⑂ main ▾   ···

gin-vue-admin / server / service / system / sys_auto_code.go / <> Jump to ▾

piexlmax 增加自动化排序功能   ⟳ History

⚇ 15 contributors   👤👤👤👤👤👤👤👤👤👤👤👤 +3

958 lines (885 sloc) | 27.6 KB   ···

```
1    package system
2
3    import (
4            "bytes"
5            "encoding/json"
6            "errors"
7            "fmt"
8            "go/ast"
9            "go/format"
10           "go/parser"
11           "go/token"
12           "io"
13           "log"
14           "mime/multipart"
15           "os"
16           "path/filepath"
17           "strconv"
18           "strings"
19           "text/template"
20
21           cp "github.com/otiai10/copy"
22           "go.uber.org/zap"
23           "golang.org/x/text/cases"
24           "golang.org/x/text/language"
25
26           "github.com/flipped-aurora/gin-vue-admin/server/resource/autocode_template/subcontract"
27
28           "github.com/flipped-aurora/gin-vue-admin/server/global"
29           "github.com/flipped-aurora/gin-vue-admin/server/model/system"
```

```go
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                      },
```

```go
            {
                    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 {
```

```go
128             path         string
129             importCodeF  string
130             structNameF  string
131             packageNameF string
132             groupName    string
133     }
134
135     type tplData struct {
136             template        *template.Template
137             autoPackage     string
138             locationPath    string
139             autoCodePath    string
140             autoMoveFilePath string
141     }
142
143     type AutoCodeService struct{}
144
145     var AutoCodeServiceApp = new(AutoCodeService)
146
147     //@author: [songzhibin97](https://github.com/songzhibin97)
148     //@function: PreviewTemp
149     //@description: 预览创建代码
150     //@param: model.AutoCodeStruct
151     //@return: map[string]string, error
152
153     func (autoCodeService *AutoCodeService) PreviewTemp(autoCode system.AutoCodeStruct) (map[string]st
154             makeDictTypes(&autoCode)
155             for i := range autoCode.Fields {
156                     if autoCode.Fields[i].FieldType == "time.Time" {
157                             autoCode.HasTimer = true
158                             break
159                     }
160                     if autoCode.Fields[i].Require {
161                             autoCode.NeedValid = true
162                             break
163                     }
164                     if autoCode.Fields[i].Sort {
165                             autoCode.NeedSort = true
166                             break
167                     }
168             }
169             dataList, _, needMkdir, err := autoCodeService.getNeedList(&autoCode)
170             if err != nil {
171                     return nil, err
172             }
173
174             // 写入文件前，先创建文件夹
175             if err = utils.CreateDir(needMkdir...); err != nil {
176                     return nil, err
```

```go
		}

		// 创建map
		ret := make(map[string]string)

		// 生成map
		for _, value := range dataList {
			ext := ""
			if ext = filepath.Ext(value.autoCodePath); ext == ".txt" {
				continue
			}
			f, err := os.OpenFile(value.autoCodePath, os.O_CREATE|os.O_WRONLY, 0o755)
			if err != nil {
				return nil, err
			}
			if err = value.template.Execute(f, autoCode); err != nil {
				return nil, err
			}
			_ = f.Close()
			f, err = os.OpenFile(value.autoCodePath, os.O_CREATE|os.O_RDONLY, 0o755)
			if err != nil {
				return nil, err
			}
			builder := strings.Builder{}
			builder.WriteString("```")

			if ext != "" && strings.Contains(ext, ".") {
				builder.WriteString(strings.Replace(ext, ".", "", -1))
			}
			builder.WriteString("\n\n")
			data, err := io.ReadAll(f)
			if err != nil {
				return nil, err
			}
			builder.Write(data)
			builder.WriteString("\n\n```")

			pathArr := strings.Split(value.autoCodePath, string(os.PathSeparator))
			ret[pathArr[1]+"-"+pathArr[3]] = builder.String()
			_ = f.Close()

		}
		defer func() { // 移除中间文件
			if err := os.RemoveAll(autoPath); err != nil {
				return
			}
		}()
		return ret, nil
}
```

```go
func makeDictTypes(autoCode *system.AutoCodeStruct) {
	DictTypeM := make(map[string]string)
	for _, v := range autoCode.Fields {
		if v.DictType != "" {
			DictTypeM[v.DictType] = ""
		}
	}

	for k := range DictTypeM {
		autoCode.DictTypes = append(autoCode.DictTypes, k)
	}
}

//@author: [piexlmax](https://github.com/piexlmax)
//@function: CreateTemp
//@description: 创建代码
//@param: model.AutoCodeStruct
//@return: err error

func (autoCodeService *AutoCodeService) CreateTemp(autoCode system.AutoCodeStruct, ids ...uint) (e
	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
		}
	}
	// 增加判断: 重复创建struct
	if autoCode.AutoMoveFile && AutoCodeHistoryServiceApp.Repeat(autoCode.BusinessDB, autoCode
		return RepeatErr
	}
	dataList, fileList, needMkdir, err := autoCodeService.getNeedList(&autoCode)
	if err != nil {
		return err
	}
	meta, _ := json.Marshal(autoCode)
	// 写入文件前, 先创建文件夹
	if err = utils.CreateDir(needMkdir...); err != nil {
		return err
	}
```

```go
        // 生成文件
        for _, value := range dataList {
                f, err := os.OpenFile(value.autoCodePath, os.O_CREATE|os.O_WRONLY, 0o755)
                if err != nil {
                        return err
                }
                if err = value.template.Execute(f, autoCode); err != nil {
                        return err
                }
                _ = f.Close()
        }

        defer func() { // 移除中间文件
                if err := os.RemoveAll(autoPath); err != nil {
                        return
                }
        }()
        bf := strings.Builder{}
        idBf := strings.Builder{}
        injectionCodeMeta := strings.Builder{}
        for _, id := range ids {
                idBf.WriteString(strconv.Itoa(int(id)))
                idBf.WriteString(";")
        }
        if autoCode.AutoMoveFile { // 判断是否需要自动转移
                Init(autoCode.Package)
                for index := range dataList {
                        autoCodeService.addAutoMoveFile(&dataList[index])
                }
                // 判断目标文件是否都可以移动
                for _, value := range dataList {
                        if utils.FileExist(value.autoMoveFilePath) {
                                return errors.New(fmt.Sprintf("目标文件已存在:%s\n", value.autoMoveI
                        }
                }
                for _, value := range dataList { // 移动文件
                        if err := utils.FileMove(value.autoCodePath, value.autoMoveFilePath); err
                                return err
                        }
                }
                err = injectionCode(autoCode.StructName, &injectionCodeMeta)
                if err != nil {
                        return
                }
                // 保存生成信息
                for _, data := range dataList {
                        if len(data.autoMoveFilePath) != 0 {
                                bf.WriteString(data.autoMoveFilePath)
```

```go
                        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
```

```go
373        }
374
375    //@author: [piexlmax](https://github.com/piexlmax)
376    //@function: GetAllTplFile
377    //@description: 获取 pathName 文件夹下所有 tpl 文件
378    //@param: pathName string, fileList []string
379    //@return: []string, error
380
381    func (autoCodeService *AutoCodeService) GetAllTplFile(pathName string, fileList []string) ([]strin
382            files, err := os.ReadDir(pathName)
383            for _, fi := range files {
384                    if fi.IsDir() {
385                            fileList, err = autoCodeService.GetAllTplFile(pathName+"/"+fi.Name(), file
386                            if err != nil {
387                                    return nil, err
388                            }
389                    } else {
390                            if strings.HasSuffix(fi.Name(), ".tpl") {
391                                    fileList = append(fileList, pathName+"/"+fi.Name())
392                            }
393                    }
394            }
395            return fileList, err
396    }
397
398    //@author: [piexlmax](https://github.com/piexlmax)
399    //@function: GetDB
400    //@description: 获取指定数据库和指定数据表的所有字段名,类型值等
401    //@param: tableName string, dbName string
402    //@return: err error, Columns []request.ColumnReq
403
404    func (autoCodeService *AutoCodeService) DropTable(BusinessDb, tableName string) error {
405            if BusinessDb != "" {
406                    return global.GVA_DB.Exec("DROP TABLE " + tableName).Error
407            } else {
408                    return global.MustGetGlobalDBByDBName(BusinessDb).Exec("DROP TABLE " + tableName).
409            }
410    }
411
412    //@author: [SliverHorn](https://github.com/SliverHorn)
413    //@author: [songzhibin97](https://github.com/songzhibin97)
414    //@function: addAutoMoveFile
415    //@description: 生成对应的迁移文件路径
416    //@param: *tplData
417    //@return: null
418
419    func (autoCodeService *AutoCodeService) addAutoMoveFile(data *tplData) {
420            base := filepath.Base(data.autoCodePath)
421            fileSlice := strings.Split(data.autoCodePath, string(os.PathSeparator))
```

```go
        n := len(fileSlice)
        if n <= 2 {
                return
        }
        if strings.Contains(fileSlice[1], "server") {
                if strings.Contains(fileSlice[n-2], "router") {
                        data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root, glo
                                fmt.Sprintf(global.GVA_CONFIG.AutoCode.SRouter, data.autoPackage),
                } else if strings.Contains(fileSlice[n-2], "api") {
                        data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
                } else if strings.Contains(fileSlice[n-2], "service") {
                        data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
                } else if strings.Contains(fileSlice[n-2], "model") {
                        data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
                } else if strings.Contains(fileSlice[n-2], "request") {
                        data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Server, fmt.Sprintf(global.GVA_CONFIG.A
                }
        } else if strings.Contains(fileSlice[1], "web") {
                if strings.Contains(fileSlice[n-1], "js") {
                        data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Web, global.GVA_CONFIG.AutoCode.WApi, b
                } else if strings.Contains(fileSlice[n-2], "form") {
                        data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Web, global.GVA_CONFIG.AutoCode.WForm,
                } else if strings.Contains(fileSlice[n-2], "table") {
                        data.autoMoveFilePath = filepath.Join(global.GVA_CONFIG.AutoCode.Root,
                                global.GVA_CONFIG.AutoCode.Web, global.GVA_CONFIG.AutoCode.WTable,
                }
        }
}

//@author: [piexlmax](https://github.com/piexlmax)
//@author: [SliverHorn](https://github.com/SliverHorn)
//@function: CreateApi
//@description: 自动创建api数据，
//@param: a *model.AutoCodeStruct
//@return: err error

func (autoCodeService *AutoCodeService) AutoCreateApi(a *system.AutoCodeStruct) (ids []uint, err e
        apiList := []system.SysApi{
                {
                        Path:        "/" + a.Abbreviation + "/" + "create" + a.StructName,
                        Description: "新增" + a.Description,
                        ApiGroup:    a.Abbreviation,
                        Method:      "POST",
```

```go
                    },
                    {
                            Path:        "/" + a.Abbreviation + "/" + "delete" + a.StructName,
                            Description: "删除" + a.Description,
                            ApiGroup:    a.Abbreviation,
                            Method:      "DELETE",
                    },
                    {
                            Path:        "/" + a.Abbreviation + "/" + "delete" + a.StructName + "ByIds
                            Description: "批量删除" + a.Description,
                            ApiGroup:    a.Abbreviation,
                            Method:      "DELETE",
                    },
                    {
                            Path:        "/" + a.Abbreviation + "/" + "update" + a.StructName,
                            Description: "更新" + a.Description,
                            ApiGroup:    a.Abbreviation,
                            Method:      "PUT",
                    },
                    {
                            Path:        "/" + a.Abbreviation + "/" + "find" + a.StructName,
                            Description: "根据ID获取" + a.Description,
                            ApiGroup:    a.Abbreviation,
                            Method:      "GET",
                    },
                    {
                            Path:        "/" + a.Abbreviation + "/" + "get" + a.StructName + "List",
                            Description: "获取" + a.Description + "列表",
                            ApiGroup:    a.Abbreviation,
                            Method:      "GET",
                    },
            }
            err = global.GVA_DB.Transaction(func(tx *gorm.DB) error {
                    for _, v := range apiList {
                            var api system.SysApi
                            if errors.Is(tx.Where("path = ? AND method = ?", v.Path, v.Method).First(&
                                    if err = tx.Create(&v).Error; err != nil { // 遇到错误时回滚事务
                                            return err
                                    } else {
                                            ids = append(ids, v.ID)
                                    }
                            }
                    }
                    return nil
            })
            return ids, err
    }

    func (autoCodeService *AutoCodeService) getNeedList(autoCode *system.AutoCodeStruct) (dataList []t
```

```go
        // 去除所有空格
    utils.TrimSpace(autoCode)
    for _, field := range autoCode.Fields {
            utils.TrimSpace(field)
    }
    // 获取 basePath 文件夹下所有tpl文件
    tplFileList, err := autoCodeService.GetAllTplFile(autocodePath, nil)
    if err != nil {
            return nil, nil, nil, err
    }
    dataList = make([]tplData, 0, len(tplFileList))
    fileList = make([]string, 0, len(tplFileList))
    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
    // 根据文件路径生成 tplData 结构体, 待填充数据
    for _, value := range tplFileList {
            dataList = append(dataList, tplData{locationPath: value, autoPackage: autoCode.Pac
    }
    // 生成 *Template, 填充 template 字段
    for index, value := range dataList {
            dataList[index].template, err = template.ParseFiles(value.locationPath)
            if err != nil {
                    return nil, nil, nil, err
            }
    }
    // 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
    // resource/template/web/api.js.tpl -> autoCode/web/autoCode.PackageName/api/autoCode.Pack
    // resource/template/readme.txt.tpl -> autoCode/readme.txt
    for index, value := range dataList {
            trimBase := strings.TrimPrefix(value.locationPath, autocodePath+"/")
            if trimBase == "readme.txt.tpl" {
                    dataList[index].autoCodePath = autoPath + "readme.txt"
                    continue
            }

            if lastSeparator := strings.LastIndex(trimBase, "/"); lastSeparator != -1 {
                    origFileName := strings.TrimSuffix(trimBase[lastSeparator+1:], ".tpl")
                    firstDot := strings.Index(origFileName, ".")
                    if firstDot != -1 {
                            var fileName string
                            if origFileName[firstDot:] != ".go" {
                                    fileName = autoCode.PackageName + origFileName[firstDot:]
                            } else {
                                    fileName = autoCode.HumpPackageName + origFileName[firstDo
                            }

                            dataList[index].autoCodePath = filepath.Join(autoPath, trimBase[:l
                                    origFileName[:firstDot], fileName)
                    }
            }
```

```go
                    if lastSeparator := strings.LastIndex(dataList[index].autoCodePath, string(os.Path
                        needMkdir = append(needMkdir, dataList[index].autoCodePath[:lastSeparator]
                    }
                }
            for _, value := range dataList {
                    fileList = append(fileList, value.autoCodePath)
            }
            return dataList, fileList, needMkdir, err
    }

    // injectionCode 封装代码注入
    func injectionCode(structName string, bf *strings.Builder) error {
            for _, meta := range injectionPaths {
                    code := fmt.Sprintf(meta.structNameF, structName)
                    if err := utils.AutoInjectionCode(meta.path, meta.funcName, code); err != nil {
                            return err
                    }
                    bf.WriteString(fmt.Sprintf("%s@%s@%s;", meta.path, meta.funcName, code))
            }
            return nil
    }

    func (autoCodeService *AutoCodeService) CreateAutoCode(s *system.SysAutoCode) error {
            if s.PackageName == "autocode" || s.PackageName == "system" || s.PackageName == "example"
                    return errors.New("不能使用已保留的package name")
            }
            if !errors.Is(global.GVA_DB.Where("package_name = ?", s.PackageName).First(&system.SysAuto
                    return errors.New("存在相同PackageName")
            }
            if e := autoCodeService.CreatePackageTemp(s.PackageName); e != nil {
                    return e
            }
            return global.GVA_DB.Create(&s).Error
    }

    func (autoCodeService *AutoCodeService) GetPackage() (pkgList []system.SysAutoCode, err error) {
            err = global.GVA_DB.Find(&pkgList).Error
            return pkgList, err
    }

    func (autoCodeService *AutoCodeService) DelPackage(a system.SysAutoCode) error {
            return global.GVA_DB.Delete(&a).Error
    }

    func (autoCodeService *AutoCodeService) CreatePackageTemp(packageName string) error {
            Init(packageName)
            pendingTemp := []autoPackage{{
                    path: packageService,
```

```go
                    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
}
```

```go
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{
```

```go
                                                X: &ast.Ident{
                                                    Name: vi.PackageName,
                                                },
                                                Sel: &ast.Ident{
                                                    Name: vi.GroupName,
                                                },
                                            },
                                        }
                                        t.Fields.List = append(t.Fields.List, f)
                                    }
                                }
                            }
                        }
                    }
        return vi
    }

    func (vi *Visitor) addImport(genDecl *ast.GenDecl) ast.Visitor {
        // 是否已经import
        hasImported := false
        for _, v := range genDecl.Specs {
            importSpec := v.(*ast.ImportSpec)
            // 如果已经包含
            if importSpec.Path.Value == strconv.Quote(vi.ImportCode) {
                hasImported = true
            }
        }
        if !hasImported {
            genDecl.Specs = append(genDecl.Specs, &ast.ImportSpec{
                Path: &ast.BasicLit{
                    Kind:  token.STRING,
                    Value: strconv.Quote(vi.ImportCode),
                },
            })
        }
        return vi
    }

    func (vi *Visitor) addFuncBodyVar(funDecl *ast.FuncDecl) ast.Visitor {
        hasVar := false
        for _, v := range funDecl.Body.List {
            switch varSpec := v.(type) {
            case *ast.AssignStmt:
                for i := range varSpec.Lhs {
                    switch nn := varSpec.Lhs[i].(type) {
                    case *ast.Ident:
                        if nn.Name == vi.PackageName+"Router" {
                            hasVar = true
                        }
                    }
```

```go
                }
            }
        }
        if !hasVar {
            assignStmt := &ast.AssignStmt{
                Lhs: []ast.Expr{
                    &ast.Ident{
                        Name: vi.PackageName + "Router",
                        Obj: &ast.Object{
                            Kind: ast.Var,
                            Name: vi.PackageName + "Router",
                        },
                    },
                },
                Tok: token.DEFINE,
                Rhs: []ast.Expr{
                    &ast.SelectorExpr{
                        X: &ast.SelectorExpr{
                            X: &ast.Ident{
                                Name: "router",
                            },
                            Sel: &ast.Ident{
                                Name: "RouterGroupApp",
                            },
                        },
                        Sel: &ast.Ident{
                            Name: caser.String(vi.PackageName),
                        },
                    },
                },
            }
            funDecl.Body.List = append(funDecl.Body.List, funDecl.Body.List[1])
            index := 1
            copy(funDecl.Body.List[index+1:], funDecl.Body.List[index:])
            funDecl.Body.List[index] = assignStmt
        }
        return vi
}

func ImportReference(filepath, importCode, structName, packageName, groupName string) error {
        fSet := token.NewFileSet()
        fParser, err := parser.ParseFile(fSet, filepath, nil, parser.ParseComments)
        if err != nil {
                return err
        }
        importCode = strings.TrimSpace(importCode)
        v := &Visitor{
                ImportCode:  importCode,
                StructName:  structName,
```

```go
                    PackageName: packageName,
                    GroupName:   groupName,
            }
            if importCode == "" {
                    ast.Print(fSet, fParser)
            }

            ast.Walk(v, fParser)

            var output []byte
            buffer := bytes.NewBuffer(output)
            err = format.Node(buffer, fSet, fParser)
            if err != nil {
                    log.Fatal(err)
            }
            // 写回数据
            return os.WriteFile(filepath, buffer.Bytes(), 0o600)
}

// CreatePlug 自动创建插件模板
func (autoCodeService *AutoCodeService) CreatePlug(plug system.AutoPlugReq) error {
            // 检查列表参数是否有效
            plug.CheckList()
            tplFileList, _ := autoCodeService.GetAllTplFile(plugPath, nil)
            for _, tpl := range tplFileList {
                    temp, err := template.ParseFiles(tpl)
                    if err != nil {
                            zap.L().Error("parse err", zap.String("tpl", tpl), zap.Error(err))
                            return err
                    }
                    pathArr := strings.SplitAfter(tpl, "/")
                    if strings.Index(pathArr[2], "tpl") < 0 {
                            dirPath := filepath.Join(global.GVA_CONFIG.AutoCode.Root, global.GVA_CONFI
                            os.MkdirAll(dirPath, 0755)
                    }
                    file := filepath.Join(global.GVA_CONFIG.AutoCode.Root, global.GVA_CONFIG.AutoCode.
                    f, err := os.OpenFile(file, os.O_WRONLY|os.O_CREATE, 0666)
                    if err != nil {
                            zap.L().Error("open file", zap.String("tpl", tpl), zap.Error(err), zap.Any
                            return err
                    }
                    defer f.Close()

                    err = temp.Execute(f, plug)
                    if err != nil {
                            zap.L().Error("exec err", zap.String("tpl", tpl), zap.Error(err), zap.Any(
                            return err
                    }
            }
```

```go
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 {
```

```go
            err = installation(paths[webIndex], global.GVA_CONFIG.AutoCode.Server, global.GVA_
            if err != nil {
                    return webIndex, serverIndex, err
            }
        }

        if serverIndex != -1 {
            err = installation(paths[serverIndex], global.GVA_CONFIG.AutoCode.Server, global.G
        }
        return webIndex, serverIndex, err
}

func installation(path string, formPath string, toPath string) error {
        arr := strings.Split(filepath.ToSlash(path), "/")
        ln := len(arr)
        if ln < 3 {
                return errors.New("arr")
        }
        name := arr[ln-3]

        var form = filepath.ToSlash(global.GVA_CONFIG.AutoCode.Root + formPath + "/" + path)
        var to = filepath.ToSlash(global.GVA_CONFIG.AutoCode.Root + toPath + "/plugin/")
        _, err := os.Stat(to + name)
        if err == nil {
                zap.L().Error("autoPath 已存在同名插件，请自行手动安装", zap.String("to", to))
                return errors.New(toPath + "已存在同名插件，请自行手动安装")
        }
        return cp.Copy(form, to, cp.Options{Skip: skipMacSpecialDocument})
}

func filterFile(paths []string) []string {
        np := make([]string, 0, len(paths))
        for _, path := range paths {
                if ok, _ := skipMacSpecialDocument(path); ok {
                        continue
                }
                np = append(np, path)
        }
        return np
}

func skipMacSpecialDocument(src string) (bool, error) {
        if strings.Contains(src, ".DS_Store") || strings.Contains(src, "__MACOSX") {
                return true, nil
        }
        return false, nil
}
```