|  |
| --- |
|  |
| 软件需求说明书 |
| **Spring framework** |
| |  |  | | --- | --- | | 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 | 2016/3/2 | 全体 | 需求规格说明书初稿 |
| 1.1 | 2016/4/6 | 王昕 | 需求规格说明书改进 |
| 1.2 | 2016/4/8 | 王昕 | 网评修改版 |
| 1.3 | 2016/4/13 | 王昕 | 复评审修改版 |
| 2.0 | 2016/5/4 | 全体 | 改进与展示后修改 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1范围 1](#_Toc447877581)

[1.1 标识 1](#_Toc447877582)

[1.2 系统概述 1](#_Toc447877583)

[1.3 数据字典 2](#_Toc447877584)

[1.4 文档概述 4](#_Toc447877585)

[2 项目描述 4](#_Toc447877586)

[2.1 模块概述 4](#_Toc447877587)

[2.2典型应用场景描述 6](#_Toc447877588)

[3 功能需求 7](#_Toc447877589)

[3.1容器基本功能 8](#_Toc447877590)

[3.1.1配置依赖关系 8](#_Toc447877591)

[3.1.2使用默认标签 8](#_Toc447877592)

[3.1.3使用自定义标签 8](#_Toc447877593)

[3.1.4注入依赖关系 9](#_Toc447877594)

[3.1.5读取配置文件 9](#_Toc447877595)

[3.1.6解析配置文件 9](#_Toc447877596)

[3.1.7自定义标签解析 10](#_Toc447877597)

[3.1.8默认标签解析 10](#_Toc447877598)

[3.1.9加载组件 11](#_Toc447877599)

[3.1.10注册组件 14](#_Toc447877600)

[3.2容器的扩展功能 15](#_Toc447877601)

[3.2.1验证环境变量 15](#_Toc447877602)

[3.2.1处理不同语言 15](#_Toc447877603)

[3.2.3使用注解注入依赖 16](#_Toc447877604)

[3.2.4扩展容器功能 16](#_Toc447877605)

[3.2.5使用广播器管理消息 17](#_Toc447877606)

[4 非功能性需求分析 17](#_Toc447877607)

[4.1 可扩展性 17](#_Toc447877608)

[4.2 安全性 17](#_Toc447877609)

[4.3 易用性 18](#_Toc447877610)

[5 参考文献 19](#_Toc447877611)

# 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 1.2

## 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

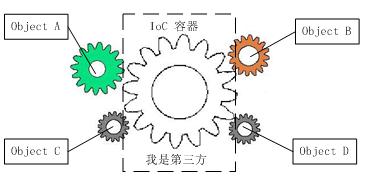
本文档是对Spring 框架的IOC模块的需求分析和规格说明书，主要借助RUCM结构化模板，采用用例图等形式进行分析。

# 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，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：

  
图1spring IOC原理

从图1中可以看到，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1中间的IOC容器拿掉，然后再来看看这套系统：如图2

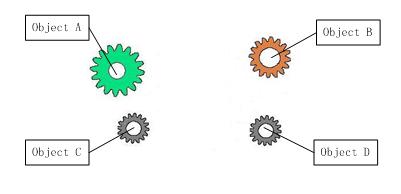


图2 无IOC的情况

## 2.2典型应用场景描述

通过介绍Spring架构设计，我们了解到Spring是一个轻量级的框架。在Spring这个一站式的应用平台或框架中，其中的各个模块除了依赖IoC容器和AOP之外，相互之间并没有很强的耦合性。Spring的最终目标是简化应用开发的编程模型。它所提供的服务，可以贯穿应用到整个软件中，从最上层的Web UI到底层的数据操作，到其他企业信息数据的集成，再到各种J2EE服务的使用，等等。这些企业应用服务，Spring都通过其特有的IoC容器和AOP模块实现。在实现过程中，Spring没有把这种复杂性转换成自己被使用的复杂性，这点无疑是成功的，同时大大拓宽了Spring的应用场景。一方面，我们可以把Spring作为一个整体来使用，另一方面，也可以各取所需，把Spring的各个模块拿出来独立使用，这取决于我们对Spring提供服务的具体需求。例如，这些需求可能来自一个完整的Java EE企业应用开发需求，可以仅使用Spring的某些模块，如IoC容器。再如，我们可以使用Spring集成其他的J2EE服务，如JavaMail、JMS、JNDI等，还可以在Android应用环境，甚至在.NET应用环境中使用Spring。使用Spring的时候，可以采用各种不同的方式，而对于这些方式的选择，完全是由应用来决定的。因而，在对Spring的使用中，我们看到应用很少依赖于Spring特有的API，同时，由于Spring本身的设计也是非常模块化的，这样，就为应用开发提供了EJB开发不曾提供的便利。

同样，因为Spring的实现中，它的核心实现，比如IoC容器实现，是直接依赖JVM虚拟机的，也就是说，在Java环境中，Spring IoC容器是可以单独使用的，特别是在BeanFactory的基本实现中，包含在一个小小的jar包里面，可以直接在应用中引用。对于Spring而言，如果要在.NET环境下使用其提供的基本特性，Spring项目也提供了Spring .NET的实现；如果需要在Android移动平台中使用Spring的基本特性，Spring也有对Spring Android项目的支持。从这些应用场景上可以看出，因为Spring设计时的轻量级特性，以及推崇POJO开发，所以使用起来非常灵活。在对Spring的应用中，Spring团队为我们列举了Spring的价值，非常值得参考.

正常的用例场景：

A．事件源：用户想运行基于Spring框架的应用程序

事件结果：系统依据用户编写的配置，运行代码并返回运行的状态和结果。

事件名：用户想实现Spring框架的IOC功能

输入数据：xml配置文件或POJO代码

输出数据：Spring框架内IOC容器或XML文件

B．结构化场景

1. 用户编写了xml配置文件，并打算运行程序

2. Spring框架需要知道配置文件的名称以及路径

3. Spring框架需要核实配置文件是否存在以及是否可用

4. Spring框架需要获取配置文件的验证信息

5. Spring框架需要根据配置文件完成IOC功能

6. Spring框架需要返回用户IOC功能执行的信息

正常用例场景如下：

1. 用户编写了xml配置文件，运行程序

2. Spring框架知道配置文件的名称以及路径

3. Spring框架核实配置文件存在且可用

4. Spring框架获取配置文件的验证信息

5. Spring框架根据配置文件完成IOC功能

6. Spring框架返回用户IOC功能执行成功

异常的用例场景：

用户运行基于Spring框架的应用程序，但应用程序编译未通过或其他原因不能运行，此时用户和系统该怎么办？（系统终止并报告出错信息，用户需根据出错情况修改代码）。Spring框架不知道用户配置文件的名称和路径，或者配置文件不存在或存在但不可用，没有访问权限怎么办？（系统报错或自动生成配置文件的名称和路径，系统修改配置文件的访问权限或从其他路径搜索并提示用户已经出现的问题）。Spring框架未能获取文件的验证信息怎么办？（使用默认的文件验证信息或提示用户未能获取配置文件的验证信息）。Spring框架未能依据配置文件完成IOC功能，中间出现异常情况怎么办？（系统提示用户出错的信息并终止程序的运行）。

# 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

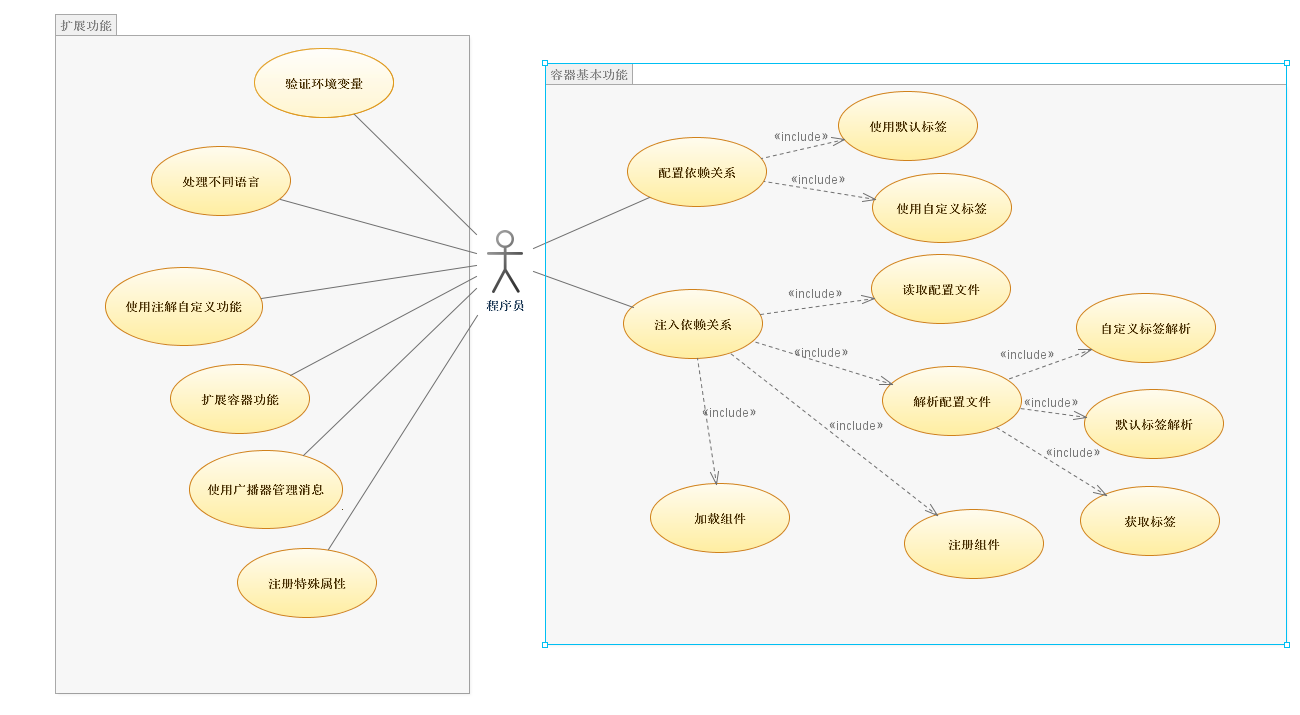


图3 IOC模块主用例图

加载组件部分细化用例图

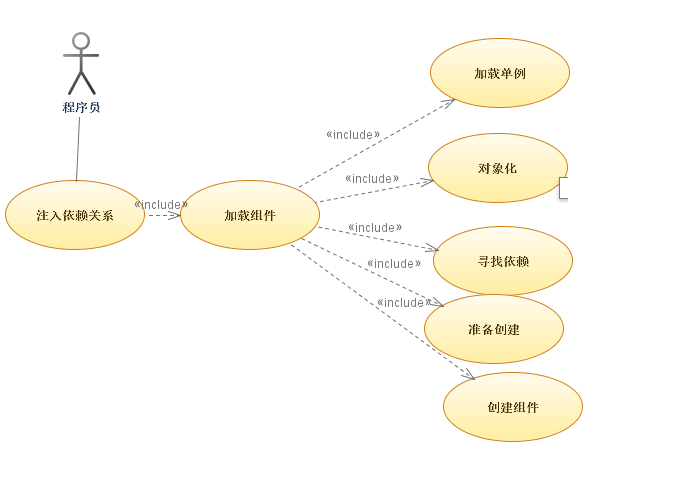


图4 加载组件模块细化用例图

将过程较为复杂的注入依赖关系中的加载组件进一步展开，将注入的过程细化。

## 3.1容器基本功能

### 3.1.1配置依赖关系

用户根据编程的需要，将现实中的类间的依赖关系反应到Spring容器中，具体是在配置文件中体现类间的依赖关系，这部分操作需要用户手动进行。之后Spring通过容器动态的向程序注入依赖关系。

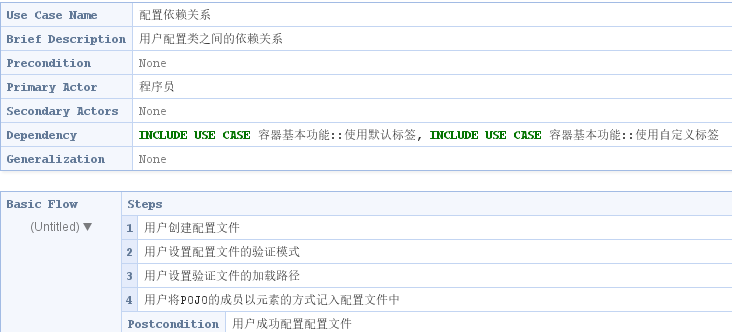


图5 配置依赖关系用例规格

### 3.1.2使用默认标签

这两种标签的使用方式基本相同，不同之处在于使用自定义标签需要指定自定义标签的解析方式，这种解析方式也是有用户自己编程实现的。Spring框架已经向用户提供了功能丰富的标签，能够满足通常的使用需求。自定义标签用于满足企业对Spring框架扩展的需求。



图6使用默认标签用例规格

### 3.1.3使用自定义标签

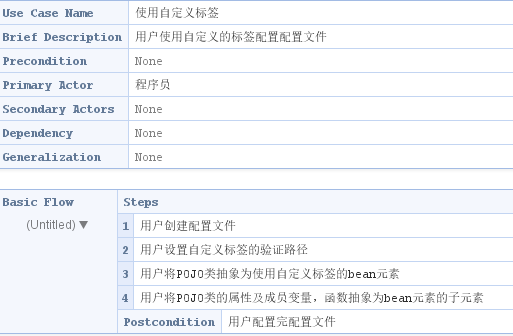


图7 使用自定义标签用例规格

### 3.1.4注入依赖关系

用户注入依赖关系的具体操作就是运行依赖于Spring框架的程序，将模块间的关系交由Spring框架来管理，之后的操作就有系统来进行处理。



图8注入依赖关系用例规格

### 3.1.5读取配置文件

用户将控制权交由Spring框架之后，Spring框架将从配置文件中获取用户所期望的对象间关系，配置文件是用户与框架间交互的接口。



图9读取配置文件用例规格

### 3.1.6解析配置文件

读取配置文件本质上是获取指向配置文件的输入流。解析配置文件模块从流中获取用户定义的元素，每个元素对应JAVA源文件中的一个类，对元素的属性解析本质上是对对应类的属性进行解析。



图10解析配置文件用例规格

### 3.1.7获取便签

标签用于是定义元素的。获取标签就是获取标签所定义的相关元素，对应于JAVA的一个类。



图11 获取标签用例规格

### 3.1.8自定义标签解析



图12 自定义标签解析用例规格

### 3.1.9默认标签解析



图13默认标签解析用例规格

### 3.1.10加载组件

在进行解析的工作结束之后，将是对组件的加载，而组件的加载是一个相当复杂的过程。后面的步骤都有较为具体的介绍，所以只简单对转换组件的名字进行介绍，由于传入的参数可能会是别名，别名是之前解析文件的过程中可能会出现的情况。所以根据别名所指向的组件返回真正的组件名。

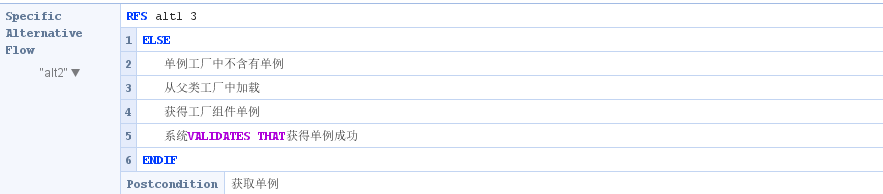


 图14加载组件用例规格

#### 3.1.10.1加载单例

由于单例在Spring的同一个容器中只会被创建一次，所以后续想要再次获取组件，就需要从缓存，从单例工厂，甚至从父类组件中寻找，以找到其中的实例为判断是否获取组件的条件。



 图15加载单例用例规格

#### 3.1.10.2对象化

从获取的单例中得到对象





图16对象化用例规格

#### 3.1.10.3寻找依赖



图17寻找依赖用例规格

#### 3.1.10.4准备创建

介绍一下循环依赖的定义：循环依赖就是循环引用，就是两个或多个组件相互之间的持有对方，例如CircleA引用CircleB,CircleB引用CircleC，CircleC引用CircleA，则它们最终反映为一个环。寻找依赖，就是要查看一下是否发生了这类情况。





图18准备创建用例规格

#### 3.1.10.5创建组件

组件Wrapper：组件Wrapper是对组件的包装，其接口中所定义的功能很简单包括设置获取被包装的对象，获取被包装组件的属性描述器。





图19创建组件用例规格

### 3.1.11注册组件

对标签做解析之后，将标签所对应的类的属性以及成员函数封装到Spring框架的内部对象中。为了在将来再次使用该类是能直接调用封装的对象，注册组件为封装的对象创建索引，并将他注册到容器中，将来通过索引就能获取所需要的内部对象。



图20注册组件用例规格

## 3.2容器的扩展功能

### 3.2.1验证环境变量

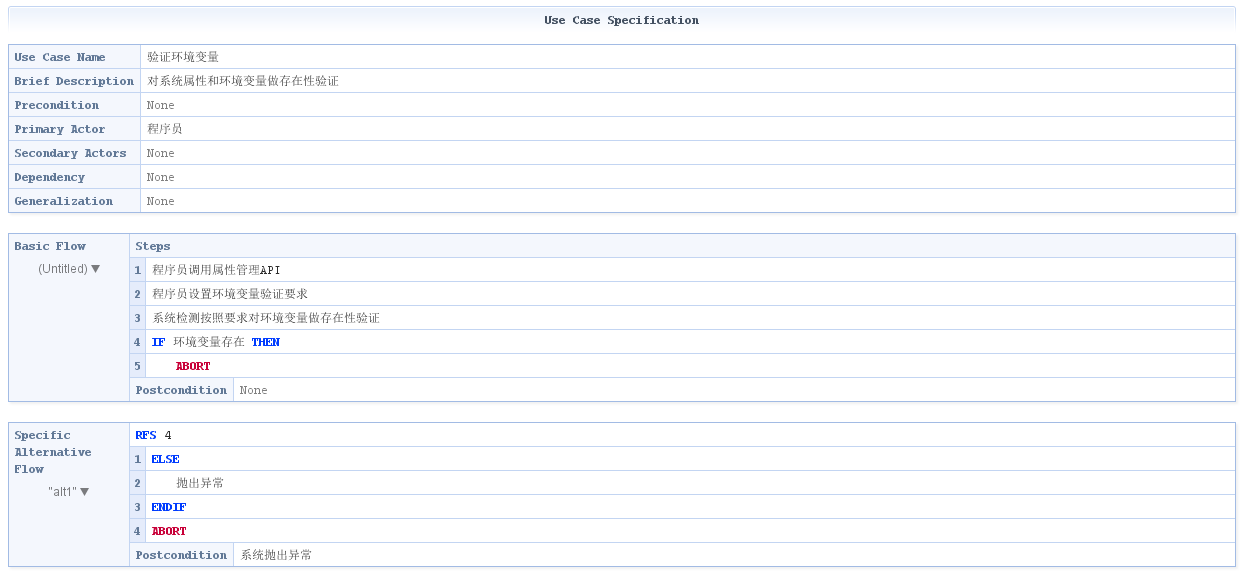


图21验证环境变量用例规格

### 3.2.1处理不同语言

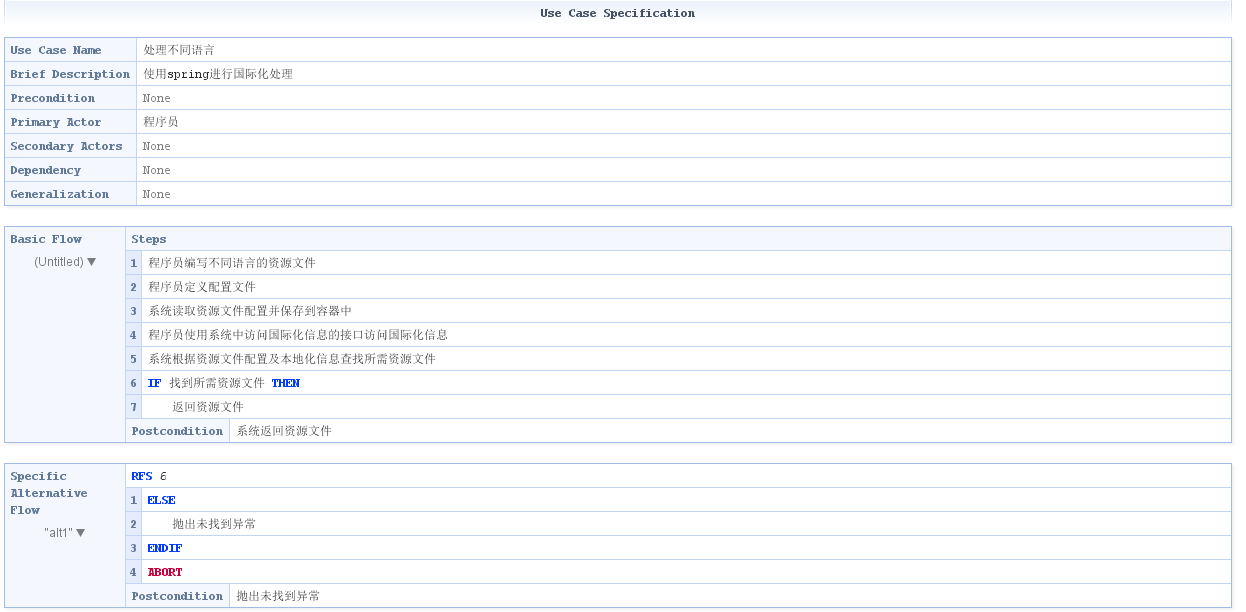
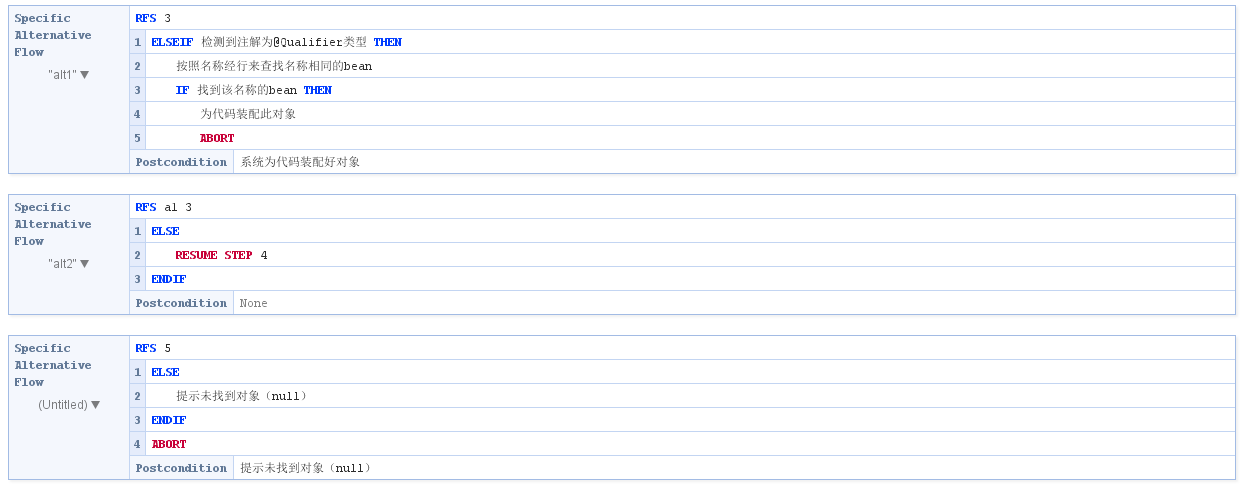


图22处理不同语言用例规格

### 3.2.3使用注解注入依赖



 图23使用注解注入依赖用例规格

### 3.2.4扩展容器功能

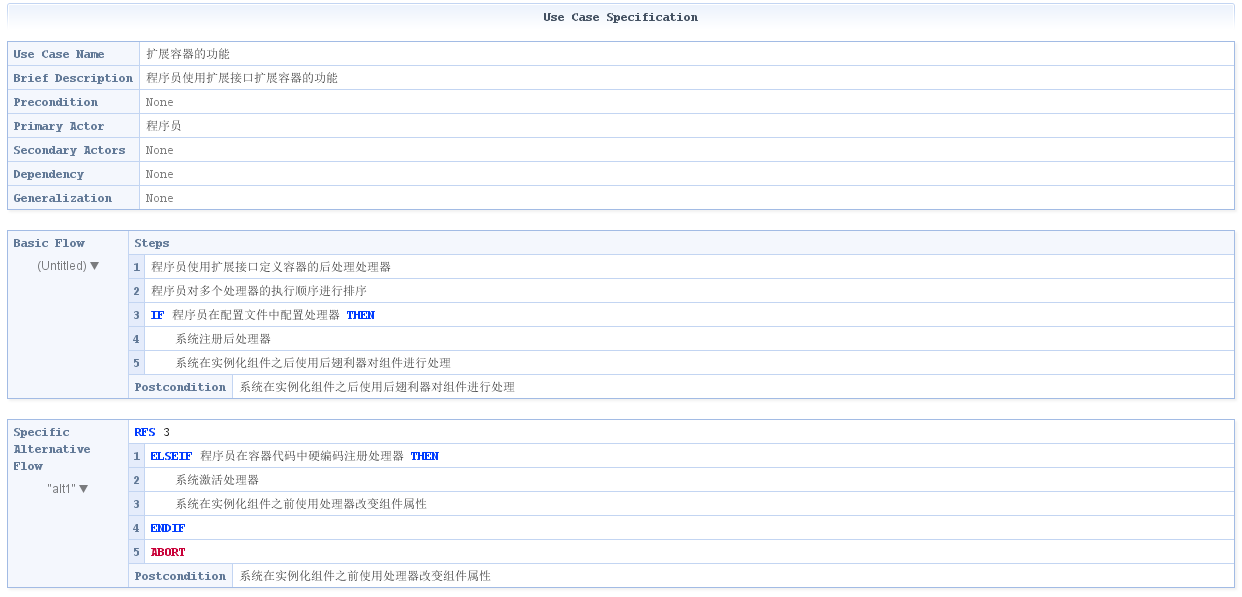


图24扩展容器功能用例规格

### 3.2.5使用广播器管理消息

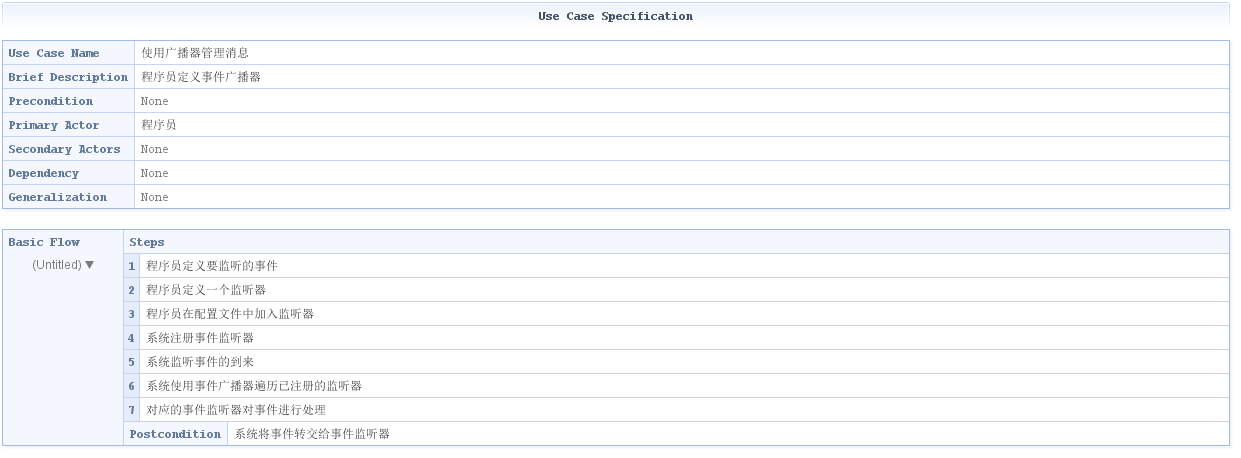


图25使用广播管理消息用例规格

### 3.2.6注册特殊属性

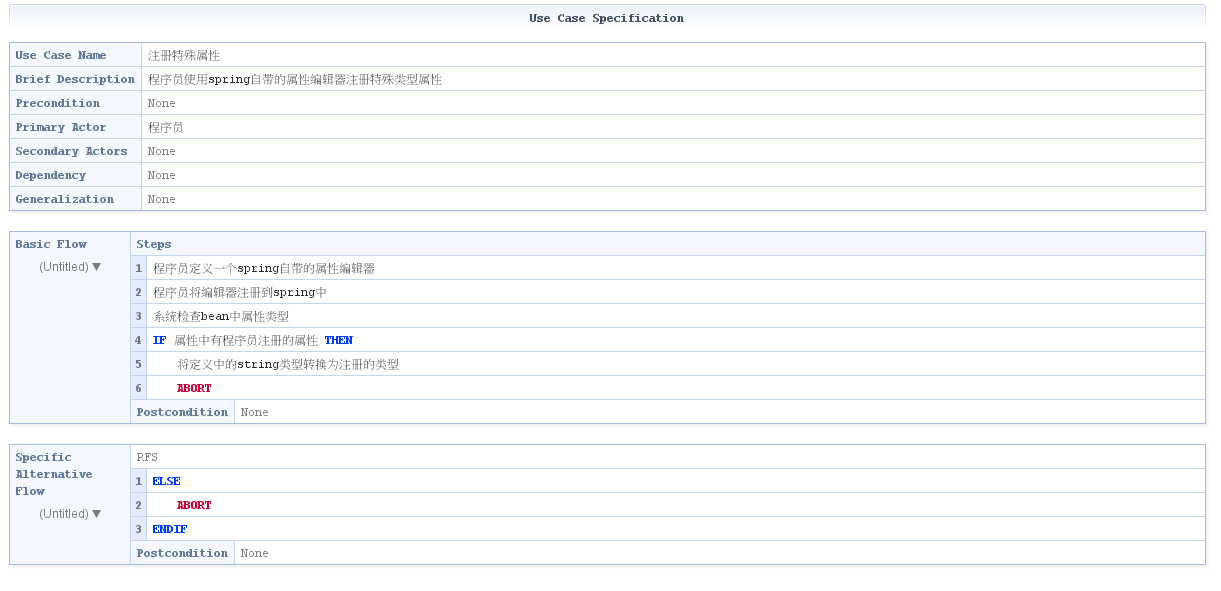


图26注册特殊属性

# 4功能需求改进部分

概述：spring IOC：优点有：

1. XML配置方式进一步降低了耦合，使得应用更加容易扩展，即使对配置文件进一步修改也不需要工程进行修改和重新编译。

2. 在处理大的业务量的时候，用XML配置应该更加好一些。因为XML更加清晰的表明了各个对象之间的关系，各个业务类之间的调用。同时spring的相关配置也能一目了然。

当然，有人会说，用XML配置，在大的业务量时候会使得XML文件过大，不容易查看。这一点我们完全可以利用业务分解书写多个XML配置文件就可以了。

xml配置文件方式缺点有：

配置文件读取和解析需要花费一定的时间，配置文件过多的时候难以管理，无法对配置的正确性进行校验，增加了测试难度。配置文件配置工作量相对较大。

总结性：数据基础配置性质，基础框架性质的bean，是写在xml里面，很多可能根据实际需要还会有调整。这种情况下使用xml比较好。

在上一周，我们通过直接读取类的文件将组件的相互依赖关系通过模拟的容器进行统一管理，本周我们又实现了直接通过xml文件获取其中的依赖关系并且利用模拟容器进行管理。而spring框架在一般情况下都是通过xml配置文件方式或注解方式（实现较为复杂）进行组件的加载管理，利用类直接转换成对应的xml配置文件。（下周内的发展点）。可有可无：

注解方式优点：

1、在class文件中，可以降低维护成本

2、提高开发效率。

注解方式缺点：

1、如果对annotation进行修改，需要重新编译整个工程。

2、业务类之间的关系不如XML配置那样一目了然。

3、程序中过多的annotation，对于代码的简洁度有一定影响。

4、annotation貌似功能没有xml配置文件齐全）

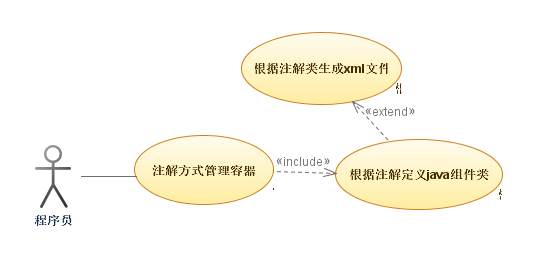


图 27 改进部分用例图

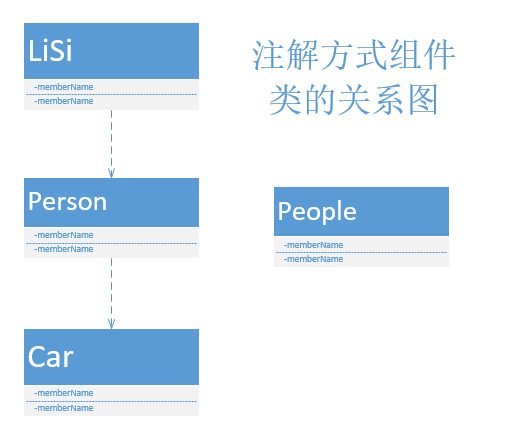


图28注解方式类图及设计

## 4.1根据注解生成Java组件类C:\Users\w\Documents\Tencent Files\781594958\FileRecv\spring需求补充\补充2.png

图29

## 4.2注解方式管理C:\Users\w\Documents\Tencent Files\781594958\FileRecv\spring需求补充\补充rucm1.png

图30

## 4.3根据注解生成xml配置文件



图31

# 5非功能性需求分析

## 5.1 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。比如一个企业在用spring开发他们的OA系统。一般的架构会包含三层：表现层，模型层，持久层。持久层也就是我们常说的数据库层。Spring框架为这三层提供了一体化的解决方案。Spring改进了体系结构的选择，虽然作为应用平台，Spring可以帮助我们选择不同的技术实现，比如从Hiberante切换到其他ORM工具，从Struts切换到Spring MVC，尽管我们通常不会这样做，但是我们在技术方案上选择使用Spring作为应用平台，Spring至少为我们提供了这种可能性和选择，从而降低了平台锁定的风险。

## 5.2 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。

## 5.3 易用性

随着Spring的版本提升，Spring版本中增加了很多方便程序员使用Spring 框架更加方便，比如注解的增加了优化了配置难度，让越来越多的公司拥抱Spring 框架。

# 6参考文献

[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