

# Как преодолеть несовместимости J2SE 1.3-1.4

*На помощь приходят Reflection API и Ant.*

JDBC Tutorial

Сэм Меффорд

Реализация одного из многочисленных Java API может оказаться трудной работой. Часто необходимо реализовывать множество взаимозависимых интерфейсов. Спрос на новые функциональные возможности движет созданием новых версий Java API, и поставщики должны постоянно обновлять свои реализации, чтобы удовлетворять этот спрос. В то время как сложность и постоянные изменения понятны и даже ожидаемы, несовместимость версий API, вынуждающая поддерживать различные варианты кода для более новых версий, может оказаться исключительно болезненной. Эта статья демонстрирует методы преодоления несовместимостей версии интерфейса, указывая, как компилировать один и тот же код для разных версий API. *(В оригинальной статье на английском языке 1500 слов)*

К стандартному набору библиотек Java добавлены многочисленные API, например, Java Database Connectivity Java (JDBC). Это помогает более широкой аудитории осваивать API, так как дополнительные пакеты не нужно связывать с развёртыванием. Для групп, пишущих реализации этих популярных API, более широкое принятие делает их предложения более ценными. Однако эти группы могут предпочитать, чтобы их API оставались опциональными (а не включёнными в набор стандартных библиотек Java), когда в свет выходит новая версия Java с изменённым API, зависящим от классов и методов, недоступных в предыдущих версиях Java. Внезапно оказывается, что нужно поддерживать две версии реализации: соответствующую старому API и соответствующую новому API. Именно это произошло с JDBC API в Java 2 Platform, Standard Edition (J2SE) версии 1.4. Из-за изменений в JDBC API реализация `java.sql.Connection` не может компилироваться и под J2SE 1.3, и под 1.4.

Вы, возможно, оказывались в таком же затруднительном положении как и я: я должен был реализовать интерфейсы JDBC типа `java.sql.Connection`, но мой код нужно было компилировать и под J2SE 1.3, и под 1.4. Я не хотел поддерживать различные файлы исходного кода для J2SE 1.3 и 1.4, так что я искал лучшее решение.

К сожалению, пресловутая мантра Java «Один раз написать, где угодно запускать» не подразумевает «Один раз написать, где угодно компилировать», если при компиляции вы полагаетесь на `javac`. К счастью, помочь могут уловки с кодом с помощью Reflection API и уловки с компиляцией с помощью Ant. Можно иметь только один набор исходных `.java`-файлов, а Ant поможет компилировать их и под J2SE 1.3, и под 1.4. Ant позволяет на лету вносить в `.java`-файлы изменения, соответствующими версии Java, используемой для компиляции. Но прежде чем полностью объяснить решение, я должен полностью объяснить проблему.

### 1. Пул подключений для бедняка

Два года назад, моя компания нуждалась в пуле подключений JDBC, но платить за это не хотела. В то время мы не могли найти хорошую бесплатную альтернативу, так что мы написали пул подключений сами. Чтобы лучше отслеживать, как подключения использовались в наших приложениях, мы создали `com.icentris.sql.ConnectionWrapper`, реализующий `java.sql.Connection`, и некоторые другие классы-обёртки, реализующие другие интерфейсы `java.sql`. Классы-обёртки только отслеживают использование базы данных в нашем приложении и затем перенаправляют запросы метода реальному ресурсу JDBC.

Когда в свет вышел J2SE 1.4, мы, естественно, захотели перевести на него некоторых наших клиентов, чтобы они могли извлечь пользу из многих улучшений в нём. Но, конечно, мы все ещё должны были поддерживать J2SE 1.3 для клиентов, которые не видели необходимости в модернизации. К нашему огорчению, `ConnectionWrapper` и другие наши классы-обёртки JDBC без изменений не компилировались под J2SE 1.4. Чтобы не усложнять эту статью, я буду использовать `ConnectionWrapper` для демонстрации методов, которые я применил ко всем классам, чтобы они компилировались и под J2SE 1.3, и под 1.4. Чтобы соответствовать обновлённому JDBC API, мне пришлось добавить в `ConnectionWrapper` несколько методов, что

## Как преодолеть несовместимости J2SE 1.3-1.4

создало две большие проблемы:

1. Так как мои классы-обёртки должны передавать вызовы методов, я должен был бы вызывать методы, не существующие в `sql`-классах J2SE 1.3.
2. Так как некоторые новые методы полагаются на новые классы, я должен был бы предусмотреть зависимости от классов, не существующих в J2SE 1.3.

## 2. На помощь приходит Reflection

Несколько примеров кода помогут лучше объяснить первую проблему. Поскольку мой `ConnectionWrapper` обёртывает `java.sql.Connection`, все мои примеры зависят от переменной экземпляра `realConnection` (выделено жирным), создаваемой в конструкторе:

```
private java.sql.Connection realConnection = null;
public ConnectionWrapper(java.sql.Connection connection) {
    realConnection = connection;
}
```

Чтобы увидеть, что я сделал бы, если бы не было проблем несовместимости, рассмотрим `setHoldability(int)` (новый метод в `java.sql.Connection` в J2SE 1.4):

```
public void setHoldability(int holdability) throws SQLException {
    realConnection.setHoldability( holdability );
}
```

К сожалению, этот код не компилируется под J2SE 1.3, потому что `java.sql.Connection` не имеет метода `setHoldability()`, который я могу вызвать под J2SE 1.3. Но чтобы скомпилировать код под J2SE 1.4, я должен иметь метод `setHoldability()`, чтобы должным образом реализовать API. Чтобы решить эту проблему, я допустил, что мой метод `setHoldability()` будет только вызываться только под J2SE 1.4, так что я мог бы использовать Reflection API для вызова метода:

```
public void setHoldability(int holdability) throws SQLException {
    Class[] argTypes = new Class[] { Integer.TYPE };
    Object[] args = new Object[] { new Integer(holdability) };
}
```

```
    callJava14Method("setHoldability", realConnection, argTypes, args);
}
public static Object callJava14Method(String methodName, Object instance,
Class[] argTypes, Object[] args)
    throws SQLException
{
    try {
        Method method = instance.getClass().getMethod(methodName, argTypes);
        return method.invoke(instance, args );
    } catch (NoSuchMethodException e) {
        e.printStackTrace();
        throw new SQLException("Ошибка при вызове метода (" + methodName + "): "
+ e);
    } catch (IllegalAccessException e) {
        e.printStackTrace();
        throw new SQLException("Ошибка при вызове метода (" + methodName + "): "
+ e);
    } catch (InvocationTargetException e) {
        e.printStackTrace();
        throw new SQLException("Ошибка при вызове метода (" + methodName + "): "
+ e);
    }
}
```

Теперь у меня есть метод `setHoldability()`, так что могу компилировать код под J2SE 1.4. Я непосредственно не вызываю прежде несуществующий метод в моем `java.sql.Connection`, так что я могу произвести компиляцию и под J2SE 1.3. Мой метод `callJava14Method()` использует Reflection API, чтобы вызывать метод, затем обёртывает любые ошибки в `SQLException`, так как это всё, что я должен выбрасывать. Я использовал эту стратегию для всех новых методов J2SE 1.4, так что мой код будет работать должным образом под 1.4, вызывая обёрнутый метод, и при этом компилироваться под 1.3. Теперь мне осталось решить вторую проблему и найти способ построить зависимости от классов, не существующих в J2SE 1.3.

### 3. Ответ — Ant

В J2SE 1.4 `java.sql.Connection` зависит от нового класса: `java.sql.Savepoint`. Так как этот новый класс находится в пакете `java.sql`, его

## Как преодолеть несовместимости J2SE 1.3-1.4

нельзя добавить к J2SE 1.3. Java не позволяет никаких сторонних добавлений к набору основных классов в пакетах `javax.*` или `java.*`. Так что это стало для меня проблемой: использовать новый класс `java.sql.Savepoint`, чтобы написать код, который работал бы под J2SE 1.4, и одновременно обеспечить, чтобы код компилировался под J2SE 1.3, где этот класс не существует. Просто, правда? Все, кто ответил «Да!», возьмите пирожок с полки. Ну, по крайней мере, просто теперь, когда я нашёл ответ.

Сначала я включил следующее условие импорта:

```
// Comment_next_line_to_compile_with_Java_1.3
import java.sql.Savepoint;
```

Затем я нашел способ для Ant закомментировать этот импорт при компиляции под J2SE 1.3. Вот ключевая часть сценария Ant в упрощённом виде:

```
<replace>
  <replacetoken>Comment_next_line_for_Java_1.3&#010;</replacetoken>
  <replacevalue>Comment_next_line_for_Java_1.3&#010;
```

## Как преодолеть несовместимости J2SE 1.3-1.4

компилироваться без инструкции импорта. Тем не менее, когда инструкция импорта незакомментирована (при компиляции под J2SE, 1.4), мой класс `Savepoint` игнорируются из-за более конкретного определения импорта. Так что я создал мой собственный фиктивный класс с именем `com.icentris.sql.Savepoint`, который (исключая комментарии javadoc), вероятно, является самым коротким допустимым классом:

```
package com.icentris.sql;

/** Фиктивный класс, позволяющий ConnectionWrapper реализовать java.sql.Connection
 * и при этом компилироваться под J2SE 1.3 и J2SE 1.4. При компиляции
 * под J2SE 1.3 этот класс компилируется как заполнитель вместо
 * java.sql.Savepoint (отсутствующего в J2SE 1.3). При компиляции
 * под J2SE 1.4 этот класс игнорируется, и ConnectionWrapper использует
 * java.sql.Savepoint, который является новым в J2SE 1.4.
 */
public class Savepoint {}
```

Под J2SE 1.4 я могу теперь должным образом импортировать `java.sql.Savepoint`. Под J2SE 1.3, Ant комментирует строку импорта, так что на месте `Savepoint`, упомянутого в моем коде, оказывается фиктивный класс, находящийся в том же самом пакете. Теперь я могу добавить все методы, ссылающиеся на `Savepoint`, и все еще использовать уловку с `Reflection`, которую объяснил ранее:

```
// Comment_next_line_to_compile_with_Java_1.3
import java.sql.Savepoint;

. . .
public Savepoint setSavepoint() throws SQLException {
    Class[] argTypes = new Class[0];
    Object[] args = new Object[0];
    return (Savepoint) callJava14Method("setSavepoint", realConnection,
    argTypes, args);
}

public Savepoint setSavepoint(String name) throws SQLException {
    Class[] argTypes = new Class[] { String.class };
}
```

## Как преодолеть несовместимости J2SE 1.3-1.4

```
    Object[] args = new Object[] { name };
    return (Savepoint) callJava14Method("setSavepoint", realConnection,
    argTypes, args);
}

public void rollback(Savepoint savepoint) throws SQLException {
    Class[] argTypes = new Class[] { Savepoint.class };
    Object[] args = new Object[] { savepoint };
    callJava14Method("rollback", realConnection, argTypes, args);
}

public void releaseSavepoint(Savepoint savepoint) throws SQLException {
    Class[] argTypes = new Class[] { Savepoint.class };
    Object[] args = new Object[] { savepoint };
    callJava14Method("releaseSavepoint", realConnection, argTypes, args);
}
```

Теперь всё, что мне нужно, — чтобы задание Ant `compile` определило J2SE 1.3 и закомментировало импорт на лету, если кто-то попытается использовать J2SE 1.3, чтобы скомпилировать `ConnectionWrapper`:

```
<target name="compile">
  <antcall target="undoJava13Tweaks" />
  <antcall target="doJava13Tweaks" />
  <javac srcdir="src" destdir="WEB-INF/classes" debug="on">
    <classpath>
      <fileset dir="WEB-INF/lib">
        <include name="*.jar"/>
      </fileset>
    </classpath>
  </javac>
  <antcall target="undoJava13Tweaks" />
</target>

<target description="Определить, идёт ли компиляция под Java 1.3"
name="isJava13">
  <echo message="java.specification.version=[${java.specification.version}]" />
  <condition property="isJava13">
    <equals arg1="${java.specification.version}" arg2="1.3" />
  </condition>
</target>
```

```
</target>

<target description="Вот пара моих уловок, чтобы произвести компиляцию под Java 1.3"
  name="doJava13Tweaks" depends="isJava13" if="isJava13">
  <echo message="Это Java 1.3, сейчас похимичим!" />
  <replace dir="src/com/icentris/" summary="true">
    <include name="sql/ConnectionWrapper.java" />
    <replacetoken>Comment_next_line_for_Java_1.3>#010;</replacetoken>
    <replacevalue>Comment_next_line_for_Java_1.3>#010;
```

## Как преодолеть несовместимости J2SE 1.3-1.4

зависимости от классов, которые могут не существовать в версии Java, с которой происходит компиляция. В то время как приведённые примеры упрощены для целей демонстрации, я успешно использовал эти и подобные методы для решения различных проблем, связанных с одновременной поддержкой J2SE версий 1.3 и 1.4. Так как так многие API Java постоянно изменяются, вы можете использовать эти методы, чтобы избежать необходимости поддержания двух вариантов кода.

## 5. Об авторе

В качестве главного разработчика для iCentris Сэм Меффорд ставит во главу угла совместимость. Он возглавляет группу, обеспечивающую возможность развёртывания одинакового кода для многих компаний; на серверах приложений, включая Tomcat, WebLogic, Resin, Orion и WebSphere; на СУБД, включая Oracle, PostgreSQL, MySQL и Informix; и на множественных средах выполнения Java.

## 6. Ресурсы

- API для java.sql.Connection (J2SE 1.3):  
<http://java.sun.com/j2se/1.3/docs/api/java/sql/Connection.html>
- API для java.sql.Connection (J2SE 1.4):  
<http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Connection.html>
- JDBC API:  
<http://java.sun.com/products/jdbc/>
- Содержание документации по Reflection:  
<http://java.sun.com/j2se/1.3/docs/guide/reflection/spec/java-reflectionTOC.doc.html>
- Учебник по Reflection API:  
<http://java.sun.com/docs/books/tutorial/reflect/>
- Javadoc (java.lang.reflect):  
<http://java.sun.com/j2se/1.3/docs/api/java/lang/reflect/package-summary.html>
- Дополнительные статьи о Reflection API на *JavaWorld*:
  - [«Reflection vs. Code Generation»](#), Майкл Дж. Реттиг и Мартин Фаулер (ноябрь 2001 г.)
  - [«Untangle Your Servlet Code with Reflection»](#), Джереми Рочелл (декабрь 2000 г.)
  - [«Создание EJB из любого Java класса с помощью Java Reflection»](#), Тони Лоутон

- (декабрь 2000 г.)
- [«Eliminate Tedious Programming: Recover Data with XML and Reflection»](#),  
Абхилаш Конери (ноябрь 2000 г.)
  - Домашняя страница Ant:  
<http://ant.apache.org/>
  - Руководство пользователя Apache Ant:  
<http://ant.apache.org/manual>
  - «Автоматизируйте процесс сборки Java программ с помощью Ant», Майкл Цимерман:  
[http://www.javaworld.com/javaworld/10\\_00/03/](http://www.javaworld.com/javaworld/10_00/03/)
  - Раздел по **Java Database Connectivity (JDBC)** на *JavaWorld*:  
[http://www.javaworld.com/channel\\_content/jw-jdbc-index.shtml](http://www.javaworld.com/channel_content/jw-jdbc-index.shtml)
  - Раздел по **Java 2 Platform, Standard Edition (J2SE)** на *JavaWorld*:  
[http://www.javaworld.com/channel\\_content/jw-j2se-index.shtml?j2se1](http://www.javaworld.com/channel_content/jw-j2se-index.shtml?j2se1)
  - Раздел по **API** на *JavaWorld*:  
[http://www.javaworld.com/channel\\_content/jw-apis-index.shtml](http://www.javaworld.com/channel_content/jw-apis-index.shtml)
  - Выскажите на форуме JavaWorld:  
<http://www.javaworld.com/javaforums/ubbthreads.php?Cat=&C==2>
  - Подпишитесь на бесплатные еженедельные информационные бюллетени *JavaWorld* по электронной почте:  
<http://www.javaworld.com/subscribe>

Reprinted with permission from the September 2003 edition of JavaWorld magazine.  
Copyright © ITworld.com, Inc., an IDG Communications company.

View the original article at: <http://www.javaworld.com/javaworld/jw-09-2003/jw-0926-overcome.html>

[Перевод на русский © Дмитрий Габинский. 2003](#)