链滴

记一次代码重构

作者：tiangao

原文链接：https://ld246.com/article/1486457832217

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

记一次代码重构

有这么一个下载 zip 文件的功能函数

函数的目的是要下载一个文件到指定目录，第一要检查大小，第二要检查是不是 zip 文件，然后下载指定目录。

现在的问题是这个没有超时控制，网络慢或者出问题的时候就死死的卡在那儿了

```
private File downloadZipFile(Task task) throws DeployScheduleException, IOException {
    LOGGER.debug("#Scheduler#{}#download start", task.getId());
    HttpClient client = HttpClientBuilder.create().build();
    HttpGet httpGet = new HttpGet(task.getArchive_url());

    dbHelper.updateTaskStatusAndReason(task.getId(), "FETCHING", "");
    HttpResponse response = client.execute(httpGet);
    long contentLength = response.getEntity().getContentLength();
    long limit = sizeLimit << 20;
    if (contentLength > limit) {
        dbHelper.updateTaskLog(task.getId(), "Site archive size is more than " + sizeLimit + "M.")

        throw new DeployScheduleException("#Scheduler#" + task.getId() + "#Archive too big#s
ze: {}" + contentLength);
    }
    String contentType = response.getEntity().getContentType().getValue();
    if (!contentType.contains("octet-stream") &&
            !contentType.contains("zip") &&
            !task.getArchive_url().endsWith(".zip")) {
        dbHelper.updateTaskLog(task.getId(), "Project or branch not found.");
        throw new DeployScheduleException("#Scheduler#" + task.getId() + "#invalid content ty
e: " + contentType);
    }
    InputStream inputStream = response.getEntity().getContent();

    LOGGER.debug("tempPath" + getTempOutFile(task));
    FileOutputStream outFile = new FileOutputStream(getTempOutFile(task));
    LOGGER.info("#Scheduler#{}#download file at {}", task.getId(), getTempOutFile(task));
    byte[] buffer=new byte[1024];
    int ch;
    while ((ch = inputStream.read(buffer)) != -1) {
        outFile.write(buffer,0,ch);
    }
    inputStream.close();
    outFile.flush();
    outFile.close();
    return new File(getTempOutFile(task));
}
```

第一个想法就是： 没超时控制，加上呗。

```
//downloadTimeout= 1200s

private File downloadZipFile(Task task) throws DeployScheduleException, IOException {
    ...
    long start = System.currentTimeMillis();
    String outFilePath = getTempOutFile(task);
    boolean successful = false;
    try (InputStream inputStream = response.getEntity().getContent();
         FileOutputStream fos = new FileOutputStream(outFilePath)) {

        LOGGER.info("#Scheduler#{}#download file at {}", task.getId(), getTempOutFile(task));
        byte[] buffer = new byte[1024];
        int ch;
        while ((ch = inputStream.read(buffer)) != -1) {
            fos.write(buffer, 0, ch);
            if (System.currentTimeMillis() - start > downloadTimeout * 1000) {
                throw new IOException(format("#Scheduler#%d#fetching timeout: %ds", task.getId(
, downloadTimeoutSeconds));
            }
        }
        successful = true;
        fos.flush();
    } finally {
        if (!successful) {
            FileUtils.deleteQuietly(new File(outFilePath));
        }
    }
    return new File(getTempOutFile(task));
}
```

Review 时 大神们的意见就来了:

- timeout 加上单位吧
- httpClient 自己有超时控制，自己造轮子有风险
- 判断条件长了语义化一下比较好

嗯嗯嗯，有道理，改改改。。。

```
//downloadTimeoutSeconds= 1200s

private File downloadZipFile(Task task) throws DeployScheduleException, IOException {
        LOGGER.debug("#Scheduler#{}#download start", task.getId());
        SocketConfig socketConfig = SocketConfig.custom()
                .setSoTimeout(downloadTimeoutSeconds * 1000)
                .build();
        HttpGet httpGet = new HttpGet(task.getArchive_url());
        final String tempOutFile = getTempOutFile(task);
        try(CloseableHttpClient client = HttpClientBuilder.create().setDefaultSocketConfig(socke
Config).build()) {
            dbHelper.updateTaskStatusAndReason(task.getId(), "FETCHING", "");

            HttpEntity resEntity = client.execute(httpGet).getEntity();
            long contentLength = resEntity.getContentLength();
            long limit = sizeLimit << 20;
            if (contentLength > limit) {
                dbHelper.updateTaskLog(task.getId(), "Site archive size is more than " + sizeLimit +
M.");
```

```
                throw new DeployScheduleException("#Scheduler#" + task.getId() + "#Archive too
ig#size: {}" + contentLength);
            }
            String contentType = resEntity.getContentType().getValue();
            if (isZipFile(task, contentType)) {
                dbHelper.updateTaskLog(task.getId(), "Project or branch not found.");
                throw new DeployScheduleException("#Scheduler#" + task.getId() + "#invalid cont
nt type: " + contentType);
            }
            try (InputStream inputStream = response.getEntity().getContent();
                 FileOutputStream fos = new FileOutputStream(outFilePath)) {

                LOGGER.info("#Scheduler#{}#download file at {}", task.getId(), getTempOutFile(task))

                byte[] buffer = new byte[1024];
                int ch;
                while ((ch = inputStream.read(buffer)) != -1) {
                    fos.write(buffer, 0, ch);
                    if (System.currentTimeMillis() - start > downloadTimeout * 1000) {
                        throw new IOException(format("#Scheduler#%d#fetching timeout: %ds", task.
etId(), downloadTimeoutSeconds));
                    }
                }
                fos.flush();
            }
        } catch (SocketTimeoutException e) {
            FileUtils.deleteQuietly(new File(tempOutFile));
            throw new IOException(String.format("#Scheduler#%d#fetching timeout: %ds", task.g
tId(), downloadTimeoutSeconds));
        }
        return new File(tempOutFile);
    }

    private boolean isZipFile(Task task, String contentType) {
        return !contentType.contains("octet-stream") &&
                !contentType.contains("zip") &&
                !task.getArchive_url().endsWith(".zip");
    }
```

这下应该没问题了吧，继续提交

又有人质疑了这段

```
try (InputStream inputStream = response.getEntity().getContent();
     FileOutputStream fos = new FileOutputStream(outFilePath)) {

    LOGGER.info("#Scheduler#{}#download file at {}", task.getId(), getTempOutFile(task));
    byte[] buffer = new byte[1024];
    int ch;
    while ((ch = inputStream.read(buffer)) != -1) {
        fos.write(buffer, 0, ch);
        if (System.currentTimeMillis() - start > downloadTimeout * 1000) {
            throw new IOException(format("#Scheduler#%d#fetching timeout: %ds", task.getId(),
ownloadTimeoutSeconds));
        }
    }
    fos.flush();
}
```

都什么时候还用这么原始的方式， JDK nio 啊，Files.copy() 一下搞定，还不比你的优秀啊

去翻翻源码还真有这么个

```
/**
 * Copies all bytes from an input stream to a file. On return, the input
 * stream will be at end of stream.
 *
 * <p> By default, the copy fails if the target file already exists or is a
 * symbolic link. If the {@link StandardCopyOption#REPLACE_EXISTING
 * REPLACE_EXISTING} option is specified, and the target file already exists,
 * then it is replaced if it is not a non-empty directory. If the target
 * file exists and is a symbolic link, then the symbolic link is replaced.
 * In this release, the {@code REPLACE_EXISTING} option is the only option
 * required to be supported by this method. Additional options may be
 * supported in future releases.
 *
 * <p>  If an I/O error occurs reading from the input stream or writing to
 * the file, then it may do so after the target file has been created and
 * after some bytes have been read or written. Consequently the input
 * stream may not be at end of stream and may be in an inconsistent state.
 * It is strongly recommended that the input stream be promptly closed if an
 * I/O error occurs.
 *
 * <p> This method may block indefinitely reading from the input stream (or
 * writing to the file). The behavior for the case that the input stream is
 * <i>asynchronously closed</i> or the thread interrupted during the copy is
 * highly input stream and file system provider specific and therefore not
 * specified.
 *
 * <p> <b>Usage example</b>: Suppose we want to capture a web page and save
 * it to a file:
 * <pre>
 *     Path path = ...
 *     URI u = URI.create("http://java.sun.com/");
 *     try (InputStream in = u.toURL().openStream()) {
 *         Files.copy(in, path);
 *     }
 * </pre>
 *
 * @param   in
 *          the input stream to read from
 * @param   target
 *          the path to the file
 * @param   options
 *          options specifying how the copy should be done
 *
 * @return  the number of bytes read or written
 *
 * @throws  IOException
 *          if an I/O error occurs when reading or writing
 * @throws  FileAlreadyExistsException
 *          if the target file exists but cannot be replaced because the
 *          {@code REPLACE_EXISTING} option is not specified <i>(optional
 *          specific exception)</i>
 * @throws  DirectoryNotEmptyException
 *          the {@code REPLACE_EXISTING} option is specified but the file
 *          cannot be replaced because it is a non-empty directory
 *          <i>(optional specific exception)</i>     *
 * @throws  UnsupportedOperationException
 *          if {@code options} contains a copy option that is not supported
 * @throws  SecurityException
 *          In the case of the default provider, and a security manager is
 *          installed, the {@link SecurityManager#checkWrite(String) checkWrite}
 *          method is invoked to check write access to the file. Where the
 *          {@code REPLACE_EXISTING} option is specified, the security
 *          manager's {@link SecurityManager#checkDelete(String) checkDelete}
 *          method is invoked to check that an existing file can be deleted.
 */
public static long copy(InputStream in, Path target, CopyOption... options)
    throws IOException
{...}
```

最后忽然觉得用 throw 来终止过程好像还可以精简。

哟西，最终结果出炉。

```
private File downloadZipFile(Task task) throws DeployScheduleException, IOException {
    LOGGER.debug("#Scheduler#{}#download start", task.getId());
    SocketConfig socketConfig = SocketConfig.custom()
            .setSoTimeout(downloadTimeoutSeconds * 1000)
            .build();
    HttpGet httpGet = new HttpGet(task.getArchive_url());
    final String tempOutFile = getTempOutFile(task);
    try(CloseableHttpClient client = HttpClientBuilder.create().setDefaultSocketConfig(socketC
nfig).build()) {
        dbHelper.updateTaskStatusAndReason(task.getId(), "FETCHING", "");

        HttpEntity responseEntity = client.execute(httpGet).getEntity();
        checkContentLength(task, responseEntity);
        checkContentType(task, responseEntity);
        try (InputStream inputStream = responseEntity.getContent()) {
            Files.copy(inputStream, Paths.get(tempOutFile), StandardCopyOption.REPLACE_EXISTI
G);
        }
    } catch (SocketTimeoutException e) {
        FileUtils.deleteQuietly(new File(tempOutFile));
        throw new IOException(String.format("#Scheduler#%d#fetching timeout: %ds", task.getI
(), downloadTimeoutSeconds));
    }
    return new File(tempOutFile);
}

private void checkContentType(Task task, HttpEntity resEntity) throws DeployScheduleExcepti
```

```
n {
    String contentType = resEntity.getContentType().getValue();
    if (isZipFile(task, contentType)) {
        dbHelper.updateTaskLog(task.getId(), "Project or branch not found.");
        throw new DeployScheduleException("#Scheduler#" + task.getId() + "#invalid content ty
e: " + contentType);
    }
}

private void checkContentLength(Task task, HttpEntity resEntity) throws DeployScheduleExce
tion {
    long contentLength = resEntity.getContentLength();
    // 1 << 20 = 1M
    long limit = sizeLimit << 20;
    if (contentLength > limit) {
        dbHelper.updateTaskLog(task.getId(), "Site archive size is more than " + sizeLimit + "M.")

        throw new DeployScheduleException("#Scheduler#" + task.getId() + "#Archive too big#s
ze: {}" + contentLength);
    }
}

private boolean isZipFile(Task task, String contentType) {
    return !contentType.contains("octet-stream") &&
            !contentType.contains("zip") &&
            !task.getArchive_url().endsWith(".zip");
}
```

重构小结

- 对类库的使用要熟悉，多测试，少写很多代码而且语义清晰
- 长判断条件提炼方法，语义化更方便看
- 要多看看 JDK 的东西，很多基础的操作里面都有
- 配置参数如果有单位最好加上单位，减少使用风险

原文链接：记一次代码重构