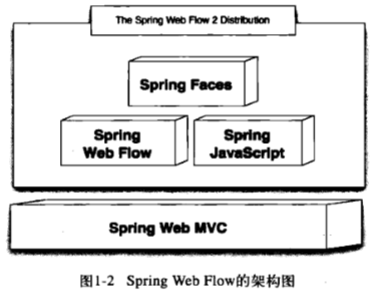
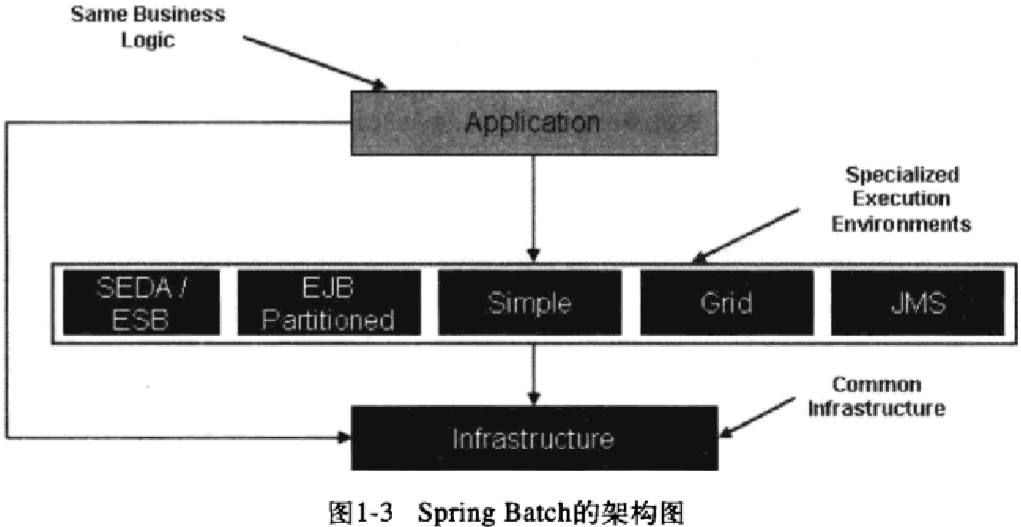
# 第一章

    spring的设计理念和整体架构

1. Spring的各个子项目
2. Spring的设计目标
3. Spring的整体架构
4. Spring的应用场景

## 1.1spring的各个子项目

* Spring Framework (core) : spring的核心项目
  + 一系列ioc容器设计
  + 依赖反转的实现
  + 集成aop功能
  + 包含其他spring的基本模块
    - SpringMvc ，jdbc ，事物处理模块等的实现
* Spring web flow ： 原先spring web flow 是建立在springmvc基础上的工作流引擎
  + 
* Spring BlazeDS integration ：spring与adobe flex技术集成的模块
* Spring security：spring认证和安全工具
* Spring security oauth：这个项目是为oauth在spring上集成提供支持。Oauth是一个第三方模块，提供一个开放的协议实现，通过这个协议，前端桌面应用可以对web应用进行简单而标准的安全吊用
* Spring dynamic modules：可以让spring运行在osgi平台上，osgi平台增加应用在部署和运行时的灵活性，eclipse就是构建在osgi平台上面的
* Spring batch ：构建批处理应用和自动化操作框架。
  + 不需要与用户交互
  + 重复的操作量大
* Spring integration：企业集成模式的具体实现。
  + 企业数据集成
  + 转换各种消息格式
* Spring amqp：高级消息队列协议的消息服务器开发的 ，rabbit mq就是基于amqp的消息服务器，使用erlang语言开发的
* Spring .net：.net版本的spring
* Spring android：
* Spring mobile：与spring android不同
  + 基于是spring mvc构建的
  + 可以在服务器自动识别连接到移动终端的相关设备信息
* Spring soical:
* Spring data：为spring提供非关系型数据，用于分布式存储，云计算等。

# 第二章

* **springframework的核心：ioc容器的实现**
* 本章内容
* spring ioc容器概述
* ioc容器系列的设计与实现 ： beanFactory和applicationContext
* ioc容器的初始化过程
* ioc容器的以来注入
* 容器其他相关特性的设计与实现

## 2.1 spring ioc容器概述

### 2.1.1 Ioc容器和依赖反转模式

* 依赖对象的获得被反转了
* 依赖对象由自己获取的话，耦合度非常高并且难以测试
* 对象的依赖交给框架或者ioc容器来完成
* 依赖控制反转有很多实现方式，
* 在spring中ioc容器是实现这个模式的载体，它可以在对象生成或初始化时直接将数据注入到对象中，也可以通过对象引用注入到对象数据域中的方法来注入对方法的调用
* 把控制权从具体的业务对象转交到平台或者框架中
* 把对象进行分类，一部分是数据对象，其他很大一部分是用来处理数据的
* 对不常变化，是系统中的基础部分，对象之间的依赖关系也是比较稳定的

### 2.1.2 spring ioc的应用场景

Ioc容器是解耦组件之间复杂关系的利器

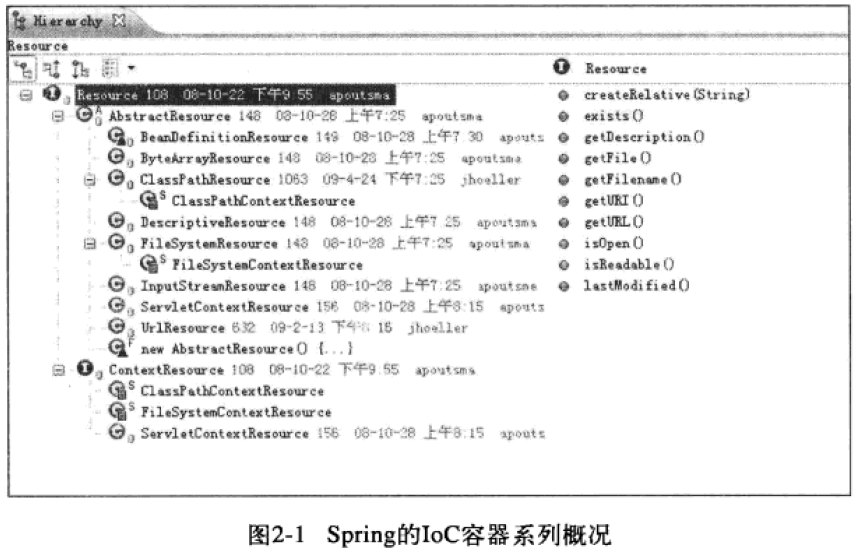
* 设计组件时，往往需要调用和引用其他组件的服务。这种依赖关系可以固化在组件设计中(**我似乎明白了为什么要写配置文件了！**)，但是修改起来非常麻烦
* 注入实现中,主要的实现方式
  + 接口注入 （type 1 ioc）
  + Setter注入 (type 2 ioc) @spring.main
  + 构造器注入 (type 3 ioc) @spring.main
  + Spring中setter是常见的注入方式，spring还提供了对特定依赖的检查，为了防止注入异常
  + 通过可读文本来完成配置，可视化管理和浏览，耦合关系变动不需要修改java源码([符合开闭准则](http://baike.baidu.com/link?url=n5ubJ8w-MVX3B_p1QqGLGZkLB-W2qM-jLaZ-ZujD7c51mzvFmaO054LYkrykcXP4o5v1b9l91P21p-1-0N2g_K))，结合osgi特性还能提高应用动态部署的能力。

## 2.2 ioc容器系列的设计与实现：beanFactory和applicationContext

* beanFactory 只实现了容器的最基本功能。
* applicationContext应用上下文

### 2.2.1 spring的ioc容器系列

Spring的各种容器相当于商店中的水桶一样，有大有小，有木质的，有金属的有各自的特点。



* 在spring提供的基本ioc容器的接口定义和实现的基础上，spring通过定义beandefinition来管理基于spring的应用中的各种对象以及他们之间的相互依赖关系。
* Beandefinition抽象了我们对bean的定义，是让容器起作用的主要数据类型。也是容器实现依赖反转的核心数据结构。

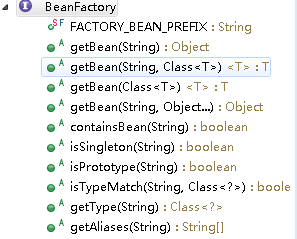


下面对接口简要分析

* beanFactory->HierarchicalBeanFactory->ConfigurableBeanFactory
* applicationContext为核心的接口。
  + beanFactory->listableBeanFactory->applicationContext->webapplicationContext/congruableApplicationContext
* 具体的ioc容器都是在这个接口体系下实现的。
* 这个接口体系是以beanFactory和applicationContext为核心的。

#### beanFactory的应用场景

* 可以通过转义符”&”来得到FactoryBean本身
* facotryBean的实现与设计模式中的工厂模式和修饰器模式类似
* ioc容器中bean的取得是通过名字来索引的。有类型时，检索时过滤类型



#### beanFactory容器的设计原理



xmlbeanFactory 初始化了一个XmlBeanDefinitionReader对象

xmlBeanFactory 🡨 resource 🡨 classpathResouce

* xmlbeanfactory 功能建立在 defaultListableBeanFactory基础之上的
* 使用ioc容器的基本过程
  1. 创建ioc配置文件的抽象资源，这个抽象资源包含了beanDefinition的定义信息
  2. 创建一个beanFactory，这里使用的defaultlistableBeanFactory
  3. 创建一个载入beandefinition的读取器，这里使用xmlbeandefinitionreader来载入xml文件形式的beandefinition，通过一个回调配置为BeanFactory
  4. 从定义好的资源位置读入配置信息，具体的解析过程有xmlbeandefinitionReader来完成。

#### applicationContext的应用场景



* 支持不同的信息源。applicationContext扩展了messagesource接口，这些信息源的扩展功能可以支持国际化的实现过程，为开发多语言版本提供服务。
* 访问资源。这一特性体现在resourceloader和resource的支持上，可以从不同的地方获取bean定义资源，尤其是从不同途径得到bean定义信息。一般来说，具体applicationContext都是继承了defaultresourceLoader。因为defaultresourceLoader是abstractapplicationContext的基类。
* 支持应用事件。继承了接口applicationEventPublisher，在上下文中引用了事件机制。
* 在applicationContext中提供的附加服务。

#### applicationContext的设计原理

以常用的FileSystemXmlApplicationContextd的实现为例来说明applicationContext的原理

* 启动ioc容器时refresh()
* 怎样从文件系统中加载xml的bean定义资源。

## 2.3 ioc容器的初始化过程

* Ioc的初始化是由前面的[refresh()](#_applicationContext的设计原理)方法来启动的。这个方法标志这ioc容器的正式启动。
* 启动过程包括
  + Beandefinition的resource定位
  + 载入
  + 注册

概述

* 第一个过程resource定位
  + Beandefinition的资源定位
  + 由resourceloader通过统一的resource接口完成，这个resource对各种形式的beandefinition的使用都提供了统一接口
    - 文件系统可以使用fileSystemResource
    - 在类路径中的bean可以使用ClassPathResource。
* Beandefinition的载入
  + 这个过程是把用户定义好的bean表示成ioc容器内部的数据结构，而这个容器内部的数据结构就是beandefinition
  + 具体来说这个beandefinition就是pojo对象在ioc容器中的抽象，通过beandefinition定义的数据结构，使得ioc容器方便对pojo对象也就是bean进行管理。
* 向ioc容器注册这些beandefinition的过程。
  + 调用beandefinitionregistry。
    - 把载入过程中解析得到的beandefinition向ioc容器注册。
    - 在ioc内部将beandefinition注入到一个hashmap中去。
    - Ico容器就是通过hashmap来持有这些beandefinition数据的。
* Ioc中bean的载入和注入是两个独立的过程。
  + 依赖一般发生在应用第一次通过getBean向容器索取bean的时候。
  + 预实例化配置。(lazyinit)会改变这个属性的依赖注入过程。
  + 通俗的说，lazyinit属性的bean的依赖注入ioc容器初始化的时候就预先完成了，而不是在应用第一次通过getbean获取的时候。

### 2.3.1 beanDefinition的resource定位

* defaultlistableBeanfactory并不能直接使用resource，必须通过beandefinitionreader来对这些信息进行处理。
* applicationContext则有spring已经实现的不同resource的读取器。Defaultlistablebeanfactory只是一个纯粹的容器，需要为它配置特定的读取器才能完成这些功能。



