Java Try with Resources

JULY 19, 2016 BY PANKAJ — 6 COMMENTS

Today we will look into Java Try with Resources. One of the Java 7 feature is **try-with-resources** statement for automatic resource management.

Try with Resources

A resource is an object that must be closed once your program is done using it. For example a File resource or JDBC resource for database connection or a Socket connection resource. Before Java 7, there was no auto resource management and we should explicitly close the resource once our work is done with it.

Usually, it was done in the `finally` block of a `try-catch` statement. This approach used to cause memory leaks and performance hit when we forgot to close the resource.

Let's see a pseudo code snippet to understand this java try with resources feature.

Before Java 7:

```
try{
        //open resources like File, Database connection, Sockets etc
} catch (FileNotFoundException e) {
        // Exception handling like FileNotFoundException, IOException etc
}finally{
        // close resources
}
```

Java 7 try with resources implementation:

```
try(// open resources here){
    // use resources
} catch (FileNotFoundException e) {
        // exception handling
}
// resources are closed as soon as try-catch block is executed.
```

Let's write a simple program to read a file and print the first line using Java 6 or older versions and Java 7 **try-with-resources** implementation.

Java 6 Resource Management Example

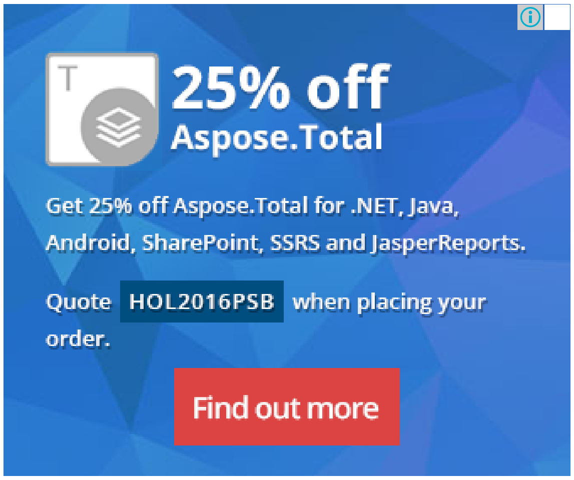
```java
package com.journaldev.util;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Java6ResourceManagement {

        public static void main(String[] args) {
                BufferedReader br = null;
                try {
                        br = new BufferedReader(new FileReader("C:\\journaldev.txt"));
                        System.out.println(br.readLine());
                } catch (IOException e) {
                        e.printStackTrace();
                } finally {
                        try {
                                if (br != null)
                                        br.close();
                        } catch (IOException ex) {
                                ex.printStackTrace();
                        }
                }
        }
}
```

Java 7 Try With Resources Example

```
package com.journaldev.util;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Java7ResourceManagement {

	public static void main(String[] args) {
		try (BufferedReader br = new BufferedReader(new FileReader(
				"C:\\journaldev.txt"))) {
			System.out.println(br.readLine());
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Java try with resources benefits

Some of the benefits of using try with resources in java are;



1. More readable code and easy to write.
2. Automatic resource management.
3. Number of lines of code is reduced.
4. No need of finally block just to close the resources.
5. We can open multiple resources in try-with-resources statement separated by a semicolon. For example, we can write following code:

```
try (BufferedReader br = new BufferedReader(new FileReader(
                                "C:\\journaldev.txt"));
                                java.io.BufferedWriter writer =
java.nio.file.Files.newBufferedWriter(FileSystems.getDefault().getPath("C:\\journald
 Charset.defaultCharset())) {
                        System.out.println(br.readLine());
                } catch (IOException e) {
                        e.printStackTrace();
                }
```

6. When multiple resources are opened in try-with-resources, it closes them in the reverse order to avoid any dependency issue. You can extend my resource program to prove that.

Java 7 has introduced a new interface `java.lang.AutoCloseable` that extends `java.io.Closeable` interface. To use any resource in try-with-resources, it must implement `AutoCloseable` interface else java compiler will throw compilation error.

Lets confirm this with an example:

```
package com.journaldev.util;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.FileSystems;

public class Java7ResourceManagement {

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

```
                        e.printStackTrace();
                }
                System.out.println("Out of try-catch block.");
        }

        static class MyResource implements AutoCloseable{

                @Override
                public void close() throws Exception {
                        System.out.println("Closing MyResource");
                }

        }
}
```

Output of above program is:

```
MyResource created in try-with-resources
Closing MyResource
Out of try-catch block.
```

From the output it's clear that as soon as try-catch block is finished, resource close method is called.

Try with Resources Exceptions

There is one difference to note between try-catch-finally and try-with-resources in case of exceptions.

If an exception is thrown in both try block and finally block, the method returns the exception thrown in finally block.

For try-with-resources, if exception is thrown in try block and in try-with-resources statement, then method returns the exception thrown in try block.

To better clarify this difference, we will see a sample code.

```
package com.journaldev.util;

public class Java7ResourceManagement {

        public static void main(String[] args) throws Exception {
```

```
                }
                try {
                        normalTryException();
                } catch (Exception e) {
                        System.out.println(e.getMessage());
                }
        }

        private static void normalTryException() throws Exception {
                MyResource mr = null;
                try {
                        mr = new MyResource();
                        System.out.println("MyResource created in try block");
                        if (true)
                                throw new Exception("Exception in try");
                } finally {
                        if (mr != null)
                                mr.close();
                }

        }

        private static void tryWithResourceException() throws Exception {
                try (MyResource mr = new MyResource()) {
                        System.out.println("MyResource created in try-with-
```

Output of the above program is:

```
MyResource created in try-with-resources
Closing MyResource
Exception in try
MyResource created in try block
Closing MyResource
Exception in Closing
```

The output of the program proves the difference given above. Thats all for the java 7 try-with-resources.

FILED UNDER: JAVA

articles directly.
Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

« Java switch case String

How to fix maven Out of Memory issu

Comments

Arun says
NOVEMBER 28, 2016 AT 4:58 AM

HI Pankaj,

I am regularly reading your articles and I have gained a lot of knowledge from this site. Thanks. In this article it is mentioned that
Java 7 has introduced a new interface java.lang.AutoCloseable that extends java.io.Closeable interface. To use any resource in try-with-resources,

but Closeable interface extends AutoCloseable interface right. Please correct me if I am wrong.

Regards
Arun

Reply

thanu says

MARCH 23, 2016 AT 12:38 AM

Java is excellent programming language.we also provide the training with more job opportunities.it is used to get better knowledge about the java.

Reply

Advanced java Training says

JULY 19, 2013 AT 1:56 PM

try (int y = a / b;)

Reply

Ralph says

NOVEMBER 6, 2012 AT 7:16 AM

Thanks for the information. One thing worth mentioning is that try() only closes resources on objects if their variable declaration is located in the try (….) statement.

The following code does not call close of object B because there is no variable declaration for B.

```
public class T {
public static void main(String[] args) throws Exception {
try (A a = new A(new B())) {
System.out.println(“inside arm block”);
}
}
private static class A implements AutoCloseable {
public A(B b) {}
public void close() throws Exception {
System.out.println(“running A.close()”);
}
}

private static class B implements AutoCloseable {
```

```
public void close() throws Exception {
System.out.println("running B.close()");
}
}
}
```

But it calls B.close if the ARM block looks like this:

```
try (B b = new B(); A a = new A(b)) {

}
```

Reply

Thats a great point to note Ralph, thanks for the code to understand this.

Reply

Rohit says

NOVEMBER 6, 2012 AT 1:26 AM

ARM Blocks are I guess most useful feature of Java 7 along with String in Switch. These helps to perfor daily programming activity with ease.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

10
Shares

POST COMMENT

8

2

SpreadsheetGear
Performance Spreadsheet Components
"Over 100 Times Faster..."
Amy Tate, Chief .NET Architect
Applied OLAP Inc
Download Free Trial

10
Shares

E-Mail Address

I AM IN!

8

2

RECOMMENDED TUTORIALS

Java Tutorials: Java IO Tutorial, Java Regular Expressions Tutorial, Multithreading in Java, Java Logging AF Tutorial, Java Annotations,Java XML Tutorial, Collections in Java, Java Gener Exception Handling in Java, Java Reflection, Java Design Patterns, JDBC Tutorial
Java EE: Servlet JSP Tutorial, Struts2 Tutorial, Spring Tutorial, Hibernate Tutorial, Primefaces Tutorial
Web Services: Apache Axis 2 Tutorial, JAX-RS Web Services Tutorial
Misc: Memcached Tutorial

Resources: Free eBooks, My Favorite Web Hosting

IMPORTANT INTERVIEW QUESTIONS

Java String Interview Questions, Java Multithreading Interview Questions, Java Programming Interview Questions, Java Interview Questions, Java Collections Interview Questions, Java Exception Interview Questions

Servlet Interview Questions, JSP Interview Questions, Struts2 Interview Questions, JDBC Interview Questions, Spring Interview

10
Shares

8

2