**背景**

最近接到一个安全生产相关需求 ：需要在开发提测时在提测卡口上增加一个校验点，当提测时代码错误码个数有变化会检测出来，继而推动开发确认新增的错误码是否可降级之后再提测。调研之后发现有一个java的开源工具包jgit可以辅助实现该功能，感觉非常方便好用，故分享一下

**jgit**

﻿<https://www.eclipse.org/jgit/>﻿

可以通过java实现git的操作，如pull，push，commit等操作，最重要可以支持diff的功能，可以解决我们本次需求

**思路**

在其他应用上提供一个接口，在提测卡口上调用，传入需要校验的代码仓库地址以及分支名称，通过jgit的diff功能，比较当前分支与master分支的差异文件，然后解析错误码文件，将错误码统计出来，返回新增的错误码

**maven坐标**

1

<dependency>

2

<groupId>org.eclipse.jgit</groupId>

3

<artifactId>org.eclipse.jgit</artifactId>

4

<version>5.13.0.202109080827-r</version>

5

</dependency>

**核心代码**

**【克隆仓库】**

http方式：

1

//clone操作默认checkout的是仓库的master分支，如果想checkout其他分支，可以通过setBranch()方法显示指定

2

git = Git.cloneRepository()

3

.setCredentialsProvider(new UsernamePasswordCredentialsProvider("域账户", "域密码"))

4

.setURI(remotePath)//远程仓库的地址，http链接

5

.setDirectory(new File(localPath))//本地仓库的路径

6

.call();

ssh方式：

用户名+密码适用于本地调试，如果要发代码最好申请公共账号，或通过ssh方式，jgit提供了抽象类JSchConfigSessionFactory可以建立ssh连接，需要重写configure和createDefaultJSch方法，高本版jgit使用ssh方式需额外引入依赖：

1

<!--低版本不需要下面这个依赖 -->

2

<dependency>

3

<groupId>org.eclipse.jgit</groupId>

4

<artifactId>org.eclipse.jgit.ssh.jsch</artifactId>

5

<version>5.13.0.202109080827-r</version>

6

</dependency>

1

SshSessionFactory sshSessionFactory = new JschConfigSessionFactory() {

2

​

3

@Override

4

protected void configure(OpenSshConfig.Host host, Session session) {

5

//不加这个配置可能会遇到UnknownHostKey异常

6

session.setConfig("StrictHostKeyChecking", "no");

7

}

8

​

9

@Override

10

protected JSch createDefaultJSch(FS fs) throws JSchException {

11

JSch defaultJSch = super.createDefaultJSch(fs);

12

defaultJSch.removeAllIdentity();

13

defaultJSch.addIdentity("/xxx/id\_rsa");//这里如果只传入私钥路径，默认会在相同路径寻找公钥文件

14

return defaultJSch;

15

}

16

};

17

​

18

git = Git.cloneRepository()

19

.setTransportConfigCallback(transport -> {

20

SshTransport sshTransport = (SshTransport) transport;

21

sshTransport.setSshSessionFactory(sshSessionFactory);

22

})

23

.setURI(projectRemotePath)

24

.setCloneAllBranches(true)

25

.setNoCheckout(true)

26

.setDirectory(new File(localPath))

27

.call();

关于如何生成公私钥文件可以去ATA上搜一下，很多文章，此处不赘述，可以利用Dockerfile上传公私钥文件到机器

可能会遇到问题【JSCH连接SSH报错：Invalid privatekey】

原因：生成秘钥的Openssl工具版本较高，工程中的jsch版本无法正常使用此版本的秘钥

解法：生成密钥时指定格式，ssh-keygen -m PEM -t rsa -C "youremail"，加上参数-m PEM

**【切换分支】**

1

/\*\*

2

\* 切换分支

3

\*

4

\* @param branchName 分支名称

5

\* @throws GitAPIException GitAPIException

6

\*/

7

public void checkOut(String branchName) throws GitAPIException, IOException {

8

log.info("check out:" + branchName);

9

//1.获取此分支的Ref信息

10

Ref branchRef = getBranchRef(branchName);

11

//2.如果没有该分信息，从远程拉取分支

12

if (branchRef == null) {

13

git.checkout()

14

.setCreateBranch(true)

15

.setName(branchName)

16

.setStartPoint("origin/" + branchName)

17

.setUpstreamMode(CreateBranchCommand.SetupUpstreamMode.SET\_UPSTREAM)

18

.call();

19

} else {

20

git.checkout().setCreateBranch(false).setName(branchName).call();

21

}

22

}

23

​

24

/\*\*

25

\* 根据branchName获取分支信息

26

\* Ref对象表示.git/refs下面的文件引用repository.getRef("refs/heads/xxx")

27

\*

28

\* @param branchName

29

\* @return

30

\*/

31

public Ref getBranchRef(String branchName) throws IOException {

32

return git.getRepository().exactRef("refs/heads/" + branchName);

33

}

**【pull分支】**

1

/\*\*

2

\* 拉取最新的提交

3

\*

4

\* @throws GitAPIException

5

\*/

6

public void pull() throws GitAPIException {

7

log.info("git pull...");

8

git.pull().setTransportConfigCallback(transport -> {

9

SshTransport sshTransport = (SshTransport) transport;

10

sshTransport.setSshSessionFactory(sshSessionFactory);

11

}).call();

12

}

**【通过分支名获取分支对应的树结构】**

1

/\*\*

2

\* 将分支解析为树结构

3

\*

4

\* @param branchName

5

\* @return

6

\* @throws Exception

7

\*/

8

public AbstractTreeIterator treeParser(String branchName) throws IOException, GitAPIException {

9

log.info("treeParser:" + branchName);

10

​

11

RevWalk walk = new RevWalk(repository);

12

Ref ref = getBranchRef(branchName);

13

if (ref == null) {

14

checkOut(branchName);

15

ref = getBranchRef(branchName);

16

}

17

RevTree tree = walk.parseTree(ref.getObjectId());

18

​

19

CanonicalTreeParser treeParser = new CanonicalTreeParser();

20

ObjectReader oldReader = repository.newObjectReader();

21

treeParser.reset(oldReader, tree.getId());

22

​

23

walk.dispose();

24

​

25

return treeParser;

26

}

**【通过commitId获取分支的树结构】**

如果想通过git.diff()比较两次commit的差异，就需要通过commitId解析出树结构

1

/\*\*

2

\* 通过CommitId获取分支树结构信息

3

\*

4

\* @param commitId

5

\* @return

6

\* @throws IOException

7

\*/

8

public AbstractTreeIterator prepareTreeParser(String commitId) throws IOException {

9

RevWalk walk = new RevWalk(repository);

10

RevCommit revCommit = walk.parseCommit(repository.resolve(commitId));

11

RevTree tree = walk.parseTree(revCommit.getTree().getId());

12

CanonicalTreeParser treeParser = new CanonicalTreeParser();

13

ObjectReader reader = repository.newObjectReader();

14

treeParser.reset(reader, tree.getId());

15

walk.dispose();

16

return treeParser;

17

}

**【比较两个分支的差异】**

1

/\*\*

2

\* 比较两个分支的差异

3

\*

4

\* @param branchName

5

\* @return

6

\* @throws IOException

7

\* @throws GitAPIException

8

\*/

9

public List<DiffEntry> diffBranchWithMaster(String branchName) throws IOException, GitAPIException {

10

log.info("diff with master:" + branchName);

11

pull();

12

​

13

AbstractTreeIterator masterTree = treeParser("master");

14

AbstractTreeIterator featureTree = treeParser(branchName);

15

​

16

return git.diff()

17

.setOldTree(masterTree)

18

.setNewTree(featureTree)

19

.setShowNameAndStatusOnly(true)

20

.call();

21

}

**【获取指定分支、指定文件内容】**

1

/\*\*

2

\* 获取指定分支的指定文件内容

3

\*

4

\* @param branchName

5

\* @param filePath

6

\* @return

7

\* @throws Exception

8

\*/

9

public byte[] fileContentParser(String branchName, String filePath) throws IOException, GitAPIException {

10

System.out.println("fileContentParser:" + branchName);

11

12

//可以遍历提交对象，并按照顺序返回提交对象

13

RevWalk walk = new RevWalk(repository);

14

Ref ref = getBranchRef(branchName);

15

if (ref == null) {

16

checkOut(branchName);

17

ref = getBranchRef(branchName);

18

}

19

//RevTree 代表树对象

20

RevTree tree = walk.parseTree(ref.getObjectId());

21

​

22

TreeWalk treeWalk = TreeWalk.forPath(repository, filePath, tree);

23

//如果master没有该文件

24

if (treeWalk == null) {

25

return new byte[0];

26

}

27

ObjectId blobId = treeWalk.getObjectId(0);

28

ObjectLoader loader = repository.open(blobId);

29

​

30

return loader.getBytes();

31

}

**【打印文件差异内容】**

1

//找到差异文件

2

List<DiffEntry> diff = diffBranchWithMaster(req.getBranchName());

3

ByteArrayOutputStream out = new ByteArrayOutputStream();

4

DiffFormatter df = new DiffFormatter(out);

5

​

6

//设置比较器为忽略空白字符对比（Ignores all whitespace）

7

df.setDiffComparator(RawTextComparator.WS\_IGNORE\_ALL);

8

df.setRepository(git.getRepository());

9

​

10

for (DiffEntry diffEntry : diff) {

11

//打印文件差异具体内容

12

df.format(diffEntry);

13

FileHeader fileHeader = df.toFileHeader(diffEntry);

14

System.out.println("----------" + fileHeader.getNewPath());

15

String diffText = out.toString("UTF-8");

16

System.out.println(diffText);

17

}

**【列出所有的远程分支】**

可通过该方法获取远程分支的remoteRefObjectId和本地的localRefObjectId比较，判断本地分支是否是最新版本

1

// 获取远程所有分支

2

Collection<Ref> remoteRefs = git.lsRemote().setTransportConfigCallback(transport -> {

3

SshTransport sshTransport = (SshTransport) transport;

4

sshTransport.setSshSessionFactory(sshSessionFactory);

5

}).setHeads(true).call();

6

​

7

for (Ref remoteRef : remoteRefs) {

8

String remoteRefName = remoteRef.getName();

9

String remoteRefObjectId = remoteRef.getObjectId().getName();

10

}

**【获取分支的提交记录】**

1

//RevCommit 代表一个提交对象

2

List<Ref> branches = git.branchList().setListMode(ListMode.REMOTE).call();

3

try(RevWalk walk = new RevWalk(git.getRepository())) {

4

for(Ref branch : branches) {

5

RevCommit commit = walk.parseCommit(branch.getObjectId());

6

System.out.println("Time committed: " + commit.getCommitterIdent().getWhen());

7

System.out.println("Time authored: " + commit.getAuthorIdent().getWhen());

8

}

9

}

﻿

因其他git操作没有使用到，故没有列出，如有好的建议欢迎指出~

﻿

**参考资料**

﻿<https://github.com/centic9/jgit-cookbook>﻿

﻿<https://wiki.eclipse.org/JGit/User_Guide>﻿

﻿<https://www.javacodegeeks.com/2015/12/clone-git-repositories-jgit.html>﻿

﻿<http://www.codeaffine.com/2014/12/09/jgit-authentication/>﻿

﻿<http://doc.yonyoucloud.com/doc/wiki/project/pro-git-two/jgit.html>﻿

﻿<https://gitee.com/RainyGao/DocSys>