**Bug 管理系统 UML 2.0 全程建模**

**——系统分析与设计报告**

**目 录**

[1．项目概述 1](#_bookmark0)

[1.1 需求分析 1](#_bookmark0)

[1.2 开发技术 3](#_bookmark1)

[1.3 UML2.0 全程建模概述 3](#_bookmark1)

[2．系统分析 4](#_bookmark2)

[2.1 用例模型 5](#_bookmark3)

[2.2 BMS用例图 5](#_bookmark3)

[2.3 BMS时序图（需求模型） 6](#_bookmark4)

[2.4 BMS状态图（需求模型） 8](#_bookmark5)

[2.5 BMS活动图（需求模型） 9](#_bookmark6)

[3．系统设计 10](#_bookmark7)

[3.1 体系结构设计 11](#_bookmark8)

[3.2 BMS类图 12](#_bookmark9)

[3.3 BMS时序图（实现模型） 13](#_bookmark10)

[3.4 BMS包图 15](#_bookmark11)

[3.5 BMS组件图 15](#_bookmark11)

[3.6 BMS部署图 16](#_bookmark12)

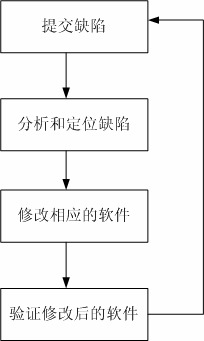
[4．参考资料 17](#_bookmark13)

# 1．项目概述

随着软件项目规模和复杂性的增大，有效跟踪和管理项目中存在的缺陷 Bug 变得越来越重要。每一个软件企业都需要妥善处理软件中的缺陷，这将直接关系 到软件过程质量与软件产品质量，但并非所有的软件组织都知道如何有效地管理 自己软件中的缺陷。在软件缺陷管理(Software Defect Management)中，软件缺陷 的分类和管理非常重要，因此软件缺陷管理工具的开发和使用将在现代软件开发 中发挥重要作用。本报告将使用 UML2.0 对 Bug 管理系统进行全程建模，该系 统名为缺陷管理系统(Bug Management System, BMS)，并按照软件工程的标准， 提供一套完整的解决方案。

## 1.1 需求分析

一个完备的 bug 管理流程通常包括如下几个步骤，如图 1-1 所示：



**图 1-1 bug 管理流程图**

图 1-1 是 bug 管理的最基本流程，而实际的 bug 管理要更加复杂，不同的步 骤由不同的角色负责，如提交 bug、验证修改后的软件是测试人员的工作，分析 和定位 bug 以及修改相应的软件是分析设计人员以及开发人员的工作，在整个过

程中项目经理还需要对 bug 信息进行统计和监控。在 BMS 的需求分析过程中， 我们发现 bug 管理流程的某些步骤可以通过一个 bug 管理系统来完成，一方面可 以提高 bug 的处理速度，另一方面便于对 bug 信息的跟踪与统计。

通过对 bug 管理流程和实际使用过程的需求分析，BMS 系统基本需求如下：

(1) 系统预设管理员帐号为 Admin，初始密码为 Admin。BMS 系统管理员在 登录系统后可修改密码，系统管理员的主要工作包括增加相关人员初始信息，包 括帐号、初始密码和项目角色，项目角色包括测试人员、开发组长、开发人员和 项目经理；另外，系统管理员还可以删除人员信息。

(2) 其他用户在登录后方可使用该系统，除了帐号和项目角色外用户可以修 改各项个人信息，包括真实姓名、联系电话和电子邮箱等。

(3) 测试人员可以利用 BMS 提交自己发现的 bug 信息，提交的信息包括 bug 类型、bug 严重程度、bug 发生的位置（如所处功能模块、测试界面的 URL 或名 称等）、测试环境描述、使用的测试工具和版本信息、测试用例信息（包括测试 数据、期望结果和实际结果等信息）、附加描述信息、附件（屏幕截图或录像等） 等。测试人员将尽量填写完整这些信息以便最大程度帮助开发人员重现 bug 以便 调试，在系统数据库中需要记录 bug 的状态。

(4) 测试人员将 bug 提交给开发组长，开发组长在查看 bug 信息之后可将 bug

分发给相关开发人员，系统可以记录开发组长的 bug 查看和分发情况。

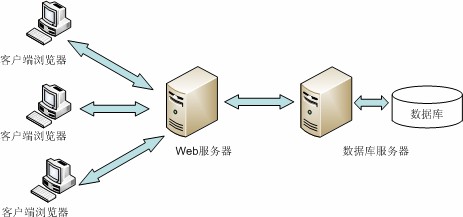
(5) 开发人员可以登录系统查看 bug 详情，系统可以记录开发人员是否已查 看 bug 详情。在对 bug 进行修复后，更新 bug 修复信息（修复内容、修复时间、 修复人等），将更新的 bug 信息发送给测试人员，系统将修改 bug 的状态，然后 通知测试人员以获取最新版本进行验证。

(6) 测试人员如验证无误，可关闭该 bug；否则可重新返回开发人员修复。 无论验证是否通过，测试人员需更新 bug 测试信息（测试结果、测试时间、测试 人等）。

(7) 项目经理可以随时查看 bug 统计报告，对 bug 信息进行分类汇总与实时 跟踪。

## 1.2 开发技术

本系统采用三层 B/S 结构进行开发，包括客户端浏览器层、Web 服务器层和 数据库服务器层，系统整体架构如图 1-2 所示：



**图 1-2 BMS 整体架构**

在实际部署和使用过程中，如果数据量较小，可以将 Web 服务器和数据库 服务器合二为一。B/S 结构具备部署和升级简单等优点，系统安装、修改和维护 全在服务器端解决，用户在使用系统时，仅仅需要一个浏览器就可运行全部的模 块；同时 B/S 结构还提供了一种异种机、异种网联机、联网解决方案，开发团队 与测试团队可以基于不同的操作系统平台和网络环境进行协同工作。

BMS 系统开发技术包括：采用 Java EE 平台，使用 MVC 架构，运用 JSP + Servlet + JavaBean 等技术来实现系统功能，数据库采用 MySQL，并使用 Navicat 8 for MySQL 对 MySQL 数据库进行可视化管理，服务器中间件使用 Tomcat 6.5， 开发工具使用 MyEclipse 8.0。

## 1.3 UML2.0 全程建模概述

为了更为高效快捷地开发 BMS ，我们采用了 UML(Unified Modeling Language, 统一建模语言) 2.0 建模技术，并充分使用 UML 2.0 建模语言的特性， 对系统进行全程建模。在使用 UML 2.0 的同时，我们提出了 UML 全程建模 (UML-Full Process Modeling, UML-FPM)的概念，将 UML 2.0 中的 13 种图应用于 系统分析与设计的全过程，通过对 UML 2.0 十三种图进行分析，并根据这些图 之间的关系及作用绘制了如图 1-3 所示的 UML 2.0 全程建模流程图，在该图中将

系统开发过程分为两个大阶段：需求分析与系统分析，系统设计与实现，该图使 用 UML 活动图绘制，较为全面、清晰地描述如何应用 UML 技术来构造系统的 分析与设计模型以及 UML 各图形之间的关系。



通信图（需求模型）

**需求分析与系统分析**

用例图（用例文档）

[是]

是否需要清晰显示交互对象关系

时序图（需求模型）

是否存在复杂对象

是否存在复杂流程

[是]

[是]

状态图（需求模型）

活动图（需求模型）

时序图（实现模型）

**系统设计与实现**

状态图（实现模型）

是否需要对时序图进行组织

活动图（实现模型）

[是]

是否需要清晰显示对象间的关系

是否需要了解每一个活动的细节

[是]

（

交互概览图

是否需要标明定时约束

是否需要一个类图实例

类图

[是]

定时图

[逻辑划分]

[物理划分]

[是]

对象图

组件图

包图

组合结构图

[是]

部署图

内部结构是否复杂

|  |  |
| --- | --- |
|  | [是] |
| 通信图 | 实现模型） |

**图 1-3 UML 2.0 全程建模流程图**

BMS 采用 UML-FPM 进行全程建模，我们构造了 BMS 的各种视图，包括用 户视图、结构视图、动态视图、实现视图和环境视图，每种视图对应一种或多种 UML 图形，通过这些图形来对系统进行全面而有效的分析与设计。

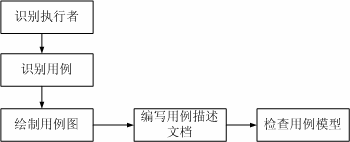
# 2．系统分析

在 BMS 的系统分析阶段，我们使用了用例图、时序图、状态图和活动图等 UML 图形构造系统的分析模型，对系统进行深入的分析，明确系统的开发目标， 更好地回答了“做什么”的问题，各种图形相互补充，从不同的角度对系统进行 全面的分析。通过使用 UML-FPM 方法，我们构造了系统的分析模型，具体分析 工作如下：

## 2.1 用例模型

在 BMS 系统中，我们首先使用用户视图即用例图来将系统功能需求图形化， 通过找出执行者与用例来明确和细化系统功能。

UML 用例建模流程如图 2-1 所示：



**图 2-1 UML 用例建模流程图**

BMS 的执行者包括系统管理员、开发组长、开发人员、测试人员和项目经 理，每个执行者对应的功能有所差异。系统提供主要功能包括人员信息的管理和 bug 信息的管理，因此用例主要包括对人员信息和 bug 信息的增删改查等操作。

## 2.2 BMS 用例图

通过对系统进行分析，BMS 用例图如图 2-2 所示：



<<include>>

<<include>>

分发bug

开发组长

修改密码

<<include>>

增加人员信息

<<include>>

<<include>>

登录

系统管理员

删除人员信息

<<include>>

修改个人信息

<<include>>

<<include>>

开发人员

用户

更新bug修复信息

<<include>>

查看bug信息

<<include>>

<<include>>

更新bug测试信息

测试人员

提交bug信息

关闭bug信息

查看bug统计报告

项目经理

**图 2-2 BMS 用例图**

## 2.3 BMS 时序图（需求模型）

在 UML-FPM 中，我们将时序图分为两类，一类用于描述系统需求，构造系 统的需求模型；另一类用于指导设计与实现，构造系统的实现模型。

在系统分析时，可以通过时序图来对执行者和系统的交互过程进行建模，方 便用户更好地理解系统的工作流程。对于需求模型时序图，一般使用用户熟悉的 业务语言来进行系统描述，不涉及到实现细节，一方面方便用户理解，另一方面 可以指导后续类图的设计。时序图可显示不同的业务对象如何交互，对于交流当 前业务如何进行很有用，一个业务级的时序图能被当作一个需求文件使用，为实 现一个未来系统传递需求；同时，时序图能够使用更为清晰形象的表达，将用例 带入下一层次，通常一个用例可以被细化为一个或者更多的时序图。时序图的主 要用途之一，是把用例表达的需求，转化为进一步、更深层次的精细表达。

根据需求我们绘制了每一个用例的时序图，由于篇幅关系，未将每个用例的 时序图一一列举。图 2-3、2-4、2-5、2-6 分别是用例“登录”、“提交 bug 信息”、 “查看 bug 信息”和“更新 bug 信息”的时序图。



登录

登录界面

业务对象

数据库操作对象

数据库

用户

打开界面

loop

[如果验证失败]

系统提示输入帐号和密码

输入帐号和密码

提交数据

验证数据

查询数据

返回结果

返回结果

返回结果

显示结果

**图 2-3 用例“登录”时序图（需求模型）**



提交bug信息

提交界面

业务对象

数据库操作对象

数据库

测试人员

打开界面

loop

[wrong]

输入bug信息

提交bug信息

验证bug信息

增加数据

返回结果

返回结果

返回结果

显示结果

**图 2-4 用例“提交 bug 信息”时序图（需求模型）**



查看bug信息

bug显示界面

业务对象

数据库操作对象

数据库

用户

打开界面

查询信息

查询信息

查询信息

返回信息

返回信息

返回信息

显示bug信息

**图 2-5 用例“查看 bug 信息”时序图（需求模型）**

息

返回



更新bug修复信息

bug显示界面

bug信息修改界面

业务对象

数据库操作对象

数据库

开发人员

ref

查看bug信息

返回bug信息

查看bug信息()

1: 显示bug信息

2: 选择一个bug

3: 选择bug信息

4: 查询bug信息

5: 查询数据库

6: 返回bug信息

7: 返回bug信息

8: 返回bug信息

9: 显示一个bug信息

10: 输入修复信息

11: 提交修复信息

12: 更新bug信息

13: 更新数据库

14: 返回信息

15: 返回信息

16: 返回信息

17: 显示更新结果

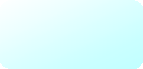
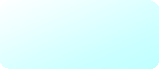
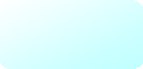
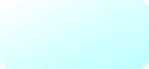
**图 2-6 用例“更新 bug 信息”时序图（需求模型）**

1

## 2.4 BMS 状态图（需求模型）

在需求分析过程中，我们发现 BMS 系统的核心对象是 bug，因此可以使用 状态图对其进行建模。UML 中的状态图可以用来描述一个特定对象的所有可能 状态及其引起状态转移的事件。只有那些具有重要交互行为的类，才会使用状态 图来描述，一个状态图包括一系列对象的状态及状态之间的转换。在实际建模中， 并不需要给出每个对象的状态图，而需要将注意力集中在整体系统或少数关键的 对象上，特别是那些状态比较多的对象。

在 BMS 系统中，最复杂也最为重要的对象是 bug，它在系统中拥有多种不 同的状态，不同类型的用户可以对其进行操作，为了更好地描述 bug 对象状态的 转换，我们绘制了 bug 对象状态图，如图 2-7 所示：



新提交bug

开发组长查看

开发组长已 查看bug

分发

已分发bug

开发人员查看

开发人员已查 看bug

已修复bug

do / 验证

已关闭bug

修复bug

[验证通过]/ 验证

[验证未通过]/ 验证

**图 2-7 bug 对象状态图**

在图 2-7 中，我们可以清晰了解 bug 对象在系统中所具有的状态以及这些状 态之间的转换过程，如测试人员提交的 bug 其状态为“新提交 bug”，开发组长 查看后该 bug 的状态将变为“开发组长已查看 bug”。

## 2.5 BMS 活动图（需求模型）

在状态图中，我们描述了 BMS 系统中 bug 对象的各种状态以及状态之间的 转换关系，但是这些状态在转换的过程中无法确定何种状态由哪类执行者负责操 作，因此可以通过活动图来进行建模，此时的活动图用于对需求模型进行进一步 细化。在系统分析过程中，我们使用活动图取代传统的流程图，在表示系统业务 流程的同时通过泳道来确定每一个活动的执行者。在活动图中我们还使用了对象 流来表示活动与对象之间的依赖关系，描述在活动中对象的状态。通过活动图建 立的模型比状态图建立的模型具有更多信息，在 BMS 中，我们描述了不同用户 对 bug 的操作活动以及在每一次活动之后 bug 对象所处于的状态，对操作流程进 行图形化建模，如图 2-8 所示：



测试人员

开发组长

开发人员

提交bug信息

新提交bug

开发组长已查看bug

开发组长查看bug信息

分发bug

已分发bug

开发人员查看bug信息

开发人员已查看bug

更新bug修复信息

测试人员查看bug信息

已修复bug

更新bug测试信息

[否]

修复是否成功

[是]

关闭bug信息

已关闭bug

**图 2-8 BMS 活动图**

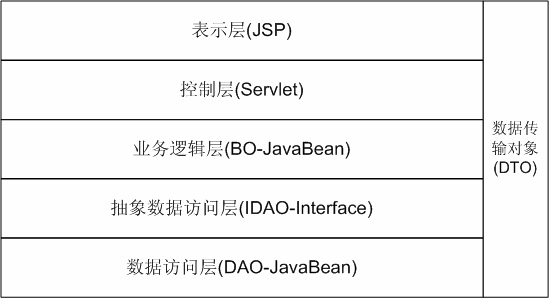
[否]

# 3．系统设计

在对系统进行全面分析后，我们开始使用 UML-FPM 方法对系统进行设计， 构造了 BMS 系统的设计模型，包括类图、包图、时序图（实现模型）、组件图 和部署图等的绘制，回答了“怎么做”的问题。具体设计工作如下：

## 3.1 体系结构设计

BMS 采用多层 Java EE 设计方案，考虑到系统的扩展性，定义了抽象的数 据访问层，系统体系结构图如图 3-1 所示：



**图 3-1 BMS 体系结构图**

在图 3-1 中，BMS 系统一共包含五层，其中表示层使用 JSP 来实现，控制 层使用 Servlet 实现，Servlet 将调用业务逻辑层中的方法实现具体业务功能，如 果业务功能在实现过程需要访问数据库，则调用数据访问层中的数据库操作方 法，为了保证系统的扩展性，我们定义了抽象的数据访问层，业务逻辑层针对抽 象数据访问层编程，而将具体数据访问类类名存储在配置文件中，使用 XML 格 式的文件作为配置文件，提高系统的可扩展性。具体实现方案如图 3-2 所示：

<className>DA O</className>



BO

config.xml

IDAO

DAO

**图 3-2 数据访问层扩展实现方案**

## 3.2 BMS 类图

类图是一个面向对象系统最核心的设计图之一，BMS 的主要功能包括 bug 管理和人员信息管理，针对这两个主要功能模块，我们绘制了两个类图，图 3-3 对用户信息管理进行建模，图 3-4 对 bug 信息管理进行建模。

为了更好地描述各种不同的类，我们使用了彩色 UML 建模方式，不同类型 的类使用不同的颜色来表示，如使用红色表示数据传输类 DTO，使用粉红色表 示 JSP 界面类，使用绿色表示 Servlet 控制类，使用蓝色表示业务逻辑类 BO，使 用浅蓝色表示数据访问接口 IDAO，使用橙色表示数据访问类 DAO。



UserDTO

* userAccount : String
* userPassword : String
* userRole
* userName
* userTel
* userEmail

: int

: String

: String

: String

+ <<Getter>> getUserAccount ()

+ <<Setter>> setUserAccount (String newUserAccount)

+ <<Getter>> getUserPassword ()

: String

: void

: String

+ <<Setter>> setUserPassword (String newUserPassword) : void

+ <<Getter>> getUserRole ()

+ <<Setter>> setUserRole (int newUserRole)

+ <<Getter>> getUserName ()

+ <<Setter>> setUserName (String newUserName)

+ <<Getter>> getUserTel ()

+ <<Setter>> setUserTel (String newUserTel)

+ <<Getter>> getUserEmail ()

+ <<Setter>> setUserEmail (String newUserEmail)

1..1 0..\*

: int

: void

: String

: void

: String

: void

: String

: void

<<JSP>>

login.jsp

<<JSP>>

modifyUserInfo.jsp

<<JSP>>

viewUser.jsp

<<JSP>>

addUser.jsp

FindUserServlet

* userdto : UserDTO
* finduserbo : FindUserBO

+ doPost () : void

LoginServlet

- loginbo : LoginBO

+ doPost () : void

1..1

AddUserServlet

* adduserbo : AddUserBO
* userdto : UserDTO

+ doPost () : void

1..1

ViewUserServlet

- findalluserbo : FindAllUserBO

- userlist

: List

+ doPost () : void

DeleteUserServlet

- deleteuserbo : DeleteUserBO

+ doPost () : void

ModifyUserServlet

LoginBO

- userdao : IUserDAO

+ validate (String account, : int Stirng password)

FindAllUserBO

- userdao : IUserDAO

+ findAllUser () : List

- userdto

: UserDTO

DeleteUserBO

- userdao : IUserDAO

+ deleteUser (String account) : int

- modifyuserbo : ModifyUserBO

+ doPost () : void

AddUserBO

- userdao : IUserDAO

+ addUser (UserDTO userdto) : int

FindUserBO

- userdao : IUserDAO

+ findUser (String account) : UserDTO

IUserDAO

......

userdao.findUserByAccount(account);

......

+ findUserByAccount (String account) : UserDTO

+ addUser (UserDTO userdto)

+ findAllUser ()

+ deleteUser (String account)

+ modifyUser (UserDTO userdto)

: int

: List

: int

: int

ModifyUserBO

- userdao : IUserDAO

+ modifyUser (UserDTO userdto) : int

UserDAO

+ <<Implement>> findUserByAccount (String account) : UserDTO

+ <<Implement>> addUser (UserDTO userdto)

+ <<Implement>> findAllUser ()

+ <<Implement>> deleteUser (String account)

+ <<Implement>> modifyUser (UserDTO userdto)

: int

: List

: int

: int

**图 3-3 用户信息管理部分类图**



BugDTO

* bugId
* bugType
* bugLevel
* bugPosition

: String

: String

: int

: String

- bugTestDescription : String

* bugTestTool
* bugTestCase
* bugDescription
* bugAttachment
* bugTestInfo
* bugModifyInfo

: String

: String

: String

: String

: String

: String

+ <<Getter>> getBugId ()

+ <<Setter>> setBugId (String newBugId)

+ <<Getter>> getBugType ()

+ <<Setter>> setBugType (String newBugType)

+ <<Getter>> getBugLevel ()

+ <<Setter>> setBugLevel (int newBugLevel)

+ <<Getter>> getBugPosition ()

+ <<Setter>> setBugPosition (String newBugPosition)

+ <<Getter>> getBugTestDescription ()

+ <<Setter>> setBugTestDescription (

String newBugTestDescription)

+ <<Getter>> getBugTestTool ()

+ <<Setter>> setBugTestTool (String newBugTestTool)

+ <<Getter>> getBugTestCase ()

: String

: void

: String

: void

: int

: void

: String

: void

: String

: void

: String

: void

: String

+ <<Setter>> setBugTestCase (String newBugTestCase) : void

+ <<Getter>> getBugDescription ()

+ <<Setter>> setBugDescription (

String newBugDescription)

+ <<Getter>> getBugAttachment ()

+ <<Setter>> setBugAttachment (

String newBugAttachment)

+ <<Getter>> getBugTestInfo ()

+ <<Setter>> setBugTestInfo (String newBugTestInfo)

+ <<Getter>> getBugModifyInfo ()

+ <<Setter>> setBugModifyInfo (

String newBugModifyInfo)

: String

: void

: String

: void

: String

: void

: String

: void

<<JSP>>

modifyBug.jsp

<<JSP>>

viewDetailBug.jsp

<<JSP>>

viewReport.jsp

<<JSP>>

addBug.jsp

<<JSP>>

viewAllBug.jsp

ModifyBugServlet

: BugDTO

: UserDTO

* bugdto
* userdto
* modifybugbo
* finduserbo

: ModifyBugBO

: FindUserBO

ViewDetailBugServlet

AddBugServlet

* bugdto : BugDTO
* addbugbo : AddUserBO

+ doPost () : void

ViewAllBugServlet

- bugdto

: BugDTO

- finddetailbugbo : FindDetailBugBO

+ doPost () : void

- buglist

: List

- viewdetailbugbo : FindDetailBugBO

- viewallbugbo : ViewAllBugBO + doPost () : void

+ doPost () : void

FindUserBO

- userdao : IUserDAO

+ findUser (String account) : UserDTO

FindDetailBugBO

- bugdao : IBugDAO

+ findDetailBug (String bugId) : BugDTO

ViewAllBugBO

- bugdao : IBugDAO

+ findAllBug () : List AddBugBO

- bugdao : IBugDAO

+ addBug (BugDTO bugdto) : int

ModifyBugBO

- bugdao : IBugDAO

+ modifyBug (BugDTO bugdto) : int

IUserDAO

......

UserDTO userdto=fi int role=userdto.getU if(role==1)

{

......

}

else if(role==2)

{

......

}

......

IBugDAO

+ findUserByAccount (String account) : UserDTO

+ addBug (BugDTO bugdto)

+ findAllBug ()

+ findDetailBug (String bugId)

: int

: List

: BugDTO

+ addUser (UserDTO userdto)

+ findAllUser ()

+ deleteUser (String account)

+ modifyUser (UserDTO userdto)

: int

: List

: int

: int

+ modifyBug (BugDTO bugdto) : int

UserDAO

BugDAO

+ <<Implement>> findUserByAccount (String account) : UserDTO

+ <<Implement>> addBug (BugDTO bugdto)

+ <<Implement>> findAllBug ()

+ <<Implement>> findDetailBug (String bugId)

: int

: List

: BugDTO

+ <<Implement>> addUser (UserDTO userdto)

+ <<Implement>> findAllUser ()

+ <<Implement>> deleteUser (String account)

+ <<Implement>> modifyUser (UserDTO userdto)

: int

: List

: int

: int

+ <<Implement>> modifyBug (BugDTO bugdto) : int

**图 3-4 bug 信息管理部分类图**

## 3.3 BMS 时序图（实现模型）

在系统设计与实现阶段我们也可以使用时序图进行建模，此时通过时序图可 以明确表示系统设计中对象之间的交互，考虑到具体系统实现，对象之间通过方 法调用传递消息。在 BMS 系统中，对每个用例均可以绘制基于实现模型的时序

图。由于篇幅关系，未将每个用例的时序图一一列举。图 3-5、3-6 分别是用例 “登录”和“提交 bug 信息”的实现模型时序图。

serB



SD-Login

login: LoginBO

userdao: UserDAO

:login.jsp

loginServlet:LoginServlet

loop

[resulst=u=b0m]

it

doPost()

validate(account,password)

findUserByAccount(account) return userdto

return result

display

**图 3-5 用例“登录”时序图（实现模型）**

val



SD-AddBug

addbugbo: AddBugBO

bugdao: BugDAO

:addBug.jsp addBugServlet:AddBugServlet

submit

new()

bugdto:BugDTO

setXXX()

return bugdto

doPost()

addBug(bugdto)

aadddBug(bugdto)

return 1

return 1

display

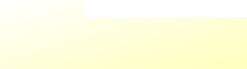
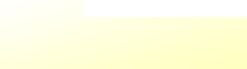
**图 3-6 用例“提交 bug 信息”时序图（实现模型）**

d

在图 3-5、3-6 中，通过实现模型时序图来表示对象之间的交互关系，每个 用例对应一个时序图，开发人员可以根据时序图编写实现代码，并更好地与其他 成员交流。

## BMS 包图

BMS 根据系统分层架构组包，并参照类之间的关系绘制包之间的关系，如 图 3-7 所示：



<<import>>

<<import>>

com.sunny.bms. servlet

<<import>>

<<import>>

<<import>>

com.sunny.bms.dto

com.sunny.bms.bl

com.sunny.bms.util

<<import>>

<<import>>

com.sunny.bms.idao

<<import>>

com.sunny.bms.dao

**图 3-7 BMS 包图**

在包图中体现了系统的分层架构，包与层的对应关系如表 3-1 所示：

**表 3-1 BMS 包与层对应关系表**

|  |  |
| --- | --- |
| **包 名** | **分 层** |
| com.sunny.bms.servlet | 控制层 |
| com.sunny.bms.bl | 业务逻辑层 |
| com.sunny.bms.idao | 抽象数据访问层 |
| com.sunny.bms.dao | 数据访问层 |

另外，在 com.sunny.bms.dto 包中封装了数据传输类，在 com.sunny.bms.util 包中封装了一些工具类，如字符编码转换类、数据加密类等，可以根据实际需要 增加到系统中。

## BMS 组件图

组件图表示了系统中物理文件之间的关系，描述了每个功能所在的组件位置 以及它们之间的关系。在 BMS 设计方案中，将类打包成 war 文件或者 jar 文件， 以便于部署和重用，BMS 组件图如图 3-8 所示：

|  |
| --- |
| BugDTO |
|  |
|  |

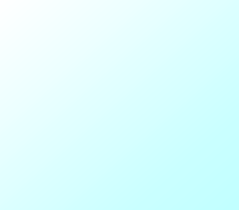
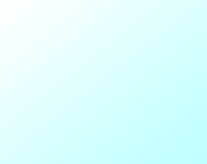
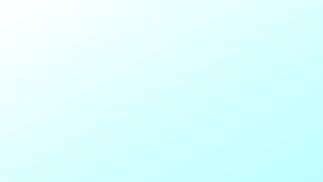
|  |
| --- |
| AddBugBO |
|  |
|  |

|  |
| --- |
| FindAllUserBO |
|  |
|  |

|  |
| --- |
| ViewAllBugBO |
|  |
|  |

|  |
| --- |
| LoginBO |
|  |
|  |

**图 3-8 BMS 组件图**



dao.jar

BugDAO

idao.jar

dto.jar

UserDAO

UserDTO

IBugDAO IUserDAO

bo.jar

DeleteUserBO

ModifyUserBO

ModifyBugBO

FindUserBO

FindDetailBugBO

AddUserBO

bug.war

<<JSP>>

addBug.jsp

<<JSP>>

addUser.jsp

AddBugServlet

AddUserServlet

<<JSP>>

viewAllBug.jsp

ViewAllBugServlet

<<JSP>>

viewUser.jsp

ViewUserServlet

<<JSP>>

login.jsp

DeleteUserServlet

<<JSP>>

modifyUserInfo.jsp

ViewDetailBugServlet

ModifyUserServlet

LoginServlet

<<JSP>>

modifyBug.jsp

<<JSP>>

viewDetailBug.jsp

<<JSP>>

viewReport.jsp

ModifyBugServlet

FindUserServlet

在图 3-8 中，将所有的 JSP 和 Servlet 打包成一个 war 文件，然后将其他类 按照分层打包成不同的 jar 文件，在 dto.jar 中包含了所有的 DTO 类，在 idao.jar 中包含了所有的抽象 DAO 接口，在 dao.jar 中包含了所有的 DAO 类，在 bo.jar 中包含了所有的 BO 类。将不同的类封装在不同的组件中以便于组件的复用，降 低系统的复杂性。

## BMS 部署图

部署图描述了系统硬件的拓扑结构以及在此结构上执行的软件，是一种常用

的系统架构建模技术。BMS 系统部署图如图 3-9 所示：

|  |  |  |
| --- | --- | --- |
| 数据库服务器 | | |
|  | MySQL |  |
|  |
|  | | |

**图 3-9 BMS 部署图**



应用服务器

客户端

web

bo

dto

Browser

idao

dao

在图 3-9 中，按照系统的整体架构，BMS 采用 B/S 结构进行开发，因此客 户端只需要安装浏览器(Browser)，将所有的 JSP 文件和类部署在应用服务器中， 通过 DAO 类操作存储在 MySQL 数据库服务器中的数据。系统维护和部署人员 可以按照该部署图部署应用程序组件。

至此，通过应用 UML-FPM 技术构造了 BMS 系统完整的分析和设计模型， 通过正向工程可以生成部分程序代码，开发人员可以基于该模型快速开发系统， 测试人员可以根据模型设计相应的测试用例。本系统全程采用 UML 2.0 进行建 模，充分发挥各种 UML 图形的特点，提升系统分析与设计水平，为后续实施和 测试工作奠定基础。

# 4．参考资料

[1] Grady Booch, James Rumbaugh, Ivar Jacobson 著. 邵维忠，麻志毅等译. UML

用户指南(第 2 版). 北京: 人民邮电出版社, 2006.

[2] Grady Booch, Ivar Jacobson, James Rumbaugh 著. UMLChina 译. UML 参考手册

(第 2 版). 北京: 机械工业出版社, 2005.

[3] Martin Fowler 著. 徐家福译. UML 精粹：标准对象建模语言简明指南(第 3 版).

北京: 清华大学出版社, 2005.

[4] Craig Larman 著. 李洋, 郑龑译. UML 和模式应用(原书第 3 版). 北京: 机械工 业出版社, 2006.

[5] 邵维忠, 杨芙清著. 面向对象的系统分析(第 2 版). 北京: 清华大学出版社, 2006.

[6] 周力, 何雪飞. UML 建模图解教程. 北京：人民邮电出版社, 2009.

[7] 谭云杰. 大象-Thinking in UML. 北京：中国水利水电出版社, 2009. [8] 王先国. UML 统一建模实用教程. 北京：清华大学出版社, 2009.

[9] Scott W.Ambler 著. 袁峰译. UML 风格(第 2 版). 北京：人民邮电出版社, 2008. [10] Kenneth E.Kendall, Julie E.Kendall 著. 施平安, 郝清赋译. 系统分析与设计 (原书第 7 版). 北京：机械工业出版社, 2010.

[11] 邱郁惠. 系统分析师 UML 实务手册. 北京：机械工业出版社, 2008.

[12] 赵韶平, 徐茂生, 周勇华, 罗海燕. PowerDesigner 系统分析与建模(第 2 版). 北京：清华大学出版社, 2010.