СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

JDBC-драйвер

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

РЕЛЭКС

Товарные знаки

РЕЛЭКС™, ЛИНТЕР® являются товарными знаками, принадлежащими АО НПП «Реляционные экспертные системы» (далее по тексту – компания РЕЛЭКС). Прочие названия и обозначения продуктов в документе являются товарными знаками их производителей, продавцов или разработчиков.

Интеллектуальная собственность

Правообладателем продуктов ЛИНТЕР® является компания РЕЛЭКС (1990-2026). Все права защищены.

Данный документ является результатом интеллектуальной деятельности, права на который принадлежат компании РЕЛЭКС.

Все материалы данного документа, а также его части/разделы могут свободно размещаться на любых сетевых ресурсах при условии указания на них источника документа и активных ссылок на сайты компании РЕЛЭКС: relex.ru и linter.ru.

При использовании любого материала из данного документа несетевым/печатным изданием обязательно указание в этом издании источника материала и ссылок на сайты компании РЕЛЭКС: relex.ru и linter.ru.

Цитирование информации из данного документа в средствах массовой информации допускается при обязательном упоминании первоисточника информации и компании РЕЛЭКС.

Любое использование в коммерческих целях информации из данного документа, включая (но не ограничиваясь этим) воспроизведение, передачу, преобразование, сохранение в системе поиска информации, перевод на другой (в том числе компьютерный) язык в какой-либо форме, какими-либо средствами, электронными, механическими, магнитными, оптическими, химическими, ручными или иными, запрещено без предварительного письменного разрешения компании РЕЛЭКС.

О документе

Материал, содержащийся в данном документе, прошел доскональную проверку, но компания РЕЛЭКС не гарантирует, что документ не содержит ошибок и пропусков, поэтому оставляет за собой право в любое время вносить в документ исправления и изменения, пересматривать и обновлять содержащуюся в нем информацию.

Контактные данные

394006, Россия, г. Воронеж, ул. Бахметьева, 2Б.

Тел./факс: (473) 2-711-711, 2-778-333.

e-mail: info@linter.ru.

Техническая поддержка

С целью повышения качества программного продукта ЛИНТЕР и предоставляемых услуг в компании РЕЛЭКС действует автоматизированная система учёта и обработки пользовательских рекламаций. Обо всех обнаруженных недостатках и ошибках в программном продукте и/или документации на него просим сообщать нам в раздел Поддержка на сайте ЛИНТЕР.

Содержание

Предисловие 2
Назначение документа 2
Для кого предназначен документ 2
Необходимые предварительные знания 2
Назначение и условия применения программы 3
Назначение драйвера 3
Условия применения 3
Характеристики JDBC-драйвера 5
Основные функции 5
Соответствие типов данных 5
Сценарии (способы) доступа к БД 5
Типы JDBC-драйверов 6
Состав прикладного интерфейса JDBC-драйвера СУБД ЛИНТЕР 8
Новые возможности JDBC 2 9
Новые возможности JDBC 3 9
Новые возможности JDBC 4 10
Новые возможности JDBC 4.1 10
Новые возможности JDBC 4.2 11
Установка и запуск драйвера 12
Состав JDBC-драйвера 12
Выбор пакета для соответствующей версии JDK 12
Установка драйвера 13
Установка JDBC-драйвера в локальный репозиторий maven 13
Запуск клиентской части драйвера 14
Запуск серверной части драйвера 14
Реализация JDBC-драйвера для СУБД ЛИНТЕР 16
java.sql.CallableStatement 16
java.sql.Connection 16
java.sql.DatabaseMetaData 20
java.sql.Driver 21
java.sql.DriverManager 21
java.sql.PreparedStatement 22
java.sql.ResultSet 24
java.sql.ResultSetMetaData 26
java.sql.Statement 27
Поддерживаемые интерфейсы соединения с СУБД 30
JNDI 30
Apache Cayenne 33
Hibernate 34
Приложение 1. Пример java-модуля 37
Приложение 2. Примеры XML-файлов для подключения к СУБД ЛИНТЕР 39
hibernate.cfg.xml 39
web.xml для tomcat 39
context.xml для tomcat 40

Предисловие

Назначение документа

Документ содержит описание JDBC-драйвера для СУБД ЛИНТЕР, реализованного на основе спецификации SUN JDBC 4.2 API. Драйвер используется совместно с J2SE и J2EE последних версий. Для каждого интерфейса драйвера дается информация о его соответствии спецификации стандарта и об особенностях реализации для СУБД ЛИНТЕР.

Дополнительно описаны установка и запуск драйвера.

Документ предназначен для СУБД ЛИНТЕР БАСТИОН 6.0 сборка 20.8, далее по тексту СУБД ЛИНТЕР.

Для кого предназначен документ

Документ предназначен для программистов, разрабатывающих приложения с использованием JDBC-доступа к СУБД ЛИНТЕР.

Необходимые предварительные знания

Для работы с LINTER JDBC API необходимо владеть:

- основами реляционных баз данных и языка баз данных SQL;
- стандартными спецификациями JDBC-интерфейса;
- языком программирования Java;
- навыками работы в соответствующей операционной системе на уровне простого пользователя.

© JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

Назначение и условия применения программы

Назначение драйвера

Java DataBase Connectivity (JDBC) – это стандартный интерфейс, предназначенный для доступа к базам данных из приложений, написанных на языке программирования Java. Использование JDBC позволяет разрабатывать независимые от платформы и используемой базы данных приложения.

Поддерживаются следующие интерфейсы соединения с СУБД: JNDI, Apache Cayenne и Hibernate.

Java-программа может быть разработана в виде апплета, загружаемого через Internet и запускаемого на стороне клиента, или в виде приложения, постоянно находящегося на стороне клиента. В любом случае интерфейс JDBC позволяет Java-приложению подключаться к удаленным базам данных, направлять к ним запросы и получать результаты обработки запросов. При этом необходимо помнить, что работа апплета может ограничиваться требованиями безопасности, поэтому следует, при необходимости, сконфигурировать соответствующим образом Web-browser для разрешения апплету доступа в сеть.

Реально интерфейс JDBC представляет собой набор абстрактных классов (интерфейсов в терминах Java), которые должны быть определены для конкретных источников данных. Поэтому возможно абстрактное представление JDBC высокого уровня и конкретное представление на низком уровне определенной базы данных. Представление высокого уровня дается прикладными интерфейсами JDBC, в которых имеются методы подключения к нескольким базам данных, запросов и манипулирования данными. Прикладные интерфейсы обеспечивают более высокий уровень абстракции, описывая только объявления методов (а не их реализацию).

Конкретное представление интерфейса JDBC, специфичное для каждой СУБД, реализуется конкретным драйвером JDBC.

Как и в ODBC, разработчики реализуют интерфейс JDBC посредством диспетчера драйверов java.sql.DriverManager (единственный реализованный самими разработчиками стандарта класс пакета `java.sql`), а он, в свою очередь, поддерживает многочисленные драйверы, позволяющие осуществлять связь с различными базами данных.

Пример Java модуля представлени в приложении 1.

Условия применения

Существует несколько реализаций платформы Java, каждая новая версия JDK работает со своим стандартом JDBC, расширенным и дополненным новыми возможностями. Соответствие версии JDK и стандарта JDBC приведено в таблице 1.

Таблица 1. Соответствие версии JDK и стандарта JDBC

Версия JDK	Стандарт JDBC
JDK 1.4	JDBC 3.0
JDK 1.6	JDBC 4.0

Версия JDK	Стандарт JDBC
JDK 1.7	JDBC 4.1
JDK 1.8	JDBC 4.2

Примечание
Если необходимая версия отсутствует в перечне поддерживаемых версий, следует обратиться в раздел Поддержка на сайте ЛИНТЕР.

В данном документе приводится описание драйвера JDBC для СУБД ЛИНТЕР. Если не оговорено дополнительно, то описание относится ко всем версиям JDBC.

© JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

Характеристики JDBC-драйвера

Основные функции

Драйвер обеспечивает:

1) передачу запроса к базе данных в виде строки. Таким образом, могут использоваться конструкции SQL, специфичные для данной БД и/или ее JDBC-драйвера;
2) выполнение SQL-запросов, базирующихся на спецификациях X/Open и SQL Access Group (SAG) SQL CAE 1992 года;
3) получение результатов обработки SQL-запросов;
4) предоставление кодов завершения обработки запросов;
5) поддержку стандартных типов данных;
6) статическое и динамическое формирование SQL-предложений;
7) прием и передачу значений данных в формате, задаваемом приложением.

Соответствие типов данных

Драйвер обеспечивает следующее соответствие типов данных (таблица 2).

Таблица 2. Соответствие типов данных

СУБД ЛИНТЕР	Аналог в языке Java
INTEGER	int (java.lang.Integer)
SMALLINT	short (java.lang.Short)
BIGINT	long (java.lang.Long)
BYTE	byte (class java.lang.Byte)
VARBYTE	byte[]
REAL	float (java.lang.Float)
DOUBLE	double (java.lang.Double)
FLOAT	double (java.lang.Double)
BOOLEAN	boolean (java.lang.Boolean)
CHAR	java.lang.String
VARCHAR	java.lang.String
NCHAR	java.lang.String
NCHAR VARYING	java.lang.String
DECIMAL	java.math.BigDecimal
BLOB	java.sql.Blob
DATE	java.sql.Timestamp
EXTFILE	java.lang.String

Сценарии (способы) доступа к БД

Способ доступа к БД (как правило, находящейся на сервере, а не на клиентском компьютере) зависит от типа приложения: апплет, загружаемый по сети на клиентский компьютер или самостоятельное приложение, выполняемое на клиентском компьютере.

В случае апплета сценарий взаимодействия с СУБД должен быть следующим:

1) апплет загружается (в виде байт-кода) на клиентский компьютер в составе web-документа;
2) виртуальная машина на клиентском компьютере стартует апплет;
3) апплет запрашивает менеджер драйверов JDBC (при этом необходимый JDBC-драйвер физически находится на клиентском компьютере);
4) апплет получает доступ к серверу БД по протоколу Internet (TCP/IP).

В случае самостоятельного клиентского приложения сценарий взаимодействия с СУБД должен быть следующим:

1) виртуальная машина на клиентском компьютере стартует самостоятельное клиентское приложение;
2) клиентское приложение запрашивает менеджер драйверов JDBC (при этом необходимый JDBC-драйвер физически находится на клиентском компьютере);
3) клиентское приложение получает доступ к серверу БД по протоколу Internet (TCP).

Типы JDBC-драйверов

Все JDBC-драйверы можно разделить на следующие классы:

1) драйвер, клиентская часть которого реализована на Java частично. В этом случае обычно используется динамически подключаемая библиотека, транслирующая вызовы Java в какие-либо иные. Данный тип драйверов преобразует вызовы Java в клиентский прикладной интерфейс непосредственно для конкретной СУБД. Подобно всем драйверам моста, драйверы этого типа предполагают загрузку некоего двоичного кода на каждый клиентский компьютер;
2) драйвер, клиентская часть которого полностью реализована на Java. Серверная часть драйвера (если существует) может быть написана на другом языке;
3) драйвер, реализованный полностью на Java с сетевым протоколом, преобразует вызовы JDBC в независящий от СУБД сетевой протокол, который далее на сервере преобразуется в протокол СУБД. Такое сетевое серверное промежуточное программное обеспечение связывает всех клиентов Java с различными БД; конкретный протокол СУБД зависит от её производителя. Как правило, этот драйвер является наиболее гибким вариантом JDBC. В драйверах подобного рода иногда используют драйверы ODBC на сервере для реального взаимодействия с СУБД;
4) драйвер, у которого клиентская и серверная часть (если существует) полностью реализованы на Java. Драйвер, реализованный полностью на Java с оригинальным протоколом, непосредственно преобразует вызовы JDBC в сетевой протокол, используемый данной СУБД. Такой подход позволяет направлять вызовы с клиентских приложений непосредственно на сервер СУБД. Данное решение имеет практический смысл, если конфигурация клиентских приложений контролируется достаточно жестко, как это бывает в типичных приложениях для Internet. Поскольку большинство протоколов для БД нестандартны, драйверы подобного типа должны поставлять сами производители СУБД для этих БД.

Помимо JDBC-драйверов, обеспечивающих доступ к конкретной БД, существуют еще универсальные JDBC-драйверы, которые могут взаимодействовать с несколькими БД с помощью других стандартных интерфейсов. Например, мост от JDBC к ODBC обеспечивает доступ к БД с помощью драйверов ODBC. Для его работы на каждом

 © JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

клиентском компьютере, кроме самого драйвера JDBC-ODBC, необходимо еще наличие соответствующего драйвера ODBC для БД, к которой производится обращение.

Описываемый JDBC-драйвер СУБД ЛИНТЕР принадлежит к четвертому классу. Все интерфейсы стандарта реализованы на Java и не требуют присутствия на клиентской стороне машинозависимого кода на каждом клиентском компьютере. Серверная часть драйвера написана на языке программирования C. Она транслирует вызовы из промежуточного сетевого протокола в вызовы СУБД и отправляет назад клиентскому приложению результаты выполнения вызовов.

Состав прикладного интерфейса JDBC-драйвера СУБД ЛИНТЕР

Так как Java является объектно-ориентированным языком, то под прикладным интерфейсом подразумевается набор классов и интерфейсов (в понимании языка Java). Эти классы и интерфейсы описываются в специальном пакете `java.sql`. JDBC-драйвер есть совокупность классов, реализующих JDBC-интерфейсы.

В данном разделе рассматриваются особенности реализации стандарта JDBC в JDBC-драйвере СУБД ЛИНТЕР и отличия версий JDBC-драйверов. Более подробную информацию можно найти в документации к соответствующей версии JDK.

Пакет `java.sql` содержит следующие классы и интерфейсы:

1) соединения с СУБД:
   - класс DriverManager создает соединения с СУБД, используя соответствующий драйвер;
   - класс SQLPermission предназначен для управления правами доступа к БД, когда программа работает в защищенной среде, например, в случае использования апплета на клиентской стороне;
   - интерфейс Driver предоставляет возможность для соединения с конкретной БД, обычно используется только классом DriverManager;
   - класс DriverPropertyInfo предназначен для хранения свойств драйвера, обычно пользователями не используется.
2) обработки SQL-запросов с помощью СУБД:
   - интерфейс Statement предназначен для обработки простых SQL-запросов;
   - интерфейс PreparedStatement предназначен для обработки SQL-запросов, содержащих параметры (наследуется от Statement);
   - интерфейс CallableStatement предназначен для вызова хранимых процедур с входными и выходными параметрами (наследуется от PreparedStatement);
   - интерфейс Connection предоставляет методы для создания соответствующих объектов для обработки SQL-запросов и управления свойствами соединения;
   - интерфейс Savepoint предназначен для управления временными точками сохранения в пределах транзакции.
3) получения и обновления информации в результатах обработки запросов:
   - интерфейс ResultSet.
4) отображения стандартных SQL-типов данных в классы Java:
   - интерфейс Blob выполняет отображение SQL BLOB;
   - интерфейс Clob выполняет отображение SQL CLOB;
   - класс Date выполняет отображение SQL DATE;
   - класс Time выполняет отображение SQL TIME;
   - класс Timestamp выполняет отображение SQL TIMESTAMP;
   - класс Types содержит константы для SQL-типов.

5) работы с метаданными:
   - интерфейс DatabaseMetaData предоставляет информацию об объектах БД;
   - интерфейс ResultSetMetaData предоставляет информацию о полях запроса выборки данных;
   - интерфейс ParameterMetaData предоставляет информацию о параметрах хранимых процедур и запросов с параметрами.
6) обработки ошибок:
   - класс SQLException применяется большинством методов в случае ошибочной ситуации;
   - класс SQLWarning применяется в случае предупреждения;
   - класс DataTruncation применяется в том случае, если данные могут быть не полностью переданы от клиентского приложения к БД или в обратном направлении;
   - класс BatchUpdateException применяется для сигнализации, что не все операции множественного изменения БД были выполнены успешно.

Новые возможности JDBC 2

В JDBC 2 реализованы следующие новые возможности:

- интерфейс DataSource для унифицированного соединения с источниками данных. Интерфейс JNDI используется для регистрации источников данных и получения соединения с ними;
- отложенное закрытие соединений, позволяющее использовать ранее открытые соединения и не тратить время на соединения с СУБД;
- распределенные транзакции, позволяющие производить транзакцию с использованием нескольких серверов БД;
- интерфейс RowSet, предоставляющий более удобный способ работы с данными;
- поддержка скроллируемых курсоров – новые методы в интерфейсе ResultSet, позволяющие перемещать курсор относительно текущего положения в выборке данных или в абсолютную позицию выборки данных;
- множественные обновления – выполнение нескольких запросов как одного целого;
- изменение данных при перемещении по выборке данных с использованием метода обновления информации в интерфейсе ResultSet;
- поддержка новых типов данных – новые интерфейсы для отображения SQL-типов данных;
- другие улучшения: оптимизация производительности, использование потоков символов, улучшенная поддержка значений типа java.math.BigDecimal, поддержка временных зон в значениях времени.

Новые возможности JDBC 3

В JDBC 3 реализованы следующие новые возможности:

- точки сохранения – программная поддержка возможности отката транзакции к обозначенной точке сохранения;

- расширенные свойства для класса ConnectionPoolDataSource, определяющие, каким образом соединения должны добавляться в пул;
- метаданные для параметров объектов класса PreparedStatement;
- возможность получить значение для автоматически генерируемых полей;
- возможность одновременно иметь несколько объектов класса ResultSet, открытых одним и тем же объектом класса CallableStatement;
- возможность для идентификации параметров объектов класса CallableStatement по имени и порядковому номеру;
- возможность указать, должен ли курсор быть закрытым после завершения транзакции или оставаться открытым;
- возможность программно изменять BLOB/CLOB-значения;
- дополнительные метаданные для получения иерархии SQL-типов.

Новые возможности JDBC 4

В JDBC 4 реализованы следующие новые возможности:

- автоматическая загрузка JDBC-драйвера;
- поддержка конвертации национальных кодировок;
- поддержка типов NCHAR, NVARCHAR, LONGNVARCHAR, NCLOB и методов их обработки в интерфейсах Connection, PreparedStatement, CallableStatement;
- расширенная поддержка BLOB/CLOB данных: добавлены дополнительные методы обработки, создания и записи в потоках;
- возможность получения нативного класса для интерфейсов;
- расширенные возможности для интерфейсов Connection и Statement: создание LOB объектов, возможность лучшего отслеживания состояния соединения, большая гибкость управления в пуле;
- получение дополнительных метаданных о функциях, схемах и свойствах объектов БД;
- расширен интерфейс PooledConnection в части поддержки нотификации о создании и удалении Statement.

Новые возможности JDBC 4.1

В JDBC 4.1 реализованы следующие новые возможности:

- поддержка технологии try-with-resources в интерфейсах Connection, ResultSet и Statement;
- возможность использования типов данных Date, Calendar для данных БД типа Timestamp;
- возможность отображения типа java.lang.BigInteger в JDBC BIGINT;
- возможность преобразования типов данных Date, Calendar в тип данных CHAR, VARCHAR, LONGVARCHAR, DATE, TIME и TIMESTAMP;
- возможность преобразования типа данных BigInteger в тип данных CHAR, VARCHAR, LONGVARCHAR, и BIGINT;

 © JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

- возможность использования схем, тайм-аута и разрыва соединения в интерфейсе Connection;
- поддержка метода getParentLogger для класса Driver;
- возможность получения значений псевдостолбцов;
- возможность закрытия statement после закрытия всех его resultset.

Новые возможности JDBC 4.2

В JDBC 4.2 реализованы следующие новые возможности:

- поддержка типизации данных с помощью интерфейсов SQLType, JDBCType;
- поддержка интерфейсов SQLType, JDBCType в функциях setObject, registerOutParameter, updateObject, интерфейсов CallableStatement, PreparedStatement и ResultSet.

Установка и запуск драйвера

Состав JDBC-драйвера

JDBC-драйвер состоит из следующих компонентов:

- серверная часть (сервис `linapid`, находящийся в каталоге `~linter/bin`);
- клиентская часть (наборы Java-классов `linjdbc-1.4.jar`, `linjdbc-1.6.jar`, `linjdbc-1.7.jar` и `linjdbc-1.8.jar` для соответствующей версии JDK);
- дополнительный JNDI-интерфейс входит в состав linjdbc.

Выбор пакета для соответствующей версии JDK

Клиентская часть драйвера JDBC поставляется в зависимости от версии JDK клиентского компьютера:

1) версия JDK: 1.4
   - версия JDBC: 3
   - имя архива: linjdbc-1.4.jar
   - имя драйвера: com.relx.jdbc.LinterDriver
   - формат URL: jdbc:linter:linapid:<host>:<port>:<db>
   - JNDI интерфейс: входит в состав linjdbc.
   - JNDI DataSource: com.relx.jdbc.LinterJNDIDataSource
2) версия JDK: 1.6
   - версия JDBC: 4
   - имя архива: linjdbc-1.6.jar
   - имя драйвера: com.relx.jdbc.LinterDriver
   - формат URL: jdbc:linter:linapid:<host>:<port>:<db>
   - JNDI интерфейс: входит в состав linjdbc.
   - JNDI DataSource: com.relx.jdbc.LinterJNDIDataSource
3) версия JDK: 1.7
   - версия JDBC: 4.1
   - имя архива: linjdbc-1.7.jar
   - имя драйвера: com.relx.jdbc.LinterDriver
   - формат URL: jdbc:linter:linapid:<host>:<port>:<db>
   - JNDI интерфейс: входит в состав linjdbc.
   - JNDI DataSource: com.relx.jdbc.LinterJNDIDataSource
4) версия JDK: 1.8
   - версия JDBC: 4.2
   - имя архива: linjdbc-1.8.jar

- имя драйвера: com.relx.jdbc.LinterDriver
- формат URL: jdbc:linter:linapid:<host>:<port>:<db>
- JNDI интерфейс: входит в состав linjdbc.
- JNDI DataSource: com.relx.jdbc.LinterJNDIDataSource

где:

<host> – сетевое имя сервера, на котором запущен сервер JDBC linapid;

<port> – порт утилиты linapid (задаётся параметром -p при запуске linapid, значение по умолчанию 1070);

<db> – имя сервера из ~linter\bin\nodetab ('local' – для доступа к серверу по умолчанию).

Пример задания url: "jdbc:linter:linapid:192.168.1.123:1070:local".

Установка драйвера

JDBC-драйвер входит в состав дистрибутива СУБД ЛИНТЕР. Процесс установки драйвера выполняется автоматически (при условии выбора драйвера как компонента СУБД). Путь к JDBC-драйверу при запуске Java-приложений должен задаваться в переменной окружения CLASSPATH или ключом компилятора -classpath (или -cp), например,

1)

```
SET CLASSPATH=%LINTER_HOME%\jdbc\linjdbc-1.4.jar
%JAVAC% -classpath %CLASSPATH% test.java
```

2)

```
C:\PROGRA~1\Java\jdk1.8.0_31\bin\javac
-classpath C:\Linter\jdbc\linjdbc-1.4.jar test.java
```

Примечание
При установке JDBC-драйвера для ранних версий JDK иногда требуется вручную разархивировать файл jclasses.zip в некоторый каталог и добавить путь к этому каталогу в переменной окружения CLASSPATH.

Установка JDBC-драйвера в локальный репозиторий maven

Для установки JDBC-драйвера в локальный репозиторий maven, необходимо выполнить команду:

- для jdbc3:

```
mvn install:install-file -Dfile=linjdbc-1.4.jar -
DgroupId=ru.relex.lintersql
 -DartifactId=linter-jdbc3 -Dversion=1.0.1 -Dpackaging=jar -
DgeneratePom=true
```

- для jdbc4:

```
mvn install:install-file -Dfile=linjdbc-1.6.jar -
DgroupId=ru.relex.lintersql
 -DartifactId=linter-jdbc4 -Dversion=1.0.1 -Dpackaging=jar -
DgeneratePom=true
```

- для jdbc4.1:

```
mvn install:install-file -Dfile=linjdbc-1.7.jar -
DgroupId=ru.relex.lintersql
 -DartifactId=linter-jdbc4.1 -Dversion=1.0.1 -Dpackaging=jar -
DgeneratePom=true
```

- для jdbc4.2:

```
mvn install:install-file -Dfile=linjdbc-1.8.jar -
DgroupId=ru.relex.lintersql
 -DartifactId=linter-jdbc4.2 -Dversion=1.0.1 -Dpackaging=jar -
DgeneratePom=true
```

Запуск клиентской части драйвера

Клиентская часть JDBC-драйвера запускается автоматически на клиентском компьютере при выполнении команды регистрации драйвера апплетом или самостоятельным клиентским приложением.

Запуск серверной части драйвера

На сервере необходимо запустить программу `linapid`, принимающую запросы от клиентской части JDBC-драйвера и адресующую их к ядру СУБД ЛИНТЕР.

Ключи запуска программы `linapid` приведены в таблице 3.

Таблица 3. Ключи запуска программы linapid

Ключ	Поддерживается ОС		Описание
	Windows	Linux	
-h, -help, -H, -?, /h, /H, /?	–	+	Выдать на консоль справочную информацию о ключах
{-p, --port, -p=, /p=}<port>	+	+	Номер порта (по умолчанию используется значение 1070)
-f, -F, /f, /F, --no-daemon	–	+	Не уходить в фоновый режим
-l, /l, --log	+	+	Выполнять протоколирование сообщений `linapid`
-log, /log		–	
-t, /t, --trace	+	+	Выполнять трассировку работы `linapid`. Трассируется: • открытие и закрытие соединений (выводится номер соединения, номер канала СУБД ЛИНТЕР, который соответствует открытому соединению и режим открытия);
-trace, /trace		–	

 © JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

Ключ	Поддерживается ОС		Описание
	Windows	Linux	
			• выполнение операции commit/ rollback; • ошибочные коды завершения. Трассировка совместима с логированием, то есть их можно запускать одновременно
-version, -VERSION /version, /VERSION	–	+	Показать полную версию СУБД ЛИНТЕР, с которой взаимодействует `linapid`
-briefversion, -BRIEFVERSION /briefversion, /BRIEFVERSION	–	+	Показать короткую версию СУБД ЛИНТЕР, с которой взаимодействует `linapid`
-w, -W, /w, /W	–	+	Работа до тех пор, пока существует родительский процесс
/HPARENT	–	–	Синоним /W
/NAME=<linter_mbx>	+	–	Выставляет переменную окружения `LINTER_MBX` для процесса `linapid`
/NONAME	+	–	Не использовать переменную окружения `LINTER_MBX`
/KEEPIDLE=<interval>	+	+	Установить интервал времени от последнего пакета с данными до первого сторожевого пакета постоянного («неразрываемого») соединения (keepalive) (сек.). Если пришел (или, наоборот, отправлен) пакет с данными раньше этого интервала, то посылки сторожевого пакета не будут производиться. Значение по умолчанию 60 сек.
/KEEPINTVL=<interval>	+	+	Установить интервал посылки сторожевых пакетов постоянного («неразрываемого») соединения (keepalive) (сек.). Значение по умолчанию 6 сек.
/KEEPCNT=<count>	–	+	Установить количество не принятых сторожевых пакетов постоянного («неразрываемого») соединения (keepalive), после которого соединение всё-таки считается разорванным. Значение по умолчанию 10

Реализация JDBC-драйвера для СУБД ЛИНТЕР

java.sql.CallableStatement

Интерфейс java.sql.CallableStatement предназначен для выполнения хранимых процедур. Он работает с параметризированными SQL-выражениями.

Пример

```
Работа с хранимой процедурой, возвращающей курсор.
Описание процедуры:
create or replace procedure getCursor() result cursor(personid
 int)
declare
  var res typeof(result);
code
  open res for direct "select personid from auto;";
  return res;
end;

Фрагмент кода в java:

CallableStatement cs = connection.prepareCall("{call
 getCursor()}");
ResultSet rs = cs.executeQuery();
```

java.sql.Connection

Интерфейс java.sql.Connection определяет характеристики (включая выбор кодировки) и состояние соединения с СУБД; кроме того, он предоставляет средства для контроля транзакций и уровня их изолированности. Интерфейс предназначен для подачи запросов к БД и получения результатов их обработки.

Возможности класса:

- методы commit, rollback и setAutoCommit управляют транзакциями;
- объекты CallableStatement, создаваемые методом prepareCall, – хранимыми процедурами;
- объекты PreparedStatement (метод prepareStatement) – претранслированными запросами;
- объекты Statement – обычными запросами.

Для получения информации о БД интерфейс Connection использует метод getMetaData, который возвращает объект DatabaseMetaData. Он предоставляет информацию с описанием таблиц БД, поддерживаемой СУБД грамматики SQL-операторов, ее хранимых процедур, возможностей соединения и т. д.

В клиентском приложении выполняется автоматическое определение кодировки канала сервера:

- на основании свойства encoding соединения (если оно задано);
- на основании свойства file.encoding, отражающего кодировку клиента по умолчанию (если свойство encoding соединения не задано).

Для соединения с конкретной БД необходимо указать строку соединения (URL), которая имеет вид (для JDBC-драйвера версии 3.0 и выше):

```
jdbc:linter:linapid:[<host>]:[<port>]:[<node>]
[;<параметр настройки>…]
```

где:

<host> – IP-адрес удаленного узла; если он не задан, используется локальный узел;

<port> – номер порта (1070, если не указан), на котором работает серверная часть JDBC-драйвера;

<node> – имя сервера из <LINTER_TOP_DIR>\bin\nodetab на удалённом узле, где <LINTER_TOP_DIR> – спецификация пути к установочному каталогу СУБД ЛИНТЕР;

<параметр настройки> – задает параметр настройки соединения.

Возможные значения параметра настройки соединения:

1) rollbackOnClose=true | false – устанавливает режим закрытия соединения:
   - если выставлен режим autocommit и rollbackOnClose=false, то при вызове метода close() измененные данные сохранятся в БД;
   - если выставлен режим autocommit и rollbackOnClose=true, то при вызове метода close() изменения в БД не сохраняются.

   Значение по умолчанию true.

   Пример url:

```
jdbc:linter:linapid:localhost:1070:DEMO;rollbackOnClose=false
```

2) emptyBlobAsNull=true | false – устанавливает правило интерпретации пустого BLOB-значения для методов интерфейса java.sql.ResultSet getBinaryStream(), getCharacterStream(), getAsciiStream():
   - true – возвращать для пустых BLOB-данных null-значение;
   - false – возвращать объекты InputStream, Reader и InputStream соответственно.

   Значение по умолчанию true.

   Пример url:

```
jdbc:linter:linapid:localhost:1070:DEMO;emptyBlobAsNull=false
```

3) autoCommit=true | false – устанавливает (true)/отменяет (false) режим autoCommit, то есть включает режим Pessimistic.

   Значение по умолчанию true.

   Пример url:

```
jdbc:linter:linapid:localhost:1070:DEMO;autoCommit=false
```

4) timeout=<значение> – задает максимальное время ожидания (в миллисекундах) клиентским приложением данных с ЛИНТЕР-сервера. Если <значение> равно 0, предполагается бесконечное ожидание.

   Значение по умолчанию 0.

   Пример url:

```
jdbc:linter:linapid:localhost:1070:DEMO;timeout=1000
```

5) ignoreTargetSqlType=true | false – не учитывать (true)/учитывать (false) параметр SQLType, передаваемый в функции setNull и setObject:

   - если настройка установлена в значение true, драйвер учитывает переданный SQLType согласно спецификации (о соответствии см. подробнее в JDBC Specification, Table B-5);
   - если настройка установлена в значение false, драйвер игнорирует переданный тип.

   Значение по умолчанию: false.

6) LOBCreateOnCopy=true | false – использовать копию LOB-данных (true)/использовать оригинальные LOB-данные (false).

   Настройка актуальна при множественном использовании одного и того же LOB-значения. Например, значение одного и того же LOB-поля выборки данных можно присвоить нескольким переменным, при этом изменение значения любой из этих переменных либо не будет влиять на значение остальных переменных (true), либо влечет изменение значения и во всех остальных переменных (false).

   Значение по умолчанию false.

   Пример url:

```
jdbc:linter:linapid:localhost:1070:DEMO;LOBCreateOnCopy=true
```

7) encoding=<значение> – задаёт кодировку соединения с ЛИНТЕР-сервером.

   Примеры значений:

   - cp866;
   - cp1251;
   - koi8-r.

   Пример url:

```
jdbc:linter:linapid:localhost:1070:DEMO;encoding=cp1251
```

При создании соединения в случае ошибки в имени/значении параметров будут создаваться предупреждения SQLWarning, доступные путем вызова метода conn.getWarnings().

Пример подсоединения к БД с помощью JDBC-драйвера приведен в приложении 2.

Примечание
Параметры строки соединения – позиционные, поэтому вместо пропускаемых параметров надо ставить двоеточие.

Пример

```
Properties info = new Properties();
info.put("user", "...");
info.put("password", "...");
info.put("encoding", "cp1251");
driver.connect(url, info);
```

Примечание
Если на клиенте свойство не указано, то для конвертации строк будут использоваться настройки Java-приложения.

Пример

```
import java.sql.*;
import com.relx.jdbc.LinterDriver;

public class Connect
{
  public static void main (String[] args)
  {
    try
    {
      Driver d =
 (Driver)Class.forName("com.relx.jdbc.LinterDriver").newInstance();
      String address =
 "jdbc:linter:linapid:localhost:1070:local;autoCommit=false";

      String user    = "SYSTEM";
      String password= "MANAGER8";

      System.out.println("Driver found. Now connecting to
 database ... ");

      Connection con =
 DriverManager.getConnection(address,user,password);

      System.out.println(" Connection established ... ");

      /*......*/
      con.close();
    }
    catch (Exception e)
    {
```

```
      System.out.println("Catched :"+e+" mess= "+e.getMessage() );
      e.printStackTrace();
    }
  }
}
```

java.sql.DatabaseMetaData

Интерфейс java.sql.DatabaseMetaData запрашивает у СУБД информацию о структуре объектов БД, поддерживаемых СУБД функциях, типах данных и другие возможности. Интерфейс java.sql.DatabaseMetaData позволяет клиентским приложениям извлекать необходимую информацию в реальном времени (при условии, что пользователь имеет соответствующие права на выполнение запросов), благодаря чему можно получать не относящиеся конкретно к выполнению/обработке запросов подробности, например, какой символ используется в качестве разделителя и как СУБД обрабатывает null-значение при сортировке.

Пример

```
import java.sql.*;
import com.relx.jdbc.LinterDriver;

public class DatabaseMetaDataDemo
{
  public static void main(String[] args)
  {
    try
    {
      Driver d = (Driver)
 Class.forName("com.relx.jdbc.LinterDriver")
          .newInstance();

      String address = "jdbc:linter:linapid:localhost:1070:DEMO";
      String user = "SYSTEM";
      String password = "MANAGER8";
      System.out.println("Driver found. Now connecting to database
 ");
      Connection con = DriverManager.getConnection(address, user,

 password);

      Statement stmt = con.createStatement();

      DatabaseMetaData dbmd = con.getMetaData();

      System.out.println("Product Name = " +

 dbmd.getDatabaseProductName());
      System.out.println("MaxLengthTableName = " +
```

 © JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

```
 dbmd.getMaxTableNameLength());

      System.out.println("===> Geting tables information");

      ResultSet tables = dbmd.getTables(null, null, "%TY%", null);
      while (tables.next())
      {
        String tableName = tables.getString("TABLE_NAME");
        String tableSchema = tables.getString("TABLE_SCHEM");
        if (tableSchema.length() > 0)
          tableName = tableSchema + "." +tableName;
        System.out.println(tableName);
      }
      stmt.close();
    }
    catch (Exception e)
    {
      System.out.println("Catched :" + e + " mess= " +
 e.getMessage());
      e.printStackTrace();
    }
  }
}
```

java.sql.Driver

Интерфейс предназначен для получения информации о JDBC-драйвере, проверки адреса на корректность и присоединения к конкретной БД (создания объекта Connection).

Для использования JDBC-драйвера СУБД ЛИНТЕР необходимо выполнить регистрацию драйвера с помощью команды:

```
// для драйвера JDBC 3.0 и поздних
Class.forName("com.relx.jdbc.LinterDriver").newInstanse();
```

java.sql.DriverManager

Интерфейс java.sql.DriverManager обеспечивает загрузку драйверов и создание новых соединений с СУБД; это основной интерфейс JDBC, определяющий корректный выбор и инициализацию драйвера для данной СУБД в данных условиях.

При активизации драйвера его метод getConnection связывается с СУБД и обращается к объекту соединения. Последний может участвовать только в одном сеансе с СУБД, поэтому, если клиентскому приложению необходимо получить доступ, например, к двум БД ЛИНТЕР, то каждая из них должна иметь свой объект соединения. Приложение, которое планирует обращаться к нескольким БД, должно будет загрузить (создать) несколько объектов соединения.

java.sql.PreparedStatement

Интерфейс java.sql.PreparedStatement предназначен для выполнения претранслированных SQL-выражений. Он работает с SQL-выражениями, имеющими параметры.

Пример

```
import java.sql.*;
import java.util.Arrays;
import com.relx.jdbc.LinterDriver;

public class PreparedStatementDemo
{
  public static void main (String[] args)
  {
    try
    {
      Driver d =
 (Driver)Class.forName("com.relx.jdbc.LinterDriver").newInstance();

      String address = "jdbc:linter:linapid:localhost:1070:DEMO";
      String user    = "SYSTEM";
      String password= "MANAGER8";

      System.out.println("Driver found. Now connecting to
 database.");
      Connection con;
      con = DriverManager.getConnection(address,user,password);

      PreparedStatement prepstmt;

      Statement stmt = con.createStatement();

      stmt.executeUpdate("create or replace table test (a blob, b
 int);");

      prepstmt = con.prepareStatement("insert into test values
 (?,?);");

      System.out.println("Prepared statement created");

      int testSize = 10;
      byte b = 126;
      byte[] buffer= new byte[testSize];
      Arrays.fill(buffer, b);
```


© JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

```
      System.out.println("Array filled");

      // SET BYTES
      prepstmt.setBytes(1, buffer);

      // SET INT PARAMETER
      prepstmt.setInt(2, 3);

      // EXECUTE PREPARED UPDATE
      int res = prepstmt.executeUpdate();

      System.out.println("Row count insert statements = " + res);

      // EXECUTE QUERY
      System.out.println("Selecting from database");
      ResultSet results;
      results = stmt.executeQuery("select * from test;");

      results.next();

      System.out.println("Second column value = " +
 results.getInt(2));
      byte bt[] = results.getBytes(1);
      System.out.println("Blob value = " + Arrays.toString(bt));

      results.close();

      try
      {
        stmt.executeUpdate("drop table test;");
      }
      catch(SQLException e)
      {
        System.out.println("Error deleting table:" +
 e.getMessage());
      }
      con.close();
    }
    catch (Exception e)
    {
        System.out.println("Catched :" + e + " mess= " +
 e.getMessage());
        e.printStackTrace();
    }
  }
}
```

java.sql.ResultSet

Интерфейс java.sql.ResultSet предоставляет доступ к набору строк (выборке данных), полученному в результате выполнения поискового SQL-запроса.

Пример

```
import java.sql.*;
import com.relx.jdbc.LinterDriver;

public class ResultSetDemo
{
  public static void main (String[] args)
  {
    try
    {
      Driver d =
 (Driver)Class.forName("com.relx.jdbc.LinterDriver").newInstance();

      String address = "jdbc:linter:linapid:localhost:1070:DEMO";
      String user    = "SYSTEM";
      String password= "MANAGER8";

      Connection con =

 DriverManager.getConnection(address,user,password);
      Statement stmt = con.createStatement();

      ResultSet results;

      stmt.executeUpdate("create table test1 (a int, b
 char(20));");

      //INSERT
      stmt.executeUpdate("insert into test1 values(1, 'First
 string');");
      stmt.executeUpdate("insert into test1 values(2, 'Another
 string');");
      stmt.executeUpdate("insert into test1 values(3,'Third
 string');");

      results = stmt.executeQuery("select * from test1;");

      while (results.next())
      {
        // Loop through each column, getting the column
```


© JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

```
        // data and displaying
        System.out.print("| "+results.getInt(1));
        System.out.println("| "+results.getString(2)+" |");
      }

      System.out.println();
      System.out.println("To previous...");
      if(results.previous())
      {
        System.out.print("| " + results.getInt(1));
        System.out.println("| " + results.getString(2)+" |");
      }
      System.out.println("To (current+1)...");
      if(results.relative(1))
      {
        System.out.print("| " + results.getInt(1));
        System.out.println("| " + results.getString(2)+" |");
      }

      System.out.println("To first...");
      if(results.absolute(1))
      {
        System.out.print("| " + results.getInt(1));
        System.out.println("| " + results.getString(2)+" |");
      }

      results.close();

      try
      {
        stmt.executeUpdate("drop table test1;");
      }
      catch(SQLException e)
      {
        System.out.println("Error deleting
table:"+e.getMessage());
      }

      con.close();
    }
    catch (Exception e)
    {
      System.out.println("Catched :" + e + " mess= " +
e.getMessage() );
      e.printStackTrace();
    }
```

```
  }
}
```

java.sql.ResultSetMetaData

Интерфейс java.sql.ResultSetMetaData позволяет получить информацию о типе данных и свойствах столбцов в ResultSet. Эта возможность особенно важна при построении динамических систем, таких, как среда разработки приложений или инструментарий для конструирования SQL-запросов, в которых информация о БД и ее объектах заранее неизвестна. Интерфейс устаревший и не рекомендуется для использования.

Примечание
Имена столбцов независимы от регистра.

Пример

```
import java.sql.*;
import com.relx.jdbc.LinterDriver;

public class ResultSetMetaDataDemo
{
  public static void main(String[] args)
  {
    try
    {
      Driver d = (Driver)
 Class.forName("com.relx.jdbc.LinterDriver")
          .newInstance();

      String address = "jdbc:linter:linapid:localhost:1070:DEMO";
      String user = "SYSTEM";
      String password = "MANAGER8";
      System.out.println("Driver found. Now connecting to
 database. ");
      Connection con = DriverManager.getConnection(address, user,
 password);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("select * from AUTO");
      ResultSetMetaData rsmd = rs.getMetaData();

      int count = rsmd.getColumnCount();
      for (int i = 1; i <=count; i++)
      {
        System.out.println("Column " + i + " = " +
                           rsmd.getColumnName(i) +
                           ", type = " +
 rsmd.getColumnTypeName(i));
      }
```


© JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

```
      stmt.close();
    }
    catch (Exception e)
    {
      System.out.println("Catched :" + e + " mess= " +
 e.getMessage());
      e.printStackTrace();
    }
  }
}
```

java.sql.Statement

Интерфейс java.sql.Statement предназначен для передачи СУБД SQL-запросов (под SQL-запросом понимается не только текст запроса, но и такие характеристики, как параметры и состояние запроса).

Особенности:

1) имена функций от EXTFILE (FILENAME, FILESIZE, FILETIME) в SQL-операторах, используемых в данном интерфейсе, должны обрамляться двойными кавычками;
2) допускаются следующие значения логического типа: `0`, `1`, `false`, `true`, `T`, `F`;
3) поддерживаемые форматы дат:
   - dd-MM-yyyy:HH:mm:ss.SS;
   - dd-MON-yyyy:HH:mm:ss.SS;
   - dd.MM.yyyy:HH:mm:ss.SS;
   - dd/MM/yyyy:HH:mm:ss.SS;
   - yyyy-MM-dd:HH:mm:ss.SS;
   - yyyyMMdd.

   Дополнительно проверяется возможное совпадение со стандартными форматами дат классов java.sql.Time и java.sql.Timestamp: yyyy-mm-dd hh:mm:ss.ffffffff и hh:mm:ss, соответственно.
4) для учета регистрозависимости идентификаторов их следует обрамлять двойными кавычками, например:

```
String user = "\"admin\"";
String password = "\"эюя\"";
```

Пример

```
import java.sql.*;
import com.relx.jdbc.LinterDriver;

public class StatementDemo
{
  public static void main(String[] args)
  {
    try
```

```
    {
      Driver d =
      (Driver)
Class.forName("com.relx.jdbc.LinterDriver").newInstance();

      String address = "jdbc:linter:linapid:localhost:1070:DEMO";
      String user = "SYSTEM";
      String password = "MANAGER8";
      System.out.println("Driver found. Now connecting to database
");
      Connection con =
                     DriverManager.getConnection(address, user,
password);

      System.out.println(" Connection established ... ");
      Statement stmt = con.createStatement();

      System.out.println(" Statement created ... ");
      ResultSet results;

      // DROP/CREATE TABLE
      stmt.executeUpdate("create table test1 (a int, b
char(20));");
      // INSERT
      stmt.executeUpdate("insert into test1 values(1, 'First
string');");
      stmt.executeUpdate("insert into test1 values(2, 'Another
string');");
      results = stmt.executeQuery("select * from test1;");
      System.out.println(" Result set retrieved... ");
      while (results.next())
      {
        // Loop through each column, getting the column
        // data and displaying
        System.out.print("| " + results.getInt(1));
        System.out.println("| " + results.getString(2) + " |");
      }
      results.close();
      try
      {
        stmt.executeUpdate("drop table test1;");
      }
      catch (SQLException e)
      {
        System.out.println("Error deleting table:" +
e.getMessage());
```


© JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

```
      }
      con.close();
    }
    catch (Exception e)
    {
      System.out.println("Catched :" + e + " mess= " +
 e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Поддерживаемые интерфейсы соединения с СУБД

JNDI

Для подсоединения к БД с помощью JNDI-интерфейса используются два класса:

1) com.relx.jdbc.LinterJNDIDataSource (com.relx.jdbc.jndi.LinterDataSource для jdbc2, jdbc.LinJdbc.jndi.LinterDataSource для jdbc1) – реализация javax.sql.DataSource интерфейса, предназначенного для создания соединения с СУБД;
2) com.relx.jdbc.LinterJNDIDataSourceFactory (com.relx.jdbc.jndi.LinterDataSourceFactory для jdbc2, jdbc.LinJdbc.jndi.LinterDataSourceFactory для jdbc1) – реализация интерфейса javax.naming.spi.ObjectFactory, предназначенного для создания DataSource объекта по его ранее сохраненным в контекст свойствам.

Класс LinterDataSource устанавливает следующие свойства:

- url – строка для соединения с СУБД;
- username – имя пользователя;
- password – пароль пользователя;
- description – вспомогательное текстовое описание источника данных;
- serverName – сетевое имя сервера;
- databaseName – имя БД на сервере;
- portNumber – номер порта сервера;
- encoding – кодировка соединения с сервером.

Каждое из свойств имеет соответствующий get/set метод, названый с учетом соглашений, принятых при написании JavaBeans.

Для соединения с СУБД с помощью JNDI-интерфейса сначала в некотором контексте (в терминах JNDI) создается описание источника данных и его свойств (это выполняется, как правило, однократно при конфигурировании приложения). Затем приложение может многократно получать из контекста источник данных и создавать соединение к этому источнику данных.

Примеры

1) Программное создание источника данных и привязка его к определенному имени в JNDI-контексте:

```
import com.relx.jdbc.LinterJNDIDataSource; // для версии JDK 1.4 и
 выше
...
   LinterDataSource ds = new LinterDataSource();
   ds.setUser("SYSTEM");
   ds.setPassword("MANAGER8");
   ds.setServerName("localhost");
```


© JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

```
   ds.setPortNumber(1070);
   ds.setDatabaseName("DEMO");

   Context ctx = new InitialContext();
   ctx.bind("jdbc/DemoDB", ds);
…
```

2) Подсоединение к БД с использованием только текстового имени источника данных, помещенного в JNDI-контекст:

```
Connection getMyLinterConnection()
  throws NamingException, SQLException
{
  Context ctx = new InitialContext();
  DataSource ds = (DataSource) ctx.lookup("jdbc/DemoDB");
  Connection conn = ds.getConnection();
  return conn;
}
```

3) Пример конфигурирования источника данных Tomcat для соединения с СУБД:
   а) скопировать файлы `linjdbc-1.x.jar` в каталог `$TOMCAT_HOME/common/lib`;
   б) в файле `$TOMCAT_HOME/conf/server.xml` или в файле приложения `META-INF/context.xml` в теге <Context> прописать источник данных:

```
<Resource name="jdbc/DB"
          auth="Container"
          type="com.relx.jdbc.LinterJNDIDataSource"
          factory="com.relx.jdbc.jndi.LinterDataSourceFactory"
          user="SYSTEM"
          password="MANAGER8"
          url="jdbc:linter:linapid:localhost:1070:local"
          encoding="cp866"/>
```

   в) если сервер JDBC запущен в ОС Linux, установить encoding="koi8-r";
   г) добавить в `WEB-INF/web.xml` ссылку на ресурс:

```
<resource-ref>
    <res-ref-name>jdbc/DB</res-ref-name>
    <res-type>com.relx.jdbc.LinterJNDIDataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

   д) теперь в jsp или сервлетах можно использовать источник данных "jdbc/DB":
   - в сервлете:

```
ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/DB");
Connection conn = ds.getConnection( );
Statement stmt = conn.createStatement();
…
```

- в jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<sql:query var="autoList" dataSource="jdbc/DB">
SELECT * FROM AUTO
</sql:query>
```

Примечание
Для использования JNDI-интерфейса со старыми JDK, не имеющими встроенной поддержки DataSource интерфейса, необходимо вручную установить соответствующие пакеты.

Для указания кодировки символов, использующейся на сервере, в JNDI-интерфейс добавлено свойство encoding.

Для использования нового свойства, например, совместно с tomcat, необходимо:

1) положить файлы `linjdbc-1.x.jar` в `$TOMCAT_HOME/common/lib`;
2) в файле `$TOMCAT_HOME/conf/server.xml` или в файле приложения `META-INF/context.xml` в теге <Context> прописать источник данных:

```
<Resource name="jdbc/DB"
          auth="Container"
          type="com.relx.jdbc.LinterJNDIDataSource"
          factory="com.relx.jdbc.LinterJNDIDataSourceFactory"
          user="SYSTEM"
          password="MANAGER8"
          url="jdbc:linter:linapid:localhost:1070:local"
          encoding="cp866"/>
```

3) если `linapid` запущен в ОС Linux, установить encoding="koi8-r";
4) добавить в `WEB-INF/web.xml` ссылку на ресурс:

```
<resource-ref>
      <res-ref-name>jdbc/DB</res-ref-name>
      <res-type>com.relx.jdbc.LinterJNDIDataSource</res-type>
      <res-auth>Container</res-auth>
 </resource-ref>
```

5) теперь в jsp или сервлетах можно использовать источник данных "jdbc/DB":
   - в сервлете:

```
Context ctx = new InitialContext( );
DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/DB");
Connection conn = ds.getConnection( );
..
```

   - в jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
```

 © JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<sql:query var="autoList" dataSource="jdbc/DB">SELECT * FROM
 AUTO</sql:query>
```

Apache Cayenne

Для подсоединения к БД с помощью Apache Cayenne:

1) выбрать пакет для соответствующей версии Apache Cayenne

```
Версия apache cayenne: 2.0 – 2.0.4.
Версия диалекта: linter-cayenne2.
Каталог: v2
```

или

```
Версия apache cayenne: 3.0 – 3.1.
Версия диалекта: linter-cayenne3.
Каталог: v3
```

или

```
Версия apache cayenne: 3.2
Версия диалекта: linter-cayenne3.2
Каталог: v3.2
```

2) для сборки пакета диалекта перейти в соответствующий каталог и подать команду:

```
mvn clean install
```

(диалект будет установлен в репозиторий maven) или

```
mvn clean package
```

(диалект будет собран в jar-файл в каталоге target).

Примечание

Можно изменить файл `pom.xml`, указав конкретную версию apache cayenne, например,

```
<dependency>
    <groupId>org.apache.cayenne</groupId>
    <artifactId>cayenne</artifactId>
    <version>2.0.4</version>
</dependency>
```

Возможны и другие способы использования интерфейса (прямое подключение к проекту, сборка в составе apache cayenne и т.п.).

3) подключиться к СУБД ЛИНТЕР.

В случае если пакет диалекта установлен в репозиторий maven, то для подключения его к приложению достаточно в соответствующий файл `pom.xml` добавить зависимость от интерфейса, например, так:

```
<dependencies>
    <dependency>
        <groupId>ru.relex.linter</groupId>
        <artifactId>linter-cayenne3</artifactId>
```

```
        <version>1.0.1</version>
    </dependency>
</dependencies>
```

Для указания параметров доступа потребуются параметры подключения в connection.properties, например, такие:

```
linter.adapter = org.apache.cayenne.dba.linter.LinterAdapter
linter.cayenne.adapter =
 org.apache.cayenne.dba.linter.LinterAdapter
linter.jdbc.username = SYSTEM
linter.jdbc.password = MANAGER8
linter.jdbc.url = jdbc:linter:linapid:localhost:1070:local;
 ignoreTargetSqlType=true
linter.jdbc.driver = com.relx.jdbc.LinterDriver
```

или

```
<properties>
<adapter>org.apache.cayenne.dba.linter.LinterAdapter</adapter>
<cayenne.adapter>org.apache.cayenne.dba.linter.LinterAdapter
</cayenne.adapter>
         <jdbc.driver>com.relx.jdbc.LinterDriver</jdbc.driver>
          <jdbc.url>jdbc:linter:linapid:localhost:1070:local;
           emptyBlobAsNull=true</jdbc.url>
          <jdbc.user>SYSTEM</jdbc.user>
          <jdbc.pass>MANAGER8</jdbc.pass>
          <jdbc.isolation/>
</properties>
```

Следует помнить, что непосредственно для доступа к СУБД требуется ещё и JDBC-драйвер СУБД ЛИНТЕР linjdbc-1.4.jar или выше.

Hibernate

Для подсоединения к БД с помощью Hibernate:

1) выбрать пакет для соответствующей версии Hibernate. Соответствие версии Hibernate и версии диалекта в таблице 4.

Таблица 4. Соответствие версии Hibernate и версии диалекта

Версия Hibernate	Версия диалекта
3.0.3 – 3.5.6	linter_hibernate3
3.6.0 – 3.6.9	linter_hibernate3.6
4.0 – 4.3.11	linter_hibernate4
5.0.0 – 5.0.7	linter_hibernate5
5.0.8, 5.0.9	linter_hibernate5.0.8
5.1	linter_hibernate5.1

© JDBC-драйвер. АО НПП «РЕЛЭКС», 1990-2026

Версия Hibernate	Версия диалекта
hibernate spatial **Примечание** Для версии hibernate до 5.x требуется наличие модуля linter_hibernate4	linter_spatial

2) для сборки пакета диалекта перейти в соответствующий каталог и подать команду:

```
mvn clean install
```

(диалект будет установлен в репозиторий maven) или

```
mvn clean package
```

(диалект будет собран в jar-файл в каталоге target).

Примечание
Можно изменить файл `pom.xml`, указав конкретную версию hibernate, например,

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.0.0</version>
    </dependency>
</dependencies>
```

Возможны и другие способы использования интерфейса (прямое подключение к проекту, сборка в составе hibernate и т. п.).

3) подключиться к СУБД ЛИНТЕР.

В случае если пакет диалекта установлен в репозиторий maven, то для подключения его к приложению достаточно в соответствующий файл `pom.xml` добавить зависимость от интерфейса, например, так:

```
<dependencies>
    <dependency>
        <groupId>ru.relex.lintersql</groupId>
        <artifactId>linter-hibernate4</artifactId>
        <version>1.0.1</version>
    </dependency>
</dependencies>
```

Для указания параметров доступа потребуются параметры подключения, например, такие:

```
<properties>
  <db.dialect>org.hibernate.dialect.LinterDialect</db.dialect>
  <jdbc.driver>com.relx.jdbc.LinterDriver</jdbc.driver>
  <jdbc.url>jdbc:linter:linapid:localhost:1070:local;
```

```
    emptyBlobAsNull=true</jdbc.url>
  <jdbc.user>SYSTEM</jdbc.user>
  <jdbc.pass>MANAGER8</jdbc.pass>
  <jdbc.isolation/>
</properties>
```

Следует помнить, что непосредственно для доступа к СУБД требуется ещё и JDBC-драйвер СУБД ЛИНТЕР linjdbc-1.x.jar.

Для работы диалекта hibernate spatial необходимо наличие диалекта hibernate и JDBC-драйвера.

Приложение 1
Пример java-модуля

Исходный текст примера

```
import com.relx.jdbc.LinterDriver;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class test
{
  public static void main(String[] args) throws SQLException
  {
    Driver driver = null;
    Connection connection = null;
    Statement statement = null;
    ResultSet resultSet = null;
    try
    {
      driver =
 (Driver)Class.forName("com.relx.jdbc.LinterDriver").newInstance();
      connection = DriverManager.getConnection
        ("jdbc:linter:linapid:localhost:1070:local", "SYSTEM",
 "MANAGER8");
      statement = connection.createStatement();
      resultSet = statement.executeQuery("SELECT DISTINCT STATUS
 from S");

      while (resultSet.next())
      {
        System.out.println(resultSet.getString(1));   // Print col
 1
      }
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      if(resultSet != null)
      {
```

```
        resultSet.close();
      }
      if(statement != null)
      {
        statement.close();
      }
      if(connection != null)
      {
        connection.close();
      }
    }
  }
}
```

Трансляция примера

```
SET JAVA_HOME=C:\PROGRA~1\Java\jdk1.8.0_31
SET LINTER_HOME=C:\Linter
SET JAVA=%JAVA_HOME%\bin\java
SET JAVAC=%JAVA_HOME%\bin\javac
SET CLASSPATH=%LINTER_HOME%\jdbc\linjdbc-1.8.jar
%JAVAC% -classpath %CLASSPATH% test.java
```

Выполнение примера

```
java -cp .;C:\Linter\jdbc\linjdbc-1.8.jar test
```

Приложение 2
Примеры XML-файлов для подключения к СУБД ЛИНТЕР

hibernate.cfg.xml

Фрагмент hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
   PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
       <property
 name="connection.driver_class">com.relx.jdbc.LinterDriver</
property>
       <property name="connection.url">
          jdbc:linter:linapid:localhost:1070:local
       </property>
       <property name="connection.username">SYSTEM</property>
       <property name="connection.password">MANAGER8</property>
       <property name="connection.datasource">java:comp/env/jdbc/
DB</property>
       <property
 name="dialect">org.hibernate.dialect.LinterDialect</property>
...
   </session-factory>
</hibernate-configuration>
```

web.xml для tomcat

Фрагмент web.xml для tomcat

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

...

   <resource-ref>
       <description>Base Main Database</description>
       <res-ref-name>jdbc/DB</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
```

```
        <res-auth>Container</res-auth>
        <res-sharing-scope>Shareable</res-sharing-scope>
    </resource-ref>

...

</web-app>
```

context.xml для tomcat

Фрагмент context.xml для tomcat

Примечание
Данный файл связывает JNDI-интерфейс с конкретным драйвером и настройками.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/base" debug="5" reloadable="true"
 crosContext="true">

...

    <Resource name="jdbc/DB" auth="Container"
        factory="com.relx.jdbc.LinterJNDIDataSourceFactory"
        type="com.relx.jdbc.LinterJNDIDataSource"
        url="jdbc:linter:linapid:localhost:1070:local"
 user="SYSTEM"
        password="MANAGER8" encoding="CP866" />

...
</Context>
```

Вместо `context.xml` в других серверах приложений может использоваться иной способ задания JDBC-ресурсов через JNDI-интерфейс (в SunApplicationServer это осуществляется в gui).

В любом случае надо указать:

- имя ресурса, по которому будет выполняться поиск: "jdbc/DB";
- фабрика для создания DataSource: "com.relx.jdbc.LinterJNDIDataSourceFactory";
- конкретная реализация DataSource: "com.relx.jdbc.LinterJNDIDataSource";
- url к БД: "jdbc:linter:linapid:localhost:1070:local";
- имя пользователя: "SYSTEM";
- пароль пользователя: "MANAGER8";
- свойство, влияющее на кодировку соединения: "encoding": "CP866".