Read/convert an InputStream to a String

If you have `java.io.InputStream` object, how should you process that object and produce a `String` ?

Suppose I have an `InputStream` that contains text data, and I want to convert this to a `String` (for example, so I can write the contents of the stream to a log file).

What is the easiest way to take the `InputStream` and convert it to a `String` ?

```
public String convertStreamToString(InputStream is) {
    // ???
}
```

java string io stream inputstream

edited Jul 16 '12 at 20:32
Robert Harvey ♦ 132k 30 232 361

asked Nov 21 '08 at 16:47
Johnny Maelstrom 12.8k 5 14 17

624 Boy, I'm absolutely in love with Java, but this question comes up so often you'd think they'd just figure out that the chaining of streams is somewhat difficult and either make helpers to create various combinations or rethink the whole thing. – Bill K Nov 21 '08 at 17:16

16 The answers to this question only work if you want to read the stream's contents *fully* (until it is closed). Since that is not always intended (http requests with a keep-alive connection won't be closed), these method calls block (not giving you the contents). – f1sh Jul 14 '10 at 13:32

10 You **need** to know and specify the character encoding for the stream, or you **will** have character encoding bugs, since you will be using a randomly chosen encoding depending on which machine/operating system/platform or version thereof your code is run on. That is, do **not** use methods that depend on the platform default encoding. – Christoffer Hammarström Dec 17 '10 at 13:50

50 Answers

1 2 next

A nice way to do this is using Apache commons `IOUtils` to copy the `InputStream` into a `StringWriter` ... something like

```
StringWriter writer = new StringWriter();
IOUtils.copy(inputStream, writer, encoding);
String theString = writer.toString();
```

or even

```
// NB: does not close inputStream, you can use IOUtils.closeQuietly for that
String theString = IOUtils.toString(inputStream, encoding);
```

Alternatively, you could use `ByteArrayOutputStream` if you don't want to mix your Streams and Writers

edited May 25 '16 at 13:59
Daniel 1,533 1 11 35

answered Nov 21 '08 at 16:54
Harry Lime 16.6k 2 18 28

i found filenotfound exception while i try to read file name with "До_свидания" file name(Russian language) i try with FileInputstream but it not cable to read this filename from sdcard. – Bhanu Sharma Feb 11 '14 at 5:39

9 For android developers, seems like android does not come with IOUtils from Apache. So you might consider referring to other answers. – Chris.Zou Aug 1 '14 at 3:57

I work in a limited footprint environment, so the solution by @PavelRepin below using the java io/util libs makes more sense. – James Sep 25 '14 at 23:34

2 This is an incredibly old question at this point (it was asked in 2008). It is worth your time to read through more modern answers. Some use native calls from the Java 8 library. – Shadoninja Mar 28 '16 at 3:13

5 This answer is heavily outdated and one should be able to mark it as such (sadly this is not possible atm). – TrudleR Apr 20 '16 at 13:00

Here's a way using only standard Java library (note that the stream is not closed, YMMV).

```
static String convertStreamToString(java.io.InputStream is) {
    java.util.Scanner s = new java.util.Scanner(is).useDelimiter("\\A");
    return s.hasNext() ? s.next() : "";
}
```

I learned this trick from "Stupid Scanner tricks" article. The reason it works is because Scanner iterates over tokens in the stream, and in this case we separate tokens using "beginning of the input boundary" (\A) thus giving us only one token for the entire contents of the stream.

Note, if you need to be specific about the input stream's encoding, you can provide the second argument to `Scanner` constructor that indicates what charset to use (e.g. "UTF-8").

Hat tip goes also to Jacob, who once pointed me to the said article.

EDITED: Thanks to a suggestion from Patrick, made the function more robust when handling an empty input stream. **One more edit:** nixed try/catch, Patrick's way is more laconic.

edited Dec 10 '15 at 17:48
RAnders00
2,061 2 15 45

answered Mar 26 '11 at 20:40
Pavel Repin
21.9k 1 21 33

3 Thanks, for my version of this I added a finally block that closes the input stream, so the user doesn't have to since you've finished reading the input. Simplifies the caller code considerably. – user486646 Apr 21 '12 at 17:07

2 @PavelRepin @Patrick in my case, an empty inputStream caused a NPE during Scanner construction. I had to add `if (is == null) return "";` right at the beginning of the method; I believe this answer needs to be updated to better handle null inputStreams. – CFL_Jeff Aug 9 '12 at 13:36

79 For Java 7 you can close in a try-with: `try(java.util.Scanner s = new java.util.Scanner(is)) { return s.useDelimiter("\\A").hasNext() ? s.next() : ""; }` – earcam Jun 13 '13 at 5:24

1 Unfortunately this solution seems to go and lose the exceptions thrown in my underlying stream implementation. – Taig Jul 16 '13 at 7:59

5 FYI, hasNext blocks on console input streams (see here). (Just ran into this issue right now.) This solution works fine otherwise... just a heads up. – Ryan Feb 24 '14 at 5:36

Apache Commons allows:

```
String myString = IOUtils.toString(myInputStream, "UTF-8");
```

Of course, you could choose other character encodings besides UTF-8.

Also see: (Docs)

edited Jan 14 '16 at 0:48
Daniel Alexiuc
6,621 8 40 66

answered Dec 8 '08 at 20:13
Chinnery
8,193 2 16 24

1 Also, there is a method that only take a inputStream argument, if you are find with the default encoding. – Guillaume Coté Feb 3 '11 at 16:07

9 @Guillaume Coté I guess the message here is that you never should be "fine with the default encoding", since you cannot be sure of what it is, depending on the platform the java code is run on. – Per Wiklander Feb 3 '11 at 21:54

5 @Per Wiklander I disagree with you. Code that is going to work on a single could be quite sure that default encoding will be fine. For code that only open local file, it is a reasonable option to ask them to be encoded in the platform default encoding. – Guillaume Coté Feb 4 '11 at 15:56

20 To save anyone the hassle of Googling - <dependency> <groupId>org.apache.commons</groupId> <artifactId>commons-io</artifactId> <version>1.3.2</version> </dependency> – Chris Mar 9 '12 at 12:04

5 Also little improvement would be to use apache io (or other) constant for character encoding instead of using plain string literal - eg: IOUtils.toString(myInputStream, Charsets.UTF_8); – user1018711 Jan 13 '14 at 12:35

Summarize other answers I found 11 main ways to do this (see below). And I wrote some performance tests (see results below):

Ways to convert an InputStream to a String:

1. Using **IOUtils.toString** (`Apache Utils`)

```
String result = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
```

2. Using **CharStreams** (`guava`)

   ```
   String result = CharStreams.toString(new InputStreamReader(
         inputStream, Charsets.UTF_8));
   ```

3. Using `Scanner` (**JDK**)

   ```
   Scanner s = new Scanner(inputStream).useDelimiter("\\A");
   String result = s.hasNext() ? s.next() : "";
   ```

4. Using **Stream Api** (`Java 8`). **Warning**: This solution convert different line breaks (like `\r\n`) to `\n` .

   ```
   String result = new BufferedReader(new InputStreamReader(inputStream))
     .lines().collect(Collectors.joining("\n"));
   ```

5. Using **parallel Stream Api** (`Java 8`). **Warning**: This solution convert different line breaks (like `\r\n`) to `\n` .

   ```
   String result = new BufferedReader(new InputStreamReader(inputStream)).lines()
      .parallel().collect(Collectors.joining("\n"));
   ```

6. Using **InputStreamReader** and **StringBuilder** (`JDK`)

   ```
   final int bufferSize = 1024;
   final char[] buffer = new char[bufferSize];
   final StringBuilder out = new StringBuilder();
   Reader in = new InputStreamReader(inputStream, "UTF-8");
   for (; ; ) {
       int rsz = in.read(buffer, 0, buffer.length);
       if (rsz < 0)
           break;
       out.append(buffer, 0, rsz);
   }
   return out.toString();
   ```

7. Using **StringWriter** and **IOUtils.copy** (`Apache Commons`)

   ```
   StringWriter writer = new StringWriter();
   IOUtils.copy(inputStream, writer, "UTF-8");
   return writer.toString();
   ```

8. Using **ByteArrayOutputStream** and **inputStream.read** (`JDK`)

   ```
   ByteArrayOutputStream result = new ByteArrayOutputStream();
   byte[] buffer = new byte[1024];
   int length;
   while ((length = inputStream.read(buffer)) != -1) {
       result.write(buffer, 0, length);
   }
   return result.toString("UTF-8");
   ```

9. Using **BufferedReader** (`JDK`). **Warning:** This solution convert different line breaks (like `\n\r`) to `line.separator` system property (for example, in Windows to "\r\n").

   ```
   String newLine = System.getProperty("line.separator");
   BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
   StringBuilder result = new StringBuilder();
   String line; boolean flag = false;
   while ((line = reader.readLine()) != null) {
       result.append(flag? newLine: "").append(line);
       flag = true;
   }
   return result.toString();
   ```

10. Using **BufferedInputStream** and **ByteArrayOutputStream** (`JDK`)

   ```
   BufferedInputStream bis = new BufferedInputStream(inputStream);
   ByteArrayOutputStream buf = new ByteArrayOutputStream();
   int result = bis.read();
   while(result != -1) {
       buf.write((byte) result);
       result = bis.read();
   }
   return buf.toString();
   ```

11. Using **inputStream.read()** and **StringBuilder** (`JDK`). **Warning**: This solution has problem with Unicode, for example with Russian text (work correctly only with non-Unicode text)

   ```
   int ch;
   StringBuilder sb = new StringBuilder();
   while((ch = inputStream.read()) != -1)
       sb.append((char)ch);
   reset();
   return sb.toString();
   ```

Warning:

1. Solutions `4` , `5` and `9` convert different line breaks to one.
2. Solution `11` can't work correctly with Unicode text

Performance tests

Performance tests for small `String` (length = 175), url in github (mode = Average Time, system = Linux, score 1,343 is the best):

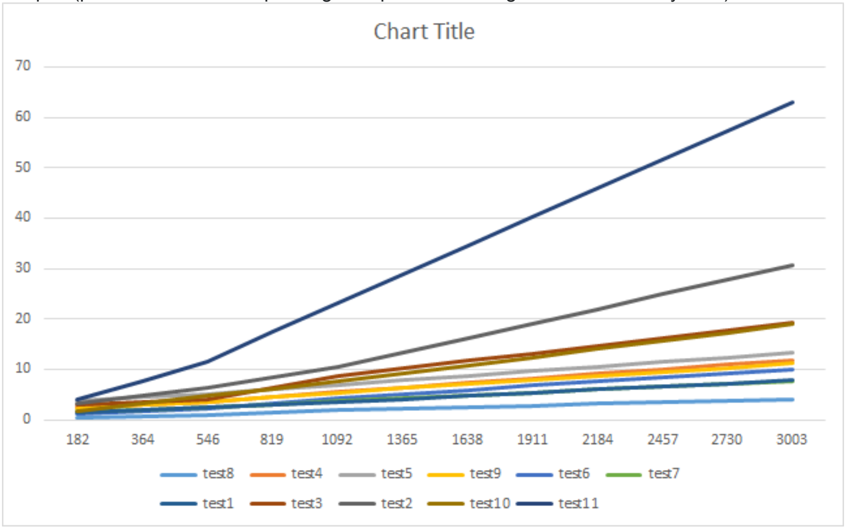
```
              Benchmark                        Mode  Cnt   Score   Error  Units
 8. ByteArrayOutputStream and read (JDK)        avgt   10   1,343 ± 0,028  us/op
 6. InputStreamReader and StringBuilder (JDK)   avgt   10   6,980 ± 0,404  us/op
```

```
10.BufferedInputStream, ByteArrayOutputStream  avgt   10   7,437 ± 0,735  us/op
11.InputStream.read() and StringBuilder (JDK)  avgt   10   8,977 ± 0,328  us/op
7. StringWriter and IOUtils.copy (Apache)      avgt   10  10,613 ± 0,599  us/op
1. IOUtils.toString (Apache Utils)             avgt   10  10,605 ± 0,527  us/op
3. Scanner (JDK)                               avgt   10  12,083 ± 0,293  us/op
2. CharStreams (guava)                         avgt   10  12,999 ± 0,514  us/op
4. Stream Api (Java 8)                         avgt   10  15,811 ± 0,605  us/op
9. BufferedReader (JDK)                        avgt   10  16,038 ± 0,711  us/op
5. parallel Stream Api (Java 8)                avgt   10  21,544 ± 0,583  us/op
```

Performance tests for big `String` (length = 50100), url in github (mode = Average Time, system = Linux, score 200,715 is the best):

```
              Benchmark                        Mode  Cnt   Score        Error  Units
8. ByteArrayOutputStream and read (JDK)        avgt   10   200,715 ±   18,103  us/op
1. IOUtils.toString (Apache Utils)             avgt   10   300,019 ±    8,751  us/op
6. InputStreamReader and StringBuilder (JDK)   avgt   10   347,616 ±  130,348  us/op
7. StringWriter and IOUtils.copy (Apache)      avgt   10   352,791 ±  105,337  us/op
2. CharStreams (guava)                         avgt   10   420,137 ±   59,877  us/op
9. BufferedReader (JDK)                        avgt   10   632,028 ±   17,002  us/op
5. parallel Stream Api (Java 8)                avgt   10   662,999 ±   46,199  us/op
4. Stream Api (Java 8)                         avgt   10   701,269 ±   82,296  us/op
10.BufferedInputStream, ByteArrayOutputStream  avgt   10   740,837 ±    5,613  us/op
3. Scanner (JDK)                               avgt   10   751,417 ±   62,026  us/op
11.InputStream.read() and StringBuilder (JDK)  avgt   10  2919,350 ± 1101,942  us/op
```

Graphs (performance tests depending on Input Stream length in Windows 7 system)



Performance test (Average Time) depending on Input Stream length in Windows 7 system:

```
length  182    546     1092    3276    9828    29484   58968

test8  0.38    0.938   1.868   4.448   13.412  36.459  72.708
test4  2.362   3.609   5.573   12.769  40.74   81.415  159.864
test5  3.881   5.075   6.904   14.123  50.258  129.937 166.162
test9  2.237   3.493   5.422   11.977  45.98   89.336  177.39
test6  1.261   2.12    4.38    10.698  31.821  86.106  186.636
test7  1.601   2.391   3.646   8.367   38.196  110.221 211.016
test1  1.529   2.381   3.527   8.411   40.551  105.16  212.573
test3  3.035   3.934   8.606   20.858  61.571  118.744 235.428
test2  3.136   6.238   10.508  33.48   43.532  118.044 239.481
test10 1.593   4.736   7.527   20.557  59.856  162.907 323.147
test11 3.913   11.506  23.26   68.644  207.591 600.444 1211.545
```

edited Jan 19 at 10:06
Martin Devillers
8,404 4 24 42

answered Feb 17 '16 at 0:58
Viacheslav Vedenin
11.1k 11 23 51

4 As you're writing the "summary answer", you should note that some solutions automatically convert different linebreaks (like `\r\n`) to `\n` which might be undesired in some cases. Also it would be nice to see the additional memory required or at least allocation pressure (at least you may run JMH with `-prof gc`). For the really cool post it would be great to see the graphs (depending on string length within the same input size and depending on the input size within the same string length). – Tagir Valeev Feb 17 '16 at 4:28

1 Update post. Add notes and graphs – Viacheslav Vedenin Feb 17 '16 at 13:54

5 Upvoted; the funniest thing is that results are more than expected: one should use standard JDK and/or Apache Commons syntactic sugar. – mudasobwa Apr 15 '16 at 9:17

1 Amazing post. Just one thing. Java 8 warns against using parallel streams on resources that will force you to lock and wait (such as this input stream) so the parallel stream option is rather cumbersome and not worth it no? – mangusbrother May 7 '16 at 20:20

4 comparison is awsome.. really good work – Ranjith Kumar Jun 24 '16 at 12:59

Taking into account file one should first get a `java.io.Reader` instance. This can then be read and added to a `StringBuilder` (we don't need `StringBuffer` if we are not accessing it in multiple threads, and `StringBuilder` is faster). The trick here is that we work in blocks, and as such don't need other buffering streams. The block size is parameterized for run-time performance optimization.

```
public static String slurp(final InputStream is, final int bufferSize) {
    final char[] buffer = new char[bufferSize];
    final StringBuilder out = new StringBuilder();
    try (Reader in = new InputStreamReader(is, "UTF-8")) {
        for (;;) {
            int rsz = in.read(buffer, 0, buffer.length);
            if (rsz < 0)
                break;
            out.append(buffer, 0, rsz);
        }
    }
    catch (UnsupportedEncodingException ex) {
        /* ... */
    }
    catch (IOException ex) {
        /* ... */
    }
    return out.toString();
}
```

edited Jul 15 '15 at 10:23

community wiki
11 revs, 8 users 39%
Paul de Vrieze

7 This solution uses multibyte characters. The example uses the UTF-8 encoding that allows expression of the full unicode range (Including Chinese). Replacing "UTF-8" with another encoding would allow that encoding to be used. – Paul de Vrieze Dec 9 '11 at 23:11

21 @User1 - I like using libraries in my code so I can get my job done faster. It's awesome when your managers say "Wow James! How did you get that done so fast?!". But when we have to spend time reinventing the wheel just because we have misplaced ideas about including a common, reusable, tried and tested utility, we're giving up time we could be spending furthering our project's goals. When we reinvent the wheel, we work twice as hard yet get to the finish line much later. Once we're at the finish line, there is no one there to congratulate us. When building a house, don't build the hammer too – jmort253 Jan 20 '12 at 1:05

7 Sorry, after re-reading my comment, it comes off a little arrogant. I just think it's important to have a good reason to avoid libraries and that the reason is a valid one, which there very well could be :) – jmort253 Jan 20 '12 at 1:35

2 @jmort253 We noticed performance regression after updating some library in our product for several times. Luckily we are building and selling our own product so we don't really have the so called deadlines. Unfortunately we are building a product that is available on many JVMs, databases and app servers on many operation systems so we have to think for the users using poor machines... And a string operation optimizing can improve the perf by 30~40%. And a fix: `In our product, I even replaced` should be 'we even replaced'. – coolcfan May 9 '12 at 8:39

8 @jmort253 If you would already use apache commons I would say, go for it. At the same time, there is a real cost to using libraries (as the dependency proliferation in many apache java libraries shows). If this would be the only use of the library, it would be overkill to use the library. On the other hand, determining your own buffer size(s) you can tune your memory/processor usage balance. – Paul de Vrieze May 22 '12 at 9:16

How about this?

```
InputStream in = /* your InputStream */;
StringBuilder sb=new StringBuilder();
BufferedReader br = new BufferedReader(new InputStreamReader(in));
String read;

while((read=br.readLine()) != null) {
    //System.out.println(read);
    sb.append(read);
}

br.close();
return sb.toString();
```

edited Jan 10 '16 at 3:58
hidro
6,643 4 27 35

answered Aug 4 '11 at 8:29
sampathpremarathna
3,044 2 15 29

7 The thing is, you're first splitting into lines, and then undoing that. It's easier and faster to just read arbitrary buffers. – Paul de Vrieze Apr 20 '12 at 18:36

12 Also, readLine does not distinguish between \n and \r, so you cannot reproduce the exact stream again. – Délawen Sep 10 '12 at 8:08

@PauldeVrieze how many lines, and how quickly do you need to process them!? I would hazard a guess that any performance loss would be small, or could be handled by every once in a while logging them to a file and destroying the old String obj's. – Thufir Aug 28 '13 at 6:52

1 very inefficient, as `readLine` read character by character to look for EOL. Also, if there is no line break in the stream, this does not really make sense. – njzk2 Apr 18 '14 at 18:05

@Delawn I am not able to understand your point. Can u pls explain? Coz if new line is part of stream then It will also be added to stringbuilder. Isn't? – Gops AB Sep 21 '15 at 18:11

If you are using Google-Collections/Guava you could do the following:

```
InputStream stream = ...
String content = CharStreams.toString(new InputStreamReader(stream, Charsets.UTF_8));
Closeables.closeQuietly(stream);
```

Note that the second parameter (i.e. Charsets.UTF_8) for the `InputStreamReader` isn't necessary, but it is generally a good idea to specify the encoding if you know it (which you should!)

edited Jan 30 '13 at 16:35
ralfoide
673 1 9 19

answered Jul 13 '10 at 15:56
Sakuraba
2,013 1 11 13

2 @harschware: Given the question was: "If you have java.io.InputStream object how should you process that object and produce a String?" I assumed that a stream is already present in the situation. – Sakuraba Apr 13 '11 at 9:41

You didn't explain your answer very well, and had extraneous variables; user359996 said the same thing as you, but much more clearly. – Uronym Sep 1 '11 at 22:10

it returns to me boxes instead of actual text characters. plz advise – Vik Nov 12 '11 at 13:20

1 +1 for guava, -1 for not specifying the encoding of the input stream. eg. new InputStreamReader(stream, "UTF-8") – andras Jul 6 '12 at 11:01

3 @plasma147 Instead of downvoting, consider editing the example (I just submitted that, adding `Closeables.closeQuietly(stream);`) – ralfoide Jan 30 '13 at 16:30

This is my pure Java & Android solution, works well...

```
public String readFullyAsString(InputStream inputStream, String encoding)
        throws IOException {
    return readFully(inputStream).toString(encoding);
}

public byte[] readFullyAsBytes(InputStream inputStream)
        throws IOException {
    return readFully(inputStream).toByteArray();
}

private ByteArrayOutputStream readFully(InputStream inputStream)
        throws IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    byte[] buffer = new byte[1024];
    int length = 0;
    while ((length = inputStream.read(buffer)) != -1) {
        baos.write(buffer, 0, length);
    }
    return baos;
}
```

edited Mar 25 '16 at 7:54

answered May 8 '12 at 20:24
TacB0sS
4,936 6 41 83

2 Works well on Android in comparison with other answers which work only in enterprise java. – vorrtex Jan 14 '13 at 19:30

Crashed in Android with OutOfMemory error on the ".write" line, every time, for short strings. – Adam Apr 15 '13 at 17:18

I've added the encoding. just as a side note, the original readFully method I have in my code does not return String, it returns byte[] for a more general purpose functionality. Implementing the new String(...) with encoding is the responsibility of the on that uses the API! – TacB0sS Aug 18 '13 at 10:50

1 Quick note: The memory footprint of this is maxed by `2*n` , where n is the size of the stream, as per the `ByteArrayInputStream` auto-growing system. – njzk2 Apr 18 '14 at 18:07

1 Unnecessarily doubles memory usage, that is precious on mobile devices. You'd better use InputStreamReader and append to StringReader, the byte to char conversion will be done on the fly, not in bulk at the end. – Oliv Jan 22 '15 at 8:33

Here's the most elegant, pure-Java (no library) solution I came up with after some experimentation:

```
public static String fromStream(InputStream in) throws IOException
{
    BufferedReader reader = new BufferedReader(new InputStreamReader(in));
    StringBuilder out = new StringBuilder();
    String newLine = System.getProperty("line.separator");
    String line;
    while ((line = reader.readLine()) != null) {
        out.append(line);
        out.append(newLine);
    }
    return out.toString();
}
```

edited Dec 7 '13 at 16:39

answered Jan 1 '13 at 3:43
Drew Noakes

136k 79 415 505

3 Isn't there a reader.close() missing? Ideally with try/finally... – Torben Kohlmeier Jun 2 '13 at 13:50

6 @TorbenKohlmeier, readers and buffers don't need to be closed. The provided `InputStream` should be closed by the caller. – Drew Noakes Jun 3 '13 at 11:37

5 Don't forget to mention that there's a more preferable constructor in InputStreamReader that takes a CharSet. – jontejj Jun 27 '13 at 12:36

4 why do people keep using `readLine` ? if you don't use the lines per se, what good is it (except being very slow?) – njzk2 Apr 18 '14 at 18:07

2 Do not read by lines. What if one line is so long so it does not fit into heap? – voho Aug 7 '14 at 9:29

How about:

```
import java.io.BufferedInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.IOException;

public static String readInputStreamAsString(InputStream in)
    throws IOException {

    BufferedInputStream bis = new BufferedInputStream(in);
    ByteArrayOutputStream buf = new ByteArrayOutputStream();
    int result = bis.read();
    while(result != -1) {
      byte b = (byte)result;
      buf.write(b);
      result = bis.read();
    }
    return buf.toString();
}
```

answered Jun 10 '09 at 21:07
Jon Moore
954 8 11

1 This one is slow, because reads byte by byte. – Daniel De León May 8 '15 at 21:56

I'd use some Java 8 tricks.

```
public static String streamToString(final InputStream inputStream) throws Exception {
    // buffering optional
    try
    (
        final BufferedReader br
           = new BufferedReader(new InputStreamReader(inputStream))
    ) {
        // parallel optional
        return br.lines().parallel().collect(Collectors.joining("\n"));
    } catch (final IOException e) {
        throw new RuntimeException(e);
        // whatever.
    }
}
```

Essentially the same as some other answers except more succinct.

edited Jul 15 '15 at 11:03
Ian2thedv
1,821 11 30

answered Jul 17 '14 at 17:58
Simon Kuang
1,822 1 12 41

4 Would that `return null` ever get called? Either the `br.lines...` returns or an exception is thrown. – Holloway Jul 23 '14 at 9:13

docs.oracle.com/javase/tutorial/essential/exceptions/try.html – Khaled.K Feb 5 '15 at 10:32

2 @Khaled A Khunaifer: yes, pretty sure... maybe you should have a look here: docs.oracle.com/javase/tutorial/essential/exceptions/…. What you wrongly edited is a "try-with-resources" statement. – jamp Feb 5 '15 at 13:13

8 Why do you call `parallel()` on the stream? – robinst Apr 20 '15 at 5:13

2 This would not result in an *honest* copy of the data if the source stream used windows line endings as all `\r\n` would end up getting converted into `\n` ... – Lucas Aug 13 '15 at 18:30

For completeness here is **Java 9** solution:

```
public static String toString(InputStream input) throws IOException {
    return new String(input.readAllBytes(), StandardCharsets.UTF_8);
}
```

The `readAllBytes` is currently in JDK 9 main codebase, so it likely to appear in the release. You can try it right now using the JDK 9 snapshot builds.

answered Sep 2 '15 at 11:50
Tagir Valeev
48.4k 7 83 160

Does not the method allocate a whole lot of memory for reading? `byte[] buf = new byte[DEFAULT_BUFFER_SIZE];` where `MAX_BUFFER_SIZE = Integer.MAX_VALUE - 8;` which gives `MAX_BUFFER_SIZE = 2147483639` . Google says its around 2.147 GB. – Rekin Sep 3 '15 at 9:22

Sorry, I made an error in calculations. It is 2 GB. I've edited the comment. So, even If I read like a 4kb file I use 2gb of memory? – Rekin Sep 3 '15 at 9:27

The method is already there in Java 8. And it only allocates `DEFAULT_BUFFER_SIZE` bytes which is `8192` , which then is increased in powers of 2, not `MAX_BUFFER_SIZE` . – Christian Hujer Jan 31 '16 at 21:58

@ChristianHujer, I don't see it in the latest jdk8u commit. AFAIK new methods are never introduced in Java updates, only in major releases. – Tagir Valeev Feb 1 '16 at 7:54

Ya you have to use `Path` instead of `InputStream` , it's here: docs.oracle.com/javase/8/docs/api/java/nio/file/… – Christian Hujer Feb 2 '16 at 11:36

I ran some timing tests because time matters, always.

I attempted to get the response into a String 3 different ways. (shown below)
I left out try/catch blocks for the sake readability.

To give context, this is the preceding code for all 3 approaches:

```
    String response;
    String url = "www.blah.com/path?key=value";
    GetMethod method = new GetMethod(url);
    int status = client.executeMethod(method);
```

1)

```
  response = method.getResponseBodyAsString();
```

2)

```
InputStream resp = method.getResponseBodyAsStream();
InputStreamReader is=new InputStreamReader(resp);
BufferedReader br=new BufferedReader(is);
String read = null;
StringBuffer sb = new StringBuffer(read);
while((read = br.readLine()) != null) {
    sb.append(read);
}
response = sb.toString();
```

3)

```
InputStream iStream  = method.getResponseBodyAsStream();
StringWriter writer = new StringWriter();
IOUtils.copy(iStream, writer, "UTF-8");
response = writer.toString();
```

So, after running 500 tests on each approach with the same request/response data, here are the numbers. Once again, these are my findings and your findings may not be exactly the same, but I wrote this to give some indication to others of the efficiency differences of these approaches.

Ranks:
Approach #1
Approach #3 - 2.6% slower than #1
Approach #2 - 4.3% slower than #1

Any of these approaches is an appropriate solution for grabbing a response and creating a String from it.

edited Feb 10 '14 at 8:06
BennX
4,266 2 16 47

answered Oct 12 '11 at 17:23
Brett Holt
906 7 13

1 2) contains an error, it adds always "null" at the end of the string as you are always makeing one more step then necessary. Performance will be the same anyway I think. This should work: String read = null; StringBuffer sb = new StringBuffer(); while((read = br.readLine()) != null) { sb.append(read); } – LukeSolar Oct 21 '11 at 13:32

Pure Java solution using Streams, works since Java 8.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.stream.Collectors;

// ...
public static String inputStreamToString(InputStream is) throws IOException {
    try (BufferedReader br = new BufferedReader(new InputStreamReader(is))) {
        return br.lines().collect(Collectors.joining(System.lineSeparator()));
    }
}
```

As mentioned by Christoffer Hammarström below other answer it is safer to explicitly specify the Charset. I.e. The InputStreamReader constructor can be changes as follows:

```
new InputStreamReader(is, Charset.forName("UTF-8"))
```

edited Feb 26 '15 at 18:50 answered Feb 26 '15 at 18:39
czerny
2,176 2 20 32

7 Instead of doing `Charset.forName("UTF-8")` , use `StandardCharsets.UTF_8` (from `java.nio.charset`).
– robinst Apr 20 '15 at 5:12

Here's more-or-less sampath's answer, cleaned up a bit and represented as a function:

```
String streamToString(InputStream in) throws IOException {
  StringBuilder out = new StringBuilder();
  BufferedReader br = new BufferedReader(new InputStreamReader(in));
  for(String line = br.readLine(); line != null; line = br.readLine())
    out.append(line);
  br.close();
  return out.toString();
}
```

edited Sep 12 '12 at 18:31 answered Mar 30 '12 at 19:52
TKH
464 3 9

If you were feeling adventurous, you could mix Scala and Java and end up with this:

```
scala.io.Source.fromInputStream(is).mkString("")
```

Mixing Java and Scala code and libraries has it's benefits.

See full description here: Idiomatic way to convert an InputStream to a String in Scala

answered Mar 7 '12 at 7:32
Jack
8,530 8 63 131

1 Nowadays simply this works fine: `Source.fromInputStream(...).mkString` – KajMagnus Jul 30 '15 at 23:32

If you can't use Commons IO (FileUtils/IOUtils/CopyUtils) here's an example using a BufferedReader to read the file line by line:

```
public class StringFromFile {
    public static void main(String[] args) /*throws UnsupportedEncodingException*/ {
        InputStream is = StringFromFile.class.getResourceAsStream("file.txt");
        BufferedReader br = new BufferedReader(new InputStreamReader(is/*, "UTF-8"*/));
        final int CHARS_PER_PAGE = 5000; //counting spaces
        StringBuilder builder = new StringBuilder(CHARS_PER_PAGE);
        try {
            for(String line=br.readLine(); line!=null; line=br.readLine()) {
                builder.append(line);
                builder.append('\n');
            }
        } catch (IOException ignore) { }
        String text = builder.toString();
        System.out.println(text);
    }
}
```

or if you want raw speed I'd propose a variation on what Paul de Vrieze suggested (which avoids using a StringWriter (which uses a StringBuffer internally) :

```
public class StringFromFileFast {
    public static void main(String[] args) /*throws UnsupportedEncodingException*/ {
        InputStream is = StringFromFileFast.class.getResourceAsStream("file.txt");
        InputStreamReader input = new InputStreamReader(is/*, "UTF-8"*/);
        final int CHARS_PER_PAGE = 5000; //counting spaces
        final char[] buffer = new char[CHARS_PER_PAGE];
        StringBuilder output = new StringBuilder(CHARS_PER_PAGE);
        try {
            for(int read = input.read(buffer, 0, buffer.length);
                    read != -1;
                    read = input.read(buffer, 0, buffer.length)) {
                output.append(buffer, 0, read);
            }
        } catch (IOException ignore) { }
        String text = output.toString();
        System.out.println(text);
    }
}
```

edited May 18 '10 at 13:05 answered May 18 '10 at 12:57
DJDaveMark
410 4 10

In order to make your code work, I had to use this.getClass().getClassLoader().getResourceAsStream() (using Eclipse with a maven project) – greuze Jan 24 '12 at 12:27

This is an answer adapted from `org.apache.commons.io.IOUtils` source code, for those who want to have the apache implementation but do not want the whole library.

```
private static final int BUFFER_SIZE = 4 * 1024;

public static String inputStreamToString(InputStream inputStream, String charsetName)
        throws IOException {
    StringBuilder builder = new StringBuilder();
    InputStreamReader reader = new InputStreamReader(inputStream, charsetName);
    char[] buffer = new char[BUFFER_SIZE];
    int length;
    while ((length = reader.read(buffer)) != -1) {
        builder.append(buffer, 0, length);
    }
    return builder.toString();
}
```

edited Oct 10 '15 at 4:37 answered Aug 3 '14 at 9:47
Dreaming in Code
1,703 16 24

Here is the complete method for converting `InputStream` into `String` without using any third party library. Use `StringBuilder` for single threaded environment otherwise use `StringBuffer`.

```
public static String getString( InputStream is) throws IOException {
    int ch;
    StringBuilder sb = new StringBuilder();
    while((ch = is.read()) != -1)
        sb.append((char)ch);
    return sb.toString();
}
```

edited Dec 16 '15 at 9:20 answered Apr 9 '14 at 10:37
rtruszk laksys
3,454 13 25 45 1,813 2 12 26

2 In this method there is no encoding applied. So let's say the data received from the InputStream is encoded using UTF-8 the output will be wrong. To fix this you could use `in = new InputStreamReader(inputStream)` and `(char)in.read()`. – Frederic Leitenberger Nov 4 '14 at 12:21

Isn't reading it character by character a bit slow? – Lennart Rolland Nov 5 '14 at 2:00

2 and memory-inefficient as well; I believe I tried using this before on a large input and StringBuilder ran out of memory – gengkev Nov 18 '14 at 3:37

1 There is another similar answer which uses a char[] buffer and is more efficient and takes care of charset. – Guillaume Perrot Apr 27 '15 at 21:39

Here's how to do it using just the JDK using byte array buffers. This is actually how the commons-io `IOUtils.copy()` methods all work. You can replace `byte[]` with `char[]` if you're copying from a `Reader` instead of an `InputStream`.

```
import java.io.ByteArrayOutputStream;
import java.io.InputStream;

...

InputStream is = ....
ByteArrayOutputStream baos = new ByteArrayOutputStream(8192);
byte[] buffer = new byte[8192];
int count = 0;
try {
  while ((count = is.read(buffer)) != -1) {
    baos.write(buffer, 0, count);
  }
}
finally {
  try {
    is.close();
  }
  catch (Exception ignore) {
  }
}
```

```
String charset = "UTF-8";
String inputStreamAsString = baos.toString(charset);
```

edited Aug 13 '14 at 4:30
Matt
489 2 9

answered Nov 2 '12 at 12:37
Matt Shannon
119 1 2

1 Please give a description on what you are trying to accomplish. – Ragunath Jawahar Nov 2 '12 at 12:58

Make sure to close the streams at end if you use Stream Readers

```
private String readStream(InputStream iStream) throws IOException {
    //build a Stream Reader, it can read char by char
    InputStreamReader iStreamReader = new InputStreamReader(iStream);
    //build a buffered Reader, so that i can read whole line at once
    BufferedReader bReader = new BufferedReader(iStreamReader);
    String line = null;
    StringBuilder builder = new StringBuilder();
    while((line = bReader.readLine()) != null) {  //Read till end
        builder.append(line);
    }
    bReader.close();         //close all opened stuff
    iStreamReader.close();
    //iStream.close(); //EDIT: Let the creator of the stream close it!
                       // some readers may auto close the inner stream
    return builder.toString();
}
```

EDIT: On JDK 7+, you can use try-with-resources construct.

```
/**
 * Reads the stream into a string
 * @param iStream the input stream
 * @return the string read from the stream
 * @throws IOException when an IO error occurs
 */
private String readStream(InputStream iStream) throws IOException {

    //Buffered reader allows us to read line by line
    try (BufferedReader bReader =
                 new BufferedReader(new InputStreamReader(iStream))){
        StringBuilder builder = new StringBuilder();
        String line;
        while((line = bReader.readLine()) != null) {  //Read till end
            builder.append(line);
        }
        return builder.toString();
    }
}
```

edited Feb 1 '16 at 0:35

answered Nov 17 '12 at 12:39
Thamme Gowda N
2,808 18 30

1 You're right about closing streams, however, the responsibility for closing streams is usually with the stream constructor (finish what you start). So, `iStream` should really rather be closed by the caller because the caller created `iStream` . Besides, closing streams should be done in a `finally` block, or even better in a Java 7 try-with-resources statement. In your code, when `readLine()` throws `IOException` , or `builder.append()` throws `OutOfMemoryError` , the streams would stay open. – Christian Hujer Jan 31 '16 at 22:01

Updated answer, Thanks! – Thamme Gowda N Feb 1 '16 at 0:35

Kotlin users simply do:

```
println(InputStreamReader(is).readText())
```

whereas

```
readText()
```

is Kotlin standard library's built-in extension method.

answered Feb 4 '15 at 1:12
Alexander
3,339 6 26 36

This one is nice because:

- Hand safety the Charset.
- You control the read buffer size.
- You can provision the length of the builder and can be not exactly.
- Is free from library dependencies.

- Is for Java 7 or higher.

What the for?

```
public static String convertStreamToString(InputStream is) {
   if (is == null) return null;
   StringBuilder sb = new StringBuilder(2048); // Define a size if you have an idea of it.
   char[] read = new char[128]; // Your buffer size.
   try (InputStreamReader ir = new InputStreamReader(is, StandardCharsets.UTF_8)) {
     for (int i; -1 != (i = ir.read(read)); sb.append(read, 0, i));
   } catch (Throwable t) {}
   return sb.toString();
}
```

edited Apr 20 '15 at 18:17 answered Jun 8 '14 at 7:46
Daniel De León
6,828 1 43 48

The `catch (Throwable)` shouldn't really be empty if this is production code. – Christian Hujer Jan 31 '16 at 22:03

The easiest way in JDK is with the following code snipplets.

```
String convertToString(InputStream in){
    String resource = new Scanner(in).useDelimiter("\\Z").next();
    return resource;
}
```

answered Aug 9 '16 at 20:18
Raghu K Nair
1,538 13 19

Well you can program it for yourself.. it's not complicated..

```
String Inputstream2String (InputStream is) throws IOException
    {
        final int PKG_SIZE = 1024;
        byte[] data = new byte [PKG_SIZE];
        StringBuilder buffer = new StringBuilder(PKG_SIZE * 10);
        int size;

        size = is.read(data, 0, data.length);
        while (size > 0)
        {
            String str = new String(data, 0, size);
            buffer.append(str);
            size = is.read(data, 0, data.length);
        }
        return buffer.toString();
    }
```

edited Nov 8 '13 at 16:20 answered Mar 9 '13 at 20:13
Victor
1,858 2 16 31

1 Since you're using `buffer` variable locally with no chance of being shared across multiple threads you should consider changing its type to `StringBuilder` , to avoid the overhead of (useless) synchronization. – user246645 Nov 8 '13 at 10:27

That's a good point alex!. I thing that we both agree that this method isn't thread-safe in many ways. Even the input stream operations aren't thread-safe. – Victor Nov 8 '13 at 16:19

If the stream contains UTF-8 character that spans across several lines, this algorithm can cut the character in two breaking the string. – Vlad Lifliand Aug 8 '14 at 22:47

1 @VladLifliand How exactly would a UTF-8 character manage to span across several lines? That's impossible by definition. You probably meant something else. – Christian Hujer Jan 31 '16 at 22:05

JDK 7/8 answer that closes the stream and still throws an IOException:

```
StringBuilder build = new StringBuilder();
byte[] buf = new byte[1024];
int length;
try (InputStream is = getInputStream()) {
  while ((length = is.read(buf)) != -1) {
    build.append(new String(buf, 0, length));
  }
}
```

answered Dec 5 '13 at 17:53
Brian Pontarelli
569 1 4 9

The below code worked for me.

```
URL url = MyClass.class.getResource("/" + configFileName);
BufferedInputStream bi = (BufferedInputStream) url.getContent();
byte[] buffer = new byte[bi.available() ];
int bytesRead = bi.read(buffer);
String out = new String(buffer);
```

Please note, according to Java docs, the `available()` method might not work with `InputStream` but always works with `BufferedInputStream` . In case you don't want to use `available()` method we can always use the below code

```
URL url = MyClass.class.getResource("/" + configFileName);
BufferedInputStream bi = (BufferedInputStream) url.getContent();
File f = new File(url.getPath());
byte[] buffer = new byte[ (int) f.length()];
int bytesRead = bi.read(buffer);
String out = new String(buffer);
```

I am not sure if there will be any encoding issues. Please comment, if there will be any issues with the code.

edited Dec 16 '15 at 9:23
rtruszk
3,454 13 25 45

answered Jul 24 '12 at 10:19
Anand N
139 1 2 10

3 The whole point of using `InputStream` is, that a) you don't know the length of the *complete* stream (which bails out anything depending on `available`) and b) the stream can be anything - a file, a socket, something internal (which bails out anything based on `File.size()`). Regarding `available` : This will cut off data if the stream is longer than the buffer size. – A.H. Jul 24 '12 at 10:26

```
InputStreamReader i = new InputStreamReader(s);
BufferedReader str = new BufferedReader(i);
String msg = str.readLine();
System.out.println(msg);
```

Here s is your `InputStream` object which will get convert into `String`

edited Dec 16 '15 at 9:24
rtruszk
3,454 13 25 45

answered May 30 '13 at 14:52
Omkar Khot
63 1 2

will it work if last 2 lines are inserted in `do-while` loop? – KNU Apr 7 '14 at 11:34

3 will work only if the InputStream is one-liner – Stavros Apr 30 '14 at 8:22

You can use apache commons. In the IOUtils you can find the toString metod with 3 helpfull implementations.

```
public static String toString(InputStream input) throws IOException {
        return toString(input, Charset.defaultCharset());
}

public static String toString(InputStream input) throws IOException {
        return toString(input, Charset.defaultCharset());
}

public static String toString(InputStream input, String encoding)
            throws IOException {
        return toString(input, Charsets.toCharset(encoding));
}
```

answered Jan 16 '14 at 14:03
Rys
568 8 23

I have written a class that does just that, so I figured I'd share it with everyone. Sometimes you don't want to add Apache Commons just for one thing, and want something dumber than Scanner that doesn't examine the content.

Usage is as follows

```
// Read from InputStream
String data = new ReaderSink(inputStream, Charset.forName("UTF-8")).drain();

// Read from File
data = new ReaderSink(file, Charset.forName("UTF-8")).drain();

// Drain input stream to console
new ReaderSink(inputStream, Charset.forName("UTF-8")).drainTo(System.out);
```

Here is the code for ReaderSink:

```
import java.io.*;
import java.nio.charset.Charset;

/**
 * A simple sink class that drains a {@link Reader} to a {@link String} or
 * to a {@link Writer}.
 *
 * @author Ben Barkay
 * @version 2/20/2014
 */
public class ReaderSink {
    /**
     * The default buffer size to use if no buffer size was specified.
     */
    public static final int DEFAULT_BUFFER_SIZE = 1024;

    /**
     * The {@link Reader} that will be drained.
     */
    private final Reader in;

    /**
     * Constructs a new {@code ReaderSink} for the specified file and charset.
     * @param file      The file to read from.
     * @param charset   The charset to use.
     * @throws FileNotFoundException    If the file was not found on the filesystem.
     */
    public ReaderSink(File file, Charset charset) throws FileNotFoundException {
        this(new FileInputStream(file), charset);
    }

    /**
     * Constructs a new {@code ReaderSink} for the specified {@link InputStream}.
     * @param in        The {@link InputStream} to drain.
     * @param charset   The charset to use.
     */
    public ReaderSink(InputStream in, Charset charset) {
        this(new InputStreamReader(in, charset));
    }

    /**
     * Constructs a new {@code ReaderSink} for the specified {@link Reader}.
     * @param in    The reader to drain.
     */
    public ReaderSink(Reader in) {
        this.in = in;
    }

    /**
     * Drains the data from the underlying {@link Reader}, returning a {@link String}
containing
     * all of the read information. This method will use {@link #DEFAULT_BUFFER_SIZE} for
     * its buffer size.
     * @return  A {@link String} containing all of the information that was read.
     */
    public String drain() throws IOException {
        return drain(DEFAULT_BUFFER_SIZE);
    }

    /**
     * Drains the data from the underlying {@link Reader}, returning a {@link String}
containing
     * all of the read information.
     * @param bufferSize    The size of the buffer to use when reading.
     * @return  A {@link String} containing all of the information that was read.
     */
    public String drain(int bufferSize) throws IOException {
        StringWriter stringWriter = new StringWriter();
        drainTo(stringWriter, bufferSize);
        return stringWriter.toString();
    }

    /**
     * Drains the data from the underlying {@link Reader}, writing it to the
     * specified {@link Writer}. This method will use {@link #DEFAULT_BUFFER_SIZE} for
     * its buffer size.
     * @param out   The {@link Writer} to write to.
     */
    public void drainTo(Writer out) throws IOException {
        drainTo(out, DEFAULT_BUFFER_SIZE);
    }

    /**
     * Drains the data from the underlying {@link Reader}, writing it to the
     * specified {@link Writer}.
     * @param out           The {@link Writer} to write to.
     * @param bufferSize    The size of the buffer to use when reader.
     */
    public void drainTo(Writer out, int bufferSize) throws IOException {
        char[] buffer = new char[bufferSize];
        int read;
        while ((read = in.read(buffer)) > -1) {
            out.write(buffer, 0, read);
        }
    }
}
```

edited Feb 20 '14 at 17:46

answered Feb 20 '14 at 17:24

Ben Barkay
3,423 11 26

1 2 next

protected by NullPoiиteя Jun 10 '13 at 5:09

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?