958 lines (885 sloc) | 27.6 KB ...

```go
package system

import (
	"bytes"
	"encoding/json"
	"errors"
	"fmt"
	"go/ast"
	"go/format"
	"go/parser"
	"go/token"
	"io"
	"log"
	"mime/multipart"
	"os"
	"path/filepath"
	"strconv"
	"strings"
	"text/template"

	cp "github.com/otiai10/copy"
	"go.uber.org/zap"
	"golang.org/x/text/cases"
	"golang.org/x/text/language"

	"github.com/flipped-aurora/gin-vue-admin/server/resource/autocode_template/subcontract"

	"github.com/flipped-aurora/gin-vue-admin/server/global"
	"github.com/flipped-aurora/gin-vue-admin/server/model/system"
```

```
30          "github.com/flipped-aurora/gin-vue-admin/server/utils"
31
32          "gorm.io/gorm"
33  )
34
35  const (
36          autoPath           = "autocode_template/"
37          autocodePath       = "resource/autocode_template"
38          plugPath           = "resource/plug_template"
39          packageService     = "service/%s/enter.go"
40          packageServiceName = "service"
41          packageRouter      = "router/%s/enter.go"
42          packageRouterName  = "router"
43          packageAPI         = "api/v1/%s/enter.go"
44          packageAPIName     = "api/v1"
45  )
46
47  type autoPackage struct {
48          path string
49          temp string
50          name string
51  }
52
53  var (
54          packageInjectionMap map[string]astInjectionMeta
55          injectionPaths      []injectionMeta
56          caser               = cases.Title(language.English)
57  )
58
59  func Init(Package string) {
60          injectionPaths = []injectionMeta{
61                  {
62                          path: filepath.Join(global.GVA_CONFIG.AutoCode.Root,
63                                  global.GVA_CONFIG.AutoCode.Server, global.GVA_CONFIG.AutoCode.SIni
64                          funcName:    "MysqlTables",
65                          structNameF: Package + ".%s{},",
66                  },
67                  {
68                          path: filepath.Join(global.GVA_CONFIG.AutoCode.Root,
69                                  global.GVA_CONFIG.AutoCode.Server, global.GVA_CONFIG.AutoCode.SIni
70                          funcName:    "Routers",
71                          structNameF: Package + "Router.Init%sRouter(PrivateGroup)",
72                  },
73                  {
74                          path: filepath.Join(global.GVA_CONFIG.AutoCode.Root,
75                                  global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
76                          funcName:    "ApiGroup",
77                          structNameF: "%sApi",
78                  },
```

```
                {
                        path: filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
                        funcName:    "RouterGroup",
                        structNameF: "%sRouter",
                },
                {
                        path: filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
                        funcName:    "ServiceGroup",
                        structNameF: "%sService",
                },
        }

        packageInjectionMap = map[string]astInjectionMeta{
                packageServiceName: {
                        path: filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Server, "service", "enter.go"),
                        importCodeF:  "github.com/flipped-aurora/gin-vue-admin/server/%s/%s",
                        packageNameF: "%s",
                        groupName:    "ServiceGroup",
                        structNameF:  "%sServiceGroup",
                },
                packageRouterName: {
                        path: filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Server, "router", "enter.go"),
                        importCodeF:  "github.com/flipped-aurora/gin-vue-admin/server/%s/%s",
                        packageNameF: "%s",
                        groupName:    "RouterGroup",
                        structNameF:  "%s",
                },
                packageAPIName: {
                        path: filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Server, "api/v1", "enter.go"),
                        importCodeF:  "github.com/flipped-aurora/gin-vue-admin/server/%s/%s",
                        packageNameF: "%s",
                        groupName:    "ApiGroup",
                        structNameF:  "%sApiGroup",
                },
        }
}

type injectionMeta struct {
        path        string
        funcName    string
        structNameF string // 带格式化的
}

type astInjectionMeta struct {
```

```
	path         string
	importCodeF  string
	structNameF  string
	packageNameF string
	groupName    string
}

type tplData struct {
	template         *template.Template
	autoPackage      string
	locationPath     string
	autoCodePath     string
	autoMoveFilePath string
}

type AutoCodeService struct{}

var AutoCodeServiceApp = new(AutoCodeService)

//@author: [songzhibin97](https://github.com/songzhibin97)
//@function: PreviewTemp
//@description: 预览创建代码
//@param: model.AutoCodeStruct
//@return: map[string]string, error

func (autoCodeService *AutoCodeService) PreviewTemp(autoCode system.AutoCodeStruct) (map[string]st
	makeDictTypes(&autoCode)
	for i := range autoCode.Fields {
		if autoCode.Fields[i].FieldType == "time.Time" {
			autoCode.HasTimer = true
			break
		}
		if autoCode.Fields[i].Require {
			autoCode.NeedValid = true
			break
		}
		if autoCode.Fields[i].Sort {
			autoCode.NeedSort = true
			break
		}
	}
	dataList, _, needMkdir, err := autoCodeService.getNeedList(&autoCode)
	if err != nil {
		return nil, err
	}

	// 写入文件前，先创建文件夹
	if err = utils.CreateDir(needMkdir...); err != nil {
		return nil, err
```

```
177         }
178
179         // 创建map
180         ret := make(map[string]string)
181
182         // 生成map
183         for _, value := range dataList {
184                 ext := ""
185                 if ext = filepath.Ext(value.autoCodePath); ext == ".txt" {
186                         continue
187                 }
188                 f, err := os.OpenFile(value.autoCodePath, os.O_CREATE|os.O_WRONLY, 0o755)
189                 if err != nil {
190                         return nil, err
191                 }
192                 if err = value.template.Execute(f, autoCode); err != nil {
193                         return nil, err
194                 }
195                 _ = f.Close()
196                 f, err = os.OpenFile(value.autoCodePath, os.O_CREATE|os.O_RDONLY, 0o755)
197                 if err != nil {
198                         return nil, err
199                 }
200                 builder := strings.Builder{}
201                 builder.WriteString("```")
202
203                 if ext != "" && strings.Contains(ext, ".") {
204                         builder.WriteString(strings.Replace(ext, ".", "", -1))
205                 }
206                 builder.WriteString("\n\n")
207                 data, err := io.ReadAll(f)
208                 if err != nil {
209                         return nil, err
210                 }
211                 builder.Write(data)
212                 builder.WriteString("\n\n```")
213
214                 pathArr := strings.Split(value.autoCodePath, string(os.PathSeparator))
215                 ret[pathArr[1]+"-"+pathArr[3]] = builder.String()
216                 _ = f.Close()
217
218         }
219         defer func() { // 移除中间文件
220                 if err := os.RemoveAll(autoPath); err != nil {
221                         return
222                 }
223         }()
224         return ret, nil
225 }
```

```
226
227 func makeDictTypes(autoCode *system.AutoCodeStruct) {
228         DictTypeM := make(map[string]string)
229         for _, v := range autoCode.Fields {
230                 if v.DictType != "" {
231                         DictTypeM[v.DictType] = ""
232                 }
233         }
234
235         for k := range DictTypeM {
236                 autoCode.DictTypes = append(autoCode.DictTypes, k)
237         }
238 }
239
240 //@author: [piexlmax](https://github.com/piexlmax)
241 //@function: CreateTemp
242 //@description: 创建代码
243 //@param: model.AutoCodeStruct
244 //@return: err error
245
246 func (autoCodeService *AutoCodeService) CreateTemp(autoCode system.AutoCodeStruct, ids ...uint) (e
247         makeDictTypes(&autoCode)
248         for i := range autoCode.Fields {
249                 if autoCode.Fields[i].FieldType == "time.Time" {
250                         autoCode.HasTimer = true
251                         break
252                 }
253                 if autoCode.Fields[i].Require {
254                         autoCode.NeedValid = true
255                         break
256                 }
257                 if autoCode.Fields[i].Sort {
258                         autoCode.NeedSort = true
259                         break
260                 }
261         }
262         // 增加判断: 重复创建struct
263         if autoCode.AutoMoveFile && AutoCodeHistoryServiceApp.Repeat(autoCode.BusinessDB, autoCode
264                 return RepeatErr
265         }
266         dataList, fileList, needMkdir, err := autoCodeService.getNeedList(&autoCode)
267         if err != nil {
268                 return err
269         }
270         meta, _ := json.Marshal(autoCode)
271         // 写入文件前，先创建文件夹
272         if err = utils.CreateDir(needMkdir...); err != nil {
273                 return err
274         }
```

```
275
276         // 生成文件
277         for _, value := range dataList {
278                 f, err := os.OpenFile(value.autoCodePath, os.O_CREATE|os.O_WRONLY, 0o755)
279                 if err != nil {
280                         return err
281                 }
282                 if err = value.template.Execute(f, autoCode); err != nil {
283                         return err
284                 }
285                 _ = f.Close()
286         }
287
288         defer func() { // 移除中间文件
289                 if err := os.RemoveAll(autoPath); err != nil {
290                         return
291                 }
292         }()
293         bf := strings.Builder{}
294         idBf := strings.Builder{}
295         injectionCodeMeta := strings.Builder{}
296         for _, id := range ids {
297                 idBf.WriteString(strconv.Itoa(int(id)))
298                 idBf.WriteString(";")
299         }
300         if autoCode.AutoMoveFile { // 判断是否需要自动转移
301                 Init(autoCode.Package)
302                 for index := range dataList {
303                         autoCodeService.addAutoMoveFile(&dataList[index])
304                 }
305                 // 判断目标文件是否都可以移动
306                 for _, value := range dataList {
307                         if utils.FileExist(value.autoMoveFilePath) {
308                                 return errors.New(fmt.Sprintf("目标文件已存在:%s\n", value.autoMoveF
309                         }
310                 }
311                 for _, value := range dataList { // 移动文件
312                         if err := utils.FileMove(value.autoCodePath, value.autoMoveFilePath); err
313                                 return err
314                         }
315                 }
316                 err = injectionCode(autoCode.StructName, &injectionCodeMeta)
317                 if err != nil {
318                         return
319                 }
320                 // 保存生成信息
321                 for _, data := range dataList {
322                         if len(data.autoMoveFilePath) != 0 {
323                                 bf.WriteString(data.autoMoveFilePath)
```

```
                                bf.WriteString(";")
                        }
                }

                var gormPath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                        global.GVA_CONFIG.AutoCode.Server, global.GVA_CONFIG.AutoCode.SInitialize,
                var routePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                        global.GVA_CONFIG.AutoCode.Server, global.GVA_CONFIG.AutoCode.SInitialize,
                var imporStr = fmt.Sprintf("github.com/flipped-aurora/gin-vue-admin/server/model/%
                _ = ImportReference(routePath, "", "", autoCode.Package, "")
                _ = ImportReference(gormPath, imporStr, "", "", "")

        } else { // 打包
                if err = utils.ZipFiles("./ginvueadmin.zip", fileList, ".", "."); err != nil {
                        return err
                }
        }
        if autoCode.AutoMoveFile || autoCode.AutoCreateApiToSql {
                if autoCode.TableName != "" {
                        err = AutoCodeHistoryServiceApp.CreateAutoCodeHistory(
                                string(meta),
                                autoCode.StructName,
                                autoCode.Description,
                                bf.String(),
                                injectionCodeMeta.String(),
                                autoCode.TableName,
                                idBf.String(),
                                autoCode.Package,
                        )
                } else {
                        err = AutoCodeHistoryServiceApp.CreateAutoCodeHistory(
                                string(meta),
                                autoCode.StructName,
                                autoCode.Description,
                                bf.String(),
                                injectionCodeMeta.String(),
                                autoCode.StructName,
                                idBf.String(),
                                autoCode.Package,
                        )
                }
        }
        if err != nil {
                return err
        }
        if autoCode.AutoMoveFile {
                return system.ErrAutoMove
        }
        return nil
```

```
}

//@author: [piexlmax](https://github.com/piexlmax)
//@function: GetAllTplFile
//@description: 获取 pathName 文件夹下所有 tpl 文件
//@param: pathName string, fileList []string
//@return: []string, error

func (autoCodeService *AutoCodeService) GetAllTplFile(pathName string, fileList []string) ([]strin
	files, err := os.ReadDir(pathName)
	for _, fi := range files {
		if fi.IsDir() {
			fileList, err = autoCodeService.GetAllTplFile(pathName+"/"+fi.Name(), file
			if err != nil {
				return nil, err
			}
		} else {
			if strings.HasSuffix(fi.Name(), ".tpl") {
				fileList = append(fileList, pathName+"/"+fi.Name())
			}
		}
	}
	return fileList, err
}

//@author: [piexlmax](https://github.com/piexlmax)
//@function: GetDB
//@description: 获取指定数据库和指定数据表的所有字段名,类型值等
//@param: tableName string, dbName string
//@return: err error, Columns []request.ColumnReq

func (autoCodeService *AutoCodeService) DropTable(BusinessDb, tableName string) error {
	if BusinessDb != "" {
		return global.GVA_DB.Exec("DROP TABLE " + tableName).Error
	} else {
		return global.MustGetGlobalDBByDBName(BusinessDb).Exec("DROP TABLE " + tableName).
	}
}

//@author: [SliverHorn](https://github.com/SliverHorn)
//@author: [songzhibin97](https://github.com/songzhibin97)
//@function: addAutoMoveFile
//@description: 生成对应的迁移文件路径
//@param: *tplData
//@return: null

func (autoCodeService *AutoCodeService) addAutoMoveFile(data *tplData) {
	base := filepath.Base(data.autoCodePath)
	fileSlice := strings.Split(data.autoCodePath, string(os.PathSeparator))
```

```
422         n := len(fileSlice)
423         if n <= 2 {
424                 return
425         }
426         if strings.Contains(fileSlice[1], "server") {
427                 if strings.Contains(fileSlice[n-2], "router") {
428                         data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root, glo
429                                 fmt.Sprintf(global.GVA_CONFIG.AutoCode.SRouter, data.autoPackage),
430                 } else if strings.Contains(fileSlice[n-2], "api") {
431                         data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
432                                 global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
433                 } else if strings.Contains(fileSlice[n-2], "service") {
434                         data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
435                                 global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
436                 } else if strings.Contains(fileSlice[n-2], "model") {
437                         data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
438                                 global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
439                 } else if strings.Contains(fileSlice[n-2], "request") {
440                         data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
441                                 global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
442                 }
443         } else if strings.Contains(fileSlice[1], "web") {
444                 if strings.Contains(fileSlice[n-1], "js") {
445                         data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
446                                 global.GVA_CONFIG.AutoCode.Web, global.GVA_CONFIG.AutoCode.WApi, b
447                 } else if strings.Contains(fileSlice[n-2], "form") {
448                         data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
449                                 global.GVA_CONFIG.AutoCode.Web, global.GVA_CONFIG.AutoCode.WForm,
450                 } else if strings.Contains(fileSlice[n-2], "table") {
451                         data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
452                                 global.GVA_CONFIG.AutoCode.Web, global.GVA_CONFIG.AutoCode.WTable,
453                 }
454         }
455 }
456
457 //@author: [piexlmax](https://github.com/piexlmax)
458 //@author: [SliverHorn](https://github.com/SliverHorn)
459 //@function: CreateApi
460 //@description: 自动创建api数据,
461 //@param: a *model.AutoCodeStruct
462 //@return: err error
463
464 func (autoCodeService *AutoCodeService) AutoCreateApi(a *system.AutoCodeStruct) (ids []uint, err e
465         apiList := []system.SysApi{
466                 {
467                         Path:        "/" + a.Abbreviation + "/" + "create" + a.StructName,
468                         Description: "新增" + a.Description,
469                         ApiGroup:    a.Abbreviation,
470                         Method:      "POST",
```

```
471                 },
472                 {
473                         Path:        "/" + a.Abbreviation + "/" + "delete" + a.StructName,
474                         Description: "删除" + a.Description,
475                         ApiGroup:    a.Abbreviation,
476                         Method:      "DELETE",
477                 },
478                 {
479                         Path:        "/" + a.Abbreviation + "/" + "delete" + a.StructName + "ByIds
480                         Description: "批量删除" + a.Description,
481                         ApiGroup:    a.Abbreviation,
482                         Method:      "DELETE",
483                 },
484                 {
485                         Path:        "/" + a.Abbreviation + "/" + "update" + a.StructName,
486                         Description: "更新" + a.Description,
487                         ApiGroup:    a.Abbreviation,
488                         Method:      "PUT",
489                 },
490                 {
491                         Path:        "/" + a.Abbreviation + "/" + "find" + a.StructName,
492                         Description: "根据ID获取" + a.Description,
493                         ApiGroup:    a.Abbreviation,
494                         Method:      "GET",
495                 },
496                 {
497                         Path:        "/" + a.Abbreviation + "/" + "get" + a.StructName + "List",
498                         Description: "获取" + a.Description + "列表",
499                         ApiGroup:    a.Abbreviation,
500                         Method:      "GET",
501                 },
502         }
503         err = global.GVA_DB.Transaction(func(tx *gorm.DB) error {
504                 for _, v := range apiList {
505                         var api system.SysApi
506                         if errors.Is(tx.Where("path = ? AND method = ?", v.Path, v.Method).First(&
507                                 if err = tx.Create(&v).Error; err != nil { // 遇到错误时回滚事务
508                                         return err
509                                 } else {
510                                         ids = append(ids, v.ID)
511                                 }
512                         }
513                 }
514                 return nil
515         })
516         return ids, err
517 }
518
519 func (autoCodeService *AutoCodeService) getNeedList(autoCode *system.AutoCodeStruct) (dataList []t
```

```
520         // 去除所有空格
521         utils.TrimSpace(autoCode)
522         for _, field := range autoCode.Fields {
523                 utils.TrimSpace(field)
524         }
525         // 获取 basePath 文件夹下所有tpl文件
526         tplFileList, err := autoCodeService.GetAllTplFile(autocodePath, nil)
527         if err != nil {
528                 return nil, nil, nil, err
529         }
530         dataList = make([]tplData, 0, len(tplFileList))
531         fileList = make([]string, 0, len(tplFileList))
532         needMkdir = make([]string, 0, len(tplFileList)) // 当文件夹下存在多个tpl文件时，改为map更合理
533         // 根据文件路径生成 tplData 结构体，待填充数据
534         for _, value := range tplFileList {
535                 dataList = append(dataList, tplData{locationPath: value, autoPackage: autoCode.Pac
536         }
537         // 生成 *Template，填充 template 字段
538         for index, value := range dataList {
539                 dataList[index].template, err = template.ParseFiles(value.locationPath)
540                 if err != nil {
541                         return nil, nil, nil, err
542                 }
543         }
544         // 生成文件路径，填充 autoCodePath 字段，readme.txt.tpl不符合规则，需要特殊处理
545         // resource/template/web/api.js.tpl -> autoCode/web/autoCode.PackageName/api/autoCode.Pack
546         // resource/template/readme.txt.tpl -> autoCode/readme.txt
547         for index, value := range dataList {
548                 trimBase := strings.TrimPrefix(value.locationPath, autocodePath+"/")
549                 if trimBase == "readme.txt.tpl" {
550                         dataList[index].autoCodePath = autoPath + "readme.txt"
551                         continue
552                 }
553
554                 if lastSeparator := strings.LastIndex(trimBase, "/"); lastSeparator != -1 {
555                         origFileName := strings.TrimSuffix(trimBase[lastSeparator+1:], ".tpl")
556                         firstDot := strings.Index(origFileName, ".")
557                         if firstDot != -1 {
558                                 var fileName string
559                                 if origFileName[firstDot:] != ".go" {
560                                         fileName = autoCode.PackageName + origFileName[firstDot:]
561                                 } else {
562                                         fileName = autoCode.HumpPackageName + origFileName[firstDo
563                                 }
564
565                                 dataList[index].autoCodePath = filepath.Join(autoPath, trimBase[:l
566                                         origFileName[:firstDot], fileName)
567                         }
568                 }
```

```
569
570                 if lastSeparator := strings.LastIndex(dataList[index].autoCodePath, string(os.Path
571                         needMkdir = append(needMkdir, dataList[index].autoCodePath[:lastSeparator]
572                 }
573         }
574         for _, value := range dataList {
575                 fileList = append(fileList, value.autoCodePath)
576         }
577         return dataList, fileList, needMkdir, err
578 }
579
580 // injectionCode 封装代码注入
581 func injectionCode(structName string, bf *strings.Builder) error {
582         for _, meta := range injectionPaths {
583                 code := fmt.Sprintf(meta.structNameF, structName)
584                 if err := utils.AutoInjectionCode(meta.path, meta.funcName, code); err != nil {
585                         return err
586                 }
587                 bf.WriteString(fmt.Sprintf("%s@%s@%s;", meta.path, meta.funcName, code))
588         }
589         return nil
590 }
591
592 func (autoCodeService *AutoCodeService) CreateAutoCode(s *system.SysAutoCode) error {
593         if s.PackageName == "autocode" || s.PackageName == "system" || s.PackageName == "example"
594                 return errors.New("不能使用已保留的package name")
595         }
596         if !errors.Is(global.GVA_DB.Where("package_name = ?", s.PackageName).First(&system.SysAuto
597                 return errors.New("存在相同PackageName")
598         }
599         if e := autoCodeService.CreatePackageTemp(s.PackageName); e != nil {
600                 return e
601         }
602         return global.GVA_DB.Create(&s).Error
603 }
604
605 func (autoCodeService *AutoCodeService) GetPackage() (pkgList []system.SysAutoCode, err error) {
606         err = global.GVA_DB.Find(&pkgList).Error
607         return pkgList, err
608 }
609
610 func (autoCodeService *AutoCodeService) DelPackage(a system.SysAutoCode) error {
611         return global.GVA_DB.Delete(&a).Error
612 }
613
614 func (autoCodeService *AutoCodeService) CreatePackageTemp(packageName string) error {
615         Init(packageName)
616         pendingTemp := []autoPackage{{
617                 path: packageService,
```

```
                name: packageServiceName,
                temp: string(subcontract.Server),
        }, {
                path: packageRouter,
                name: packageRouterName,
                temp: string(subcontract.Router),
        }, {
                path: packageAPI,
                name: packageAPIName,
                temp: string(subcontract.API),
        }}
        for i, s := range pendingTemp {
                pendingTemp[i].path = filepath.Join(global.GVA_CONFIG.AutoCode.Root, global.GVA_CO
        }
        // 选择模板
        for _, s := range pendingTemp {
                err := os.MkdirAll(filepath.Dir(s.path), 0755)
                if err != nil {
                        return err
                }

                f, err := os.Create(s.path)
                if err != nil {
                        return err
                }

                defer f.Close()

                temp, err := template.New("").Parse(s.temp)
                if err != nil {
                        return err
                }
                err = temp.Execute(f, struct {
                        PackageName string `json:"package_name"`
                }{packageName})
                if err != nil {
                        return err
                }
        }
        // 创建完成后在对应的位置插入结构代码
        for _, v := range pendingTemp {
                meta := packageInjectionMap[v.name]
                if err := ImportReference(meta.path, fmt.Sprintf(meta.importCodeF, v.name, package
                        return err
                }
        }
        return nil
}

```

```
type Visitor struct {
        ImportCode  string
        StructName  string
        PackageName string
        GroupName   string
}

func (vi *Visitor) Visit(node ast.Node) ast.Visitor {
        switch n := node.(type) {
        case *ast.GenDecl:
                // 查找有没有import context包
                // Notice: 没有考虑没有import任何包的情况
                if n.Tok == token.IMPORT && vi.ImportCode != "" {
                        vi.addImport(n)
                        // 不需要再遍历子树
                        return nil
                }
                if n.Tok == token.TYPE && vi.StructName != "" && vi.PackageName != "" && vi.GroupN
                        vi.addStruct(n)
                        return nil
                }
        case *ast.FuncDecl:
                if n.Name.Name == "Routers" {
                        vi.addFuncBodyVar(n)
                        return nil
                }

        }
        return vi
}

func (vi *Visitor) addStruct(genDecl *ast.GenDecl) ast.Visitor {
        for i := range genDecl.Specs {
                switch n := genDecl.Specs[i].(type) {
                case *ast.TypeSpec:
                        if strings.Index(n.Name.Name, "Group") > -1 {
                                switch t := n.Type.(type) {
                                case *ast.StructType:
                                        f := &ast.Field{
                                                Names: []*ast.Ident{
                                                        {
                                                                Name: vi.StructName,
                                                                Obj: &ast.Object{
                                                                        Kind: ast.Var,
                                                                        Name: vi.StructName,
                                                                },
                                                        },
                                                },
                                                Type: &ast.SelectorExpr{
```

```
716                                                     X: &ast.Ident{
717                                                             Name: vi.PackageName,
718                                                     },
719                                                     Sel: &ast.Ident{
720                                                             Name: vi.GroupName,
721                                                     },
722                                             },
723                                     }
724                                     t.Fields.List = append(t.Fields.List, f)
725                             }
726                     }
727             }
728     }
729     return vi
730 }
731
732 func (vi *Visitor) addImport(genDecl *ast.GenDecl) ast.Visitor {
733     // 是否已经import
734     hasImported := false
735     for _, v := range genDecl.Specs {
736             importSpec := v.(*ast.ImportSpec)
737             // 如果已经包含
738             if importSpec.Path.Value == strconv.Quote(vi.ImportCode) {
739                     hasImported = true
740             }
741     }
742     if !hasImported {
743             genDecl.Specs = append(genDecl.Specs, &ast.ImportSpec{
744                     Path: &ast.BasicLit{
745                             Kind:  token.STRING,
746                             Value: strconv.Quote(vi.ImportCode),
747                     },
748             })
749     }
750     return vi
751 }
752
753 func (vi *Visitor) addFuncBodyVar(funDecl *ast.FuncDecl) ast.Visitor {
754     hasVar := false
755     for _, v := range funDecl.Body.List {
756             switch varSpec := v.(type) {
757             case *ast.AssignStmt:
758                     for i := range varSpec.Lhs {
759                             switch nn := varSpec.Lhs[i].(type) {
760                             case *ast.Ident:
761                                     if nn.Name == vi.PackageName+"Router" {
762                                             hasVar = true
763                                     }
764                             }
```

```
765                         }
766                 }
767         }
768         if !hasVar {
769                 assignStmt := &ast.AssignStmt{
770                         Lhs: []ast.Expr{
771                                 &ast.Ident{
772                                         Name: vi.PackageName + "Router",
773                                         Obj: &ast.Object{
774                                                 Kind: ast.Var,
775                                                 Name: vi.PackageName + "Router",
776                                         },
777                                 },
778                         },
779                         Tok: token.DEFINE,
780                         Rhs: []ast.Expr{
781                                 &ast.SelectorExpr{
782                                         X: &ast.SelectorExpr{
783                                                 X: &ast.Ident{
784                                                         Name: "router",
785                                                 },
786                                                 Sel: &ast.Ident{
787                                                         Name: "RouterGroupApp",
788                                                 },
789                                         },
790                                         Sel: &ast.Ident{
791                                                 Name: caser.String(vi.PackageName),
792                                         },
793                                 },
794                         },
795                 }
796                 funDecl.Body.List = append(funDecl.Body.List, funDecl.Body.List[1])
797                 index := 1
798                 copy(funDecl.Body.List[index+1:], funDecl.Body.List[index:])
799                 funDecl.Body.List[index] = assignStmt
800         }
801         return vi
802 }
803
804 func ImportReference(filepath, importCode, structName, packageName, groupName string) error {
805         fSet := token.NewFileSet()
806         fParser, err := parser.ParseFile(fSet, filepath, nil, parser.ParseComments)
807         if err != nil {
808                 return err
809         }
810         importCode = strings.TrimSpace(importCode)
811         v := &Visitor{
812                 ImportCode:  importCode,
813                 StructName:  structName,
```

```
814                 PackageName: packageName,
815                 GroupName:   groupName,
816         }
817         if importCode == "" {
818                 ast.Print(fSet, fParser)
819         }
820
821         ast.Walk(v, fParser)
822
823         var output []byte
824         buffer := bytes.NewBuffer(output)
825         err = format.Node(buffer, fSet, fParser)
826         if err != nil {
827                 log.Fatal(err)
828         }
829         // 写回数据
830         return os.WriteFile(filepath, buffer.Bytes(), 0o600)
831 }
832
833 // CreatePlug 自动创建插件模板
834 func (autoCodeService *AutoCodeService) CreatePlug(plug system.AutoPlugReq) error {
835         // 检查列表参数是否有效
836         plug.CheckList()
837         tplFileList, _ := autoCodeService.GetAllTplFile(plugPath, nil)
838         for _, tpl := range tplFileList {
839                 temp, err := template.ParseFiles(tpl)
840                 if err != nil {
841                         zap.L().Error("parse err", zap.String("tpl", tpl), zap.Error(err))
842                         return err
843                 }
844                 pathArr := strings.SplitAfter(tpl, "/")
845                 if strings.Index(pathArr[2], "tpl") < 0 {
846                         dirPath := filepath.Join(global.GVA_CONFIG.AutoCode.Root, global.GVA_CONFI
847                         os.MkdirAll(dirPath, 0755)
848                 }
849                 file := filepath.Join(global.GVA_CONFIG.AutoCode.Root, global.GVA_CONFIG.AutoCode.
850                 f, err := os.OpenFile(file, os.O_WRONLY|os.O_CREATE, 0666)
851                 if err != nil {
852                         zap.L().Error("open file", zap.String("tpl", tpl), zap.Error(err), zap.Any
853                         return err
854                 }
855                 defer f.Close()
856
857                 err = temp.Execute(f, plug)
858                 if err != nil {
859                         zap.L().Error("exec err", zap.String("tpl", tpl), zap.Error(err), zap.Any(
860                         return err
861                 }
862         }
```

```
863         return nil
864 }
865
866 func (autoCodeService *AutoCodeService) InstallPlugin(file *multipart.FileHeader) (web, server int
867         const GVAPLUGPINATH = "./gva-plug-temp/"
868         defer os.RemoveAll(GVAPLUGPINATH)
869         _, err = os.Stat(GVAPLUGPINATH)
870         if os.IsNotExist(err) {
871                 os.Mkdir(GVAPLUGPINATH, os.ModePerm)
872         }
873
874         src, err := file.Open()
875         if err != nil {
876                 return -1, -1, err
877         }
878         defer src.Close()
879
880         out, err := os.Create(GVAPLUGPINATH + file.Filename)
881         if err != nil {
882                 return -1, -1, err
883         }
884         defer out.Close()
885
886         _, err = io.Copy(out, src)
887
888         paths, err := utils.Unzip(GVAPLUGPINATH+file.Filename, GVAPLUGPINATH)
889         paths = filterFile(paths)
890         var webIndex = -1
891         var serverIndex = -1
892         for i := range paths {
893                 paths[i] = filepath.ToSlash(paths[i])
894                 pathArr := strings.Split(paths[i], "/")
895                 ln := len(pathArr)
896                 if ln < 2 {
897                         continue
898                 }
899                 if pathArr[ln-2] == "server" && pathArr[ln-1] == "plugin" {
900                         serverIndex = i
901                 }
902                 if pathArr[ln-2] == "web" && pathArr[ln-1] == "plugin" {
903                         webIndex = i
904                 }
905         }
906         if webIndex == -1 && serverIndex == -1 {
907                 zap.L().Error("非标准插件，请按照文档自动迁移使用")
908                 return webIndex, serverIndex, errors.New("非标准插件，请按照文档自动迁移使用")
909         }
910
911         if webIndex != -1 {
```

```
912                 err = installation(paths[webIndex], global.GVA_CONFIG.AutoCode.Server, global.GVA_
913                 if err != nil {
914                         return webIndex, serverIndex, err
915                 }
916         }
917
918         if serverIndex != -1 {
919                 err = installation(paths[serverIndex], global.GVA_CONFIG.AutoCode.Server, global.G
920         }
921         return webIndex, serverIndex, err
922 }
923
924 func installation(path string, formPath string, toPath string) error {
925         arr := strings.Split(filepath.ToSlash(path), "/")
926         ln := len(arr)
927         if ln < 3 {
928                 return errors.New("arr")
929         }
930         name := arr[ln-3]
931
932         var form = filepath.ToSlash(global.GVA_CONFIG.AutoCode.Root + formPath + "/" + path)
933         var to = filepath.ToSlash(global.GVA_CONFIG.AutoCode.Root + toPath + "/plugin/")
934         _, err := os.Stat(to + name)
935         if err == nil {
936                 zap.L().Error("autoPath 已存在同名插件，请自行手动安装", zap.String("to", to))
937                 return errors.New(toPath + "已存在同名插件，请自行手动安装")
938         }
939         return cp.Copy(form, to, cp.Options{Skip: skipMacSpecialDocument})
940 }
941
942 func filterFile(paths []string) []string {
943         np := make([]string, 0, len(paths))
944         for _, path := range paths {
945                 if ok, _ := skipMacSpecialDocument(path); ok {
946                         continue
947                 }
948                 np = append(np, path)
949         }
950         return np
951 }
952
953 func skipMacSpecialDocument(src string) (bool, error) {
954         if strings.Contains(src, ".DS_Store") || strings.Contains(src, "__MACOSX") {
955                 return true, nil
956         }
957         return false, nil
958 }
```