# 1.文档介绍

## 1.1目的

在做一个项目中需要用到远程仓库，与git相比，svn显得笨重且不方便，因此放弃了svn转而使用git。java git api - jgit的资料还是比较多的，而且git的操作比svn更容易理解。

如果你想在一个 Java 程序中使用 Git ，有一个功能齐全的 Git 库，那就是 JGit 。 JGit 是一个用 Java 写成的功能相对健全的 Git 的实现，它在 Java 社区中被广泛使用。 JGit 项目由 Eclipse 维护，它的主页在 <http://www.eclipse.org/jgit> 。

非常好的例子：https://github.com/centic9/jgit-cookbook

# 2.背景介绍

基于项目管理工具需求，我们需要自动创建新分支、自动合并分支、自动删除分支的功能。这样就涉及到用java代码调用git操作git分支。那么此时使用JGit就十分合适了，正好符合我们的需求。

# 3.JGit使用

## 3.1使用JGit

**<repositories>**

**<repository>**

**<id>jgit-repository</id>**

**<url>https://repo.eclipse.org/content/groups/releases/</url>**

**</repository>**

**</repositories>**

**<!-- Core Library -->**

**<dependencies>**

**<dependency>**

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

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

**<version>4.6.1.201703071140-r</version>**

**</dependency>**

**</dependencies>**

**<!-- Smart HTTP Servlet -->**

**<dependencies>**

**<dependency>**

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

**<artifactId>org.eclipse.jgit.http.server</artifactId>**

**<version>4.6.1.201703071140-r</version>**

**</dependency>**

**</dependencies>**

**<!-- AWT UI Helpers -->**

**<dependencies>**

**<dependency>**

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

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

**<version>4.6.1.201703071140-r</version>**

**</dependency>**

**</dependencies>**

**<!-- JUnit Test Support -->**

**<dependencies>**

**<dependency>**

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

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

**<version>4.6.1.201703071140-r</version>**

**</dependency>**

**</dependencies>**

在pom文件中引入JGit的储藏库和核心包

## 3.2新建分支并同步到远程

public class GitUtilClass {

public static String localRepoPath = "D:/repo";

public static String localRepoGitConfig = "D:/repo/.git";

public static String remoteRepoURI = "git@gitlab.com:wilson/test.git";

public static String localCodeDir = "D:/platplat";

/\*\*

\* 新建一个分支并同步到远程仓库

\* @param branchName

\* @throws IOException

\* @throws GitAPIException

\*/

public static String newBranch(String branchName){

String newBranchIndex = "refs/heads/"+branchName;

String gitPathURI = "";

Git git;

try {

//检查新建的分支是否已经存在

List<Ref> refs = git.branchList().call();

for (Ref ref : refs) {

if (ref.getName().equals(newBranchIndex)) {

System.out.println("have branch before");

break;

}

}

//新建分支

Ref ref = git.branchCreate().setName(branchName).call();

//推送到远程

git.push().add(ref).call();

gitPathURI = remoteRepoURI + " " + "feature/" + branchName;

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (GitAPIException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return gitPathURI;

}

## 3.3合并分支

## 3.4删除分支

public class GitUtilClass {

public static String localRepoPath = "D:/repo";

public static String localRepoGitConfig = "D:/repo/.git";

public static String remoteRepoURI = "git@gitlab.com:wilson/test.git";

public static String localCodeDir = "D:/platplat";

/\*\*

\* 删除一个分支

\* @param branchName

\* @throws IOException

\* @throws GitAPIException

\*/

public static boolean deleteBranch(String branchName){

String newBranchIndex = "refs/heads/"+branchName;

String gitPathURI = "";

Git git;

try {

//检查新建的分支是否存在，如果存在则强制删除

List<Ref> refs = git.branchList().call();

for (Ref ref : refs) {

if (ref.getName().equals(newBranchIndex)) {

System.out.println("Removing branch before");

git.branchDelete().setBranchNames(branchName).setForce(true).call();

break;

}

}

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (GitAPIException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return true;

}