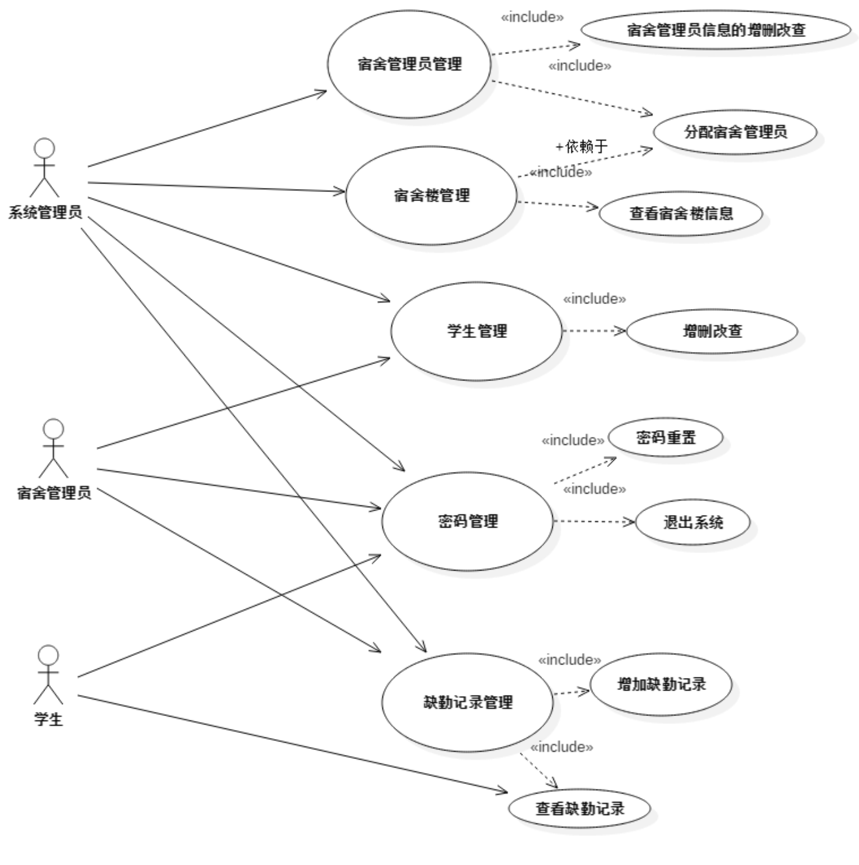
完成者：楚宇 14130140380 马翔宇 14130140382

一、功能需求建模

采用 UML 用例图全局功能建模
- 基于参与者的视图
  至少一个参与者是人且通常是匿名的
- 通过 UML activity 进行精化
- 两类需求
  功能
  导航(Navigational, Web 应用典型需求)

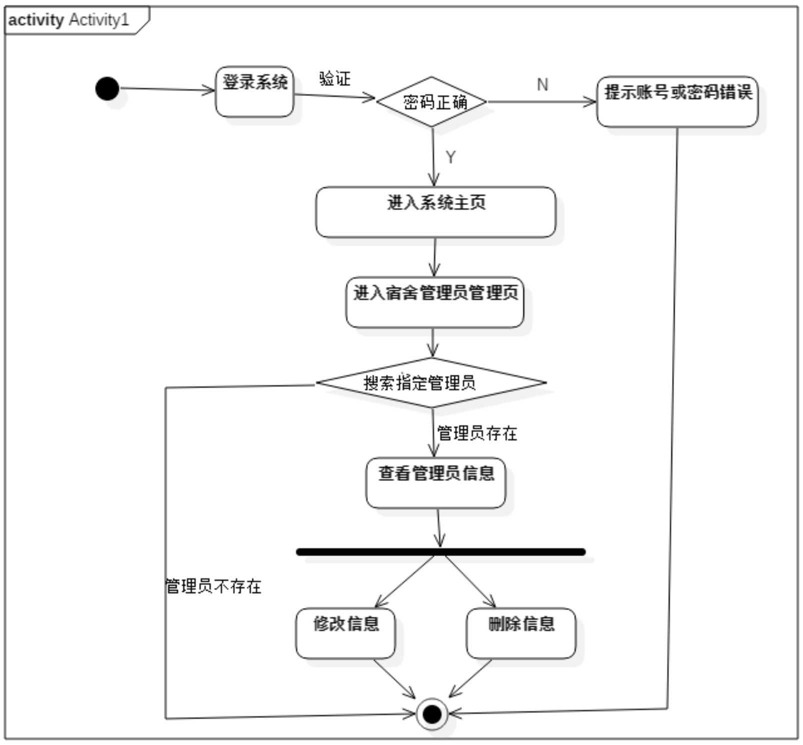
该系统的用例一共有三个：系统管理员、宿舍管理员、学生，每个用例可以实现不同的相对活动。

宿舍管理系统用例图如下：

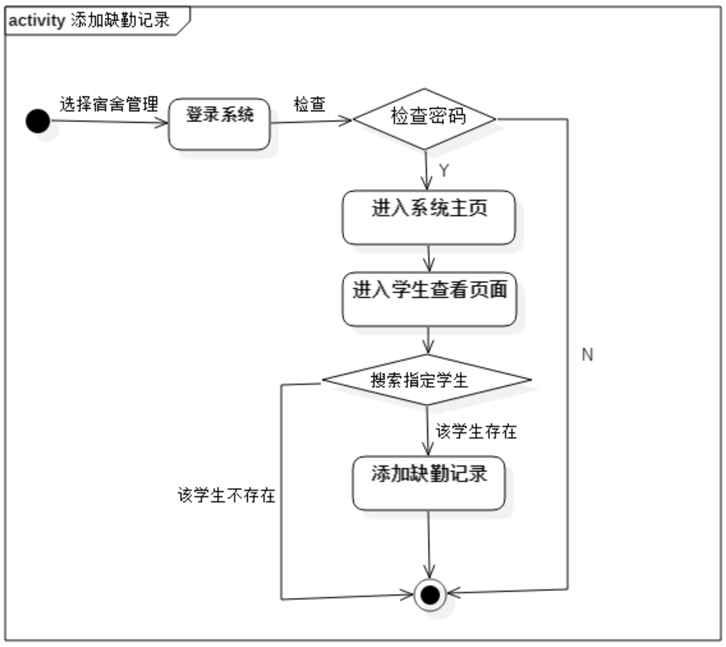


活动图用于表示针对相应的需求，用例在使用该系统时经过的详细步骤。

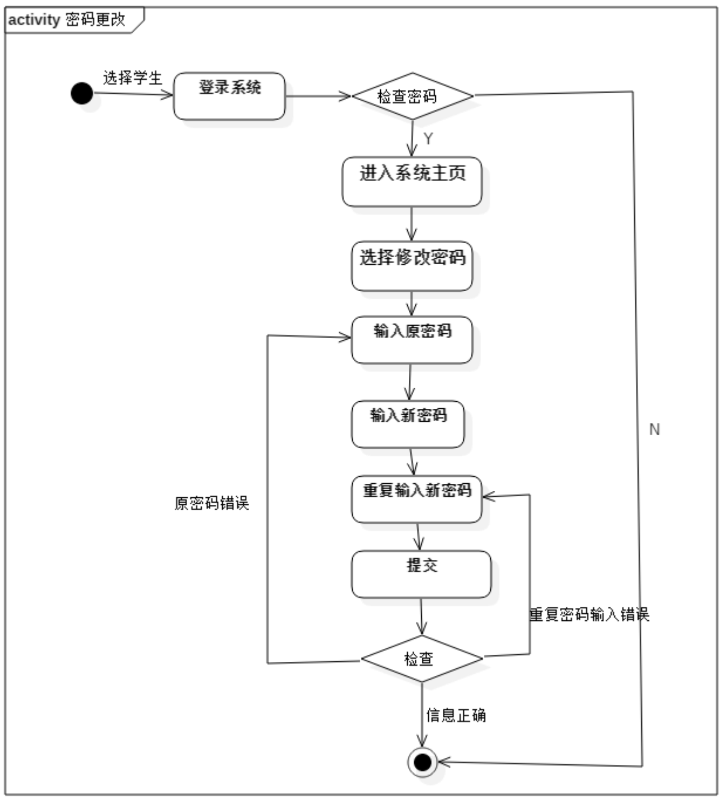
系统管理员登入系统完成修改管理员信息的活动图：



宿舍管理员登入系统添加学生缺勤记录的活动图：



学生密码更改的活动图：



二、内容建模

目标是将从需求工程中决定的 Web 应用信息和功能需求转换为模型

图形化信息的结构(i.e., information objects) 和行为方面，不关注导航

- 内容的结构方面

  问题域模型

  UML 类图
- 内容的行为方面

  根据 Web 应用的类型和复杂程度

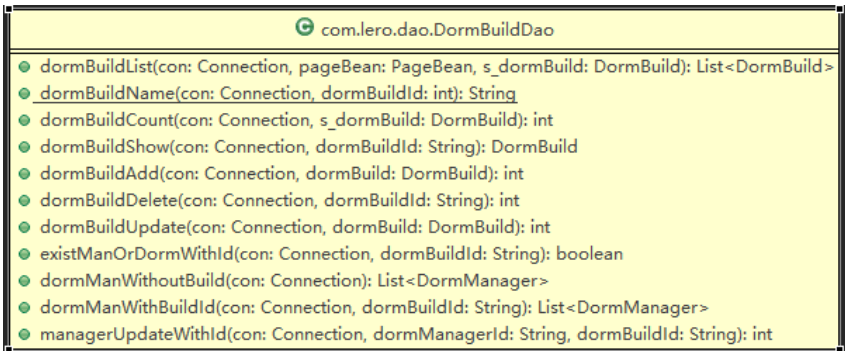
  UML 状态图(state charts)/交互图(interaction diagrams)
- 只是内容层, i.e.,

  静态方面使用类图来体现。

  该系统中一共存在四个包：

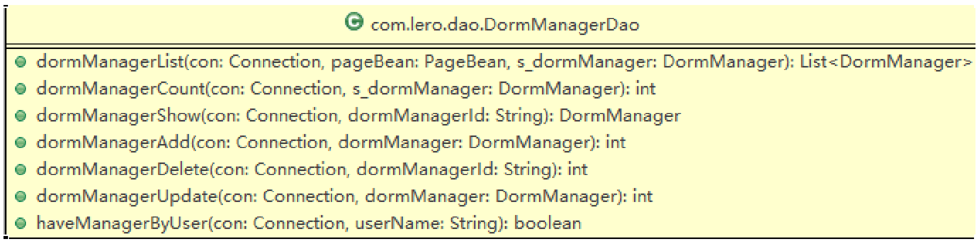
  com.lero.dao 包：提供不同角色对数据库操作的封装

  DormBuild 类：系统管理员对数据库的操作封装。



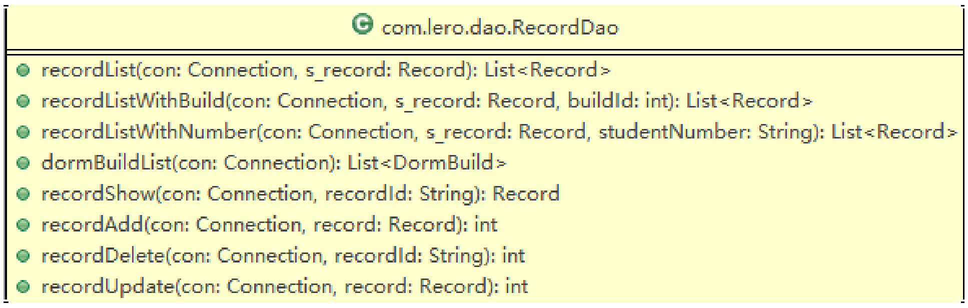
```
com.lero.dao.DormBuildDao
dormBuildList(con: Connection, pageBean: PageBean, s_dormBuild: DormBuild): List<DormBuild>
dormBuildName(con: Connection, dormBuildId: int): String
dormBuildCount(con: Connection, s_dormBuild: DormBuild): int
dormBuildShow(con: Connection, dormBuildId: String): DormBuild
dormBuildAdd(con: Connection, dormBuild: DormBuild): int
dormBuildDelete(con: Connection, dormBuildId: String): int
dormBuildUpdate(con: Connection, dormBuild: DormBuild): int
existManOrDormWithId(con: Connection, dormBuildId: String): boolean
dormManWithoutBuild(con: Connection): List<DormManager>
dormManWithBuildId(con: Connection, dormBuildId: String): List<DormManager>
managerUpdateWithId(con: Connection, dormManagerId: String, dormBuildId: String): int
```

DormManagerDao 类：宿舍管理员对数据库的操作封装。



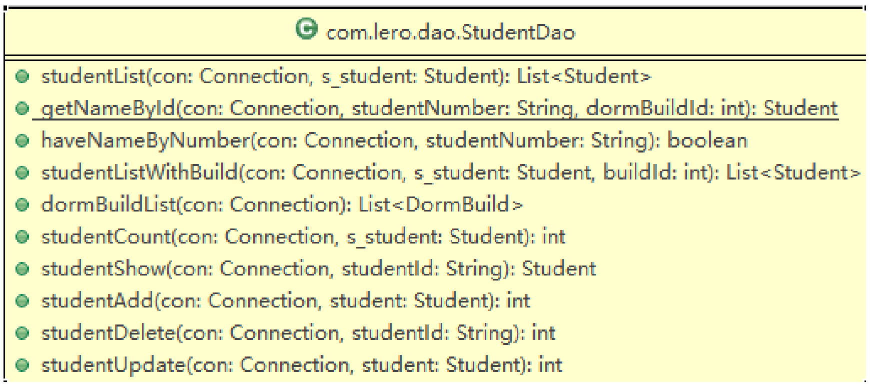
```
com.lero.dao.DormManagerDao
dormManagerList(con: Connection, pageBean: PageBean, s_dormManager: DormManager): List<DormManager>
dormManagerCount(con: Connection, s_dormManager: DormManager): int
dormManagerShow(con: Connection, dormManagerId: String): DormManager
dormManagerAdd(con: Connection, dormManager: DormManager): int
dormManagerDelete(con: Connection, dormManagerId: String): int
dormManagerUpdate(con: Connection, dormManager: DormManager): int
haveManagerByUser(con: Connection, userName: String): boolean
```

RecordDao 类：缺勤记录对数据库的操作封装。



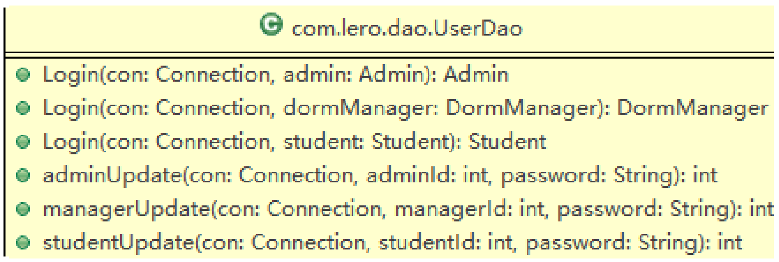
```
com.lero.dao.RecordDao
recordList(con: Connection, s_record: Record): List<Record>
recordListWithBuild(con: Connection, s_record: Record, buildId: int): List<Record>
recordListWithNumber(con: Connection, s_record: Record, studentNumber: String): List<Record>
dormBuildList(con: Connection): List<DormBuild>
recordShow(con: Connection, recordId: String): Record
recordAdd(con: Connection, record: Record): int
recordDelete(con: Connection, recordId: String): int
recordUpdate(con: Connection, record: Record): int
```

StudentDao 类：学生对数据库的操作封装。



```
com.lero.dao.StudentDao
studentList(con: Connection, s_student: Student): List<Student>
getNameById(con: Connection, studentNumber: String, dormBuildId: int): Student
haveNameByNumber(con: Connection, studentNumber: String): boolean
studentListWithBuild(con: Connection, s_student: Student, buildId: int): List<Student>
dormBuildList(con: Connection): List<DormBuild>
studentCount(con: Connection, s_student: Student): int
studentShow(con: Connection, studentId: String): Student
studentAdd(con: Connection, student: Student): int
studentDelete(con: Connection, studentId: String): int
studentUpdate(con: Connection, student: Student): int
```

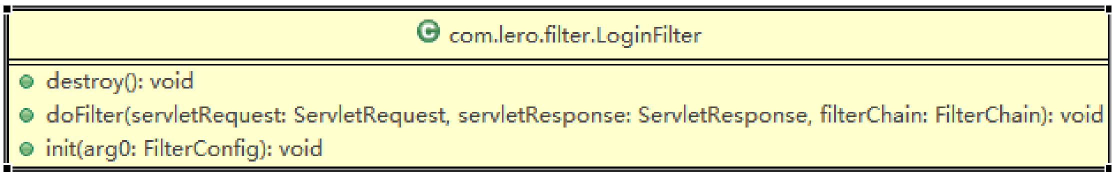
UserDao 类：



```
com.lero.dao.UserDao
Login(con: Connection, admin: Admin): Admin
Login(con: Connection, dormManager: DormManager): DormManager
Login(con: Connection, student: Student): Student
adminUpdate(con: Connection, adminId: int, password: String): int
managerUpdate(con: Connection, managerId: int, password: String): int
studentUpdate(con: Connection, studentId: int, password: String): int
```

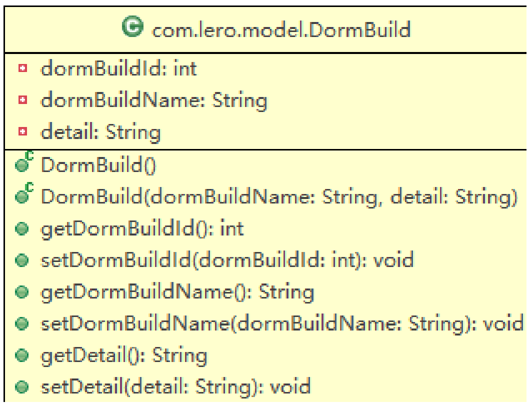
com.lero.filter 包：过滤器

LoginFilter 类：用于对登录时用户的过滤



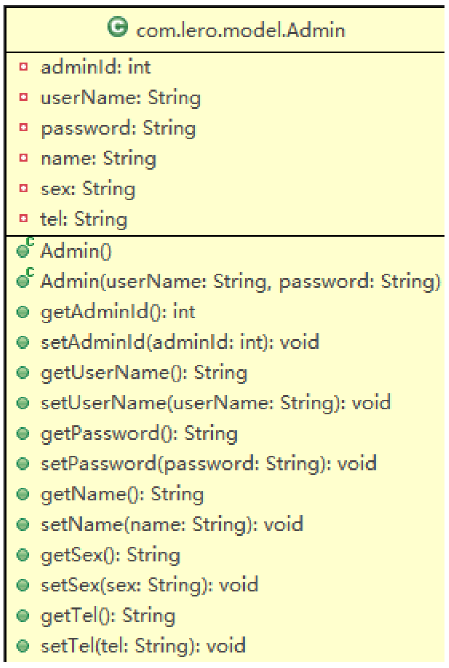
```
com.lero.filter.LoginFilter
destroy(): void
doFilter(servletRequest: ServletRequest, servletResponse: ServletResponse, filterChain: FilterChain): void
init(arg0: FilterConfig): void
```

com.lero.model 包：提供角色模型



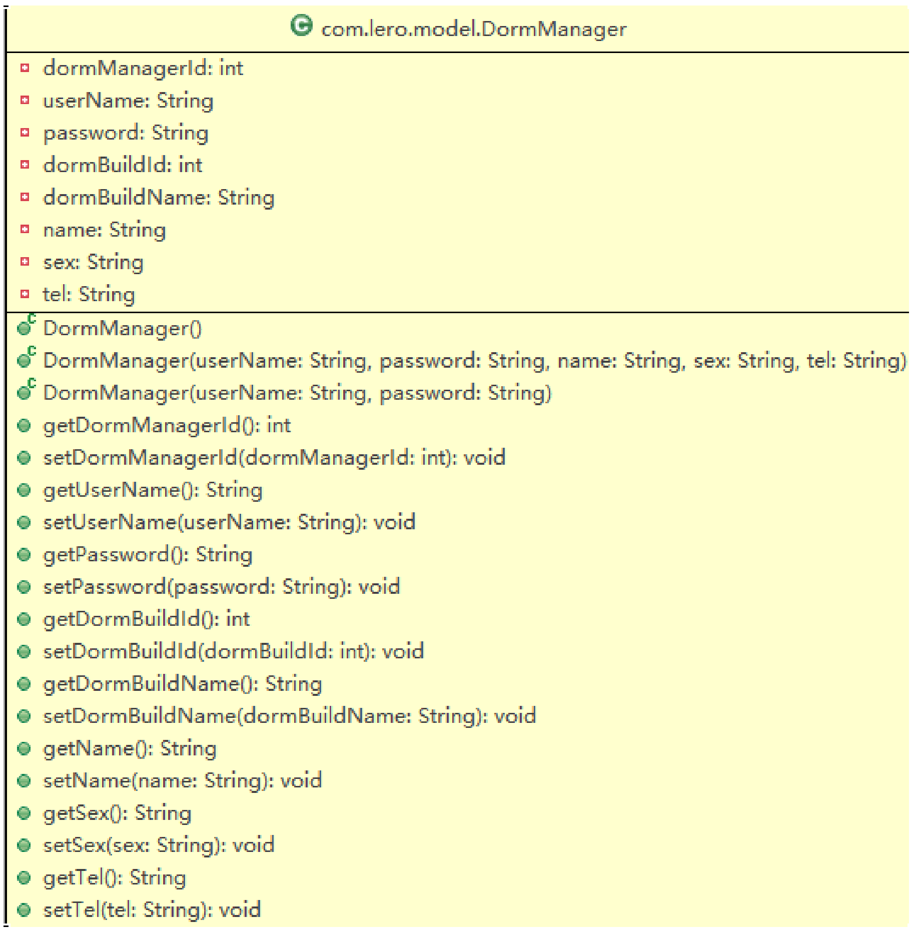
```
com.lero.model.DormBuild
dormBuildId: int
dormBuildName: String
detail: String
DormBuild()
DormBuild(dormBuildName: String, detail: String)
getDormBuildId(): int
setDormBuildId(dormBuildId: int): void
getDormBuildName(): String
setDormBuildName(dormBuildName: String): void
getDetail(): String
setDetail(detail: String): void
```

Admin 类



```
com.lero.model.Admin
adminId: int
userName: String
password: String
name: String
sex: String
tel: String
Admin()
Admin(userName: String, password: String)
getAdminId(): int
setAdminId(adminId: int): void
getUserName(): String
setUserName(userName: String): void
getPassword(): String
setPassword(password: String): void
getName(): String
setName(name: String): void
getSex(): String
setSex(sex: String): void
getTel(): String
setTel(tel: String): void
```

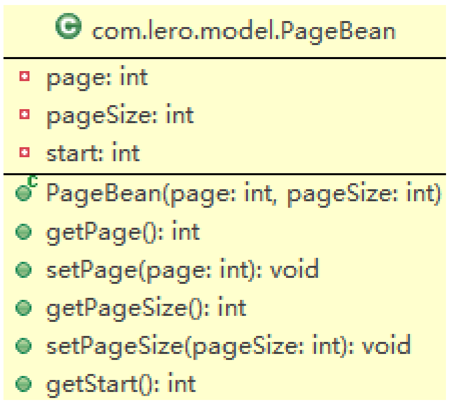
DormManagr 类



com.lero.model.DormManager

```
dormManagerId: int
userName: String
password: String
dormBuildId: int
dormBuildName: String
name: String
sex: String
tel: String
DormManager()
DormManager(userName: String, password: String, name: String, sex: String, tel: String)
DormManager(userName: String, password: String)
getDormManagerId(): int
setDormManagerId(dormManagerId: int): void
getUserName(): String
setUserName(userName: String): void
getPassword(): String
setPassword(password: String): void
getDormBuildId(): int
setDormBuildId(dormBuildId: int): void
getDormBuildName(): String
setDormBuildName(dormBuildName: String): void
getName(): String
setName(name: String): void
getSex(): String
setSex(sex: String): void
getTel(): String
setTel(tel: String): void
```

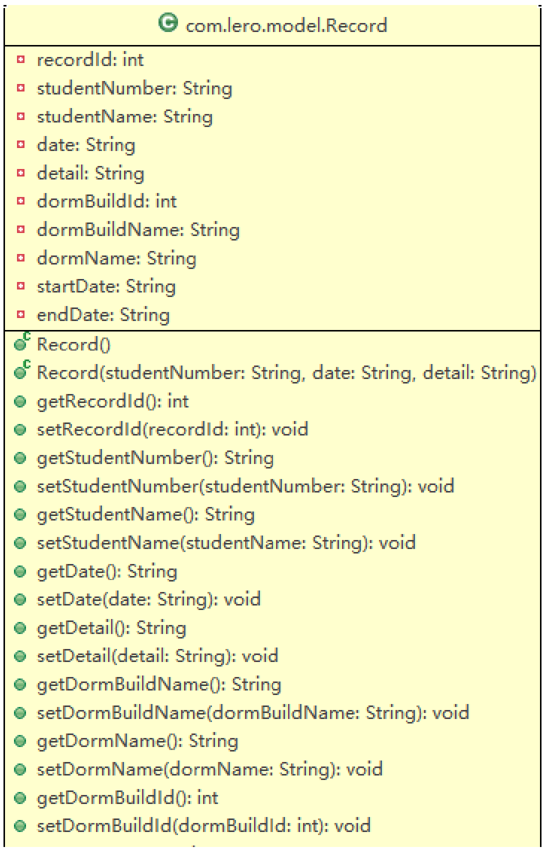
PageBane 类



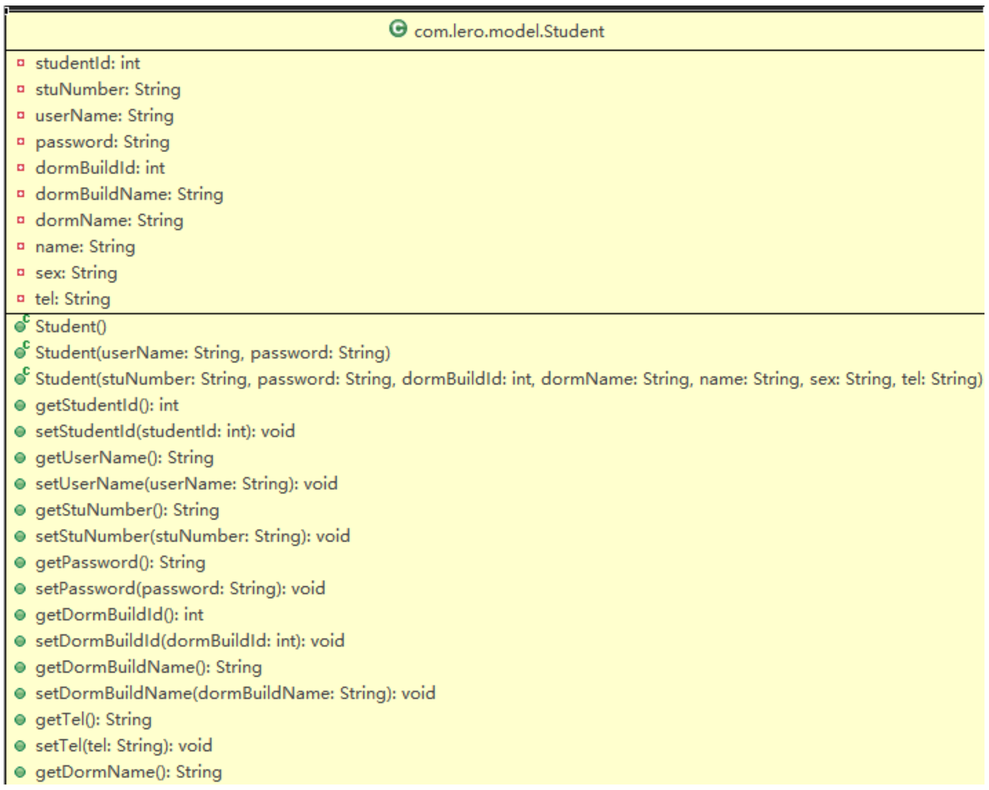
com.lero.model.PageBean

```
page: int
pageSize: int
start: int
PageBean(page: int, pageSize: int)
getPage(): int
setPage(page: int): void
getPageSize(): int
setPageSize(pageSize: int): void
getStart(): int
```

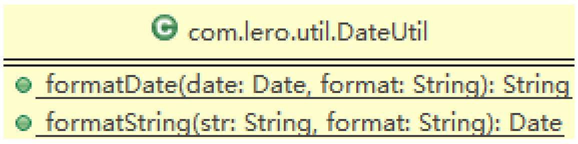
Record 类


```
com.lero.model.Record
recordId: int
studentNumber: String
studentName: String
date: String
detail: String
dormBuildId: int
dormBuildName: String
dormName: String
startDate: String
endDate: String
Record()
Record(studentNumber: String, date: String, detail: String)
getRecordId(): int
setRecordId(recordId: int): void
getStudentNumber(): String
setStudentNumber(studentNumber: String): void
getStudentName(): String
setStudentName(studentName: String): void
getDate(): String
setDate(date: String): void
getDetail(): String
setDetail(detail: String): void
getDormBuildName(): String
setDormBuildName(dormBuildName: String): void
getDormName(): String
setDormName(dormName: String): void
getDormBuildId(): int
setDormBuildId(dormBuildId: int): void
```

Student 类

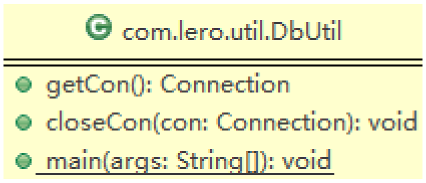

```
com.lero.model.Student
studentId: int
stuNumber: String
userName: String
password: String
dormBuildId: int
dormBuildName: String
dormName: String
name: String
sex: String
tel: String
Student()
Student(userName: String, password: String)
Student(stuNumber: String, password: String, dormBuildId: int, dormName: String, name: String, sex: String, tel: String)
getStudentId(): int
setStudentId(studentId: int): void
getUserName(): String
setUserName(userName: String): void
getStuNumber(): String
setStuNumber(stuNumber: String): void
getPassword(): String
setPassword(password: String): void
getDormBuildId(): int
setDormBuildId(dormBuildId: int): void
getDormBuildName(): String
setDormBuildName(dormBuildName: String): void
getTel(): String
setTel(tel: String): void
getDormName(): String
```

com.lero.util 包：提供工具类
DataUtil 类


com.lero.util.DateUtil

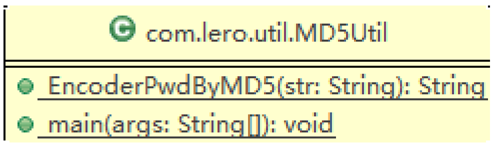
formatDate(date: Date, format: String): String
formatString(str: String, format: String): Date

DBUtil 类


com.lero.util.DbUtil

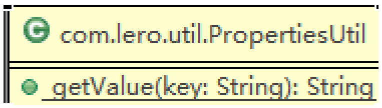
getCon(): Connection
closeCon(con: Connection): void
main(args: String[]): void

MD5Util 类


com.lero.util.MD5Util

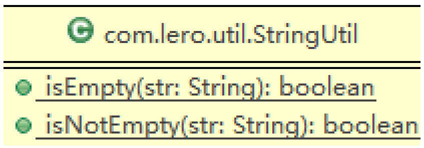
EncoderPwdByMD5(str: String): String
main(args: String[]): void

PropertiesUtil 类


com.lero.util.PropertiesUtil

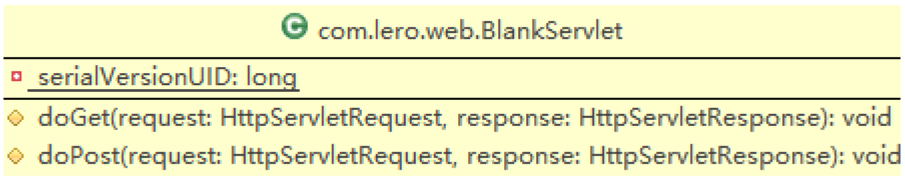
getValue(key: String): String

StringUtil 类


com.lero.util.StringUtil

isEmpty(str: String): boolean
isNotEmpty(str: String): boolean

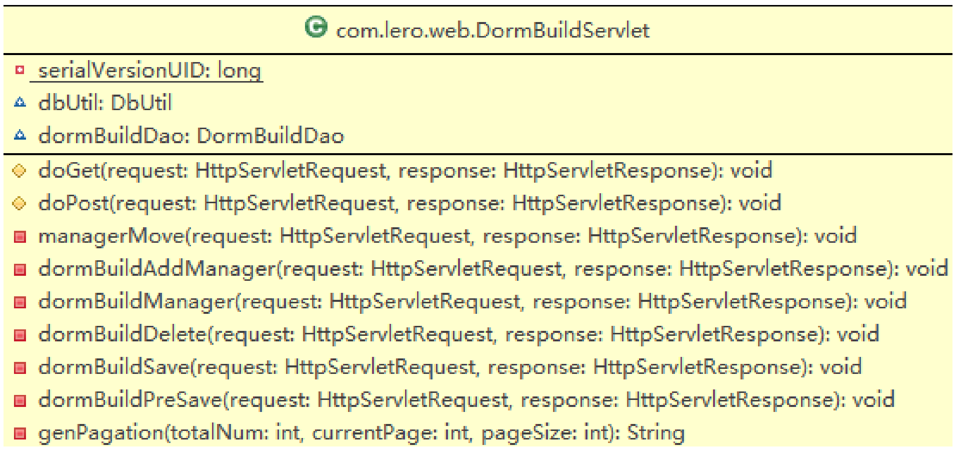
com.lero.web 包：servlet 包，提供后台的业务控制
BlankServer 类


com.lero.web.BlankServlet

serialVersionUID: long

doGet(request: HttpServletRequest, response: HttpServletResponse): void
doPost(request: HttpServletRequest, response: HttpServletResponse): void

DormBulidServlet 类

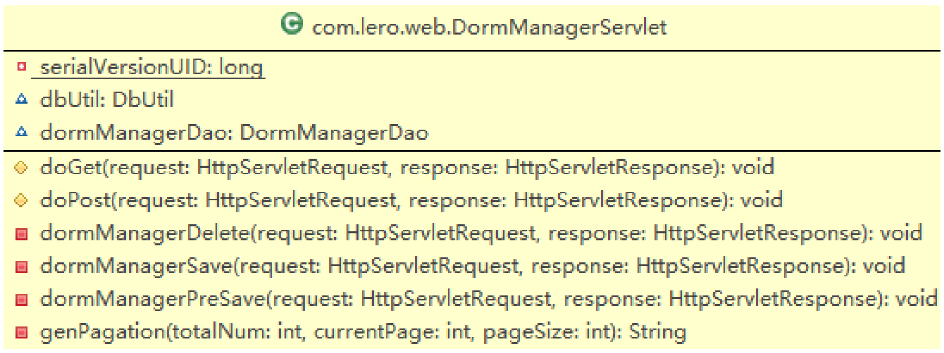


com.lero.web.DormBuildServlet

serialVersionUID: long
dbUtil: DbUtil
dormBuildDao: DormBuildDao

doGet(request: HttpServletRequest, response: HttpServletResponse): void
doPost(request: HttpServletRequest, response: HttpServletResponse): void
managerMove(request: HttpServletRequest, response: HttpServletResponse): void
dormBuildAddManager(request: HttpServletRequest, response: HttpServletResponse): void
dormBuildManager(request: HttpServletRequest, response: HttpServletResponse): void
dormBuildDelete(request: HttpServletRequest, response: HttpServletResponse): void
dormBuildSave(request: HttpServletRequest, response: HttpServletResponse): void
dormBuildPreSave(request: HttpServletRequest, response: HttpServletResponse): void
genPagation(totalNum: int, currentPage: int, pageSize: int): String

DormManagerSerlet 类

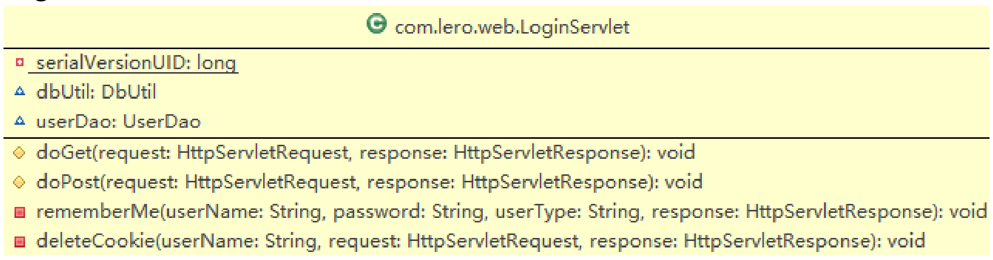


com.lero.web.DormManagerServlet

serialVersionUID: long
dbUtil: DbUtil
dormManagerDao: DormManagerDao

doGet(request: HttpServletRequest, response: HttpServletResponse): void
doPost(request: HttpServletRequest, response: HttpServletResponse): void
dormManagerDelete(request: HttpServletRequest, response: HttpServletResponse): void
dormManagerSave(request: HttpServletRequest, response: HttpServletResponse): void
dormManagerPreSave(request: HttpServletRequest, response: HttpServletResponse): void
genPagation(totalNum: int, currentPage: int, pageSize: int): String

LoginSerlet 类

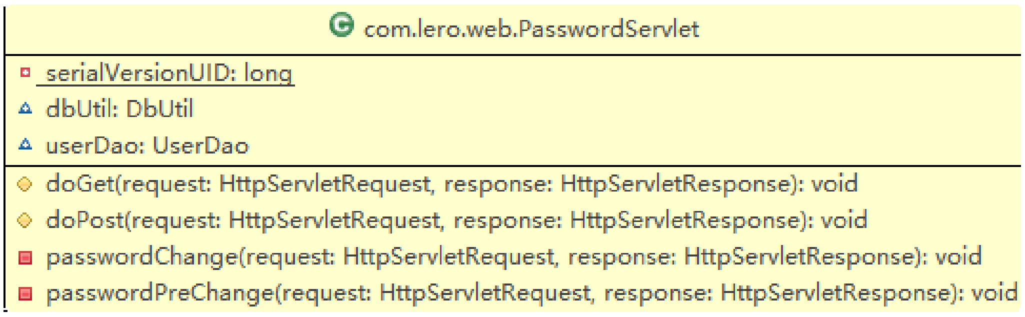


com.lero.web.LoginServlet

serialVersionUID: long
dbUtil: DbUtil
userDao: UserDao

doGet(request: HttpServletRequest, response: HttpServletResponse): void
doPost(request: HttpServletRequest, response: HttpServletResponse): void
rememberMe(userName: String, password: String, userType: String, response: HttpServletResponse): void
deleteCookie(userName: String, request: HttpServletRequest, response: HttpServletResponse): void

PasswordSerlet 类

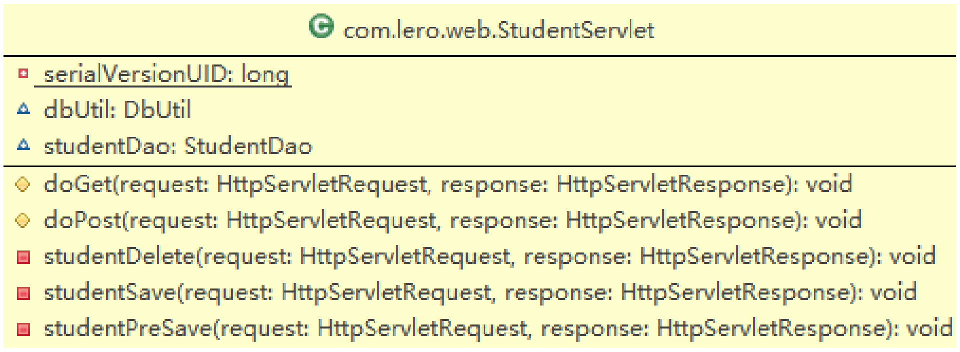


com.lero.web.PasswordServlet

serialVersionUID: long
dbUtil: DbUtil
userDao: UserDao

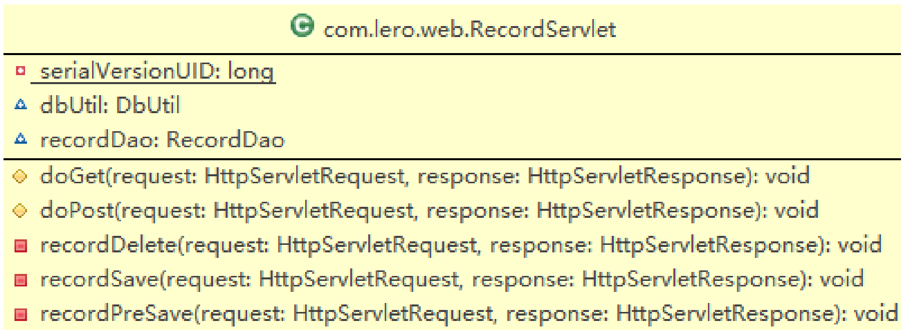
doGet(request: HttpServletRequest, response: HttpServletResponse): void
doPost(request: HttpServletRequest, response: HttpServletResponse): void
passwordChange(request: HttpServletRequest, response: HttpServletResponse): void
passwordPreChange(request: HttpServletRequest, response: HttpServletResponse): void

StudentSerlet 类



```
serialVersionUID: long
dbUtil: DbUtil
studentDao: StudentDao
doGet(request: HttpServletRequest, response: HttpServletResponse): void
doPost(request: HttpServletRequest, response: HttpServletResponse): void
studentDelete(request: HttpServletRequest, response: HttpServletResponse): void
studentSave(request: HttpServletRequest, response: HttpServletResponse): void
studentPreSave(request: HttpServletRequest, response: HttpServletResponse): void
```

RecordServlet 类



```
serialVersionUID: long
dbUtil: DbUtil
recordDao: RecordDao
doGet(request: HttpServletRequest, response: HttpServletResponse): void
doPost(request: HttpServletRequest, response: HttpServletResponse): void
recordDelete(request: HttpServletRequest, response: HttpServletResponse): void
recordSave(request: HttpServletRequest, response: HttpServletResponse): void
recordPreSave(request: HttpServletRequest, response: HttpServletResponse): void
```

动态：

使用状态图进行动态建模。

状态机用于对模型元素的动态行为进行建模，更具体地说，就是对系统行为中受事件驱动的方面进行建模（请参见概念：事件与信号）。状态机专门用于定义依赖于状态的行为（即根据模型元素所处的状态而有所变化的行为）。其行为不会随着其元素状态发生变化的模型元素不需要用状态机来描述其行为（这些元素通常是主要负载管理数据的被动类）。

因为该系统中对象的生命周期很简单，所以就不再具体描述具体对象的状态图。

三、超文本建模

1. 目标

通过 web 应用的内容构建导航，也称作导航建模

1） 建模节点和超文本结构

2） 建模导航路径

2. 产出

超文本/导航结构模型：导航类图

表述超文本的结构，内容模型可以通过导航来访问

超文本访问模型

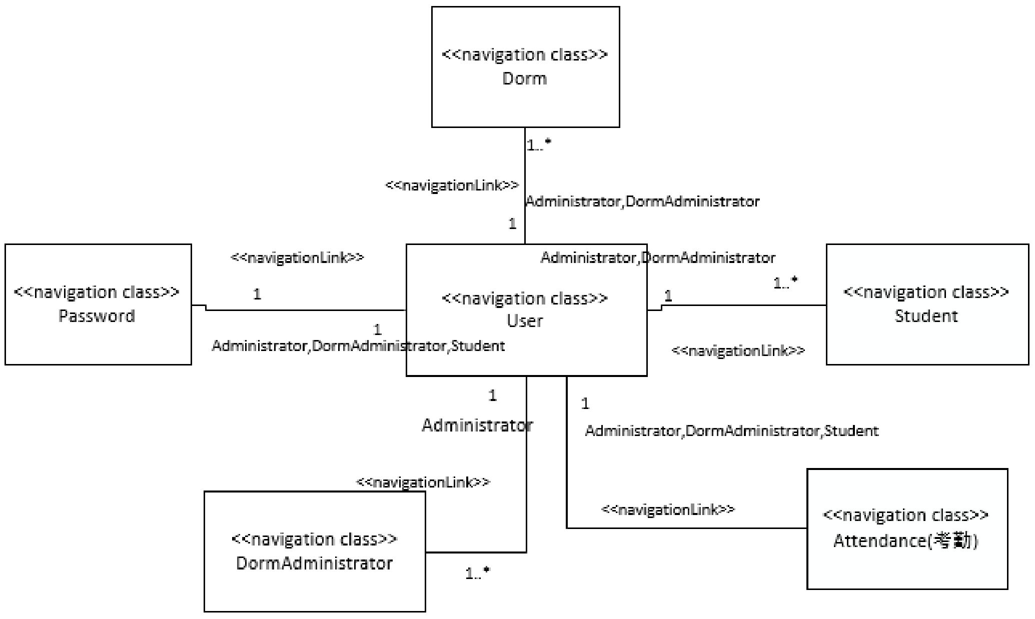
使用访问模型中的访问元素精华超文本结构模型

3. 静态建模

以内容模型为基础，类和对象在超文本中表示为节点，转换规则和按需添加一些链接。还包含一些特定的符号，如 UWE

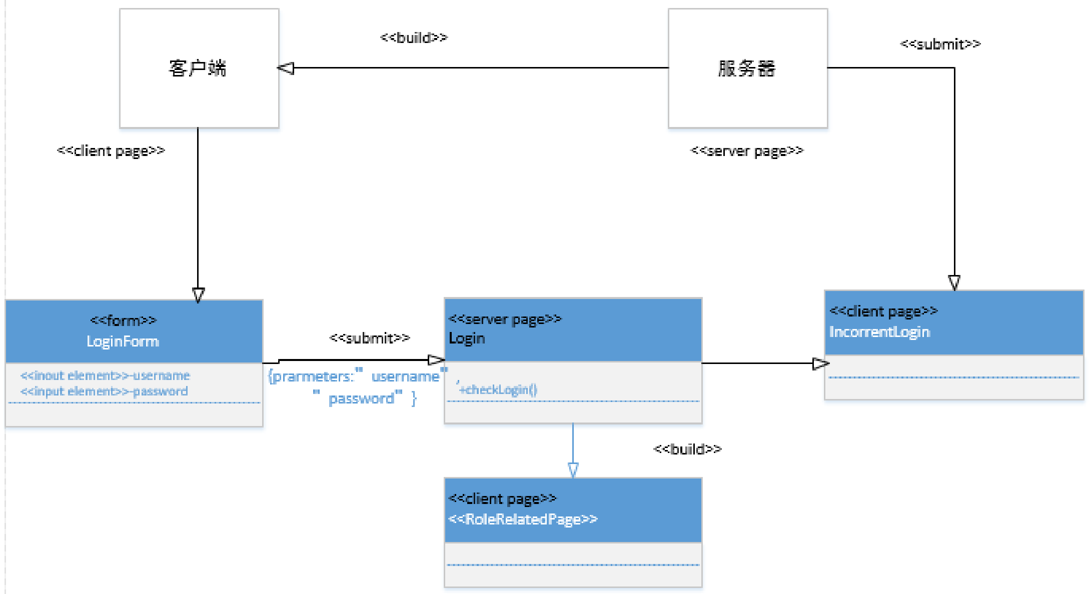
<<navigation class>> : 导航节点

<<navigation link>> : 导航链接
<<process link>>: 过程链接
<<external link>>: 外部链接
如下是宿舍管理系统的超文本视图模型：



4. 动态建模

超文本访问模型：如何通过导航访问到节点



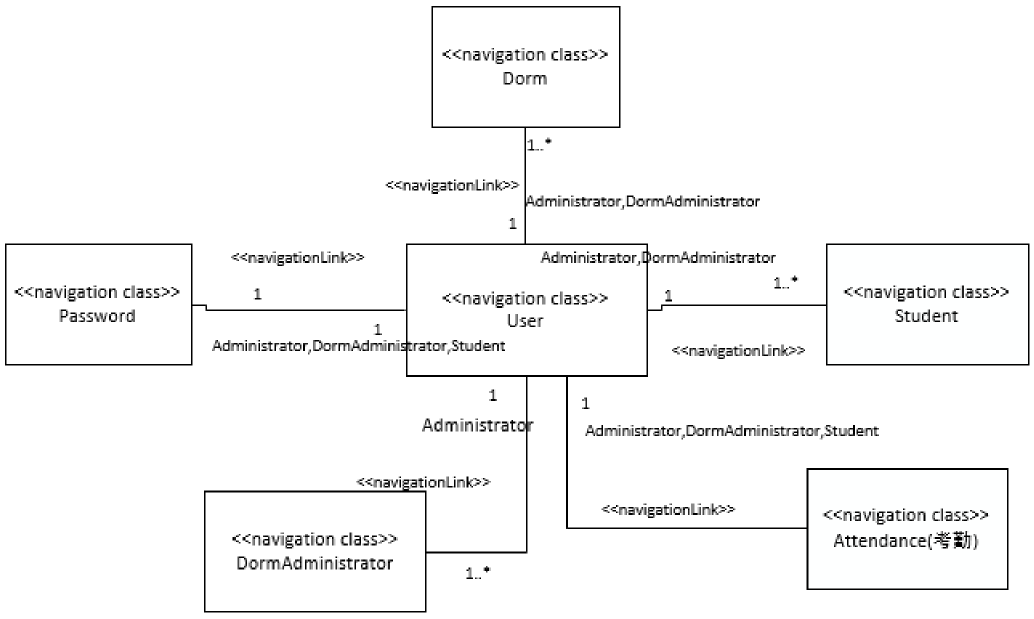
四、适应性建模

1. 目标

根据用户上下文特性，给用户合适的展示

2. 方法

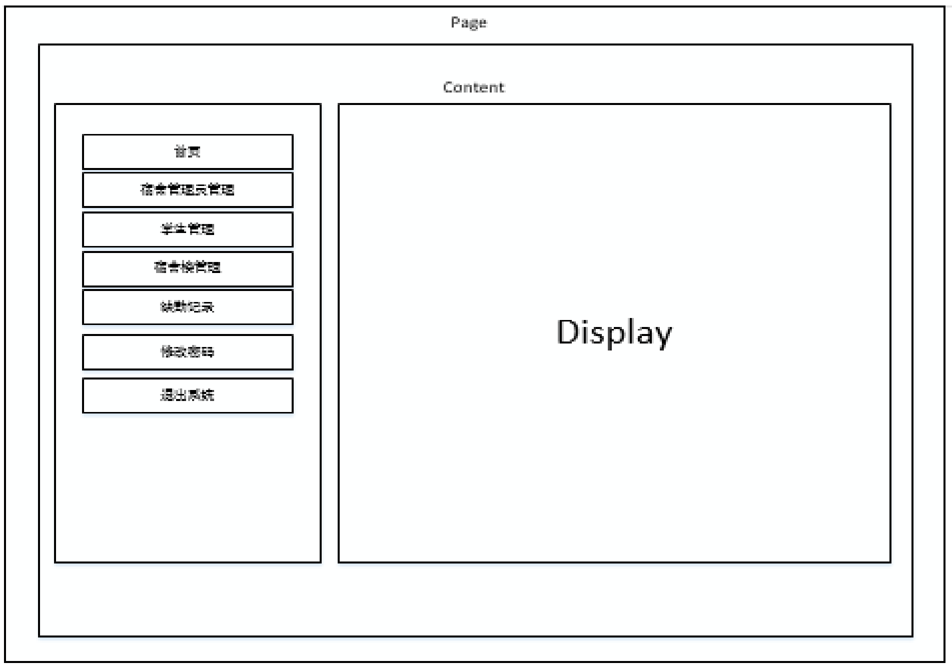
1） 静态建模



<<navigation class>> Dorm
1..*
<<navigationLink>>
Administrator,DormAdministrator
1
<<navigation class>> Password
<<navigationLink>>
1
1
Administrator,DormAdministrator,Student
<<navigation class>> User
Administrator,DormAdministrator
1
1..*
<<navigation class>> Student
<<navigationLink>>
1
Administrator
1
Administrator,DormAdministrator,Student
<<navigationLink>>
<<navigation class>> DormAdministrator
1..*
<<navigationLink>>
<<navigation class>> Attendance(考勤)

2） 动态建模

系统管理员视图

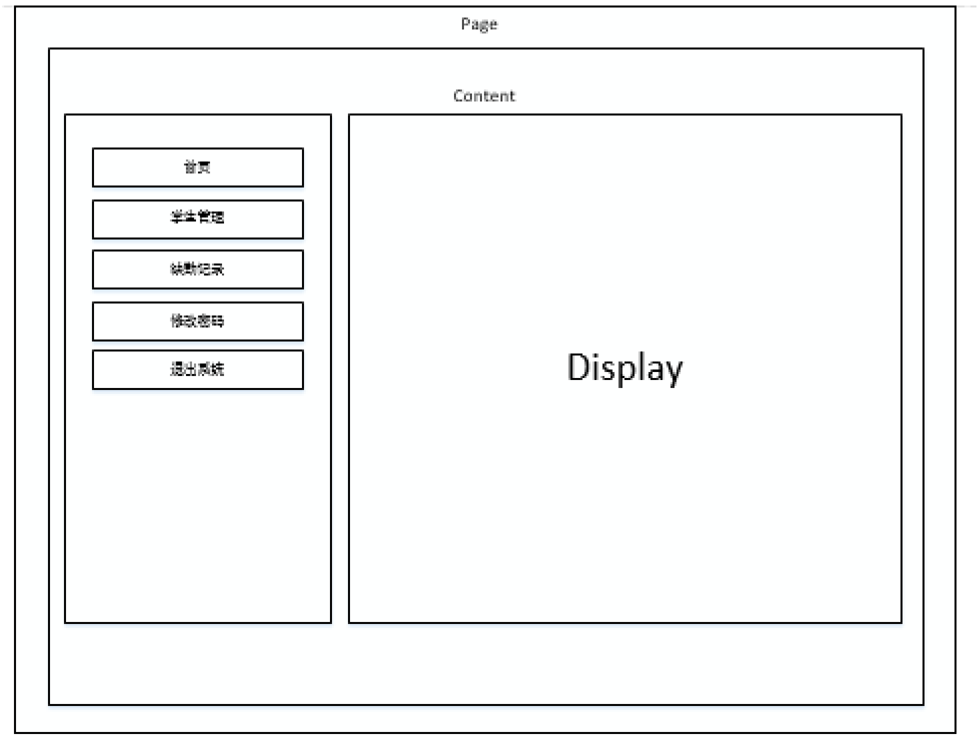


宿舍管理员视图

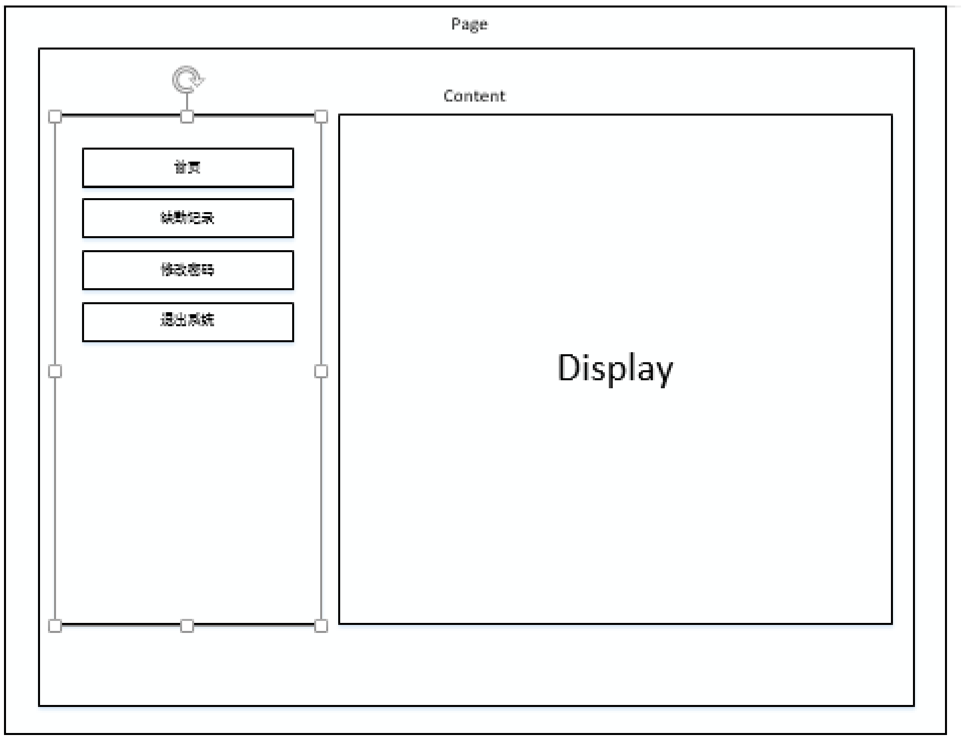
主要是索引的动态适应以及页面的动态适应

3.产出

适应性模型



学生视图：



主要是索引的动态适应以及页面的动态适应

3.产出

适应性模型