# PROJETO CUSTOMIZAÇÕES

Sincronismo de AD via robô

Amanda Alvim

# 1. Objetivo

O objetivo dessa customização é auxiliar para realizar o sincronismo automático de usuários do ad do cliente para o Lecom BPM, esse robô deve ser utilizado em versões anteriores a 5.40 1.04.3, pois nessa versão para frente terá automatizado esse sincronismo via ferramenta.

O fonte principal que fará esse sincronismo é o robô **RbSincronizaAD** que está no projeto do git (http://git.lecom.com.br/PSP/Projeto-Base-BPM) no pacote com.lecom.workflow.robo.satelite.LDAP, para o funcionamento dele ele utilizará algumas outras classes que estão no pacote com.lecom.workflow.robo.satelite.LDAP.dao e com.lecom.workflow.robo.satelite.LDAP.model.

# 2. Como usar

No momento em que forem utilizar podem apenas gerar o jar do RbSincronizaAd usando o jardesc já salvo nesse projeto na pasta jardesc/RbSincronizarAD.jardesc e pegar o arquivo RbSincronizaAd.properties que está em upload/cadastros/config/producao alterar com as informações do ambiente do cliente, subir o properties no servidor seja da forma nova no /opt/lecom/custom/web-content/config/ ou da forma antiga no /opt/lecom/app/tomcat/webapps/bpm/upload/cadastros/config posteriormente subir o jar no modulo serviços -> robô e avaliar o log da execução vendo que irá inserir todos os usuários, funções, departamentos.

# 3. Robô de Sincronismo

RbSincronizaAD.java

```
48
50  * <p>
72  @RobotModule("RbSincronizaAD")
73  @Version({ 1, 1, 12 })
74  public class RbSincronizaAD implements WebServices {
75      private Integer TESTANDO = 0;
76
77      private static final Logger logger = Logger.getLogger(RbSincronizaAD.class);
78
79      private String configPath = Funcoes.getWFRootDir() + "/upload/cadastros/config/";
80      private static final String COD_USUARIO = "COD_USUARIO";
81      private static final String DEPARTAMENTO = "DEPARTAMENTO";
82      private static final String FUNCAO = "FUNCAO";
83
84      // tags para serem coletadas do ad
85      private static final String NOME_AD = "name";
86      private static final String EMAIL_AD = "mail";
87      private static final String LOGIN_AD = "SamAccountName";
88      private static final String FUNCAO_AD = "title";
89      private static final String LIDER_AD = "Manager";
90      private static final String DEPARTAMENTO_AD = "department";
91      private static final String DES_CAMPO_INTEGRACAO1 = "employeeType";
92      private static final String DES_CAMPO_INTEGRACAO2 = "employeeNumber";
93      private static final String DES_CAMPO_INTEGRACAO3 = "employeeID";
94
95      private String EMAIL_DEFAULT = null;
96      private String LOGINS_EXECOES = null;
97      private List<BPMUsuario> usuariosBPM = null;
98      private Integer COD_LIDER_DEFAULT = null;
99      private Integer COD_DEPARTAMENTO_DEFAULT = null;
100
101     Map<String, String> parametros = null;
102
103     @Execution
104     public void executar() {
105         logger.info("\n\n" + new String(new char[150]).replace("\0", "#") + "\n\n");
106         logger.info("[==== INICIO " + getClass().getSimpleName() + " ====]\n");
107
108         try (Connection connection = DBUtils.getConnection("workflow")) {
109             connection.setAutoCommit(false);
110             try {
111                 parametros = Funcoes.getParametrosIntegracao(configPath + getClass().getSimpleName());
112
113                 Calendar dataAtual = Calendar.getInstance();
```

RbSincronizaAD.java

```
String horaAtual = Integer.toString(dataAtual.get(Calendar.HOUR_OF_DAY));

logger.debug("horaAtual : " + horaAtual);

EMAIL_DEFAULT = parametros.get("emailGenerico");
logger.debug("EMAIL_DEFAULT : " + EMAIL_DEFAULT);

String horasExecucao = parametros.get("horasExecucao");
logger.info("horaExecucao : " + horasExecucao);

boolean executarRobo = Arrays.asList(horasExecucao.split(";")).contains(horaAtual);

TESTANDO = Integer.parseInt(Funcoes.nulo(parametros.get("teste"), "0"));

if (!executarRobo && !horasExecucao.equals("0")) {
    logger.info("[ ===== ROTINAS NÃO DEVEM SER EXECUTADAS AGORA ===== ]");

} else {

    LOGINS_EXECOES = parametros.get("loginsExcecao");
    COD_LIDER_DEFAULT = Integer.parseInt(parametros.get("liderPadrao"));
    COD_DEPARTAMENTO_DEFAULT = Integer.parseInt(parametros.get("departamentoPadrao"));

    // Informações do bpm
    logger.info("[===== GERANDO LISTA DE USUARIO BPM =====]");
    usuariosBPM = BPMUsuarioClusterDAO.getTodosUsuarios(connection, logger);

    logger.debug("usuariosBPM : " + usuariosBPM);

    logger.info("[===== GERANDO LISTA DE DEPARTAMENTOS BPM =====]");
    List<String> departamentosBPM = BPMDeptoClusterDAO.getTodosDeptos(connection, logger).stream()
            .map(BPMDepto::getComando).collect(Collectors.toList());

    logger.info("[===== GERANDO LISTA DE FUNCOES BPM =====]");
    List<String> funcoesBPM = BPMFuncaoClusterDAO.getTodosFuncoes(connection, logger).stream()
            .map(BPMFuncao::getComando).collect(Collectors.toList());

    logger.debug("funcoesBPM -> " + funcoesBPM);

    // Informações do LDAP
    logger.info("[===== GERANDO LISTA DE USUARIO AD =====]");
    List<Map<String, String>> usuariosAtivosAD = getUsuariosAD("S");
    logger.info("[===== COLETADOS " + usuariosAtivosAD.size() + " USUARIOS NO AD =====]");
```

RbSincronizaAD.java

```
logger.debug("usuariosAtivosAD -> " + usuariosAtivosAD);

logger.info("[===== GERANDO LISTA DE FUNCOES AD =====]");
List<String> funcoesAD = getInfoUnica(usuariosAtivosAD, FUNCAO_AD);
logger.info("[===== COLETADOS " + funcoesAD.size() + " FUNCOES NO AD =====]");

logger.info("[===== GERANDO LISTA DE DEPARTAMENTOS AD =====]");
List<String> departamentosAD = getInfoUnica(usuariosAtivosAD, DEPARTAMENTO_AD);
logger.info("[===== COLETADOS " + departamentosAD.size() + " DEPARTAMENTOS NO AD =====]");

ArrayList<Map<String, String>> departamentosCadastrados = new ArrayList<Map<String, String>>();

logger.info("[===== CRIANDO DEPARTAMENTOS NO BPM =====]");

for (String nomeDepartamento : departamentosAD) {

    String nomeLimitado = limitaDado(nomeDepartamento, 15);
    String apelido = geraApelido(corrigeEspecial(nomeLimitado));

    if (!departamentosBPM.contains(apelido) && !departamentosBPM.contains(nomeLimitado)
            && !departamentosBPM.contains(nomeDepartamento) && !"".equals(nomeDepartamento)) {
        try {
            int codigoDepartamento = BPMDeptoClusterDAO.incluirDepto(connection, logger,
                    nomeLimitado, apelido, nomeDepartamento, COD_LIDER_DEFAULT, COD_LIDER_DEFAULT);

            Map<String, String> retorno = new HashMap<String, String>();

            retorno.put(DEPARTAMENTO, nomeLimitado);
            retorno.put(COD_USUARIO, Integer.toString(codigoDepartamento));

            departamentosCadastrados.add(retorno);

        } catch (Exception e) {
            logger.error(
                    "ERRO AO CADASTRAR O DEPARTAMENTO " + nomeLimitado + ": " + e.getMessage());
        }
    } else {
        int id = BPMDeptoClusterDAO.getCodDepto(connection, apelido);
        BPMDeptoClusterDAO.atualizaCampoDepto(connection, logger, "DES_DESCRICAO", nomeDepartamento,
                id);
        logger.info(nomeDepartamento + " Atualizado");
    }
}
```

RbSincronizaAD.java

```
201
202                     logger.info("[===== " + departamentosCadastrados.size() + " DEPARTAMENTOS CADASTRADOS =====]");
203
204                     ArrayList<Map<String, String>> funcoesCadastradas = new ArrayList<Map<String, String>>();
205
206                     logger.info("[===== CRIANDO FUNCOES NO BPM =====]");
207                     for (String nomeFuncao : funcoesAD) {
208                         String nomeLimitado = limitaDado(nomeFuncao, 15);
209                         String apelido = geraApelido(corrigeEspecial(nomeLimitado));
210
211                         if (!funcoesBPM.contains(apelido) && !funcoesBPM.contains(nomeLimitado)
212                                 && !funcoesBPM.contains(nomeFuncao) && !"".equals(nomeLimitado)) {
213                             try {
214                                 int codigoFuncao = BPMFuncaoClusterDAO.incluirFuncao(connection, logger, nomeLimitado,
215                                         apelido, nomeFuncao, COD_LIDER_DEFAULT.toString());
216
217                                 Map<String, String> retorno = new HashMap<String, String>();
218
219                                 retorno.put(FUNCAO, nomeLimitado);
220                                 retorno.put(COD_USUARIO, Integer.toString(codigoFuncao));
221
222                                 funcoesCadastradas.add(retorno);
223
224                             } catch (Exception e) {
225                                 logger.error("ERRO AO CADASTRAR A FUNÇÃO " + nomeLimitado + ": " + e.getMessage());
226                             }
227                         } else {
228                             int id = BPMFuncaoClusterDAO.getCodFuncoes(connection, logger, apelido);
229                             BPMFuncaoClusterDAO.atualizaCampoFuncoes(connection, logger, "DES_DESCRICAO", nomeFuncao,
230                                     id);
231                             logger.info(nomeFuncao + " Atualizado");
232                         }
233                     }
234
235                     logger.info("[===== " + funcoesCadastradas.size() + " FUNCOES CADASTRADAS =====]");
236
237                     logger.info("[===== CADASTRANDO USUARIOS NO BPM =====]");
238
239                     for (int i = 0; i < usuariosAtivosAD.size(); i++) {
240                         String loginUsuario = usuariosAtivosAD.get(i).get(LOGIN_AD);
241                         String nomeUsuario = usuariosAtivosAD.get(i).get(NOME_AD);
242                         String emailUsuario = usuariosAtivosAD.get(i).get(EMAIL_AD);
243                         String campoIntegracao1 = usuariosAtivosAD.get(i).get(DES_CAMPO_INTEGRACAO1);
244                         String campoIntegracao2 = usuariosAtivosAD.get(i).get(DES_CAMPO_INTEGRACAO2);
245                         String campoIntegracao3 = usuariosAtivosAD.get(i).get(DES_CAMPO_INTEGRACAO3);
```

RbSincronizaAD.java

```
247                         Integer codUsuario = insereAtualizaUsuario(connection, loginUsuario, nomeUsuario, emailUsuario,
248                                 COD_LIDER_DEFAULT, COD_DEPARTAMENTO_DEFAULT, campoIntegracao1, campoIntegracao2,
249                                 campoIntegracao3);
250                         usuariosAtivosAD.get(i).put(COD_USUARIO, Integer.toString(codUsuario));
251                     }
252
253                     logger.info("[===== " + usuariosAtivosAD.size() + " USUARIOS MANIPULADOS =====]");
254
255                     logger.info("[===== ATUALIZA LIDER USUARIOS NO BPM =====]");
256                     for (Map<String, String> usuarioAD : usuariosAtivosAD) {
257                         String nomeLider = usuarioAD.get(LIDER_AD);
258                         Optional<BPMUsuario> optioncalUsuario = usuariosBPM.stream()
259                                 .filter(u -> nomeLider.equalsIgnoreCase(u.getNomUsuario())).findFirst();
260
261                         if (optioncalUsuario.isPresent()) {
262                             Integer codLider = optioncalUsuario.get().getCodUsuario();
263                             String loginUsuario = usuarioAD.get(LOGIN_AD);
264                             String nomeUsuario = usuarioAD.get(NOME_AD);
265                             String emailUsuario = usuarioAD.get(EMAIL_AD);
266                             String campoIntegracao1 = usuarioAD.get(DES_CAMPO_INTEGRACAO1);
267                             String campoIntegracao2 = usuarioAD.get(DES_CAMPO_INTEGRACAO2);
268                             String campoIntegracao3 = usuarioAD.get(DES_CAMPO_INTEGRACAO3);
269
270                             insereAtualizaUsuario(connection, loginUsuario, nomeUsuario, emailUsuario, codLider,
271                                     COD_DEPARTAMENTO_DEFAULT, campoIntegracao1, campoIntegracao2, campoIntegracao3);
272                         }
273                     }
274
275                     logger.info("[===== GERANDO LISTA DE DEPARTAMENTOS BPM ATUALIZADA =====]");
276                     List<BPMDepto> departamentosBPMObj = BPMDeptoClusterDAO.getTodosDeptos(connection, logger);
277
278                     logger.info("[===== GERANDO LISTA DE FUNCOES BPM ATUALIZADA =====]");
279                     List<BPMFuncao> funcoesBPMObj = BPMFuncaoClusterDAO.getTodosFuncoes(connection, logger);
280
281                     logger.info("[===== ATUALIZA FUNCOES DOS USUARIOS NO BPM =====]");
282                     for (Map<String, String> usuarioAD : usuariosAtivosAD) {
283                         String funcao = limitaDado(usuarioAD.get(FUNCAO_AD), 15);
284
285                         Optional<BPMFuncao> optFuncao = funcoesBPMObj.stream()
286                                 .filter(f -> funcao.equalsIgnoreCase(f.getNome())).findFirst();
287
288                         if (optFuncao.isPresent()) {
289                             Integer codFuncao = optFuncao.get().getId();
290
```

RbSincronizaAD.java

```
290
291                         if (!BPMFuncaoClusterDAO.isUsuarioNoFuncoes(connection, logger,
292                                 Integer.parseInt(usuarioAD.get(COD_USUARIO)), codFuncao)) {
293                             BPMFuncaoClusterDAO.insereUsuarioNaFuncao(connection, logger,
294                                     Integer.parseInt(usuarioAD.get(COD_USUARIO)), codFuncao);
295                         }
296                     }
297                 }
298
299                 logger.info("[===== ATUALIZA DEPARTAMENTOS DOS USUARIOS NO BPM =====]");
300                 for (Map<String, String> usu : usuariosAtivosAD) {
301                     String departamento = limitaDado(usu.get(DEPARTAMENTO_AD), 15);
302
303                     Optional<BPMDepto> optDepartamento = departamentosBPMObj.stream()
304                             .filter(d -> departamento.equalsIgnoreCase(d.getNome())).findFirst();
305
306                     if (optDepartamento.isPresent()) {
307                         Integer codigoDepartamento = optDepartamento.get().getId();
308
309                         int codigoUsuario = Integer.parseInt(usu.get(COD_USUARIO));
310                         if (!BPMDeptoClusterDAO.isUsuarioNoDepto(connection, codigoUsuario, codigoDepartamento)) {
311                             BPMDeptoClusterDAO.insereUsuarioNoDepto(connection, codigoUsuario, codigoDepartamento);
312                         }
313                     }
314                 }
315
316                 logger.info("[===== GERANDO LISTA DE USUARIO INATIVOS NO AD =====]");
317                 List<Map<String, String>> usuariosInativosAD = getUsuariosAD("N");
318                 logger.info("[===== COLETADOS " + usuariosInativosAD.size() + " USUARIOS INATIVOS NO AD =====]");
319
320                 logger.info("[===== INATIVANDO USUARIOS DO AD =====]");
321
322                 for (Map<String, String> usuarioInativo : usuariosInativosAD) {
323                     String loginUsuario = usuarioInativo.get(LOGIN_AD);
324                     String nomeUsuario = usuarioInativo.get(NOME_AD);
325
326                     logger.debug("INATIVANDO USUARIO -> " + loginUsuario + " - " + nomeUsuario);
327
328                     BPMUsuarioClusterDAO.setUsuarioAtivo(connection, loginUsuario, false);
329                 }
330
331                 logger.info("[===== USUARIOS INATIVADOS =====]");
332             }
333         } catch (Exception e) {
```

RbSincronizaAD.java

```
329                 }
330
331                 logger.info("[===== USUARIOS INATIVADOS =====]");
332             }
333         } catch (Exception e) {
334             connection.rollback();
335             logger.error("[==== ROLLBACK ATIVADO ====]");
336             throw e;
337         } finally {
338             if (TESTANDO == 1) {
339                 connection.rollback();
340                 logger.error("[==== ROLLBACK PARA TESTES ====]");
341             }
342             connection.setAutoCommit(true);
343         }
344     } catch (Exception e) {
345         logger.error("[==== ERRO DURANTE A EXECUÇÃO ====]");
346         logger.error(Funcoes.exceptionPrinter(e));
347         e.printStackTrace();
348     } finally {
349         logger.info("\n");
350         logger.info("[==== FIM " + getClass().getSimpleName() + " ====]");
351     }
352 }
```

Nas imagens acima temos o método principal da classe onde ele irá coletar as informações e chamar alguns métodos abaixo e as outras classes que utilizam.

Nas **linhas 85 a 93** estão sendo definidos variáveis estáticas com nomes dos atributos que o ad usa em relação essas informações.

Na **linha 128** é validado está dentro do período de execução configurado no properties.

A partir da **linha 133** começa pegando algumas informações do arquivo de properties.

Na **linha 139** utiliza-se um DAO para buscar todos os usuários cadastrados no Lecom BPM.

Na **linha 144 e 145** está utilizando outro DAO para buscar todos os departamentos cadastrados no Lecom BPM, trazendo uma lista de String dos desComando cadastrados.

Na **linha 148 e 149** está utilizando outro DAO para buscar todos as funções cadastradas no Lecom BPM, trazendo uma lista de String dos comando cadastrados.

Na **linha 155** ele retorna todos os usuários que estão no AD, na sequência ele pega funções, departamentos que existem no AD, tudo colocando numa lista de String.

Nas **linhas 172 a 200** está sendo lido os departamentos existentes no AD para comparar com o que temos no Lecom BPM e assim na **linha 180** ele insere o departamento na nossa base caso não tenha ainda, ou simplesmente atualiza a informação na **linha 196.**

Nas **linhas 207 a 233** está sendo lida as funções existentes no Ad, para comparar com as existentes no Lecom BPM e se não existir é feito a inserção como na **linha 214** e uma atualização caso precise na **linha 229.**

Nas **linhas 237 a 251** está pegando os usuários do ad, e passando informações para o método insereAtualizaUsuario, internamente no método é avaliado se existe o login desse usuário dentro do Lecom BPM para realizar a inserção ou atualização.

Já nas **linhas 256 a 273** ele pega os usuários do ad, pegando o seu líder, e passando as informações desse líder caso ele seja encontrado dentro do Lecom BPM (**linha 258,**está utilizando mecanismo do java 8 para encontrar dentro de uma lista a informação que desejamos), assim chama-se a função insereAtualizaUsuario.

Nas **linhas 276 e 279** ele busca novamente todos os departamentos e funções cadastrados no Lecom BPM, nesse momento essa lista já vem atualizada.

Nas **linhas 282 a 297** ele novamente faz uma leitura dos usuários do ad, e atualiza as funções desses usuários.

Nas **linhas 300 a 314** ele novamente faz uma leitura dos usuários do ad, e atualiza os departamentos desses usuários.

Nas **linhas 317 a 329** ele retorna todos os usuários que estão inativos no AD e assim inativa o usuário no Lecom BPM.

RbSincronizaAD.java

```
356     * @param cnLecom
368    private int insereAtualizaUsuario(Connection cnLecom, String usuarioLogin, String usuarioNome, String usuarioEmail,
369            int codLider, int codDepto, String campoIntegracao1, String campoIntegracao2, String campoIntegracao3)
370            throws Exception {
371        logger.debug("insereAtualizaUsuario -- usuarioLogin -> " + usuarioLogin + " ; usuarioNome -> " + usuarioNome
372                + " ; usuarioEmail -> " + usuarioEmail + " ; codLider -> " + codLider + " ; codDepto -> " + codDepto
373                + " ; campoIntegracao1 -> " + campoIntegracao1 + " ; campoIntegracao2 -> " + campoIntegracao2
374                + " ; campoIntegracao3 -> " + campoIntegracao3);
375
376
377        int grupoPadrao = BPMDeptoClusterDAO.getCodDepto(cnLecom, parametros.get("gruposPadrao"));
378
379        logger.debug("GRUPO PADRÃO : " + grupoPadrao);
380
381        // Verifica se USUARIO já está cadastrado
382        Optional<BPMUsuario> optioncalUsuario = usuariosBPM.stream()
383                .filter(u -> usuarioLogin.equalsIgnoreCase(u.getDesLogin())).findFirst();
384        int codUsuarioBPM = -1;
385
386        // Caso exista pega o código dele
387        if (optioncalUsuario.isPresent()) {
388
389
390            codUsuarioBPM = optioncalUsuario.get().getCodUsuario();
391
392            logger.debug("codUsuarioBPM Atualizado : " + codUsuarioBPM);
393            logger.debug("validarAD : " + parametros.get("validarAD"));
394
395            // Update do usuário
396            BPMUsuarioClusterDAO.atualizaUsuario(cnLecom, logger, codUsuarioBPM, usuarioNome, usuarioEmail, codLider,
397                    codDepto, campoIntegracao1, campoIntegracao2, campoIntegracao3, "N", parametros.get("validarAD"));
398        }
399        else {
400            // Caso usuário não exista deve ser inserido
401            codUsuarioBPM = BPMUsuarioClusterDAO.inserirUsuarioAutenticacao(cnLecom, logger, usuarioNome, usuarioLogin,
402                    usuarioLogin, usuarioEmail, COD_LIDER_DEFAULT, "N", "P", codDepto, campoIntegracao1,
403                    campoIntegracao2, campoIntegracao3, "LDAP", parametros.get("validarAD"));
404
405            // Insere todo usuário no grupo de Acesso BPM
406            BPMGrupoClusterDAO.insereUsuarioNoGrupo(cnLecom, logger, codUsuarioBPM, grupoPadrao);
407
408            // Atualiza usuariosBPM com novo usuário cadastrado
409            BPMUsuario usuario = new BPMUsuario();
```

RbSincronizaAD.java

```
408            // Atualiza usuariosBPM com novo usuario cadastrado
409            BPMUsuario usuario = new BPMUsuario();
410
411            usuario.setCodUsuario(codUsuarioBPM);
412            usuario.setDesLogin(usuarioLogin);
413            usuario.setNomUsuario(usuarioNome);
414            usuario.setDesEmail(usuarioEmail);
415            usuario.setDesCampoIntegracao1(campoIntegracao1);
416            usuario.setDesCampoIntegracao2(campoIntegracao2);
417            usuario.setDesCampoIntegracao3(campoIntegracao3);
418
419            usuariosBPM.add(usuario);
420        }
421
422        return codUsuarioBPM;
423    }
424
426     * Pega informações unicas da lista
433    private List<String> getInfoUnica(List<Map<String, String>> dados, String valor) {
434        List<String> retorno = new ArrayList<String>();
435
436        // coleta os valores unicos do map e remove valor em branco
437        retorno = dados.stream().map(d -> d.get(valor)).distinct().filter(v -> !Funcoes.nulo(v, "").trim().equals(""))
438                .collect(Collectors.toList());
439
440        return retorno;
441    }
```

Nas imagens acima temos dois métodos o **insereAtualizaUsuario** e o **getInfoUnica**, o primeiro faz conforme foi explicado na página anterior ele verifica se encontra usuário do ad no Lecom BPM e insere ou atualiza a informação chamando uma classe externa. O método getInfoUnica ele está realizando um distinct diante dos valores que estão sendo passado para eles, então de um map de funções, departamentos retorna uma lista de String.

RbSincronizaAD.java

```
442
444⊕     * Coleta usuários válidos do AD
454⊖    private List<Map<String, String>> getUsuariosAD(String usuariosAtivos) throws Exception {
455
456         String urlServer = parametros.get("url_server");
457         String loginAd = parametros.get("login_ad");
458         String senhaAd = parametros.get("senha_ad");
459         String filtroLDAP = usuariosAtivos.equals("S") ? Funcoes.nulo(parametros.get("ldapFilter"), "")
460                 : Funcoes.nulo(parametros.get("ldapInactive"), "");
461         String sDesBase = parametros.get("sDesBase");
462
463         logger.info("UsuariosAtivios: "+usuariosAtivos);
464         logger.info("ldapFilter: "+Funcoes.nulo(parametros.get("ldapFilter"), ""));
465         logger.info("ldapInactive: "+Funcoes.nulo(parametros.get("ldapInactive"), ""));
466         logger.info("sDesBase: "+ parametros.get("sDesBase"));
467
468         List<Map<String, String>> usuariosAD = new ArrayList<Map<String, String>>();
469
470         if (!urlServer.toLowerCase().startsWith("ldap"))
471             urlServer = "ldap://" + urlServer;
472
473         Hashtable<String, Object> ldapEnv = new Hashtable<String, Object>(6);
474         ldapEnv.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
475
476         // dados de conexão
477         ldapEnv.put(Context.PROVIDER_URL, urlServer);
478         ldapEnv.put(Context.SECURITY_AUTHENTICATION, "simple");
479         ldapEnv.put(Context.SECURITY_PRINCIPAL, loginAd);
480         ldapEnv.put(Context.SECURITY_CREDENTIALS, senhaAd);
481         ldapEnv.put(Context.SECURITY_PROTOCOL, "simple");
482
483         LdapContext ctx = new InitialLdapContext(ldapEnv, null);
484
485         // Quantidade de resultados por vez
486         Integer pageSize = 2000;
487         byte[] cookie = null;
488         ctx.setRequestControls(new Control[] { new PagedResultsControl(pageSize, Control.NONCRITICAL) });
489
490         do {
491             SearchControls searchControl = new SearchControls();
492             searchControl.setSearchScope(SearchControls.SUBTREE_SCOPE);
493             NamingEnumeration<SearchResult> results = ctx.search(sDesBase, filtroLDAP, searchControl);
494
495             try {
```

RbSincronizaAD.java

```
494
495             try {
496                 while (results != null && results.hasMore()) {
497                     SearchResult entry = results.next();
498                     Attributes entryValues = entry.getAttributes();
499
500                     // logger.debug("entryValues : " + entryValues.toString());
501
502                     String name = limitaDado(coletaDado(entryValues, NOME_AD), 50);
503                     String email = limitaDado(coletaDado(entryValues, EMAIL_AD), 255);
504                     String login = limitaDado(coletaDado(entryValues, LOGIN_AD), 50);
505                     String campoIntegracao1 = limitaDado(coletaDado(entryValues, DES_CAMPO_INTEGRACAO1), 50);
506                     String campoIntegracao2 = limitaDado(coletaDado(entryValues, DES_CAMPO_INTEGRACAO2), 50);
507                     String campoIntegracao3 = limitaDado(coletaDado(entryValues, DES_CAMPO_INTEGRACAO3), 50);
508                     String funcao = corrigeEspecial(coletaDado(entryValues, FUNCAO_AD));
509                     String departamento = corrigeEspecial(coletaDado(entryValues, DEPARTAMENTO_AD));
510                     String lider = coletaLider(entryValues);
511
512                     // informações cadastro
513                     logger.debug("name : " + name);
514                     logger.debug("email : " + email);
515                     logger.debug("login : " + login);
516                     logger.debug("campoIntegracao1 : " + campoIntegracao1);
517                     logger.debug("campoIntegracao2 : " + campoIntegracao2);
518                     logger.debug("campoIntegracao3 : " + campoIntegracao3);
519                     logger.debug("funcao : " + funcao);
520                     logger.debug("departamento : " + departamento);
521                     logger.debug("lider : " + lider);
522
523                     if ("".equals(name) || "".equals(login) || Arrays.asList(LOGINS_EXECOES.split(";")).contains(login))
524                         continue;
525
526                     if ("".equals(email))
527                         email = EMAIL_DEFAULT;
528
529                     Map<String, String> ldapUserMap = new HashMap<String, String>();
530                     ldapUserMap.put(NOME_AD, name.trim());
531                     ldapUserMap.put(EMAIL_AD, email.trim());
532                     ldapUserMap.put(LOGIN_AD, login);
533                     ldapUserMap.put(LIDER_AD, Funcoes.nulo(lider.trim(), "Administrador"));
534                     ldapUserMap.put(DES_CAMPO_INTEGRACAO1, campoIntegracao1.trim());
535                     ldapUserMap.put(DES_CAMPO_INTEGRACAO2, campoIntegracao2.trim());
536                     ldapUserMap.put(DES_CAMPO_INTEGRACAO3, campoIntegracao3.trim());
537                     ldapUserMap.put(DEPARTAMENTO_AD, Funcoes.nulo(departamento.trim(), "geral"));
```

RbSincronizaAD.java

```
536                     ldapUserMap.put(DES_CAMPO_INTEGRACAO3, campoIntegracao3.trim());
537                     ldapUserMap.put(DEPARTAMENTO_AD, Funcoes.nulo(departamento.trim(), "geral"));
538                     ldapUserMap.put(FUNCAO_AD, funcao.trim());
539
540                     usuariosAD.add(ldapUserMap);
541                 }
542             } catch (PartialResultException e) {
543                 // Ignored
544             }
545
546             Control[] controls = ctx.getResponseControls();
547
548             if (controls == null) {
549                 logger.info("No controls were sent from the server");
550             } else {
551                 for (Control control : controls) {
552                     if (control instanceof PagedResultsResponseControl) {
553                         PagedResultsResponseControl prrc = (PagedResultsResponseControl) control;
554                         cookie = prrc.getCookie();
555                     }
556                 }
557             }
558
559             ctx.setRequestControls(new Control[] { new PagedResultsControl(pageSize, cookie, Control.CRITICAL) });
560
561         } while (cookie != null);
562
563         logger.debug("usuariosAD.size() : " + usuariosAD.size());
564
565         return usuariosAD;
566
567     }
```

Nas imagens acima tem o método getUsuariosAD onde nesse método utiliza uma api para poder realizar a conexão com o AD a partir dos dados configurados e assim utilizar essas informações nos outros métodos.

RbSincronizaAD.java

```
566
567     }
568
570     * Limpa as informações do lider para conter apenas o nome do lider
576     private String coletaLider(Attributes entryValues) throws NamingException {
577         Attribute entrada = entryValues.get(LIDER_AD);
578         String lider = entrada != null ? Funcoes.nulo(entrada.get(), "") : "";
579
580         lider = lider.length() > 55 ? lider.substring(0, 55) : lider;
581
582         String regex = "CN=(.*?),";
583         Pattern pattern = Pattern.compile(regex);
584         Matcher matcher = pattern.matcher(lider);
585
586         if (matcher.find())
587             lider = matcher.group(1);
588
589         return Funcoes.nulo(lider, "");
590     }
591
593     * Coleta valor retornado pelo ad Obs: limite 0 = sem limite
601     private String coletaDado(Attributes entryValues, String valor) throws NamingException {
602         Attribute entrada = entryValues.get(valor);
603         String retorno = entrada != null ? Funcoes.nulo(entrada.get(), "") : "";
604
605         return retorno;
606     }
607
609     * Limita tamanho do valor
615     private String limitaDado(String valor, int limite) {
616
617         if (limite != 0) {
618             valor = valor.length() > limite ? valor.substring(0, limite) : valor;
619         }
620
621         return Normalizer.normalize(valor.trim(), Normalizer.Form.NFD).replaceAll("[^\\p{ASCII}]", "");
622     }
```

Na imagem acima tem 3 métodos **coletaLider**, **coletaDado** e **limitaDado**, o método **coletaLider** está pegando a informação do líder do usuário, o método **coletaDado** pega o valor do atributo passado como parâmetro, o método **limitaDado** retornando o valor com quantidade de caracteres que precisa.

RbSincronizaAD.java

```
        Attribute entrada = entryValues.get(valor);
        String retorno = entrada != null ? Funcoes.nulo(entrada.get(), "") : "";

        return retorno;
    }

     * Limita tamanho do valor
    private String limitaDado(String valor, int limite) {

        if (limite != 0) {
            valor = valor.length() > limite ? valor.substring(0, limite) : valor;
        }

        return Normalizer.normalize(valor.trim(), Normalizer.Form.NFD).replaceAll("[^\\p{ASCII}]", "");
    }

     * Remove corrige acentuação e caracteres invalidos
    private String corrigeEspecial(String valor) {
        String result = valor.trim().replaceAll("[!@#$%¨?&*/()+=\\\\\\\\-]", "");

        return result;
    }

    /**
     * Tira espacos do nome
     *
     * @param valor
     * @return
     */
    private String geraApelido(String valor) {
        return valor.trim().replaceAll("[ ]", "_").toLowerCase();
    }

    @Override
    public List<WebServicesVO> getWebServices() {
        return null;
    }

    @Override
    public void setWebServices(WebServicesVO arg0) {
    }

}
```

Na imagem acima tem o método **corrigeEspecial** ajustando os caracteres especiais.

As outras classes conforme mencionado no começo são classes internas que estão fazendo selects e dados uteis para a utilização na classe vista acima. Caso queiram fiquem a vontade a olhar, mas para utilizar não precisam somente configurar o properties, gerar o jar do robô a partir do jardesc e utilizar.