|  |
| --- |
|  |
| 软件需求说明书 |
| **Spring framwork** |
| |  |  | | --- | --- | | 王昕 SY1506415 | 王旭辰 SY1506413 | | 林森 SY1506416 | 李勃 SY1506402 | |
|  |
|  |

**版本变更历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 提交日期 | 编制人 | 说明 |
| 1.0 | 2016/3/23 | 全体 | 需求规格说明书初稿 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1范围 1](#_Toc446538299)

[1.1 标识 1](#_Toc446538300)

[1.2 系统概述 1](#_Toc446538301)

[1.3 数据字典 2](#_Toc446538302)

[1.4 文档概述 4](#_Toc446538303)

[2 项目描述 4](#_Toc446538304)

[2.1 模块概述 4](#_Toc446538305)

[3 功能需求 6](#_Toc446538306)

[3.1容器基本功能 6](#_Toc446538307)

[3.1.1配置依赖关系 6](#_Toc446538308)

[3.1.2使用默认标签 6](#_Toc446538309)

[3.1.3使用自定义标签 6](#_Toc446538310)

[3.1.4注入依赖关系 7](#_Toc446538311)

[3.1.5读取配置文件 7](#_Toc446538312)

[3.1.6解析配置文件 7](#_Toc446538313)

[3.1.7自定义标签解析 8](#_Toc446538314)

[3.1.8默认标签解析 8](#_Toc446538315)

[3.1.9加载组件 8](#_Toc446538316)

[3.1.10注册组件 8](#_Toc446538317)

[3.2容器的扩展功能 9](#_Toc446538318)

[3.2.1验证环境变量 9](#_Toc446538319)

[3.2.1处理不同语言 9](#_Toc446538320)

[3.2.3使用注解注入依赖 10](#_Toc446538321)

[3.2.4扩展容器功能 10](#_Toc446538322)

[3.2.5使用广播器管理消息 11](#_Toc446538323)

[4 非功能性需求分析 11](#_Toc446538324)

[4.1 可扩展性 11](#_Toc446538325)

[4.2 安全性 11](#_Toc446538326)

[4.3 易用性 12](#_Toc446538327)

[5 参考文献 12](#_Toc446538328)

# 1范围

## 1.1 标识

Spring版本号： 3.2

模 块 名 称： spring core

需求报告版本：Version 1.0

## 1.2 系统概述

Spring是一个开源框架，它由Rod Johnson创建。它是为了解决企业应用开发的复杂性而创建的。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。Spring是一个轻量级的控制反转(IoC)和面向切面(AOP)的容器框架。轻量——从大小与开销两方面而言Spring都是轻量的。完整的Spring框架可以在一个大小只有1MB多的JAR文件里发布。并且Spring所需的处理开销也是微不足道的。此外，Spring是非侵入式的：典型地，Spring应用中的对象不依赖于Spring的特定类。控制反转——Spring通过一种称作控制反转（IoC）的技术促进了松耦合。当应用了IoC，一个对象依赖的其它对象会通过被动的方式传递进来，而不是这个对象自己创建或者查找依赖对象。你可以认为IoC与JNDI相反——不是对象从容器中查找依赖，而是容器在对象初始化时不等对象请求就主动将依赖传递给它。面向切面——Spring提供了面向切面编程的丰富支持，允许通过分离应用的业务逻辑与系统级服务（例如审计（auditing）和事务（transaction）管理）进行内聚性的开发。应用对象只实现它们应该做的——完成业务逻辑——仅此而已。它们并不负责（甚至是意识）其它的系统级关注点，例如日志或事务支持。容器——Spring包含并管理应用对象的配置和生命周期，在这个意义上它是一种容器，你可以配置你的每个bean如何被创建——基于一个可配置原型（prototype），你的bean可以创建一个单独的实例或者每次需要时都生成一个新的实例——以及它们是如何相互关联的。然而，Spring不应该被混同于传统的重量级的EJB容器，它们经常是庞大与笨重的，难以使用。框架——Spring可以将简单的组件配置、组合成为复杂的应用。在Spring中，应用对象被声明式地组合，典型地是在一个XML文件里。Spring也提供了很多基础功能（事务管理、持久化框架集成等等），将应用逻辑的开发留给了你。所有Spring的这些特征使你能够编写更干净、更可管理、并且更易于测试的代码。它们也为Spring中的各种模块提供了基础支持。

## 1.3 数据字典

表格 1 Spring需求分析数据字典

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 术语 | 英文 | 说明 |
|  | 框架 | Framework | 框架（Framework）是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法;另一种定义认为，框架是可被应用开发者定制的应用骨架。前者是从应用方面而后者是从目的方面给出的定义。 |
|  | 控制反转 | Inversion of Control | 控制反转（Inversion of Control，英文缩写为IoC）是一个重要的面向对象编程的法则来削减计算机程序的耦合问题，也是轻量级的Spring框架的核心。 控制反转一般分为两种类型，依赖注入（Dependency Injection，简称DI）和依赖查找（Dependency Lookup）。依赖注入应用比较广泛。 |
|  | 依赖注入 | Dependency Injection | 依赖注入(Dependency Injection)和控制反转(Inversion of Control)是同一个概念。具体含义是:当某个角色(可能是一个Java实例，调用者)需要另一个角色(另一个Java实例，被调用者)的协助时，在 传统的程序设计过程中，通常由调用者来创建被调用者的实例。但在Spring里，创建被调用者的工作不再由调用者来完成，因此称为控制反转;创建被调用者 实例的工作通常由Spring容器来完成，然后注入调用者，因此也称为依赖注入。 |
|  | 服务器端组件模型 | EJB | EJB是sun的JavaEE服务器端组件模型，设计目标与核心应用是部署分布式应用程序。简单来说就是把已经编写好的程序（即：类）打包放在服务器上执行。凭借java跨平台的优势，用EJB技术部署的分布式系统可以不限于特定的平台。EJB (Enterprise JavaBean)是J2EE(javaEE)的一部分，定义了一个用于开发基于组件的企业多重应用程序的标准。其特点包括网络服务支持和核心开发工具(SDK)。 |
|  | 可扩展标记语言 | XML | 1998年2月，W3C正式批准了可扩展标记语言的标准定义，可扩展标记语言可以对文档和数据进行结构化处理，从而能够在部门、客户和供应商之间进行交换，实现动态内容生成，企业集成和应用开发。可扩展标记语言可以使我们能够更准确的搜索，更方便的传送软件组件，更好的描述一些事物。例如电子商务交易等。 |
|  | 事务 | Transaction | 事务提供了一种“将多个命令打包，然后一次性、按顺序地执行”的机制，并且事务在执行的期间不会主动中断——服务器在执行完事务中的所有命令之后，才会继续处理其他客户端的其他  命令。 |
|  | 面向切面 | AOP | 面向切面编程，是一个比较热门的话题。AOP主要实现的目的是针对业务处理过程中的切面进行提取，它所面对的是处理过程中的某个步骤或阶段，以获得逻辑过程中各部分之间低耦合性的隔离效果。比如我们最常见的就是日志记录了， |
|  | 容器 | vessel | 容器是伴随着瘦客户端系统的发展而诞生的。在开发瘦客户端系统时，开发人员要花费大量的精力去关注线程安全、事务、网络、资源等等细节，从而降低了开发效率。由于这些对这些细节的解决方法一般是固定不变，或者只有参数改变的，所以从代码重用和设计模式的角度出发，开发人员将这些底层细节提取出来，做成平台，并提供一定的接口。这样，业务开发人员就不需要在关注与这些底层细节的实现，而专注于业务逻辑的实现。 |

## 1.4 文档概述

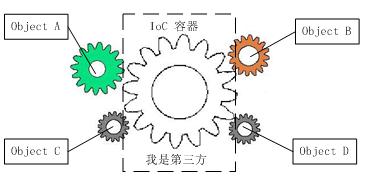
本文档是对Spring 框架的部分模块的需求分析和规格说明书，主要借助RUCM结构化模板，采用用例图等形式进行分析。

# 2 项目描述

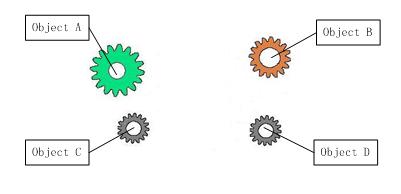
## 2.1 模块概述

(1)Ioc基本模块

IOC是Inversion of Control的缩写，多数书籍翻译成“控制反转”，还有些书籍翻译成为“控制反向”或者“控制倒置”。   
1996年，Michael Mattson在一篇有关探讨面向对象框架的文章中，首先提出了IOC 这个概念。对于面向对象设计及编程的基本思想，前面我们已经讲了很多了，不再赘述，简单来说就是把复杂系统分解成相互合作的对象，这些对象类通过封装以后，内部实现对外部是透明的，从而降低了解决问题的复杂度，而且可以灵活地被重用和扩展。IOC理论提出的观点大体是这样的：借助于“第三方”实现具有依赖关系的对象之间的解耦，如下图：



大家看到了吧，由于引进了中间位置的“第三方”，也就是IOC容器，使得A、B、C、D这4个对象没有了耦合关系，齿轮之间的传动全部依靠“第三方”了，全部对象的控制权全部上缴给“第三方”IOC容器，所以，IOC容器成了整个系统的关键核心，它起到了一种类似“粘合剂”的作用，把系统中的所有对象粘合在一起发挥作用，如果没有这个“粘合剂”，对象与对象之间会彼此失去联系，这就是有人把IOC容器比喻成“粘合剂”的由来。我们再来做个试验：把上图中间的IOC容器拿掉，然后再来看看这套系统：



（2）Ioc扩展模块

# 3 功能需求

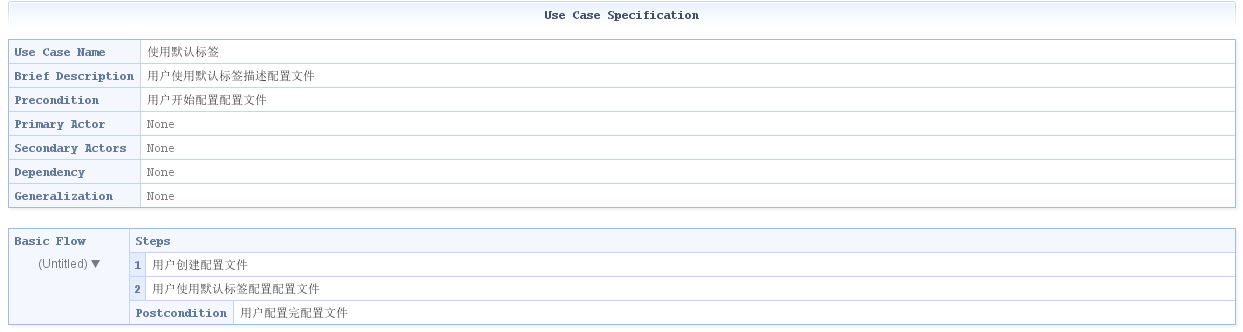
Ioc是spring的核心模块，它管理着程序员交给他的组件的整个生命周期，下面是Ioc在管理组件时的用例图

## 3.1容器基本功能

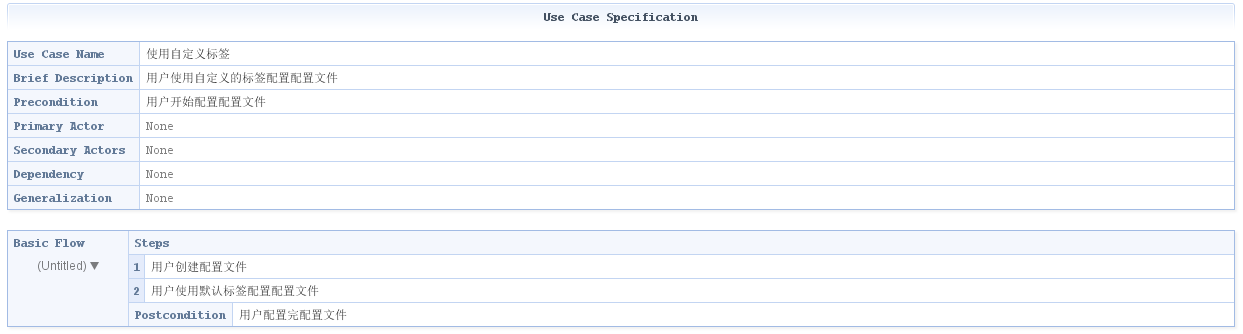
### 3.1.1配置依赖关系



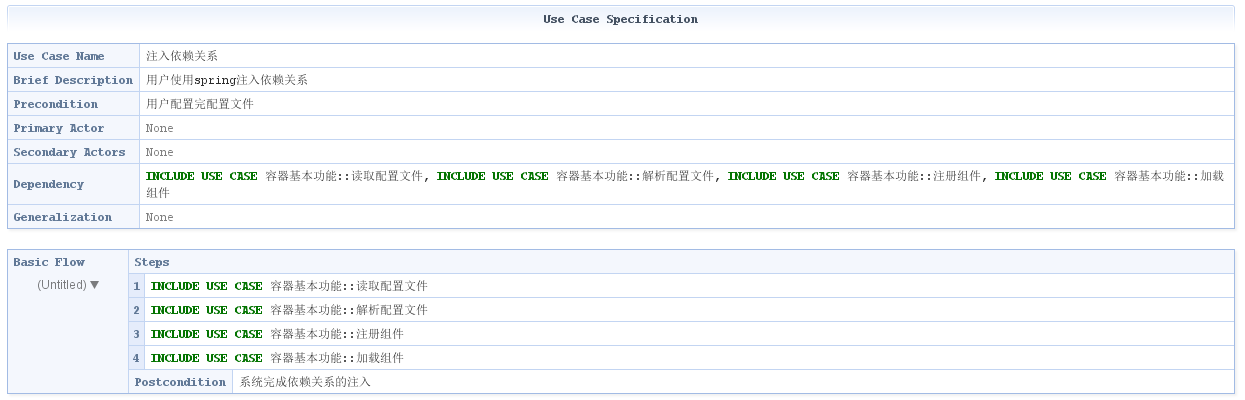
### 3.1.2使用默认标签



### 3.1.3使用自定义标签



### 3.1.4注入依赖关系



3

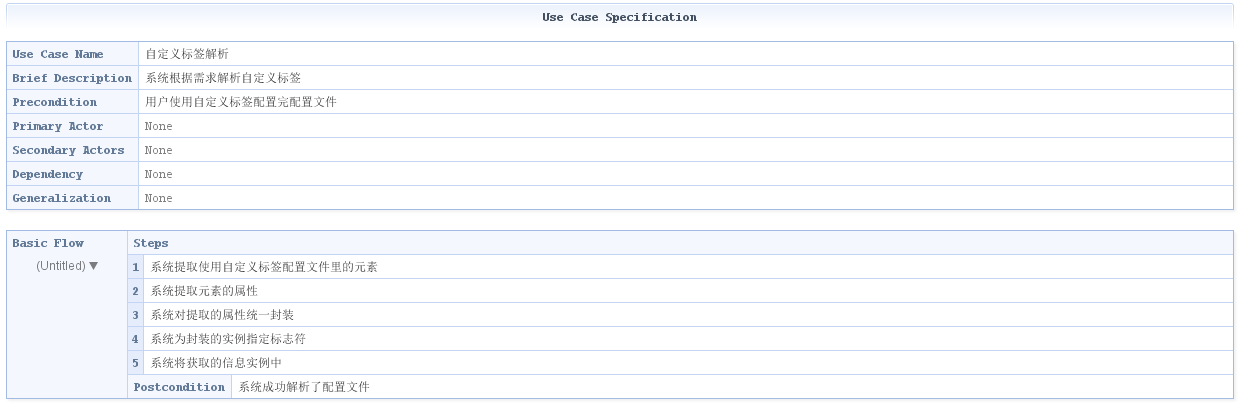
### 3.1.5读取配置文件



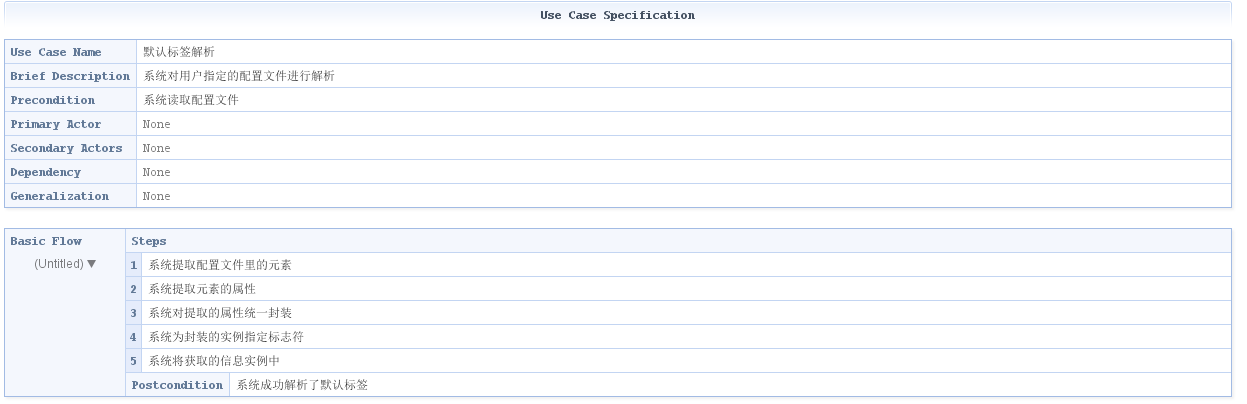
### 3.1.6解析配置文件



### 3.1.7自定义标签解析

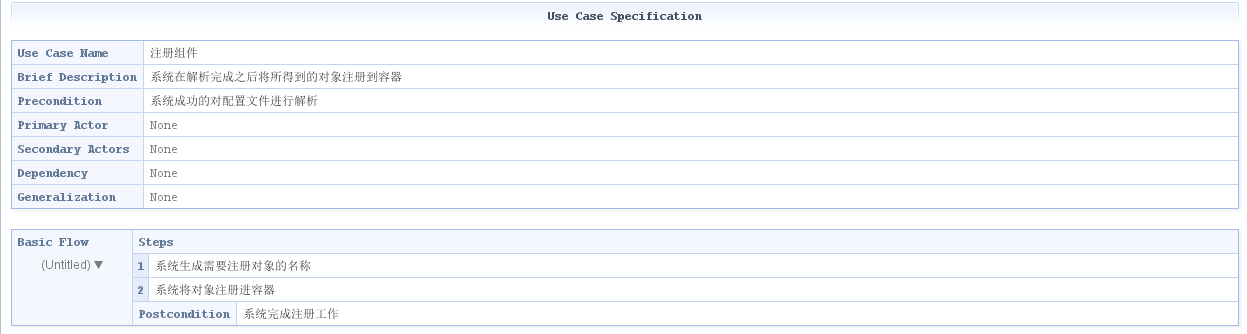


### 3.1.8默认标签解析



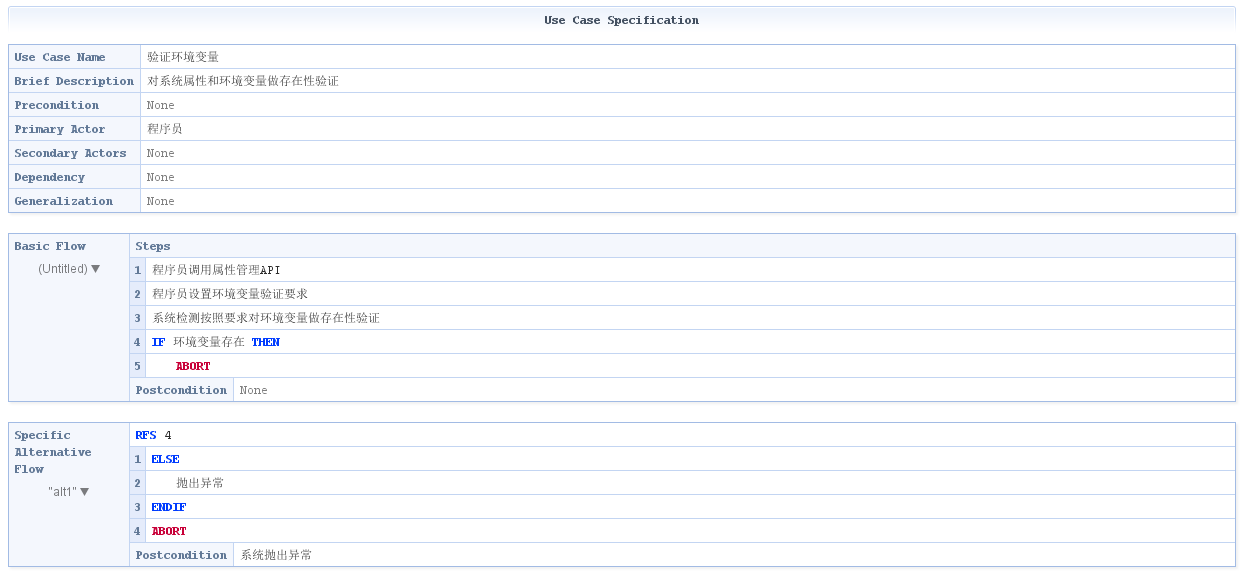
### 3.1.9加载组件

### 3.1.10注册组件

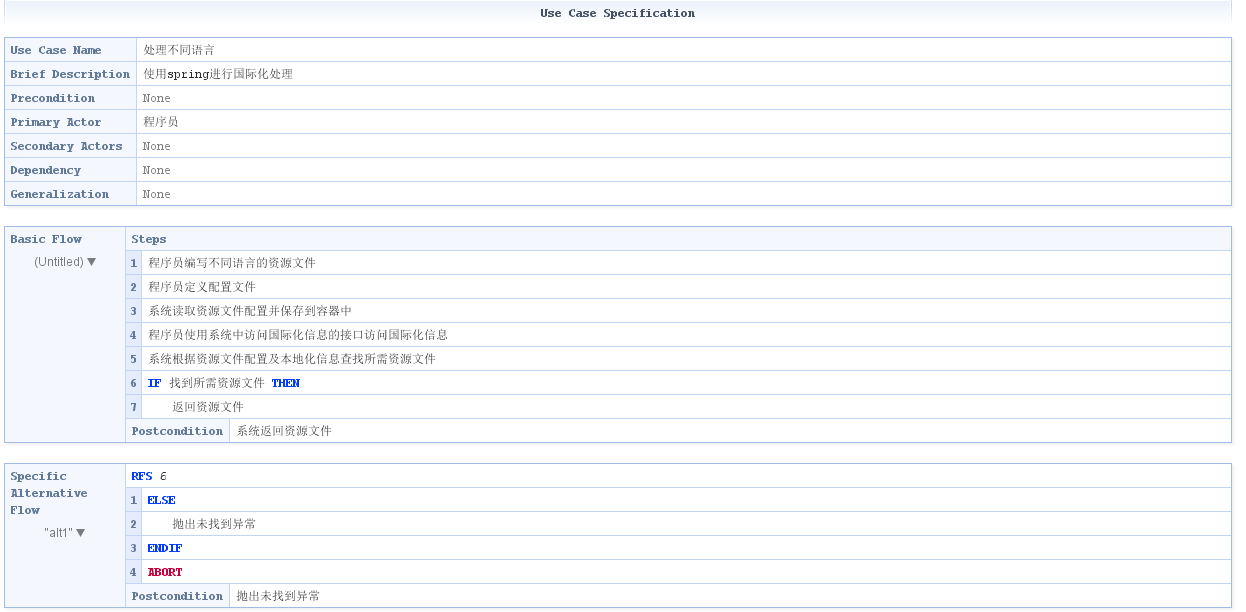


## 3.2容器的扩展功能

### 3.2.1验证环境变量

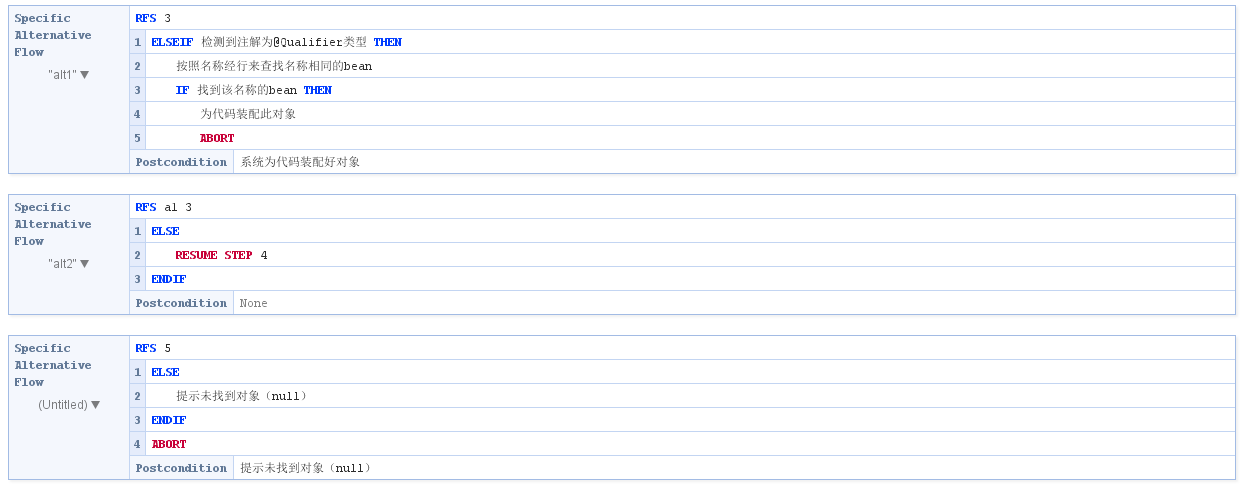


### 3.2.1处理不同语言

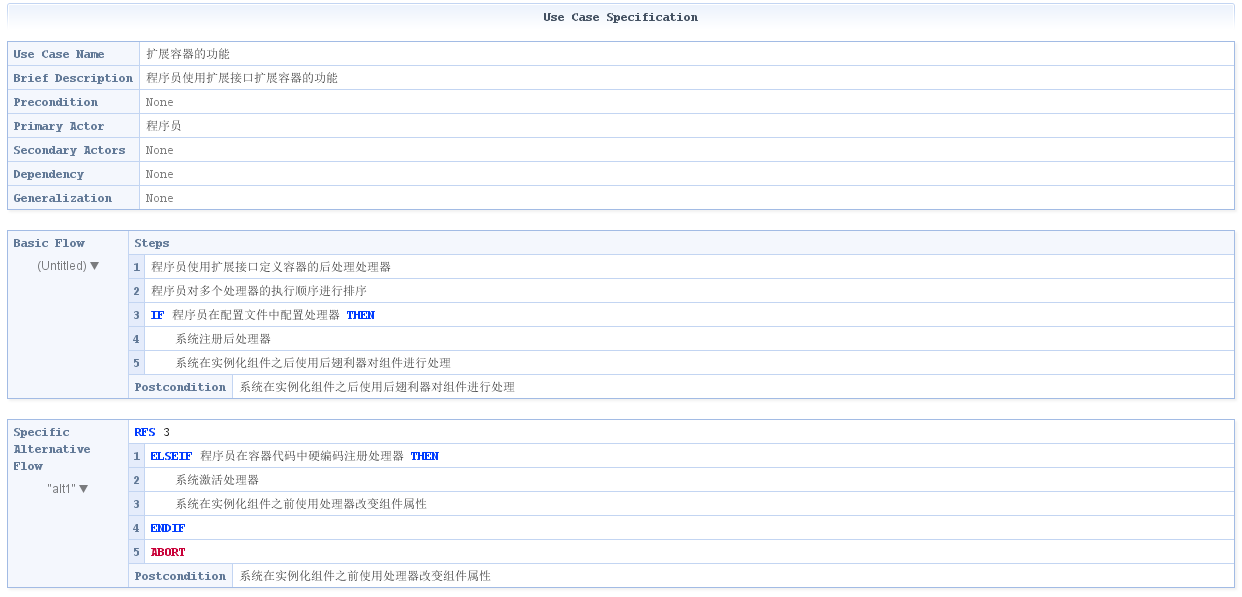


### 3.2.3使用注解注入依赖

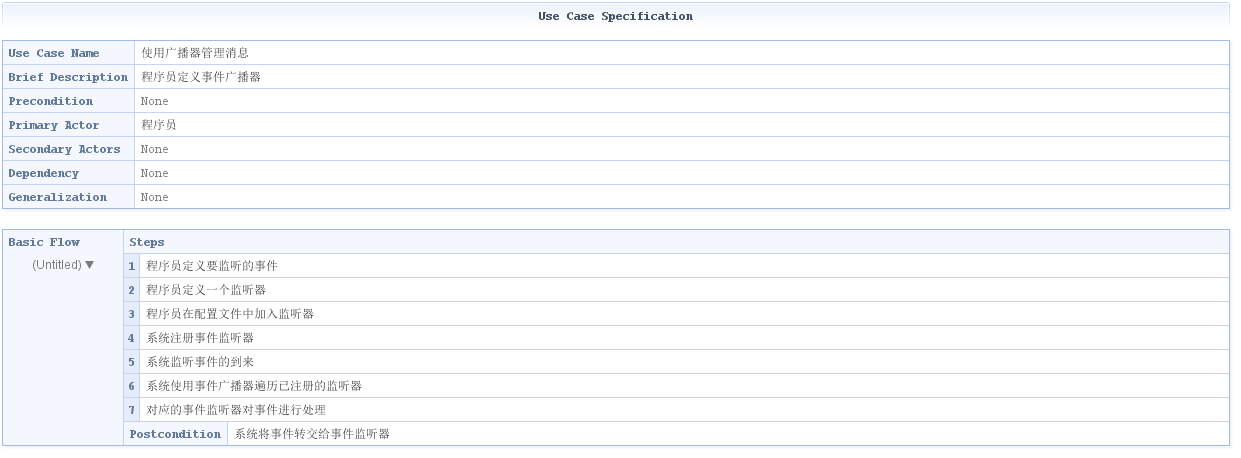




### 3.2.4扩展容器功能



### 3.2.5使用广播器管理消息



# 4 非功能性需求分析

## 4.1 可扩展性

Spring提供了对现在的主流框架的整合支持，在组件功能上提供了对各部分模块的深度定制，大部分的功能需求都能在spring上面得到支持。

## 4.2 安全性

Spring 是一个非常流行和成功的 Java 应用开发框架。Spring Security 基于 Spring 框架，提供了一套 Web 应用安全性的完整解决方案。一般来说，Web 应用的安全性包括用户认证（Authentication）和用户授权（Authorization）两个部分。用户认证指的是验证某个用户是否为系统中的合法主体，也就是说用户能否访问该系统。用户认证一般要求用户提供用户名和密码。系统通过校验用户名和密码来完成认证过程。用户授权指的是验证某个用户是否有权限执行某个操作。在一个系统中，不同用户所具有的权限是不同的。比如对一个文件来说，有的用户只能进行读取，而有的用户可以进行修改。一般来说，系统会为不同的用户分配不同的角色，而每个角色则对应一系列的权限。

对于上面提到的两种应用情景，Spring Security 框架都有很好的支持。在用户认证方面，Spring Security 框架支持主流的认证方式，包括 HTTP 基本认证、HTTP 表单验证、HTTP 摘要认证、OpenID 和 LDAP 等。在用户授权方面，Spring Security 提供了基于角色的访问控制和访问控制列表（Access Control List，ACL），可以对应用中的领域对象进行细粒度的控制。

本文将通过三个具体的示例来介绍 Spring Security 的使用。第一个示例是一个简单的企业员工管理系统。该系统中存在三类用户，分别是普通员工、经理和总裁。不同类别的用户所能访问的资源不同。对这些资源所能执行的操作也不相同。Spring Security 能帮助开发人员以简单的方式满足这些安全性相关的需求。第二个示例展示了如何与 LDAP 服务器进行集成。第三个示例展示了如何与 OAuth 进行集成。

## 4.3 易用性

(1)Spring能有效地组织你的中间层对象。

(2)Spring能消除在许多工程中常见的对Singleton的过多使用。

(3)Spring能消除各种各样自定义格式的属性文件的需要，使配置信息一元化。

(4)Spring能够帮助我们真正意义上实现针对接口编程。

(5)在Spring应用中的大多数业务对象没有依赖于Spring。

(6)使用Spring构建的应用程序易于单元测试。

(7)Spring支持JDBC和O/R Mapping产品(Hibernate)

(8)MVC Web框架，提供一种清晰，无侵略性的MVC实现方式。

(9)JNDI抽象层，便于改变实现细节，可以方便地在远程服务和本地服务间切换。

(10)简化访问数据库时的例外处理。

(11)Spring能使用AOP提供声明性事务管理，可以不直接操作JTA也能够对事务进行管理。

(12)提供了JavaMail或其他邮件系统的支持

图 27 内存操作用例规格

# 5 参考文献

[1]《spring技术内幕：深入解析spring架构与计原理(第2版)》计文柯，机械工业出版社

[2]《Spring源码深度解析》郝佳，人民邮电出版社

[3]http://www.ibm.com/developerworks/cn/java/j-lo-spring-principle/

[4]http://blog.csdn.net/lishuangzhe7047/article/details/20740209