再造一次FTP连接池轮子

Feb 2, 2019

背景

在此之前，本人曾经写过两篇总结文章，《封装一个FTP工具类》和《使用commons-pool管理FTP连接》，相应地造了两个小项目轮子。两个项目之间有相同点，也有不同的侧重点：都是使用`commons-net`和`commons-pool`库构建的，前者最大的特点是使用了模板方法设计模式，借鉴了`JdbcTemplate`的实现，造了上传下载工具类方法；后者最大的特点是使用`AutoCloseable`的方式去设计FTP连接（`FTPConnection`），借鉴的是`BasicDataSource`的实现，重点是连接的管理重用。

第一个项目的问题是和`Spring`绑定了，且底层使用了**KeyedObjectPool**模式，实现上比较简陋，不利于扩展；第二个项目倒不是存在很大的问题，只是不够简洁，还有改进的空间，但是由于一开始设计时上考虑太多，束缚了自己，项目无法演进。不难看出，本人关心的是项目的扩展性，在项目上能不能实践一些知识点，是一种练笔的心态；因此，姑且将这个新轮子取名ftpcp2；

重构

本次重构的想法酝酿了很久了，总结过去的经验教训，这次重构本着一种alpha的心态，保持代码思路简洁，一点点地加功能，没有考虑API是否稳定，将近一个月的时间收获不少，一些是关于代码架构设计的，一些是关于工具链的；

架构思路

设计一个`Manager`类，底层是一个连接池，能够返回一个封装后的`Client`对象，这个`Client`对象`close`时并不是真正的关闭连接，而是回收到连接池，再次从`Manager`获取连接时就能重用之前的`Client`对象；

```
public interface FTPClientManager extends Closeable {

    PooledFTPClient getFTPClient() throws Exception;
```

```
}
```

封装后的`Client`类型实际上继承自`FTPClient`，特别的地方是拥有连接池对象的引用，

```
public class PooledFTPClient extends FTPClient implements AutoCloseable {

    /**
     * ref to object pool which this object will return to
     */
    protected GenericObjectPool<PooledFTPClient> connectionPool;
    ...
}
```

具体的`FTPClientManager`实现类，需要考虑FTP连接参数，对象池参数的设置，它的签名大致是这样子的，

```
public class BasicFTPClientManager extends FTPClientManagerConfig implements
FTPClientManager {}
```

将参数设置移动到父类，父类的成员属性包含FTP连接相关的参数即可，再继承`GenericObjectPoolConfig`也拥有对象池参数设置的能力了，

```
public class FTPClientManagerConfig extends GenericObjectPoolConfig {

    protected int initialSize;

    protected String serverTimeZoneId;
    protected Integer bufferSize;
    protected String controlEncoding;
    protected Long keepAliveTimeout;
    protected Integer controlKeepAliveReplyTimeout;
    protected Integer connectTimeout;
    protected String dataConnectionMode;


    protected String host;
    protected int port;

    protected String username;
    protected String password;
}
```

这些API方法参数 不一定是固定的，更多的要求进来也会修改的，比如FTP 连接参数还有其他 可以设置的；如果需要支持ftps协议，`PooledFTPClient`应该如何扩展呢，会不会改动到现在的设计呢，这些都是 值得思考的，开放的；

具体实现简述

既然是一个简洁的轮子，那么它的实现应该也没有那么复杂的，通过看源码的方式也能看懂整体脉络；

首先根据参数配置创建**FTP**连接工厂；
- 然后根据**FTP**连接工厂创建连接池，将连接池设置到 **FTP**连接工厂（这里出现了循环依赖）；
- 然后根据连接池创建新的 `Manager` 实现类（它不暴露到外部）；
- 最后，获取连接就委托新的 `Manager`，再委托给连接池，向连接池借用；
- 连接使用后，调用 `close` 方法 即返回连接池；

实现上已经考虑多线程初次获取连接多次创建连接工厂的并非发问题，使用了 `DCL + volatile` 组合方式解决；多线程向连接池获取（borrow）连接的并发问题，由连接池自己解锁解决的；

jitpack.io

如果想把这个项目作为项目的依赖，一种做法是将它打包上传到maven中央仓库，一种做法是克隆这个工程，使用maven等构建工具install到本地仓库；后面发现了一种更加方便的方式，使用jitpack.io，它就像一个特殊的中央仓库，会根据你的依赖去github等源码库下载并打包，然后分发下来；

比如要使用这个工程作为依赖，只需要两步，在 `pom.xml` 文件中添加一个仓库源，

```
<repositories>
    <repository>
        <id>jitpack.io</id>
        <url>https://jitpack.io</url>
    </repository>
</repositories>
```

再添加约定方式的依赖，

```
<dependency>
    <groupId>com.github.honwhy</groupId>
    <artifactId>ftpcp2</artifactId>
    <version>9212f64a2b</version>
</dependency>
```

这个 `version` 版本号可以写源码库的commitId；groupId 约定为com.github.{user}，也因此我修改了这个工程的package名字；

为了表示这个项目是不稳定的，还特地加上了官网提供的badge， 刚好jitpack.io也提供了badge，所以最终的效果是这样的，

spring-boot-starter

在上一版本的 项目ftpcp，有人向我提议写一个 `spring-boot-starter`，当时还没有具体去了解这是什么东西，这次趁机也写了一个，ftpcp2-spring-boot-starter，名字符合 `spring-boot-starter` 风格约定，当然这也是一个不稳定的项目，写这个项目 的时候刚好验证了 jitpack.io功能；

`ftpcp2-spring-boot-starter` 能够根据参数配置，当发现需要 `Manager` 时会实例化一个，注册到Spring容器，后续就能通过依赖注入的方式使用它，也即能使用FTP连接了；

jmx

连接池已经具有了jmx的支持了，不过它只有对象池属性相关的监控管理，这次给`Manager`加上了jmx功能，关注的是FTP参数方面，并且将底层连接池的jmx也关联起来，这在一个项目中有多`Manager`时才能分辨`Manager`关联的连接池；实现它时改动了`Manager`，为它继承实现一个`MBean`接口，其他莫过如此；

集成ftpserver到单元测试

不管是前面两个项目，还是这个项目开始阶段，跑单元测试之前必做的一件事就是启动一个ftp server，然后才是跑单元测试，结束后还需要自己停止ftp server；联想到jdbc连接池的单元测试一般使用的是内存数据库h2，在单元测试方法前启动，在方法后停止，是不是ftpserver也有In memory的实现方式呢。经过一番搜索发现了Apache的开源项目`ftpserver`，这是一个ftp server的实现，读了它的单元测试类就发现类似的原理，启动ftp server，用`FTPClient`连接测试，结束后停止ftp server；

到这里我的想法就变成了将`ftpserver`集成到`ftpcp2`的单元测试中，将它作为test级别的依赖...最后面实现的效果很满意，拷贝了`ftpserver`的单元测试的代码（这些代码引用不到，它们本身是`ftpserver`的test部分），实现自己的`TestCase`再改造一点原来的单元测试就顺利串联起来了；

不过这里引入了新的问题，关于如何使用开源项目授权的问题，我拷贝了`apache license 2.0`授权项目的代码，我的项目应该以什么license开源呢，需要对这些拷贝的代码做什么说明吗。

总结

这个重构项目比上一版本简洁了许多了，目前实践了jmx，spring-boot-starter等技能点，学到了更方便合理的单元测试方式；目前对这个项目的进度把握还算满意，可能和它没有高深的知识技能要求有关，后续还会继续借鉴优秀开源项目在它之上实践；

tbc;

Honwhy Wang's Blog

Honwhy Wang's Blog Honwhy Be good to each other.
honwhy.wang@gmail.com