откату с помощью метода `EJBContext.setRollbackOnly()`. Но если бизнес-метод помечен транзакционным атрибутом со значением SUPPORTS, NOT_SUPPORTED или NEVER, то при вызове методов `getRollbackOnly()` и `setRollbackOnly()` интерфейса `EJBContext` возникает исключение `java.lang.IllegalStateException`.
Класс сессионного компонента с состоянием может реализовать интерфейс `javax.ejb.SessionSynchronization`, который позволяет его экземпляру получать от контейнера дополнительные уведомления, связанные с началом и окончанием транзакции. После начала транзакции и перед вызовом бизнес-метода компонента контейнер вызывает метод `afterBegin()`. Перед подтверждением транзакции вызывается метод `beforeCompletion()`, который предоставляет компоненту последнюю возможность инициировать откат транзакции. После завершения транзакции вызывается метод `afterCompletion(boolean)`, которому в качестве параметра передается исход транзакции (`true` – фиксация, `false` – откат).
Если класс компонента реализует интерфейс `SessionSynchronization`, то для его методов допускаются только следующие значения транзакционного атрибута: REQUIRED, REQUIRES_NEW, MANDATORY.
При декларативном управлении транзакциями запрещается использовать любые API, специфичные для менеджера ресурсов, предназначенные для управления транзакциями. Так, если в качестве менеджера ресурсов используется источник данных JDBC, то это

означает, что запрещено использование методов `commit()`, `rollback()`, `setAutoCommit()` интерфейса `java.sql.Connection`.

При декларативном управлении транзакциями в сессионных компонентах, использующих JDBC для непосредственного доступа к БД, можно изменять уровень изоляции транзакции методом `setTransactionIsolation()` интерфейса `java.sql.Connection` (как и в случае транзакции, управляемой компонентом). Однако нужно быть осторожным: изменение уровня транзакции в середине транзакции может привести к неявному подтверждению или откату той части работы, которая была уже сделана. Желательно этого избегать.



Рассмотрим пример вызова из клиента метода `a()` сессионного компонента без состояния A, который поочередно вызывает методы `b()` и `c()` сессионных компонентов без состояния B и C, соответственно (рис. 4). Все указанные методы выполняют некоторые операции с источником данных. Кроме того, метод C.c() после операций с источником данных вызывает метод контекста `setRollbackOnly()`. Во всех компонентах используется декларативное управление транзакциями и транзакционные атрибуты настроены так, как указано в табл. 2. Кроме того, вызов метода `A.a()` на клиенте производится в контексте транзакции T1.

Client TX	A		B		C	
	TX Attr	`a()`	TX Attr	`b()`	TX Attr	`c()`
T1	SUPPORTS	–	NOT_SUPPORTED	–	REQUIRES_NEW	setRollbackOnly()

Бизнес-методы компонентов выполняются в контексте транзакций, приведенных в табл. 3. Так как метод `A.a()` поддерживает транзакции, то он выполняется в контексте клиентской транзакции T1. Метод `B.b()` не поддерживает транзакции, поэтому он выполняется вне контекста транзакции, и производимые им с источником данных операции не обладают транзакционными свойствами. Так как не сказано иного, то предполагается, что метод `B.b()` завершается успешно, а изменения сохраняются в источнике данных. Метод `C.c()` всегда выполняется в контексте особой транзакции, поэтому на время его выполнения транзакция T1 приостанавливается и начинается новая транзакция T2. Так как в методе `C.c()` запрашивается откат транзакции, то транзакция T2 завершается откатом и изменения, произведенные данным методом в источнике данных, не фиксируются. Судьба же изменений, сделанных методом `A.a()`, зависит от того, подтвердит или откатит транзакцию T1 клиентское приложение.

	Client	`A.a()`	`B.b()`	`C.c()`
Контекст транзакции	T1	T1	–	T2
Исход транзакции	?	Зависит от клиента	–	Откат
Изменения	?	Зависит от клиента	Сохранение	Откат

Программное управление транзакциями

Данный вид управления транзакциями может использоваться в сессионных компонентах любого типа. Для управления транзакциями используется интерфейс `javax.transaction.UserTransaction`.

Для сессионного компонента с состоянием допустима ситуация, когда бизнес-метод, начавший транзакцию, заканчивает свое выполнение без подтверждения или отката транзакции. Тогда контейнер обязан поддерживать связь между экземпляром компонента (а точнее потоком, в котором выполняются его методы) и транзакцией, чтобы последующие вызовы бизнес-методов этого экземпляра компонента выполнялись в контексте той же транзакции. Связь между экземпляром компонента и транзакцией поддерживается до тех пор, пока транзакция не завершится подтверждением или откатом.

Для сессионного компонента без состояния такая ситуация недопустима, и любая транзакция, начатая в рамках бизнес-метода компонента, должна быть завершена подтверждением или откатом до возврата из этого метода. В противном случае контейнер выбрасывает системное исключение.

Поведение контейнера при вызове бизнес-метода компонента, использующего программное управление транзакциями, определено в табл. 4. Последний столбец указывает связь бизнес-метода компонента с транзакцией сразу после начала его исполнения. Эта связь может измениться самим методом, поскольку используется программное управление транзакциями. Строки 1 и 2 применимы для любых сессионных компонентов, а строки 3 и 4 – только для компонентов с состоянием.

№ п/п	Транзакция клиента	Транзакции, связанная с экземпляром компонента	Транзакция, связанная с вызванным методом
1	Нет	Нет	Нет
2	T1	Нет	Нет
3	Нет	T2	T2
4	T1	T2	T2

Компонент может получить объект, реализующий этот интерфейс, любым из трех способов:

1) с помощью метода `getUserTransaction()` интерфейса `SessionContext`;
2) с помощью аннотации `@Resource`;
3) выполнить поиск этого объекта в окружении компонента по имени `java:comp/UserTransaction`, которое закреплено спецификацией платформы Java EE.

Рассмотрим подробнее методы интерфейса `UserTransaction`.

1. Метод `begin()` используется для начала новой транзакции, может выбросить исключения:

- `NotSupportedException` выбрасывается в случае, если с данным потоком выполнения уже связана какая-либо транзакция, а служба управления транзакциями не поддерживает вложенные транзакции.

2. Метод `commit()` используется для подтверждения транзакции и может выбрасывать исключения:

- `RollbackException` – возникает в случае, если по объективным причинам не удается подтвердить транзакцию, она вместо подтверждения закончилась откатом;
- `HeuristicMixedException` – выбрасывается в случае, если отдельными источниками данных, охваченными распределенной транзакцией, еще до ее завершения были приняты эвристические решения о подтверждении и откате их локальных транзакций так, что часть локальных транзакций завершилась откатом, а часть – подтверждением;

- `HeuristicRollbackException` выбрасывается в случае, если отдельными источниками данных, охваченных глобальной транзакцией, были приняты эвристические решения об откате их локальных транзакций еще до завершения глобальной транзакции;
- `SecurityException` выбрасывается, если данный поток выполнения не имеет права подтверждать глобальную транзакцию;
- `IllegalStateException` выбрасывается в случае, если с данным потоком выполнения не связана ни какая глобальная транзакция.

3. Метод `rollback()` используется для отката глобальной транзакции. Он может выбрасывать следующие исключения:
   - `SecurityException` выбрасывается в случае, если поток выполнения не имеет права выполнять откат транзакции;
   - `IllegalStateException` выбрасывается, если с потоком выполнения не связана никакая глобальная транзакция.

4. Метод `setRollbackOnly()` позволяет пометить транзакцию, связанную с потоком выполнения, для отката; подтверждением такая транзакция завершиться не может. Выбрасывается `IllegalStateException`, если с потоком выполнения не связана никакая транзакция.

5. Метод `setTransactionTimeout()` позволяет указать интервал времени в секундах, по истечении которого транзакция автоматически завершается. Если транзакция завершается по таймауту, то происходит откат. Если приложение не вызывает этого метода, то используется значение по умолчанию, специфичное для реализации JTS. Параметр 0 также означает, что будет использовано значение по умолчанию.

6. Метод `getStatus()` позволяет получить информацию о состоянии транзакции, связанной с данным потоком выполнения, в виде одной из констант, определенных в интерфейсе `Status`:
   - `STATUS_ACTIVE` – транзакция была начата, не помечена к откату и выполняется;
   - `STATUS_MARKED_ROLLBACK` – транзакция была начата, была помечена для отката, но все еще выполняется;
   - `STATUS_NO_TRANSACTION` – с данным потоком выполнения не связано никакой транзакции, т.е. транзакция никогда не начиналась, либо успешно завершена подтверждением или откатом, в ходе которых не было обнаружено эвристических решений;
   - `STATUS_UNKNOWN` – означает, что состояние транзакции не может быть определено, потому что оно в данный момент изменяется (переходное состояние);
   - `STATUS_PREPARING` – транзакция, связанная с потоком выполнения, подготавливается к подтверждению (то есть выполняется фаза подготовки протокола 2PC);
   - `STATUS_PREPARED` – фаза подготовки протокола 2PC завершена и ожидается начало выполнения второй фазы;
   - `STATUS_COMMITTING` – выполняется фаза подтверждения протокола 2PC;
   - `STATUS_ROLLING_BACK` – выполняется откат транзакции, связанной с потоком выполнения;
   - `STATUS_COMMITTED` – транзакция завершилась подтверждением, однако в процессе подтверждения было обнаружено, что часть интерфейсов данных приняли эвристическое решение;
   - `STATUS_ROLLEDBACK` – транзакция завершилась откатом, в процессе выполнения отката было обнаружено, что часть источников данных приняли эвристическое решение.

Если при выполнении любого из методов интерфейса `UserTransaction` происходит внутренняя ошибка JTS, то выбрасывается исключение `SystemException`.

При программном управлении транзакциями для начала и окончания (подтверждения либо отката) транзакции необходимо использовать интерфейс `UserTransaction`. При этом запрещено использование методов управления транзакциями, специфичных для менеджеров ресурсов (например, методы `commit()` и `rollback()` интерфейса `java.sql.Connection` для JDBC источников данных), т.е. начав транзакцию методом `begin()` интерфейса `UserTransaction`, подтвердить или откатить ее необходимо методом `commit()` или `rollback()` интерфейса `UserTransaction`.
При управлении транзакциями через службу JTS (в любом режиме управления транзакциями) соединение с источниками данных JDBC по умолчанию работают с выключенным режимом AUTOCOMMIT. Если режим AUTOCOMMIT включен, то выполнять подтверждение транзакции нельзя ни через интерфейс `java.sql,Connection`, ни через `javax.transaction.UserTransaction`.
Если сессионный компонент осуществляет прямой доступ к БД через интерфейсы JDBC и использует программное управление транзакциями, то он может самостоятельно управлять уровнем изоляции транзакции, применяя рассмотренные средства JDBC.

Пример программного управления транзакциями

Пример сессионного компонента с состоянием `MySessionBean`, использующего программное управление транзакциями. Для получения объекта, реализующего интерфейс `UserTransaction`, используется аннотация `@Resource`. Метод `method1()` начинает транзакцию и заканчивает свое выполнение без подтверждения или отката транзакции. После этого клиент вызывает последовательно методы `method2()` и `method3()`, которые выполняют изменения в БД и подтверждение транзакции, соответственно.

```
@Stateful
@TransactionManagement(BEAN)
public class MySessionBean implements MySession {
    @Resource javax.transaction.UserTransaction ut;
    @Resource javax.sql.DataSource database1;

    public void method1(...) {
        // начало транзакции
        ut.begin();
    }

    public void method2(...) {
        java.sql.Connection con;
        java.sql.Statement stmt;
        // открытие соединения
        con = database1.getConnection();
        // выполнение изменений в БД
        stmt = con.createStatement();
        stmt.executeUpdate(...);
        stmt.executeUpdate(...);
        // закрытие соединения
        stmt.close();
        con.close();
    }

    public void method3(...) {
        // подтверждение транзакции
        ut.commit();
    }
    ...
}
```