链滴

spring中的流工具

作者：felayman

原文链接：https://ld246.com/article/1459333331465

来源网站：链滴

```
<pre class="prettyprint">/*
 * Copyright 2002-2015 the original author or authors.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.springframework.util;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FilterInputStream;
import java.io.FilterOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.nio.charset.Charset;

/**
 * Simple utility methods for dealing with streams. The copy methods of this class are
 * similar to those defined in {@link FileCopyUtils} except that all affected streams are
 * left open when done. All copy methods use a block size of 4096 bytes.
 *
 * &lt;p&gt;Mainly for use within the framework, but also useful for application code.
 *
 * @author Juergen Hoeller
 * @author Phillip Webb
 * @since 3.2.2
 * @see FileCopyUtils
 */
public abstract class StreamUtils {

    public static final int BUFFER_SIZE = 4096;

    private static final byte[] EMPTY_CONTENT = new byte[0];


    /**
     * Copy the contents of the given InputStream into a new byte array.
     * Leaves the stream open when done.
     * @param in the stream to copy from
     * @return the new byte array that has been copied to
```

```
     * @throws IOException in case of I/O errors
     */
    public static byte[] copyToByteArray(InputStream in) throws IOException {
        ByteArrayOutputStream out = new ByteArrayOutputStream(BUFFER_SIZE);
        copy(in, out);
        return out.toByteArray();
    }

    /**
     * Copy the contents of the given InputStream into a String.
     * Leaves the stream open when done.
     * @param in the InputStream to copy from
     * @return the String that has been copied to
     * @throws IOException in case of I/O errors
     */
    public static String copyToString(InputStream in, Charset charset) throws IOException {
        Assert.notNull(in, "No InputStream specified");
        StringBuilder out = new StringBuilder();
        InputStreamReader reader = new InputStreamReader(in, charset);
        char[] buffer = new char[BUFFER_SIZE];
        int bytesRead = -1;
        while ((bytesRead = reader.read(buffer)) != -1) {
            out.append(buffer, 0, bytesRead);
        }
        return out.toString();
    }

    /**
     * Copy the contents of the given byte array to the given OutputStream.
     * Leaves the stream open when done.
     * @param in the byte array to copy from
     * @param out the OutputStream to copy to
     * @throws IOException in case of I/O errors
     */
    public static void copy(byte[] in, OutputStream out) throws IOException {
        Assert.notNull(in, "No input byte array specified");
        Assert.notNull(out, "No OutputStream specified");
        out.write(in);
    }

    /**
     * Copy the contents of the given String to the given output OutputStream.
     * Leaves the stream open when done.
     * @param in the String to copy from
     * @param charset the Charset
     * @param out the OutputStream to copy to
     * @throws IOException in case of I/O errors
     */
    public static void copy(String in, Charset charset, OutputStream out) throws IOException {
        Assert.notNull(in, "No input String specified");
        Assert.notNull(charset, "No charset specified");
        Assert.notNull(out, "No OutputStream specified");
        Writer writer = new OutputStreamWriter(out, charset);
        writer.write(in);
```

```
        writer.flush();
    }

    /**
     * Copy the contents of the given InputStream to the given OutputStream.
     * Leaves both streams open when done.
     * @param in the InputStream to copy from
     * @param out the OutputStream to copy to
     * @return the number of bytes copied
     * @throws IOException in case of I/O errors
     */
    public static int copy(InputStream in, OutputStream out) throws IOException {
        Assert.notNull(in, "No InputStream specified");
        Assert.notNull(out, "No OutputStream specified");
        int byteCount = 0;
        byte[] buffer = new byte[BUFFER_SIZE];
        int bytesRead = -1;
        while ((bytesRead = in.read(buffer)) != -1) {
            out.write(buffer, 0, bytesRead);
            byteCount += bytesRead;
        }
        out.flush();
        return byteCount;
    }

    /**
     * Return an efficient empty {@link InputStream}.
     * @return a {@link ByteArrayInputStream} based on an empty byte array
     * @since 4.2.2
     */
    public static InputStream emptyInput() {
        return new ByteArrayInputStream(EMPTY_CONTENT);
    }

    /**
     * Return a variant of the given {@link InputStream} where calling
     * {@link InputStream#close() close()} has no effect.
     * @param in the InputStream to decorate
     * @return a version of the InputStream that ignores calls to close
     */
    public static InputStream nonClosing(InputStream in) {
        Assert.notNull(in, "No InputStream specified");
        return new NonClosingInputStream(in);
    }

    /**
     * Return a variant of the given {@link OutputStream} where calling
     * {@link OutputStream#close() close()} has no effect.
     * @param out the OutputStream to decorate
     * @return a version of the OutputStream that ignores calls to close
     */
    public static OutputStream nonClosing(OutputStream out) {
        Assert.notNull(out, "No OutputStream specified");
        return new NonClosingOutputStream(out);
```

```
    }


    private static class NonClosingInputStream extends FilterInputStream {

        public NonClosingInputStream(InputStream in) {
            super(in);
        }

        @Override
        public void close() throws IOException {
        }
    }


    private static class NonClosingOutputStream extends FilterOutputStream {

        public NonClosingOutputStream(OutputStream out) {
            super(out);
        }

        @Override
        public void write(byte[] b, int off, int let) throws IOException {
            // It is critical that we override this method for performance
            out.write(b, off, let);
        }

        @Override
        public void close() throws IOException {
        }
    }
}
</pre>
```