

Миграция из dbf формата dos кодировки cp866 в mySql utf-8

@Component  
@RequiredArgsConstructor  
public class DbfToDbMigratorImpl implements DbfToDbMigrator {  
  
 private static final String *EMPTY\_STRING* = "";  
  
 @Value(value = "${declaration.dbf}")  
 private String declaration\_dbf;  
  
 @Value(value = "${export.dbf}")  
 private String export\_dbf;  
  
 @Value(value = "${consumption.dbf}")  
 private String consumption\_dbf;  
  
 @Value(value = "${arrival.dbf}")  
 private String arrival\_dbf;  
  
 @Value(value = "${directory}")  
 private String directory;  
  
 private final ExportService exportService;  
 private final ImportService importService;  
 private final DeclarationService declarationService;  
 private final CurrencyService currencyService;  
 private final UnitService unitService;  
 private final ChangelogService changelogService;  
  
 @Override  
 @Transactional  
 public synchronized MessageDto migrateAll() {  
 deleteAll();  
  
 declarationService.importDeclarationsFromDbf(directory + declaration\_dbf);  
 exportService.importExportsFromDbf(directory + export\_dbf);  
 importService.importImportsFromDbf(directory + consumption\_dbf, directory + arrival\_dbf);  
 return MessageDto.*builder*().message(*EMPTY\_STRING*).build();  
 }  
  
 @Override  
 @Transactional  
 public synchronized MessageDto migrateLastUpdate() {  
 try {  
 declarationService.importLastUpdateDeclarationsFromDbf(directory + declaration\_dbf);  
 exportService.importLastUpdateExportsFromDbf(directory + export\_dbf);  
 importService.importLastUpdateImportsFromDbf(directory + consumption\_dbf,  
 directory + arrival\_dbf);  
 } catch (Exception exception) {  
 return MessageDto.*builder*().message(exception.getMessage()).build();  
 }  
 return MessageDto.*builder*().message(*EMPTY\_STRING*).build();  
 }  
  
 @Override  
 public synchronized Boolean isActive() {  
 boolean isDeclarationActive = declarationService.isMigrateActive(directory + declaration\_dbf);  
 boolean isExportActive = exportService.isMigrateActive(directory + export\_dbf);  
 boolean isImportActive = importService.isMigrateActive(directory + consumption\_dbf,  
 directory + arrival\_dbf);  
 return isDeclarationActive && isExportActive && isImportActive;  
 }  
  
 @Override  
 public synchronized void deleteAll() {  
 exportService.deleteAll();  
 importService.deleteAll();  
 declarationService.deleteAll();  
 unitService.deleteAll();  
 currencyService.deleteAll();  
 changelogService.deleteAll();  
 }  
}