YOU ARE HERE: HOME > CORE JAVA > MULTI THREADING > INTER-THREAD COMMUNICATION USING PIPED STREAMS IN JAVA

Inter-thread communication using piped streams in java

NOVEMBER 17, 2012 LOKESH 2 COMMENTS

Follow 2.6k +5 Recommend this on Google

This topic has been a popular interview question for a long time. With the JDK 1.5 release, ExecutorService and BlockingQueue brought another way of doing it more effectively, but this approach is also worth knowing and might be useful in certain scenarios.

Sections in this post:

What are piped streams

Pipe streams are just like real plumbing pipes. You put things into to a pipe at one end – using some methods. Then you receive the same things back from the pipe stream at the other end – using some other methods. They come out in **FIFO order**, first-in first-out, just like from real plumbing pipes.

PipedReader and PipedWriter

PipedReader is an extension of **Reader** class which is used for reading character streams. Its read() method reads the connected PipedWriter's stream. Similarly, **PipedWriter** is an extension of Writer class and does all the things which Reader class contracts.

A writer can be connected to a reader by following two methods:

1. Using constructor PipedWriter(PipedReader pr)
2. Using connect(PipedReader pr) method

Once connected through any of above ways, any thread can write data in stream using write(....) methods, and data will be available to reader and can be read using read() method.

Running example

Below given sample program creates two threads. One thread is responsible for writing into stream and second one is only reading the data to print them in console.

```
package multiThread;

```

YOU MAY ALSO LIKE

Read file line by line in java 8: Streams of Lines Example

Java 8 Tutorial: Streams by Examples

Difference between "implements

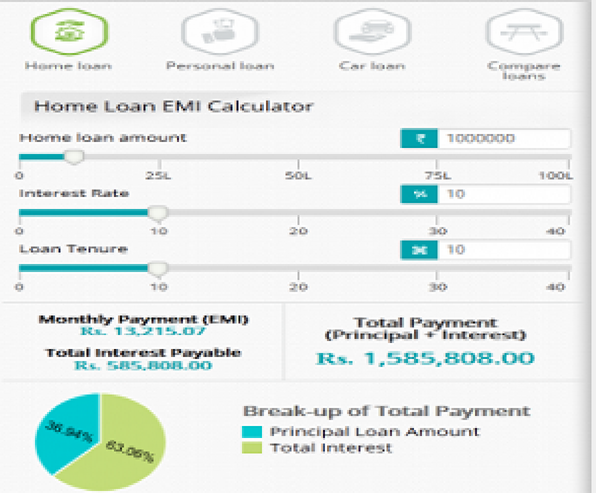
```java
import java.io.*;

public class PipedCommunicationTest {

<pre><code>public static void main(String[] args) {
    new PipedCommunicationTest();
}

public PipedCommunicationTest() {
    try {
        // Create writer and reader instances
        PipedReader pr = new PipedReader();
        PipedWriter pw = new PipedWriter();
        // Connect the writer with reader
        pw.connect(pr);
        // Create one writer thread and one reader thread
        Thread thread1 = new Thread(
                new PipeReaderThread("ReaderThread", pr));
        Thread thread2 = new Thread(
                new PipeWriterThread("WriterThread", pw));
        // start both threads
        thread1.start();
        thread2.start();
    } catch (Exception e) {
        System.out.println("PipeThread Exception: " + e);
    }
}
</code></pre>

}

class PipeReaderThread implements Runnable {
    PipedReader pr;
    String name = null;

<pre><code>public PipeReaderThread(String name, PipedReader pr) {
    this.name = name;
    this.pr = pr;
}

public void run() {
    try {
        // continuously read data from stream and print it in console
        while (true) {
            char c = (char) pr.read(); // read a char
            if (c != -1) { // check for -1 indicating end of file
                System.out.print(c);
            }
        }
    } catch (Exception e) {
        System.out.println(" PipeThread Exception: " + e);
```

Runnable” and “extends Thread” in java

Thread synchronization, object level locking and class level locking

How to read data from InputStream into String in java

Base64 encoding and decoding example in java 8



```
    }
}
</code></pre>

}

class PipeWriterThread implements Runnable {
    PipedWriter pw;
    String name = null;

<pre><code>public PipeWriterThread(String name, PipedWriter pw) {
    this.name = name;
    this.pw = pw;
}

public void run() {
    try {
        while (true) {
            // Write some data after every two seconds
            pw.write("Testing data written...n");
            pw.flush();
            Thread.sleep(2000);
        }
    } catch (Exception e) {
        System.out.println(" PipeThread Exception: " + e);
    }
}
</code></pre>

}

Output:

Testing data written...
Testing data written...
Testing data written...
```

Important notes

- You cannot write to a pipe without having some sort of reader created and connected to it. In other words, both ends must be present and already connected for the writing end to work.
- You cannot switch to another reader, to which the pipe was not originally connected, once you are done writing to a pipe.
- You cannot read back from the pipe if you close the reader. You can close the writing end successfully, however, and still read from the pipe.

- You cannot read back from the pipe if the thread which wrote to it ends.

Happy Learning !!

Related Posts:

1. Read file line by line in java 8: Streams of Lines Example
2. Java 8 Tutorial: Streams by Examples
3. Difference between “implements Runnable” and “extends Thread” in java
4. Thread synchronization, object level locking and class level locking
5. How to read data from InputStream into String in java
6. Base64 encoding and decoding example in java 8

CHARACTER STREAMS INTER THREAD COMMUNICATION MULTI THREADING PIPED STREAMS

PREVIOUS POST

SQLException: Access denied for user ‘root’@’localhost’ after re-installation of MySQL server

NEXT POST

Hibernate many-to-many mapping using annotations

2 THOUGHTS ON “INTER-THREAD COMMUNICATION USING PIPED STREAMS IN JAVA”

Manjunath.S

JULY 27, 2014 AT 4:15 AM

Hello lokesh,
i have one scenario in java i am struggling to do that could you please help me.
scenario is
1.creating two process make communicate with each other using named pipes.
2. if i send the text in process1 it must display it in other process,that is process2 and vice verse.

thanks in advance

↳ REPLY

Surjeet

DECEMBER 16, 2013 AT 10:53 AM

Hi lokesh, at one point i read, all primitives types support atomic operation except long and double, if we declare a field of type int, is it necessary to declare that variable as volatile

↳ REPLY

Note:- In comment box, please put your code inside **[java] ... [/java]** OR **[xml] ... [/xml]** tags otherwise it may not appear as intended.

LEAVE A REPLY

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

Name *

Email *

Website

Are you Human and NOT robot? Solve this to prove !! *

four + 1 =

Comment

You may use these HTML tags and attributes:

```
<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>
```

POST COMMENT

□ Notify me of follow-up comments by email.

▣ Notify me of new posts by email.

SUBSCRIBE TO BLOG VIA EMAIL

Join 3,666 other subscribers

Email Address

Email Address

SUBSCRIBE

REFERENCES

- Oracle Java Docs
- Spring 3 Reference
- Spring 4 References
- Jboss RESTEasy JAX-RS
- Hibernate Developer Guide
- Junit Wiki
- Maven FAQs

Dzone Dev Links

JSP Homepage

ANT Homepage

META LINKS

Advertise

Share your knowledge

Privacy policy

Contact Us

About Us

POPULAR POSTS

Spring 3 and hibernate integration tutorial with example

Reading/writing excel files in java : POI tutorial

How hashmap works in java

Singleton design pattern in java

Useful java collection interview questions

Understanding hibernate first level cache with example

How hibernate second level cache works?

Working with hashCode and equals methods in java

How to make a java class immutable

4 ways to schedule tasks in Spring 3 : @Scheduled example