

软 件 工 程 综 合 实 验

Blade框架分析

|  |  |
| --- | --- |
| 团队名称 | **B 组** |
| 指导教师 | **刘 超 任 健** |
| 培养学院 | **计 算 机 学 院** |

版本变更历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 变更时间 | 修改人 | 审核人 | 备注 |
| 1.0 | 2017/03/24 | 胡明昊 | 汪晓燕，穆鹏飞，刘晔 | 初稿 |
| 1.1 | 2017/04/01 | 汪晓燕 | 胡明昊，穆鹏飞，刘晔 | 二稿 |
| 1.2 | 2017/04/05 | 汪晓燕 | 胡明昊，穆鹏飞，刘晔 | 三稿 |
| 1.3 | 2017/04/05 | 穆鹏飞 | 胡明昊，穆鹏飞，刘晔 | 四稿 |
| 1.4 | 2017/04/12 | 穆鹏飞 | 胡明昊，汪晓燕，刘晔 | 五稿 |
| 1.5 | 2017/04/23 | 刘晔 | 胡明昊，汪晓燕，穆鹏飞 | 更改非功能性需求 |
| 2017/04/23 | 汪晓燕 | 胡明昊，穆鹏飞，刘晔 | 对刘超老师的批注进行修改 |
| 2017/04/23 | 穆鹏飞 | 胡明昊，汪晓燕，刘晔 | 对刘超老师的批注进行修改 |
| 1.6 | 2017/04/27 | 胡明昊 | 穆鹏飞，汪晓燕，刘晔 | 对照D组E组的批注进行修改 |
| 1.7 | 2017/05/03 | 汪晓燕 | 胡明昊，穆鹏飞，刘晔 | 对刘超老师的批注进行修改 |
| 2017/05/03 | 刘晔 | 汪晓燕, 胡明昊，穆鹏飞 | 对刘超老师的批注进行修改 |
| 2017/05/03 | 穆鹏飞 | 胡明昊，汪晓燕，刘晔 | 对刘超老师的批注进行修改 |

目 录

[1 引言 1](#_Toc481746411)

[1.1 目的 1](#_Toc481746412)

[1.2 文档约定 1](#_Toc481746413)

[1.3 术语和缩略语 1](#_Toc481746414)

[2 系统概述 4](#_Toc481746415)

[2.1 任务背景 4](#_Toc481746416)

[2.2 系统概述 5](#_Toc481746417)

[2.3 假设和约束 7](#_Toc481746418)

[2.4 运行环境 7](#_Toc481746419)

[3需求分析 7](#_Toc481746420)

[3.1 业务需求 8](#_Toc481746421)

[3.2 功能需求 8](#_Toc481746422)

[3.3 功能需求用例图建模 10](#_Toc481746423)

[3.4 RUCM 模型 11](#_Toc481746424)

[3.5 类图建模 21](#_Toc481746425)

[3.5.1 IOC类图建模 21](#_Toc481746426)

[3.5.2 配置类图建模 22](#_Toc481746427)

[3.5.2.1 配置RUCM与配置类图基本对应 23](#_Toc481746428)

[3.5.3 数据库类图建模 24](#_Toc481746429)

[3.5.3.1 数据库增加RUCM与数据库类图基本对应 25](#_Toc481746430)

[3.5.3.2 数据库删除RUCM与数据库类图基本对应 25](#_Toc481746431)

[3.5.3.3 数据库修改RUCM与数据库类图基本对应 26](#_Toc481746432)

[3.5.3.4 数据库查看RUCM与数据库类图基本对应 26](#_Toc481746433)

[3.5.4 请求响应类图建模 27](#_Toc481746434)

[3.5.4.1 请求响应RUCM与请求响应类图基本对应关系 27](#_Toc481746435)

[3.5.5 拦截请求类图建模 28](#_Toc481746436)

[3.6 非功能需求 28](#_Toc481746437)

[3.6.1 数据库操作中适当利用缓存 29](#_Toc481746438)

[3.6.2 MVC框架必须能够支持较大规模的并发请求 30](#_Toc481746439)

[3.6.3 ＭVC框架需要避免过度解耦 30](#_Toc481746440)

[3.6.4 ＭVC框架应当提供一定的灵活性 31](#_Toc481746441)

[3.7 输入和输出 32](#_Toc481746442)

[3.8 数据库特性 32](#_Toc481746443)

[3.9 故障处理 32](#_Toc481746444)

[3.10 安全和保密 34](#_Toc481746445)

[4 时序图 34](#_Toc481746446)

[4.1 IOC时序图 34](#_Toc481746447)

[4.2 配置时序图 35](#_Toc481746448)

[4.3 数据库增加时序图 36](#_Toc481746449)

[4.4 数据库删除时序图 37](#_Toc481746450)

[4.5 数据库查询时序图 37](#_Toc481746451)

[4.6 数据库修改时序图 38](#_Toc481746452)

[4.7 请求响应时序图 39](#_Toc481746453)

[4.8 拦截请求时序图 39](#_Toc481746454)

[5 状态图 40](#_Toc481746455)

[5.1数据库模块状态图 40](#_Toc481746456)

[5.2 Blade框架状态图 41](#_Toc481746457)

[6 改进方案设想 42](#_Toc481746458)

[6.1 需求分析 42](#_Toc481746459)

[6.1.1 需求描述 42](#_Toc481746460)

[6.1.2 RUCM建模 43](#_Toc481746461)

[6.1.3 需求目标 45](#_Toc481746462)

[6.2 可行性分析 45](#_Toc481746463)

[6.3 测试用例类图 46](#_Toc481746464)

软件需求分析说明书

# 1 引言

## 1.1 目的

本文档旨在对Blade框架进行总体分析后，得出的对Blade框架系统的需求说明。

## 1.2 文档约定

文档在编辑时，遵守IEEE发布的对软件需求说明书的文档约定，版本号为IEEE Std 802.11-1999。

## 1.3 术语和缩略语

在Blade框架中，涉及了多项Java语言和计算机网络的术语，较为核心的概念术语及其解释如下所示：

1）MVC：Model View Controller，即模型—视图—控制器的缩写，一种软件设计模式。使用业务逻辑、数据、界面分离的方法组织代码，将业务逻辑聚集于一个部件。在改进和个性化定制界面及用户交互的同时，不需要重新改写业务逻辑。

2）IOC：Inversion of control的缩写，意为控制反转，一种重要的面向对象编程的法则。它能指导我们如何设计出松耦合、更优良的程序。传统应用程序都是由开发人员在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试；有了IOC容器后，把创建和查找依赖对象的控制权交给了容器，由容器对组合对象进行“注入”。对象间互相不知道对方的存在，而统一由容器进行管理，有利于功能复用。更重要的是使得程序的整个体系结构变得非常灵活，耦合性低。将设计好的对象交给容器控制，而不是传统的在对象内部直接控制。

3）DI：Dependency Injection，即“依赖注入”： 组件之间的依赖关系由容器在运行期决定。形象地说，即由容器动态的将某个依赖关系注入到组件之中。依赖注入的目的并非为软件系统带来更多功能，而是为了提升组件的重用率，并为系统搭建一个灵活、可扩展的平台。通过依赖注入机制，开发人员只需要通过简单的配置，而无需任何代码就可指定目标需要的资源，完成自身的业务逻辑，而不需要关心具体的资源来自何处，由谁实现。

4）路由：在Blade中，路由是一个HTTP方法配对一个URL匹配模型，每一个路由可以应对一个处理方法。

5）拦截器：Blade中的拦截器用于接收请求时做额外操作，比如存储数据，校验数据，过滤请求等。

6）HTML：超文本标记语言。

7）RESTful架构：Representational State Transfer，一种软件架构风格，提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

8）服务器：一个管理资源并为用户提供服务的计算机软件，用于接收用户请求并响应相应的数据给用户。

9）客户端：客户端（Client），是指与服务器相对应，为客户提供本地服务的程序。一般安装在普通的用户机上，需要与服务端互相配合运行。

10）JSON：JSON(JavaScript Object Notation, JS 对象标记) 是一种轻量级的数据交换格式。它基于 ECMAScript 规范的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

11）JDBC: JDBC（Java Data Base Connectivity, java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序。

12）数据库：简单来说可视为电子化的文件柜—存储电子文件的处所，用户可以对文件中的数据运行新增、截取、更新、删除等操作。

13）Java：Java是一种广泛使用的计算机编程语言，拥有跨平台、面向对象、泛型编程的特性，广泛应用于企业级Web应用开发和移动应用开发。

14）Java web：是用Java技术来解决相关web互联网领域的技术总和。web包括：web服务器和web客户端两部分。

15）Tomcat：Tomcat是由Apache软件基金会下属的Jakarta项目开发的一个Servlet容器，按照Sun Microsystems提供的技术规范，实现了对Servlet和JavaServer Page（JSP）的支持，并提供了作为Web服务器的一些特有功能，如Tomcat管理和控制平台、安全域管理和Tomcat阀等。由于Tomcat本身也内含了一个HTTP服务器，它也可以被视作一个单独的Web服务器。

16）HTTP协议：HTTP是一个客户端终端（用户）和服务器端（网站）请求和应答的标准（TCP）。

17）GET：常用的HTTP请求方法，从指定的资源请求数据。GET请求可被缓存，并且保留在浏览器历史记录中，所以GET请求不应在处理敏感数据时使用。

18）POST：另一种常用的HTTP请求方法，不仅可以请求数据还可以向指定的资源提交要被处理的数据。POST请求不会被缓存，并且不会保留在浏览器历史纪录中。

19）Request: Request对象是从客户端向服务器发出请求，包括用户提交的信息以及客户端的一些信息。客户端可通过HTML表单或在网页地址后面提供参数的方法提交数据，然后通过request对象的相关方法来获取这些数据。

20）Response: Response对象用于动态响应客户端请求，控制发送给用户的信息，并将动态生成响应。

21) Maven项目对象模型(POM)，可以通过一小段描述信息来管理项目的构建，报告和文档的软件项目管理工具。

22) Restful: 一种软件架构风格，设计风格而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

23）模板引擎不属于特定技术领域，它是跨领域跨平台的概念。在[Asp](http://baike.baidu.com/item/Asp)下有模板引擎，在[PHP](http://baike.baidu.com/item/PHP)下也有模板引擎，在[C#](http://baike.baidu.com/item/C%23)下也有，甚至[JavaScript](http://baike.baidu.com/item/JavaScript)、[WinForm](http://baike.baidu.com/item/WinForm)开发都会用到模板引擎技术。模板引擎的实现方式有很多，最简单的是“置换型”模板引擎，这[类模板](http://baike.baidu.com/item/%E7%B1%BB%E6%A8%A1%E6%9D%BF)引擎只是将指定模板内容（字符串）中的特定标记（子字符串）替换一下便生成了最终需要的业务数据（比如网页）。置换型模板引擎实现简单，但其效率低下，无法满足高负载的应用需求（比如有海量访问的网站），因此还出现了“解释型”模板引擎和“编译型”模板引擎等。

24) 缓存插件。插件(Plug-in,又称addin、add-in、addon或add-on,又译外挂)是一种遵循一定规范的应用程序接口编写出来的程序。缓存插件主要是对用户经常访问的页面或查询的数据预先缓存在内存中，能够加速页面响应时间，节省服务器资源请求。

25) 负载均衡。英文名称为Load Balance，其意思就是分摊到多个操作单元上进行执行，例如Web[服务器](http://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)、[FTP服务器](http://baike.baidu.com/item/FTP%E6%9C%8D%E5%8A%A1%E5%99%A8)、[企业](http://baike.baidu.com/item/%E4%BC%81%E4%B8%9A)关键应用服务器和其它关键任务服务器等，从而共同完成工作任务，以增加[吞吐量](http://baike.baidu.com/item/%E5%90%9E%E5%90%90%E9%87%8F)、加强网络数据处理能力、提高网络的灵活性和可用性。

# 2 系统概述

## 2.1 任务背景

90年代末期，JavaWeb技术开始应用于服务器、网站开发。Sun公司制定了J2EE标准，借助编程语言Java快速的发展并流行起来。J2EE的广泛实现是在1999年和2000年开始的，它的出现带来了诸如事务管理之类的核心中间层概念的标准化，但是在实践中并没有获得绝对的成功。因为J2EE的开发效率，开发难度和实际的性能都令人失望。而Spring框架出现的初衷就是为了解决类似的这些问题，提供了一整套的JavaWeb支持，包括安全、事务、数据库等操作的简化。但是Spring是企业级的，所以更关注于市场需求，扩展了越来越多的功能，衍生出许多其他的分支项目，导致框架越来越大，也越来越杂，甚至演化出Spring Boot用于简化Spring自身的配置。因此我组没有选择Spring如此庞大、复杂的框架，因为在短期内分析Spring是非常困难和不现实的。

Blade是一款简洁易用的JavaWeb框架，它抽取了Spring的核心功能并重新实现。Blade在简洁和兼容两者之间选择了简洁，摒弃了繁复的配置，选择了Java 1.8，以及内嵌的服务器和数据库。它提供了IOC容器、MVC架构支持、模板引擎以及注解功能，并基于Maven进行管理。

它的主要特点如下：

* + 基本零配置。
  + 轻量级,不依赖于更多的库，摆脱SSH的臃肿，模块化设计，使用起来更轻便。
  + Restful风格的路由接口。
  + 单jar运行，易于部署。

因此，本次实验选择Blade框架作为分析的目标，并撰写此需求说明书。

## 2.2 系统概述

Blade是一个简洁强大的web框架，它内置了IOC管理，拦截器配置，REST API开发等众多主流web特性，集成了模板引擎，缓存插件，数据库操作，邮件发送，HTTP请求等常用功能。

**1）框架的整体设计**

Blade框架是一种基于MVC设计的框架，它基于Blade-core模块为核心进行构建，是一个高度解耦的框架。

Blade框架在设计之初就考虑了模块化使用，如图1[[1]](#footnote-1)所示。为了达到这一目的，故基于独立的组件进行开发，使得开发出的组件不依赖于Blade**。**

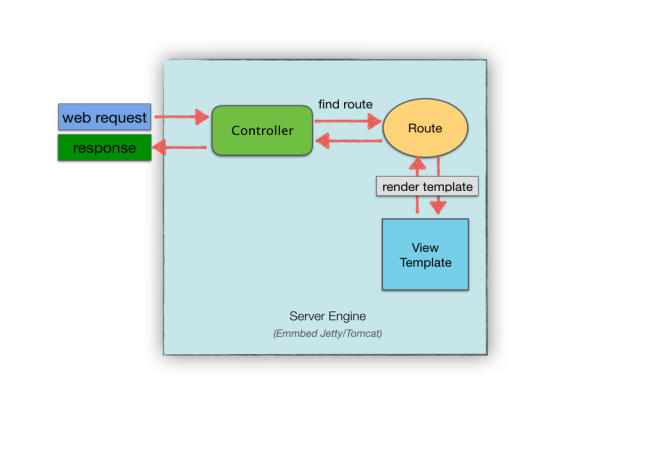


图 1 Blade MVC图示

**2）执行逻辑**

Blade的执行逻辑如图2所示，首先进行文件的配置，配置好的文件经过配置路由进行转发，在参数过滤后，到达控制器，之后加载辅助的工具包，并通过service部分与数据库进行交互。

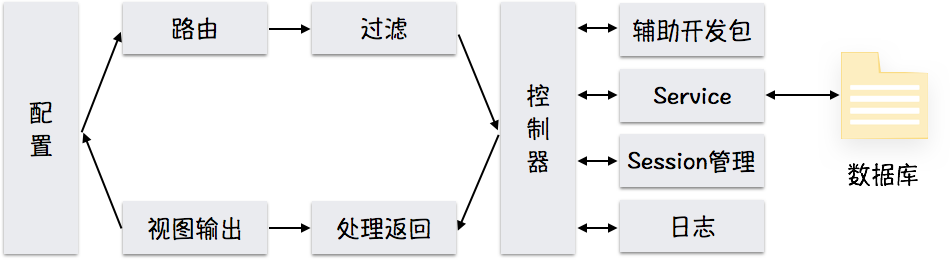


图 2 Blade 系统架构

**3）项目结构**

Blade整个的项目结构如图3所示。在整个结构中，有两大部分，分别为JAVA源码与resources资源。在JAVA源码中，各部分参数的意义如表1所示。

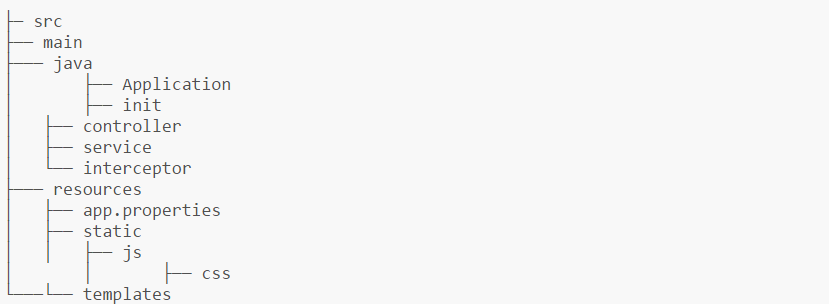


图 3 Blade工程资源结构图

表 1 Blade工程资源结构说明

|  |  |
| --- | --- |
| 路径 | 说明 |
| Application | 启动程序 |
| init | web初始化操作，比如数据库配置，模板引擎配置 |
| controller | 控制器和路由的存储位置 |
| service | 服务接口和实现 (非必需部分) |
| interceptor | 路由拦截器所在包 |

而在resources资源部分各部分参数的意义如表2所示。

表 2 Blade Resources文件资源结构说明

|  |  |
| --- | --- |
| 路径 | 说明 |
| app.properties | 程序主配置文件(非必需部分) |
| static | 静态资源存放文件夹 |
| templates | 模板文件存放文件夹 |

2.3 假设和约束

为了保证Blade框架的正常运行或发布，对其运行环境和过程做如下假设和约束。

1）在程序运行时，限制同时访问量，以保证程序安全稳定运行。当超出同时访问的限制时，进行阻塞等待，当同时访问量达到要求时，使阻塞的操作获取访问权限。

2）在程序运行的过程中，如遇突发事件时，如断电、断网，要保证操作的一致性。遇到突发事件时，对于涉及的数据操作，要保证能够自动回滚。

## 2.4 运行环境

需安装1.8版本及以上的JDK，同时需对Maven进行如下安装配置：

|  |
| --- |
| <dependencies>  <dependency>  <groupId>com.Bladejava</groupId>  <artifactId>Blade-core</artifactId>  <version>1.7.2-beta</version>  </dependency>  <dependency>  <groupId>com.Bladejava</groupId>  <artifactId>Blade-embed-jetty</artifactId>  <version>0.1.3</version>  </dependency>  </dependencies> |

# 3需求分析

软件需求包括3个不同的层次――业务需求、功能需求，非功能需求和系统需求。

业务需求（Business requirement）表示组织或客户高层次的目标。业务需求通常来自项目投资人、购买产品的客户、实际用户的管理者、市场营销部门或产品策划部门。业务需求描述了组织为什么要开发一个系统，即组织希望达到的目标。使用前景和范围（vision and scope）文档来记录业务需求，这份文档有时也被称作项目轮廓图或市场需求（project charter 或 market requirement）文档。

功能需求（functional requirement）规定开发人员必须在产品中实现的软件功能，用户利用这些功能来完成任务，满足业务需求。功能需求有时也被称作行为需求（behavoral requirement），因为习惯上总是用“应该”对其进行描述：“系统应该发送电子邮件来通知用户已接受其预定”。功能需求描述是开发人员需要实现什么。

系统需求（system requirement）用于描述系统软硬件的运行时需求。

## 3.1 业务需求

Blade旨在提供轻便简洁的JavaWeb开发，从开发人员角度，Blade框架应满足用户如下业务需求：

1） 开发人员能够在默认配置环境下快速开发，而无需手动配置众多配置项。

2）开发人员能够在必要时刻快速修改和配置选项。

3）开发人员能够通过框架访问数据库摆脱JDBC的繁复操作。

4）开发人员能够通过框架构建网络请求响应的处理逻辑。

5）开发人员能够选择性的过滤不安全的请求。

从浏览器用户的角度看，Blade应满足用户如下基本的业务需求：

1）Blade应接受浏览器发送的http请求，并经过业务逻辑处理后向浏览器返回预期的响应结果。

2）浏览器应接收到符合Http响应规范的响应数据。

## 3.2 功能需求

**1）****IOC****功能**

IOC容器是Blade的核心功能模块。IOC 不仅是一种技术，更是一种思想，一个重要的面向对象编程的法则，它能指导我们如何设计出松耦合、更优良的程序。传统应用程序都是由我们在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试；有了IOC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是 松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活。

IOC很好的体现了面向对象设计法则中的好莱坞法则：“别找我们，我们找你”。即由IOC容器帮对象找相应的依赖对象并注入，而不是由对象主动去找。通过IOC部分的实现，实现依赖注入功能，简化代码的编写，在编写程序时，能更好的运用面向对象的法则。

**2）****配置管理****功能**

Blade框架的一切功能实现的前提就是配置，任何功能在实现之前都需要加载配置文件，配置文件使得Blade框架在运行时自动加载写好的配置文件，通过配置文件的加载，Blade系统可以完成以下功能：

a）获取404，500界面。

b）获取默认字符编码。

c）设置开发者模式、

d）获取静态文件资源

e）获取Port并对Port进行监听

f）将获取到的静态资源添加到静态资源文件夹

**3）****数据库增加功能**

作为一款轻量级的框架，Blade框架，通过数据操作代码的封装，简化用户操作数据库的难度，在数据库增加模块中，用户可以使用一条语句，调用封装好的增加数据操作功能，在程序运行过程中，后台代码会自动进行数据库连接，SQL语句建立，并对SQL语句执行的过程。

**4）****数据库删除功能**

在数据库删除模块中，提供多种删除方式，可以按主键删除，也可以按特定字段删除，还可以删除所有数据，在使用任一操作时，用户只需使用一条语句，调用封装好的增加数据操作功能，在程序运行过程中，后台代码会自动进行数据库连接，SQL语句建立，并对SQL语句执行的过程。

**5）****数据库修改功能**

在数据库修改模块中，提供多种修改方式，可以修改某项数据，也可以修改所有数据，在使用任一操作时，用户只需使用一条语句，调用封装好的增加数据操作功能。在程序运行过程中，后台代码会自动进行数据库连接、SQL语句建立，并执行SQL语句。

**6）****数据库查看功能**

在数据库查看模块中，提供多种查看方式，可以按主键查看，也可以查看按特定列进行查询，查询结果可以单条显示，也可分页显示，同时还支持查询所有数据操作，在使用任一操作时，用户只需使用一条语句，调用封装好的增加数据操作功能，在程序运行过程中，后台代码会自动进行数据库连接，SQL语句建立，并对SQL语句执行的过程。

**7）****请求响应功能**

在浏览器向系统发送http请求，系统在接收到浏览器的请求后，解析该请求并寻找其匹配的路由，找到路由后，执行对应的操作并返回结果，结果经http响应报文封装后,发送该报文数据至浏览器,浏览器接收到系统的响应的数据后，加载数据,更新页面内容。

**8）****拦截请求功能**

拦截请求用于拦截部分不符合要求的请求。在请求被处理之前，对请求进行过滤，从而达到增强安全性、减少代码冗余的目的，方便开发人员编写的功能。

## 3.3 功能需求用例图建模

1）参与者分析

对于Blade框架来说，核心的用户首先是开发人员，因为框架本身的特点和样式只有开发人员才能看见，而对浏览网页的用户透明。另外，对于JavaWeb框架，浏览器或其他类浏览器的组件（比如能够发送请求的终端），则是另外一个参与者，用于交互发送请求和接受响应。

2）用例图绘制

根据参与者、功能分析，绘制用例图如图4所示。

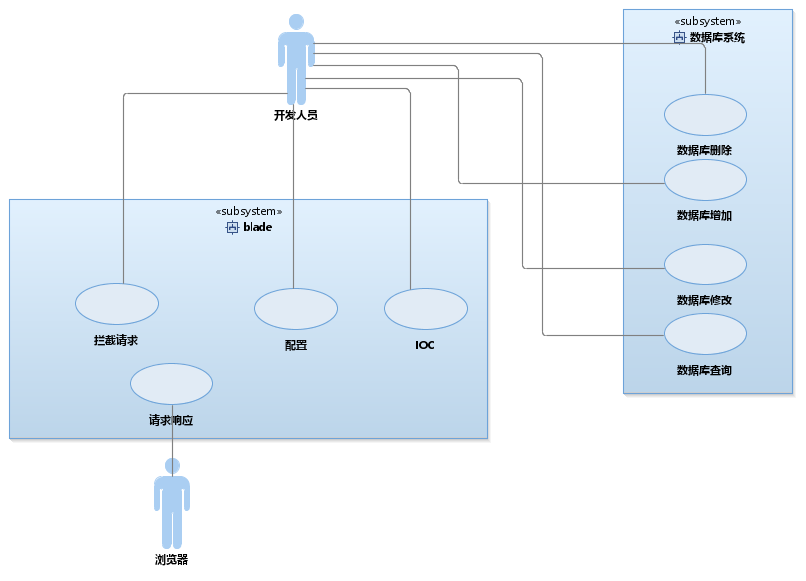


图 4 用例图

## 3.4 RUCM 模型

RUCM 即限制性用例建模。它的目标是：

1）使 UCMs 更加可理解并且更精确。

2）从 UCMs 自动生成分析模型。

RUCM 由以下两部分组成：

1）一个用于系统组织 UCSs 的用例模板。

2）限制用户写 UCSs 的一系列规则。

在对blade框架分析中，一共涉及到了8个用例，分别是IOC用例、配置用例、数据库增加用例、数据库删除用例、数据库修改用例、数据库查询用例、请求响应用例、拦截请求用例。

通过 RUCM 模型能够对用例进行规范的描述，接下来将使用 RUCM 模型描述上述用例。

1) IOC用例

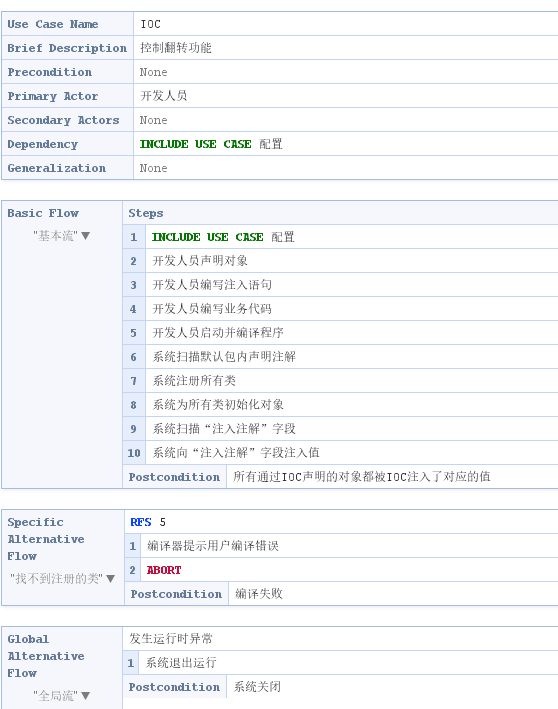
IOC是Blade提供的核心功能，其存在是有必要价值的。在Java Web开发盛行的当下，许多开源公司和项目都推出了专用的开发工具包以节省编程人员的开发时间，提高开发效率。但是对于繁复的工具类，如何有效的管理和控制各个对象成为一个挑战。在服务器上，内存和空间都极为有限，因此IOC可以作为一个有效的解决方案。在Blade中，所有的对象都可以交由IOC容器统一管理。用户只需要在一处声明，在使用的地方添加@Inject注解，系统自动将对应的对象从IOC容器中取出，并注入@Inject声明的字段索引中，而开发人员完全无需考虑何时新建资源和释放资源。建立RUCM模型如图 5所示。

图 5 IOC用例RUCM模型

2）配置



图 6 配置用例RUCM模型

3）数据库增加



图 7 数据库增加RUCM

4）数据库删除



图 8 数据库删除用例RUCM模型

5）数据库查询



图 9数据库查询用例RUCM模型

6）数据库修改



图 10 数据库修改用例RUCM 模型

1. 请求响应

请求响应作为web的两个最基本的抽象概念.因此其应当是MVC框架在内的任何web框架都必须满足的业务需求。



图 11 请求响应用例RUCM模型 a



图 12 请求响应用例RUCM模型

8）拦截请求

拦截请求是Web开发必不可少的功能，需要其提供对某些路由索引的拦截，用于对不同权限做功能划分、限制IP登录等等操作。建立拦截请求用例RUCM如图 13所示。



图 13 拦截请求用例RUCM模型

## 3.5 类图建模

### 3.5.1 IOC类图建模

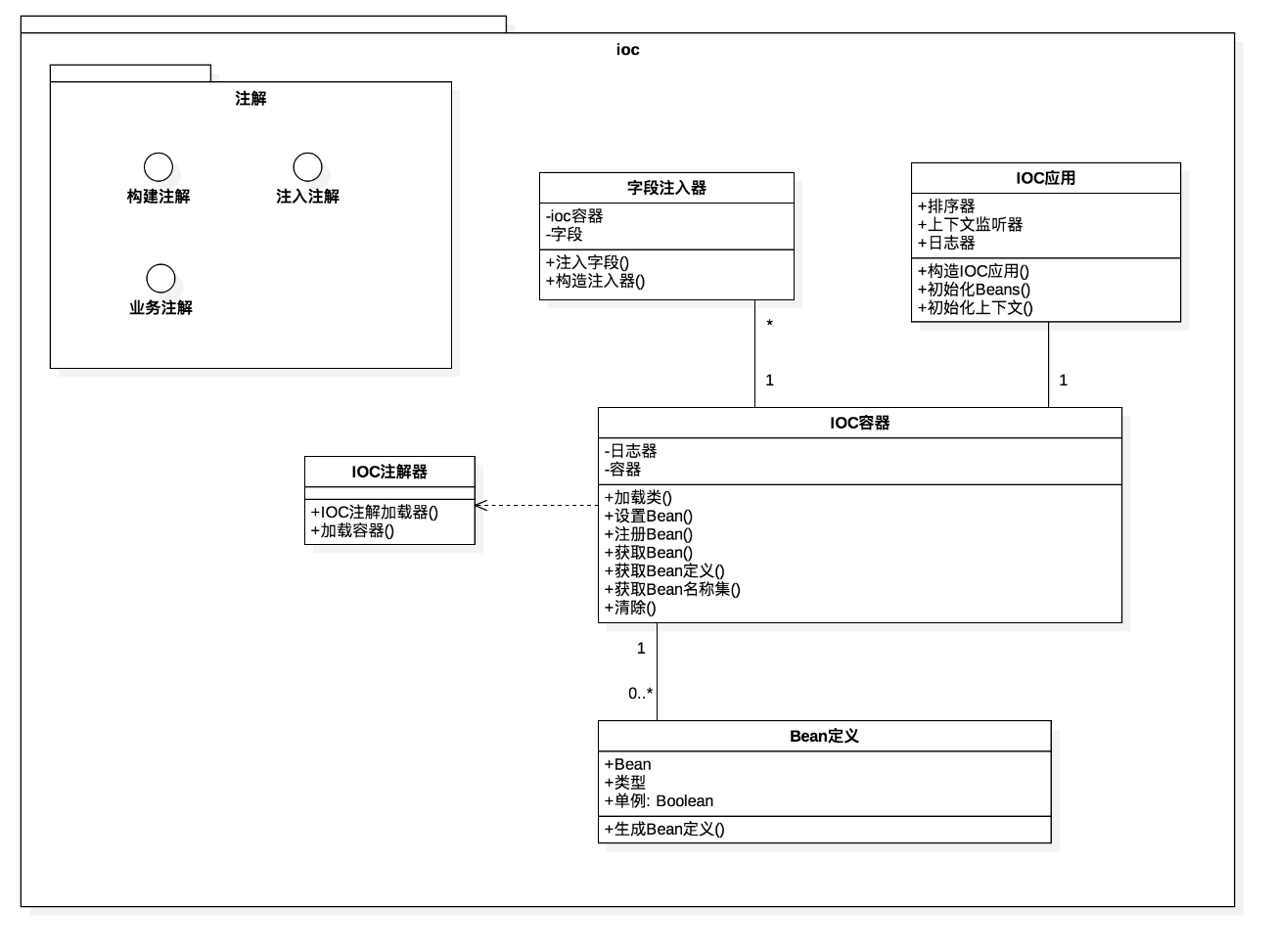


图 14 IOC用例模块类图

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类图名称 | RUCM名称 | 基本事件流 | 对应类图中的类 | 属性/方法 | 描述 |
| IOC | IOC | Step6 | IOC容器 | 获取Bean() | 从包中扫描所有的类 |
| IOC | IOC | Step7 | IOC容器 | 注册Bean() | 在容器中记录所有的类 |
| IOC | IOC | Step8 | IOC应用 | 初始化Beans() | 为所有记录的类生成对象 |
| IOC | IOC | Step9 | 注入注解 |  | 标记对象需要IOC容器注入值 |
| IOC | IOC | Step10 | 字段注入器 | 注入字段 | 向注入标记字段注入对象值 |

### 3.5.2 配置类图建模

配置类图中，主要包含“开发者模式类”、“基本配置”、“出错界面设定”和“配置”四个类，其中开发者模式类负责配置的初始化任务，基本配置类负责程序所需的基本配置，出错界面类负责接收用户对出错界面的配置，开发人员可以通过配置类获取配置信息。

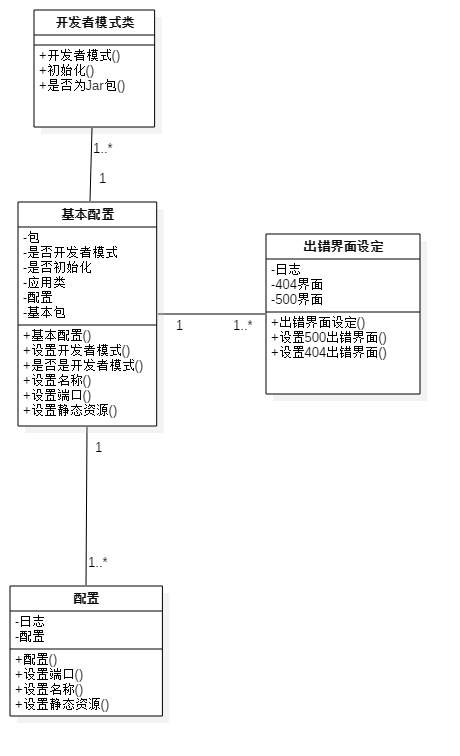


图 15 配置用例模块类图

### 3.5.2.1 配置RUCM与配置类图基本对应

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类图名称 | RUCM名称 | 基本事件流 | 对应类图中的类 | 属性/方法 | 描述 |
| 配置 | 配置 | Step1 | 基本配置类 | 设置端口 | 配置web服务器端口 |
| 配置 | 配置 | Step2 | 基本配置类 | 设置名称 | 配置当前应用别名 |
| 配置 | 配置 | Step4 | 基本配置类 | 设置开发者模式 | 是否设置开发者模式 |
| 配置 | 配置 | Step5 | 出错界面类 | 设置404出错界面 | 出现404统一界面 |
| 配置 | 配置 | Step6 | 出错界面类 | 设置500出错界面 | 出现500统一界面 |
| 配置 | 配置 | Step7 | 配置类 | 配置 | 获取配置信息 |
| 配置 | 配置 | Step8 | 响应包装 | 响应 | 响应浏览器 |

表3.1 配置RUCM与配置类图的基本对应关系及简要描述

### 3.5.3 数据库类图建模

数据库类图中，一共有4个类，其中“数据操作类”负责外面访问数据操作的接口，当有其他类调用数据操作类时数据操作类将调用数据具体操作类执行数据操作。

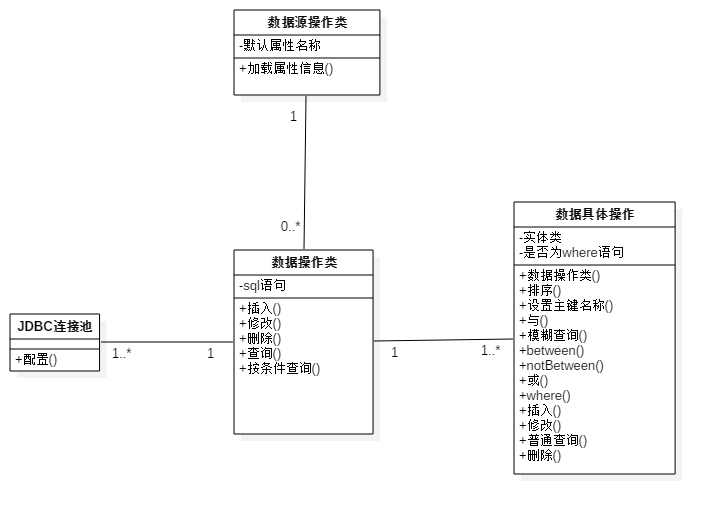


图 16 数据库模块类图

### 3.5.3.1 数据库增加RUCM与数据库类图基本对应

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类图名称 | RUCM名称 | 基本事件流 | 对应类图中的类 | 属性/方法 | 描述 |
| 数据库类图 | 数据库增加 | Step2 | 数据源操作 | 默认属性名称/加载属性信息（） | 加载数据库的属性信息 |
| 数据库类图 | 数据库增加 | Step2 | JDBC连接池 | 配置（） | 对加载出的属性信息进行配置 |
| 数据库类图 | 数据库增加 | Step4 | 数据操作类 | 构造方法 | 开发人员新建数据操作类 |
| 数据库类图 | 数据库增加 | Step5 | 数据操作类，数据具体操作类 | 数据操作类：Sql语句/插入（）  数据具体操作类：实体类/插入（） | 生成sql语句，之后在插入方法中调用数据具体操作类的插入操作 |

表3.2 数据库增加RUCM与数据操作类图的基本对应关系及简要描述

### 3.5.3.2 数据库删除RUCM与数据库类图基本对应

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类图名称 | RUCM名称 | 基本事件流 | 对应类图中的类 | 属性/方法 | 描述 |
| 数据库类图 | 数据库删除 | Step2 | 数据源操作 | 默认属性名称/加载属性信息（） | 加载数据库的属性信息 |
| 数据库类图 | 数据库删除 | Step2 | JDBC连接池 | 配置（） | 对加载出的属性信息进行配置 |
| 数据库类图 | 数据库删除 | Step4 | 数据操作类 | 构造方法 | 开发人员新建数据操作类 |
| 数据库类图 | 数据库删除 | Step5 | 数据操作类，数据具体操作类 | 数据操作类：Sql语句/删除（）  数据具体操作类：实体类/删除（） | 生成sql语句，之后在删除方法中调用数据具体操作类的删除操作 |

表3.3 数据库删除RUCM与数据操作类图的基本对应关系及简要描述

### 3.5.3.3 数据库修改RUCM与数据库类图基本对应

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类图名称 | RUCM名称 | 基本事件流 | 对应类图中的类 | 属性/方法 | 描述 |
| 数据库类图 | 数据库修改 | Step2 | 数据源操作 | 默认属性名称/加载属性信息（） | 加载数据库的属性信息 |
| 数据库类图 | 数据库修改 | Step2 | JDBC连接池 | 配置（） | 对加载出的属性信息进行配置 |
| 数据库类图 | 数据库修改 | Step4 | 数据操作类 | 构造方法 | 开发人员新建数据操作类 |
| 数据库类图 | 数据库修改 | Step5 | 数据操作类，数据具体操作类 | 数据操作类：Sql语句/修改（）  数据具体操作类：实体类/修改（） | 生成sql语句，之后在修改方法中调用数据具体操作类的修改操作 |

表3.4 数据库修改RUCM与数据操作类图的基本对应关系及简要描述

### 3.5.3.4 数据库查看RUCM与数据库类图基本对应

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类图名称 | RUCM名称 | 基本事件流 | 对应类图中的类 | 属性/方法 | 描述 |
| 数据库类图 | 数据库查看 | Step2 | 数据源操作 | 默认属性名称/加载属性信息（） | 加载数据库的属性信息 |
| 数据库类图 | 数据库查看 | Step2 | JDBC连接池 | 配置（） | 对加载出的属性信息进行配置 |
| 数据库类图 | 数据库查看 | Step4 | 数据操作类 | 构造方法 | 开发人员新建数据操作类 |
| 数据库类图 | 数据库查看 | Step5 | 数据操作类，数据具体操作类 | 数据操作类：Sql语句/查看（）  数据具体操作类：实体类/查看（） | 生成sql语句，之后在查看方法中调用数据具体操作类的具体查看操作 |

表3.5 数据库查看RUCM与数据操作类图的基本对应关系及简要描述

### 3.5.4 请求响应类图建模

基本类图介绍.

http请求解析器: 解析从浏览器发送给系统的http请求.

转发器: 根据请求url找到对应的物理资源的视图.

视图: 物理资源的逻辑表示.

控制器: 根据请求url及其参数找到其对应的逻辑处理模型.

模型: 业务处理逻辑的一簇操作集合.

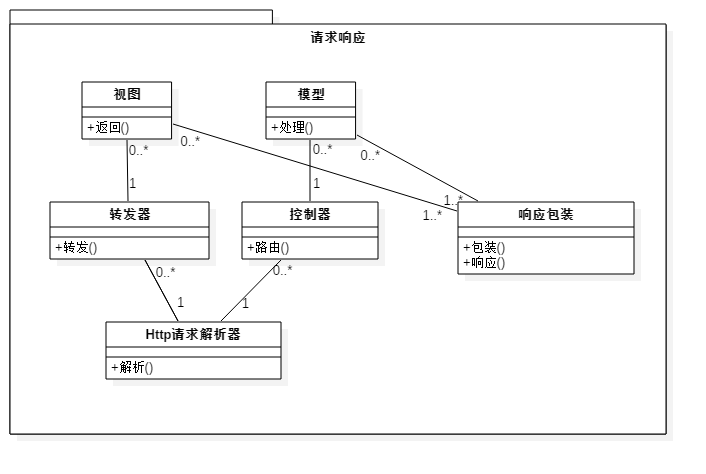


图 17 请求响应模块类图

### 3.5.4.1 请求响应RUCM与请求响应类图基本对应关系

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类图名称 | RUCM名称 | 基本事件流 | 对应类图中的类 | 属性/方法 | 描述 |
| 请求响应 | 请求响应 | Step2 | http解析器 | 解析 | 解析http请求 |
| 请求响应 | 请求响应 | Step3 | 转发器 | 转发 | 根据url寻找资源视图 |
| 请求响应 | 请求响应 | Step4 | 控制器 | 路由 | 根据url及参数找到模型 |
| 请求响应 | 请求响应 | Step5 | 视图 | 返回 | 返回资源内容的数字表示 |
| 请求响应 | 请求响应 | Step6 | 模型 | 处理 | 业务逻辑处理 |
| 请求响应 | 请求响应 | Step7 | 响应包装 | 包装 | 包装资源内容及业务处理结果 |
| 请求响应 | 请求响应 | Step8 | 响应包装 | 响应 | 响应浏览器 |

表3.5 用例与类图的基本对应关系及简要描述

### 3.5.5 拦截请求类图建模

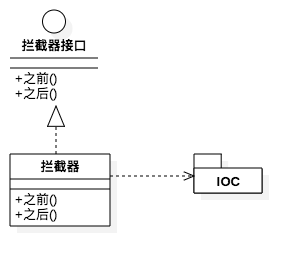


图 18 拦截请求模块类图

## 3.6 非功能需求

对于一个系统而言，功能性需求必须得到满足。不同的系统，根据系统的种类及其应用领域，有着不同的非功能性需求，对非功能性需求满足的程度也有所差别。对于一个应用于网站开发的MVC框架而言，框架的设计实现一开始就要考虑框架使用者以及网站用户的基本需求。目前总结为以下展示的四个主要的非功能性需求。

### 3.6.1 数据库操作中适当利用缓存

在数据库操作中适当的利用缓存，能够有效的降低数据库访问开销。主要的关注点在于以下三个方面。

性能:将相应数据存储起来以避免数据的重复创建、处理和传输，可有效提高性能。比如将不改变的数据缓存起来，例如国家列表等，这样能明显提高web程序的反应速度；

稳定性:同一个应用中，对同一数据、逻辑功能和用户界面的多次请求时经常发生的。当用户基数很大时，如果每次请求都进行处理，消耗的资源是很大的浪费，也同时造成系统的不稳定。例如，web应用中，对一些静态页面的呈现内容进行缓存能有效的节省资源，提高稳定性。而缓存数据也能降低对[数据库](http://lib.csdn.net/base/mysql)的访问次数，降低数据库的负担和提高数据库的服务能力；

可用性:有时，提供数据信息的服务可能会意外停止，如果使用了缓存技术，可以在一定时间内仍正常提供对最终用户的支持，提高了系统的可用性。

数据库缓存技术的一般场景：



图 19 数据库缓存的一般场景

数据库缓存技术的一般场景生成：



图 20数据库缓存技术的一般场景生成

### 3.6.2 MVC框架必须能够支持较大规模的并发请求

在网站开发过程中，随着应用市场的扩大，用户访问量的增加，有必要在网站的开始设计阶段，考虑使用具有能处理高并发特性的框架进行后续的开发实现。

主要的关注点是框架能否提供较为多的轻量级线程以及页面缓存等技术进行用户请求的处理，以支持大规模的并发请求。

大规模并发请求的一般场景：

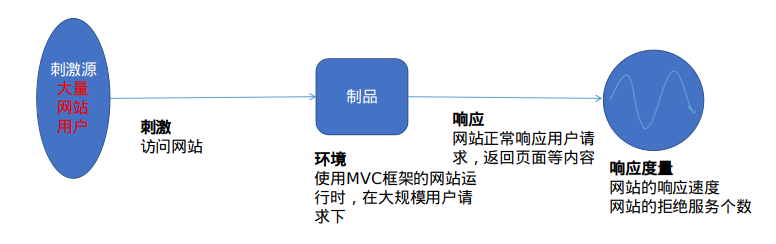


图 21大规模并发请求的一般场景

大规模并发请求的一般场景生成：



图 22 大规模并发请求的一般场景生成

### 3.6.3 ＭVC框架需要避免过度解耦

MVC框架的主要特征是提供了模型,视图，控制等的分离，这为网站系统的开发和维护提供了积极的帮助。但是，如果MVC框架过分解耦，引入过多的中间抽象层，那么就反而会降低MVC框架的执行效率。

避免过度解耦的主要的关注点是MVC框架中间抽象层数的多少，这决定了模型，视图，控制等的相互协作的程度。

避免过度解耦的一般场景：

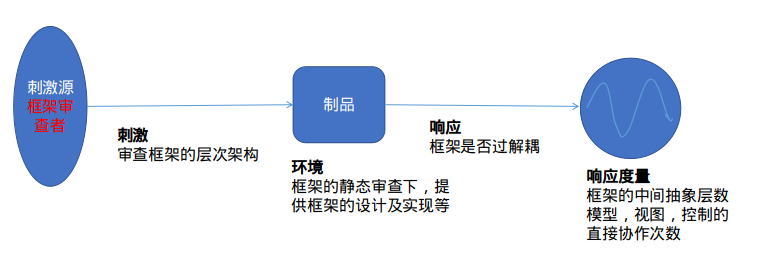


图 23避免过度解耦的一般场景

避免过度解耦的一般场景生成：

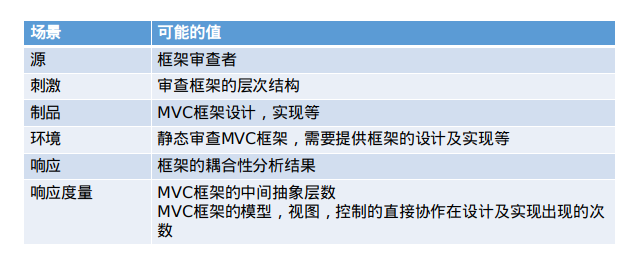


图 24避免过度解耦的一般场景生成

### 3.6.4 ＭVC框架应当提供一定的灵活性

MVC框架需要为使用者，即开发人员提供比较大的灵活性。主要的关注点是MVC应当为用户提供丰富的配置接口以及扩展接口等，方便开发人员的使用和对框架的二次开发，以有利于发挥开发人员的创造力，扩大框架的使用价值等。

框架的灵活性的一般场景：

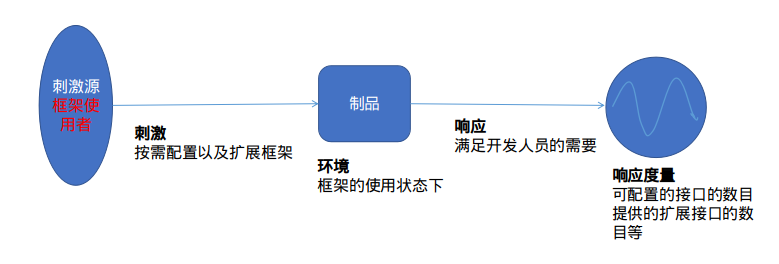


图 25 框架灵活性的一般场景

框架的灵活性的一般场景生成：



图 26 框架灵活性的一般场景生成

## 3.7 输入和输出

对于浏览器来说，输入是http请求，输出是http响应。

对于开发人员来说，输入是代码，输出是可运行的二进制jar文件。

## 3.8 数据库特性

Bladed框架的数据库有以下特性：

1）语法简介，代码量少,用极少的代码量就能实现相应的数据访问功能。

2）除了日志接口，不依赖第三方框架。

3）DSL风格，在程序编写过程中，类似于一种链式风格。

4）内置连接池，支持与其他连接池共用。

## 3.9 故障处理

在浏览器执行请求操作时，如若出现以下错误，则调用框架内置的400错误界面，并显示，同时提示出错信息。

|  |  |
| --- | --- |
| |  | | --- | | HTTP 400 - 请求无效  HTTP 401.1 - 未授权：登录失败  HTTP 401.2 - 未授权：服务器配置问题导致登录失败  HTTP 401.3 - ACL 禁止访问资源  HTTP 401.4 - 未授权：授权被筛选器拒绝  HTTP 401.5 - 未授权：ISAPI 或 CGI 授权失败  HTTP 403 - 禁止访问  HTTP 403 - 对 Internet 服务管理器 (HTML) 的访问仅限于 Localhost  HTTP 403.1 禁止访问：禁止可执行访问  HTTP 403.2 - 禁止访问：禁止读访问  HTTP 403.3 - 禁止访问：禁止写访问  HTTP 403.4 - 禁止访问：要求 SSL  HTTP 403.5 - 禁止访问：要求 SSL 128  HTTP 403.6 - 禁止访问：IP 地址被拒绝  HTTP 403.7 - 禁止访问：要求客户证书  HTTP 403.8 - 禁止访问：禁止站点访问  HTTP 403.9 - 禁止访问：连接的用户过多  HTTP 403.10 - 禁止访问：配置无效  HTTP 403.11 - 禁止访问：密码更改  HTTP 403.12 - 禁止访问：映射器拒绝访问  HTTP 403.13 - 禁止访问：客户证书已被吊销  HTTP 403.15 - 禁止访问：客户访问许可过多  HTTP 403.16 - 禁止访问：客户证书不可信或者无效  HTTP 403.17 - 禁止访问：客户证书已经到期或者尚未生效  HTTP 404.1 - 无法找到 Web 站点  HTTP 404 - 无法找到文件  HTTP 405 - 资源被禁止  HTTP 406 - 无法接受  HTTP 407 - 要求代理身份验证  HTTP 410 - 永远不可用  HTTP 412 - 先决条件失败  HTTP 414 - 请求 - URI 太长 | |

在浏览器执行请求操作时，如若出现以下错误，则调用框架内置的500错误界面，并显示，同时提示出错信息。

|  |
| --- |
| HTTP 500.100 - 内部服务器错误 - ASP 错误  HTTP 500-11 服务器关闭  HTTP 500-12 应用程序重新启动  HTTP 500-13 - 服务器太忙  HTTP 500-14 - 应用程序无效  HTTP 500-15 - 不允许请求 global.asa |

在浏览器执行请求操作时，如若出现以下错误，则调用框架内置的501错误界面，并显示，同时提示出错信息。

|  |
| --- |
| Error 501 - 未实现 |

在浏览器执行请求操作时，如若出现以下错误，则调用框架内置的502错误界面，并显示，同时提示出错信息。

|  |
| --- |
| HTTP 502 - 网关错误 |

## 3.10 安全和保密

1）在对数据库进行操作的时候，如若遇到突发的意外情况，如网络通信故障，突然断电等情况，要保证对数据库操作的ACID属性。

2）对于数据库的各种操作而言，要防止非法用户进行SQL注入，危害内部数据。

3）加密数据库中的敏感数据。

4）尽量避免使用不成熟的第三方库。

5）应该正确处理所有可能的非法操作。

6）应该考虑使用加密算法保护用户cookie和session等信息。

# 4 时序图

## 4.1 IOC时序图

根据IOC用例的RUCM描述，对开发人员使用IOC功能的工作流程如下描述：

1）配置

2）开发人员对声明交由IOC容器管理的对象。

3）开发人员需要编写注入的字段

4）开发人员编写其他业务代码

5）开发人员编译并启动程序

6）IOC应用扫描声明的注解

7）IOC应用对象扫描配置目录下的所有JavaBean类，并在存储在系统中

8）IOC应用为所有记录的类初始化对象

9）IOC应用扫描注入注解字段

10）IOC应用向所有需要注入的字段注入对象

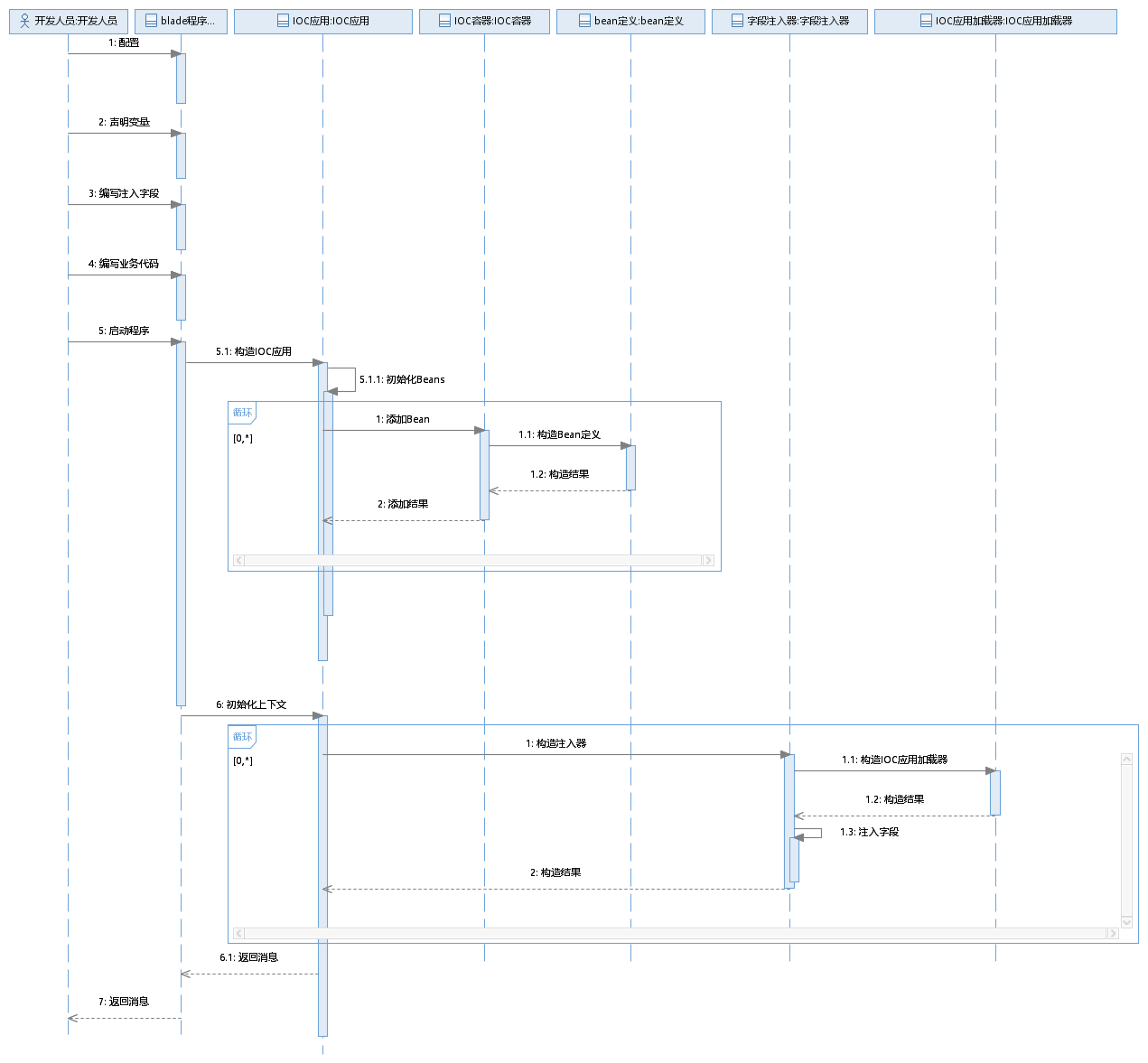


图 27 IOC用例时序图

## 4.2 配置时序图

在配置过程中，首先配置端口、应用信息别名等基本信息，之后设置开发者模式，设置完成之后对静态资源与出错界面进行设置，最后进行最终的配置过程，从而完成整个配置。

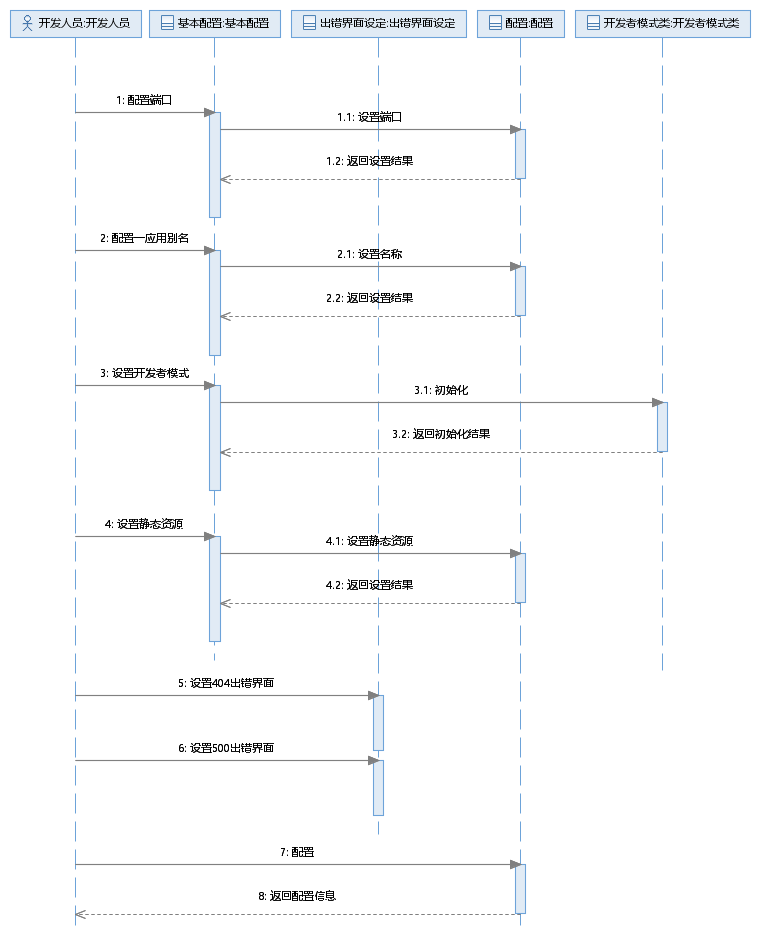


图 28 配置用例时序图

## 4.3 数据库增加时序图

在数据库增加过程中，首先对数据库的属性信息进行加载，之后对加载出的信息进行配置，在一切都完成之后进行插入操作。

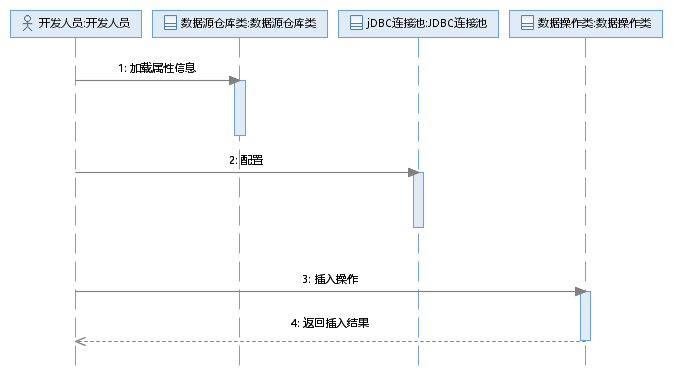


图 29 数据库增加用例时序图

## 4.4 数据库删除时序图

在数据库增加过程中，首先对数据库的属性信息进行加载，之后对加载出的信息进行配置，在一切都完成之后进行删除操作。

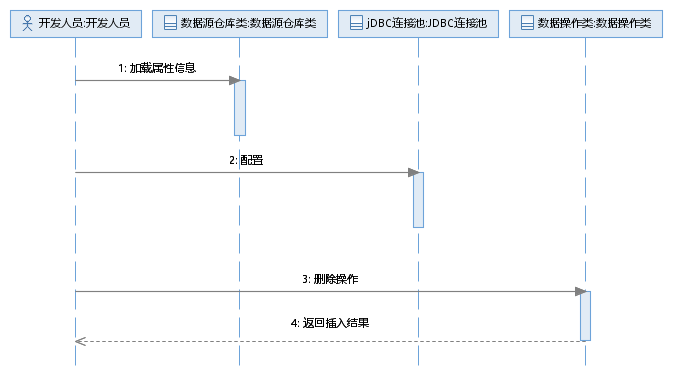


图 30 数据库删除用例时序图

## 4.5 数据库查询时序图

在数据库增加过程中，首先对数据库的属性信息进行加载，之后对加载出的信息进行配置，在一切都完成之后进行相应的查询操作操作。

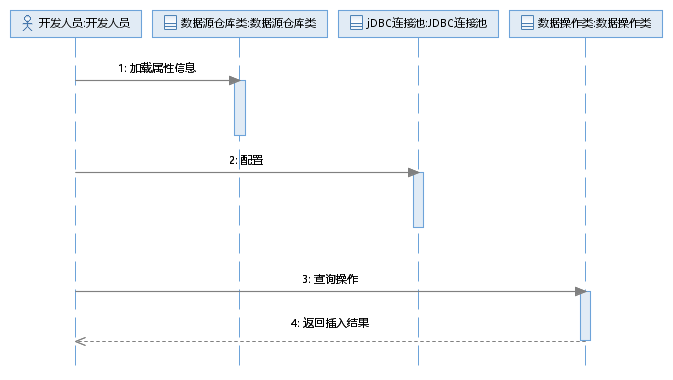


图 31 数据库查询用例时序图

## 4.6 数据库修改时序图

在数据库增加过程中，首先对数据库的属性信息进行加载，之后对加载出的信息进行配置，在一切都完成之后进行修改操作。

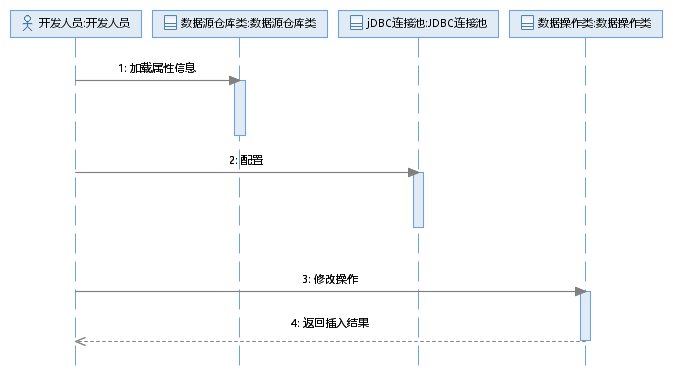


图 32 数据库修改用例时序图

## 4.7 请求响应时序图

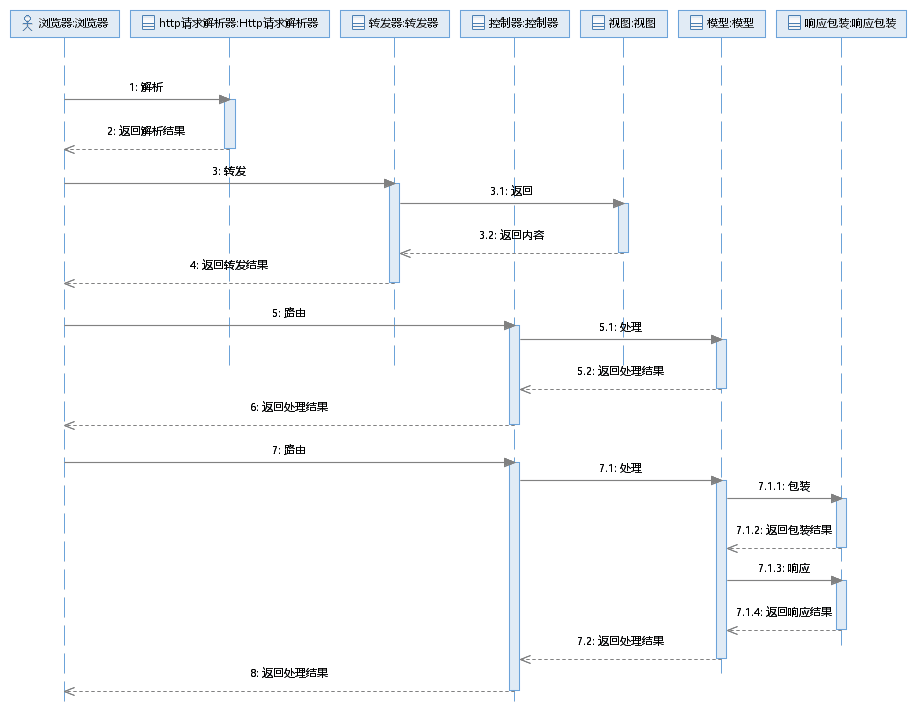


图 33 请求响应用例时序图

## 4.8 拦截请求时序图

根据拦截功能的RUCM描述，对开发人员使用拦截请求的工作流程做如下描述：

1）开发人员实现拦截器接口

2）开发人员配置拦截前置操作

3）开发人员配置拦截器后置操作

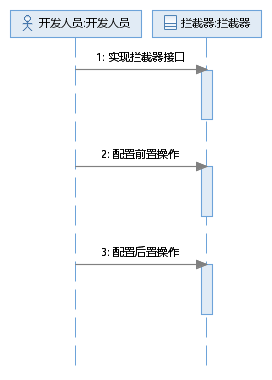


图 25 拦截请求时序图

# 5 状态图

Blade框架的数据库模块以单独的模块作为balde框架数据库接口使用，故状态图分为框架状态图与数据库状态图模块两部分。

## 5.1数据库模块状态图

在数据库模块状态图中，初始时数据库是关闭状态，在有数据库连接时数据库变为打开状态。之后若有数据库操作则数据库转为运行状态，判断是否运行完毕，运行完毕则重新回到数据库打开状态，否则依旧为数据库运行状态，如果遇到关闭数据库操作，则状态结束。

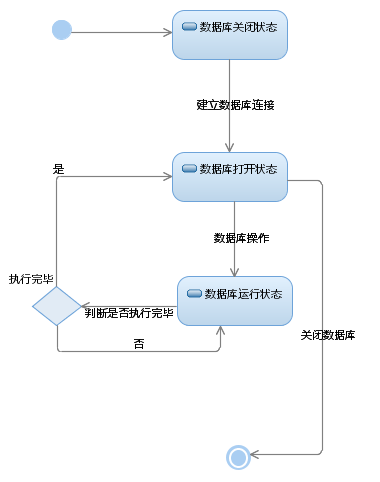


图 26 数据库操作状态迁移图

## 5.2 Blade框架状态图

在Blade框架状态图中，遇到开启程序操作时框架转为空闲状态，当开发人员运行程序时，框架转为执行状态，之后判断程序是否执行完毕，若执行完毕则框架重新变为空闲状态，否则框架依然为执行状态，当有关闭程序时，状态结束。

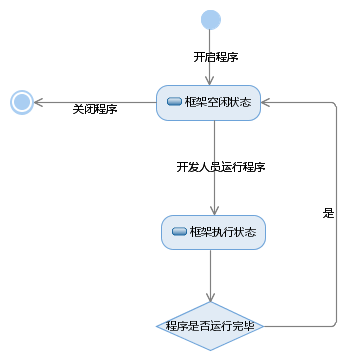


图 27 Blade运行状态迁移图

# 6 改进方案设想

## 6.1 需求分析

### 6.1.1 需求描述

从Blade的架构来看，Blade提供了相当充足的功能用于简化Java开发。但从开发过程来看，Blade尚有不足。其中最为明显的地方在于Blade并未针对测试提供解决方案。世界上不存在没有漏洞的系统，无一例外地都会在运行时都会出现错误、异常以及其他非期待的运行结果。想要尽可能的减少运行时出现的问题，提前的测试是开发中必不可少的一环。软件测试能够提前发现生产环境中可能出现的问题并提前处理，避免财产、生命的损失。根据软件工程和软件测试的规范，开发人员应先进行单元测试。单元测试中的黑盒测试将测试用例的输入作为模块的输入，将预期结果与和模块输出相比较来判断模块工作是否正常。

人工对一整个系统的测试时非常耗时费力的，因此开发人员可以选用各种常用的测试框架对程序进行测试。JUnit是Java语言中最为广泛使用和强力的测试框架，其能够提供Java运行必须的测试环境，并运行指定的待测试方法。JUnit能够采用断言的方式判断预期结果是否与模块输出结果相同，并循环或批量对不同的方法进行测试。这给开发者测试提供了极大的便利。

Blade是基于IOC技术的JavaWeb框架，其特性就在于开发者无需声明变量即可使用。但IOC必须在Blade框架完全启动才能生效，JUnit无法自动生成IOC环境，因此无法对基于Blade框架编写的应用程序进行测试。这使得Blade开发的应用程序变得不安全，我们小组认为易用的JavaWeb框架不应该有此缺陷，因此致力于完善Blade框架在测试方面的不足。

### 6.1.2 RUCM建模

测试不应该加重开发者的工作负担，因此借助IOC容器应该同样能够提供简单有效的测试方案。结合Spring的测试方案，使用RUCM描述测试用例的行为模型如图 34至图 35所示：

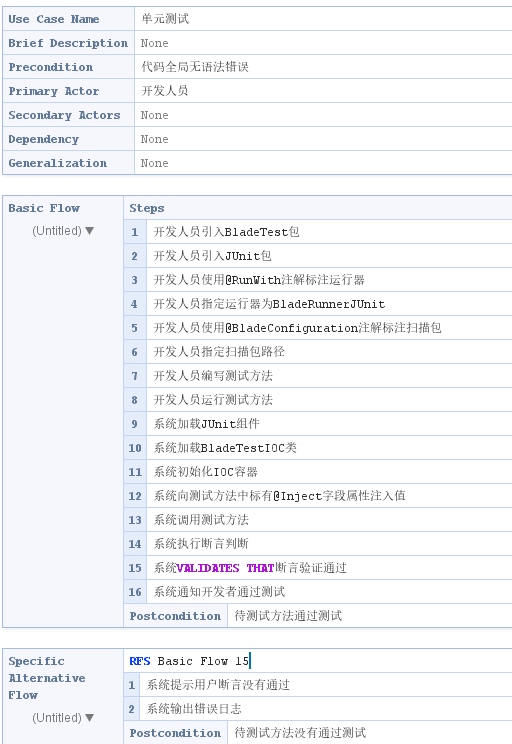


图 34 单元测试用例RUCM（a）



图 35 单元测试用例RUCM（b）

### 6.1.3 需求目标

根据需求描述，小组计划给予Blade的基本结构设计一套测试组件BladeTest，期望能够满足开发人员的日常测试功能。BladeTest应满足如下要求：

1）开发人员无需关心IoC容器细节，可在测试环境内直接使用IOC容器。

2）开发人员能够在测试环境内获取IOC容器内容

3）IOC容器在单元测试开始和结束时自动初始化

4）测试可以批量进行

5）尽可能减少与Blade的重复代码，保证同步更新的可能性

6）添加其他方便开发人员测试的功能并保证兼容性

## 6.2 可行性分析

作为一个Java语言开发的框架，Blade本应能与JUnit之间共同合作。但由于IOC容器的缺失，JUnit与Blade之间无法正常的交互。小组认为手动在JUnit运行时提供适合Blade使用的IOC容器能够解决该问题。

小组参考了Spring的Spring是JavaWeb框架的领头羊，其在SpringTest子项目中提供了基于JUnit的测试方案。JUnit在运行之前会通过@RunWith注解获取Spring默认IOC容器，该IOC进针对测试类进行管理，足够支持待测试方法的运行和检查。因此小组认为这是一条可行的技术方案。

## 6.3 测试用例类图

测试用例基于BladeTest类提供前置的IOC容器初始化，并依赖Blade的IOC组件进行执行。JUnit提供了RunWith接口，BladeTest在实现了该接口后能够预加载JUnit的环境，并提供IOC容器的实现。绘制类图如图 36所示。

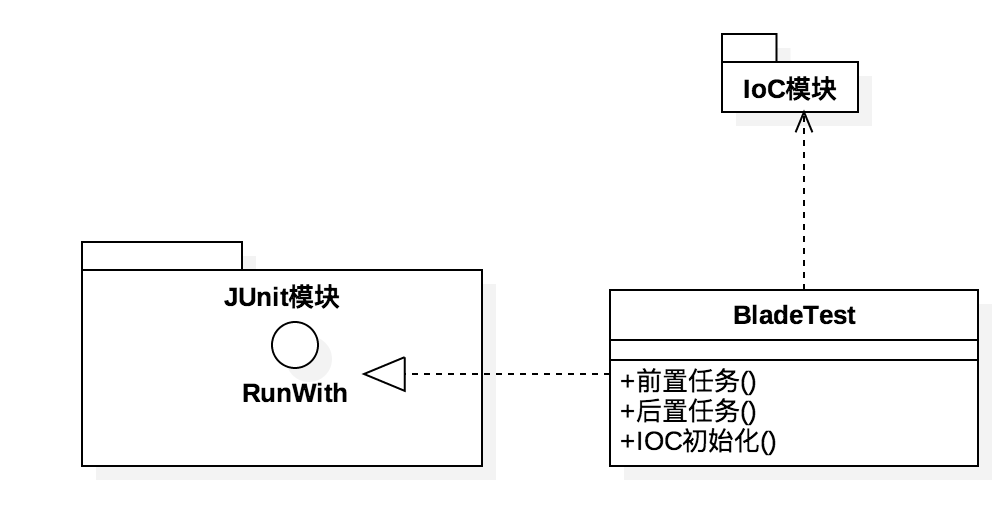


图 36 测试用例类图

1. bladejava.com [↑](#footnote-ref-1)