上品折扣网

系统架构设计说明书

（V1.0）

上品科技发展有限公司

2009 年 12 月

〔目录〕

第 1 章 引言

1.1 文档控制

文档更新记录

日期	更新人	版本	部门	备注
2009-11-01	闫文涛	创建	信息系统部	
2009-12-01	闫文涛	1.0	信息系统部	

文档审核记录

日期	审核人	职务	备注

文档去向记录

日期	接收人	职务	备注

1.2 目的

本文档从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已经对系统的构架方面作出的重要应用决策。

本文档重在帮助人们快速理解本系统的主要架构设计思想。

本文档的另外一个作用是做为概要设计和详细设计的参考文档，通过对系统涉及到的各个技术层面的阐述，来屏蔽概要设计和详细设计阶段遇到的各种技术

难点，通过对架构中各个机制的实现的说明，使概要设计和详细设计集中于系统的功能性需求的设计，非功能需求通过架构中的各项机制得到有效的解决。

1.3 范围

本文档覆盖了上品折扣网的所有子模块，包括：

√上品折扣电子商务网站系统

√上品流量统计系统

√上品广告联盟系统

1.4 缩略语

Framework（框架）

Framework 是一个相关功能的可扩展的对象集，其显著特点是实现了框架的核心和稳定的功能，并包含了允许开发者嵌入可变的功能或扩展功能的机制。

Component

component 是指业务相关的可复用组件。

Webapp

webapp 是指运行于 servlet 容器中的系统访问逻辑。

Httpserver

专指运行静态页面的服务器，如 apache 等。

Webserver

专指带有 servlet container 的服务器，如 tomcat、weblogic 的 web container 等。

Appserver

专指带有 ejb container 的服务器，如 weblogic、websphere 等。

MVC

模型－视图－控制器，是设计模式的一种，它的基本思想是将系统中的数据显示层、业务模型层和逻辑控制层分离来增强系统的可维护性。

Servlet

小型的、与平台无关的 Java 类，由容器管理并编译成与平台无关的字节码，动态地加载到 Web 服务器上，并被 Web 服务器执行。

Persistence Object（PO）

持久性对象，通常用来完成在数据库和 Java 对象之间的同步。

POJO

Plain Old Java Object，简单的普通的 Java 对象。

POJO 仅是一个普通的 Java 对象，并不是 JavaBean、EntityBean、SessionBean，也不是完成任何特定的任务或实现某个 Java 框架，例如：JDBC、EJB、JDO 等的接口

View Object（VO）

视图对象，通常用来完成用户界面和后台业务层对象之间的传递。

JTA

指定事务管理器和分布式事务中涉及的其它系统组件之间的各种高级接口，JTA 允许事务由应用程序本身、应用服务器或者一个外部事务管理器来管理。

JDBC

Java DataBase Connectivity，一个在 Java 中以面向对象的方法来连接数据库的技术。

AOP

面向方面编程。

1.5 参考资料

《需求规格说明书》

第 2 章 架构的目标和约束

上品折扣网是一个基于单品管理的电子商务网站系统。架构必需从各个方面满足实际需求，这是架构的设计目标。同时在设计与开发的过程中严格遵守架构的特殊约束，从而保证可以实现架构设计的最终目标。

2.1 架构的目标

架构的目标包括：高效目标、安全目标、保密目标、最大化的重用、简洁、灵活等。

（一）高效目标

由于系统核心是电子商务网站，系统的访问速度是系统成败的关键，如果系统响应速度过慢，很可能丧失大部分用户。

（二）安全目标

架构从管理安全、系统安全、数据安全三个方面来具体考虑和保障的安全的。管理安全是指建立相应的安全管理制度，最终决定是否安全的是人来决定的。系统安全从网络、硬件、系统软件、应用软件方面考虑如何加强整个系统的安全性。数据安全是指企业数据的安全，包括备份策略，加密等。

（三）保密目标

对于敏感信息实行加密保存，不允许外部应用读取。对于需要在网络上传输的数据，需要加密处理，并在敏感网段考虑使用硬件加密设备进行加密处理。

（四）最大化的重用

重用是减少冗余的一个有效的途径，包括组件级别的重用、组件的实现通常会应用一种或几种设计模式，这些模式与组件的结合使用大大提升了架构的质量。这类组件包括日志组件、报表组件、WEB 组件等。

（五）简洁

把复杂问题简单化是架构设计的一个重要的目标，明确类以及类之间

的职责关系，简洁的好处是明显的，复杂的代价是高昂的，将给开发维护带来诸多的问题。

（六）灵活

架构可扩展性强，能够以最小的代价满足变化，为了满足出入库等不确定的流程的变化处理，在总分公司和直属库系统的架构中设计了工作流引擎。

（七）可靠性

系统采用集群设计实现负载均衡，可以提供高可用的系统，增强容错处理能力。

（八）易维护性

系统在设计时，充分考虑到应用分布于较广的区域，系统的维护和升级将会占用较大的成本，易维护性是系统架构设计的重要原则。

（九）可扩展性

系统在设计时，充分考虑未来公司业务的变化，组织架构的变化，业务流程的变化，将因业务或者需求的变更带来的系统升级和改造，降低到最小的开发量。

2.2 架构的约束

为了实现架构的目标，遵循 RUP 软件开发过程，实行全程的质量控制。在设计策略、开发策略、技术规范、平台规范方面进行约束。

（一）分析、设计策略

采用面向对象的分析和设计方法，UML 建模语言，使用 ROSE 建模工具。

（二）开发策略

开发工具支持，测试驱动开发，J2EE 团队，任务分工。

（三）技术规范

编码规范、数据库设计规范等规范。

（四）平台规范

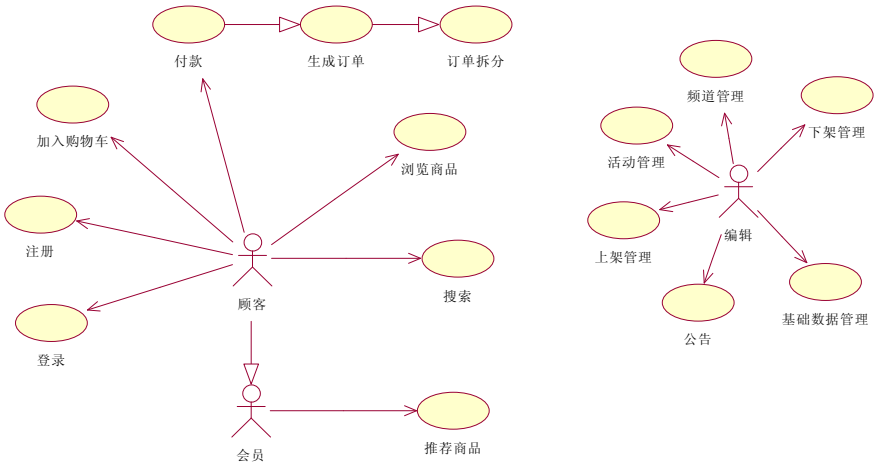
J2EE6 版本，服务器采用 Linux 系统，客户端浏览器支持 IE6 以上版本。基于 Web 页的界面应和 Java 1.6 VM 运行环境相兼容。

第 3 章 架构视图

本系统的软件构架是采用 UML 标准规范进行描述的，分别从用例视图、逻辑视图、部署视图、实施视图几个视图来描述。下面分别介绍几种视图包含哪些类型的模型元素。

3.1 用例视图

用例视图是用来捕获需求的，是将系统所需要完成的功能性需求进行描述的一个视图。用例视图包含包、包关系图、用例图，用例视图中描述软件构架的内容必须包括包关系图，用来描述包之间的依赖关系。包之间的依赖关系表示了业务之间的关联，系统开发的先后顺序。还要指出系统的关键用例，系统的关键用例组成了系统的功能构架。



3.2 逻辑视图

逻辑视图从计算机实现的角度，对需求加以分析，经过分解、合并、抽象等多种方式，形成便于计算机实现的系统功能说明。其中逻辑视图中有子系统、包、关键类，以及类关系图，子系统、包之间的关系，子系统的接口。系统在进行分析时，重点描述业务方面的逻辑，而在设计时，重点结合技术平台描述实现的逻辑。

（一）分析

在描述业务时，对业务进行抽象，分层表示，表示的元素包括

1、包；

2、分析类(界面类、控制类、实体类)，界面类对应界面原型，控制类用于调用实体类封装的业务逻辑，实体类封装实体和方法；

3、用例实现-分析，以时序图、类关系图、协作图来描述，参与对象为分析类以及分析对象。

（二）设计

在描述技术实现时，按照技术架构的设计机制进行分层，例如按照 Spring+Hibernate 模式分层。表示的元素包括：

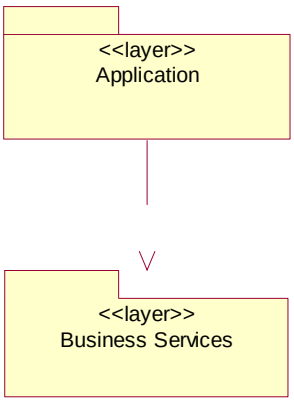
1、子系统、包、接口；

2、设计类(界面类、控制类、实体类)，其中，界面类对应 Controller+JSP, 控制类对应 Domain Object,实体类对应 Pojo，是不包含方法的实体，直接对应 Java 类。

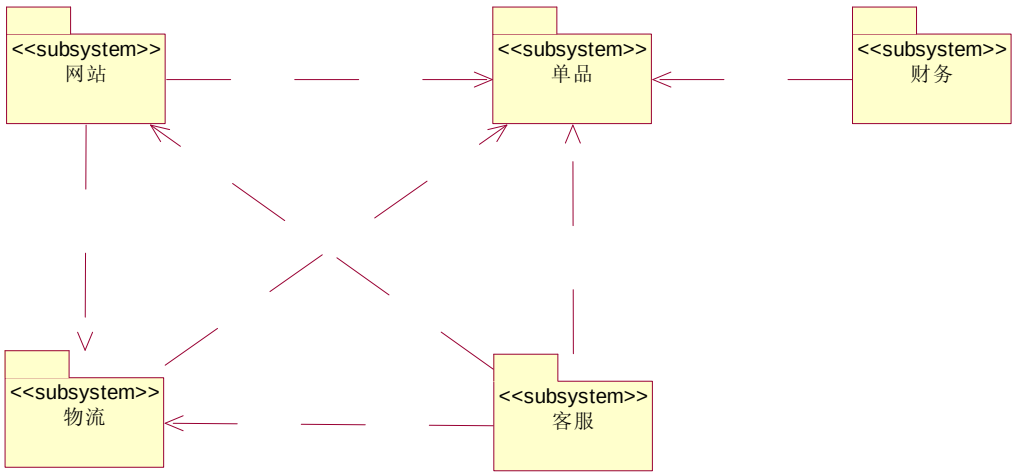
3、用例实现-设计，以时序图，类关系图，协作图来描述。参与对象为设计类以及设计对象。

3.2.1 应用层

根据面向对象的分析和设计思想，按照本系统承建的业务范围，分析零售业务与其他行业的共性和差异，借鉴现有产品的成功经验和设计思想，将系统的应用架构分层设计。最外层为应用层(Application)，用于完成电子商务系统的全面业务。里面一层为通用业务层(Business Services),



3.2.1.1应用系统



3.2.1.2核心用例

在描述系统的架构时，首先识别系统的核心用例，一方面为了为迭代开发打下基础，另一方面为识别系统的骨架提供依据。

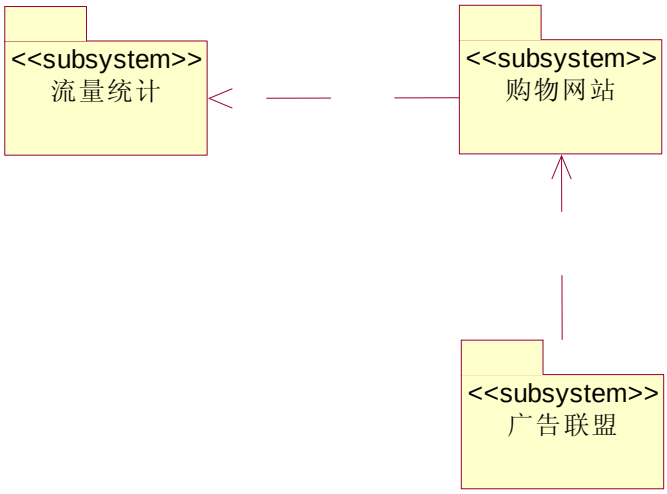
系统	核心用例
网站	商品展示
	搜索
	购物车
	付款结算
	订单管理
流量统计	访问记录
网站联盟	

3.2.1.3关键类

3.2.1.3.1关键类列表

系统	关键类名

3.2.1.3.2类关系图



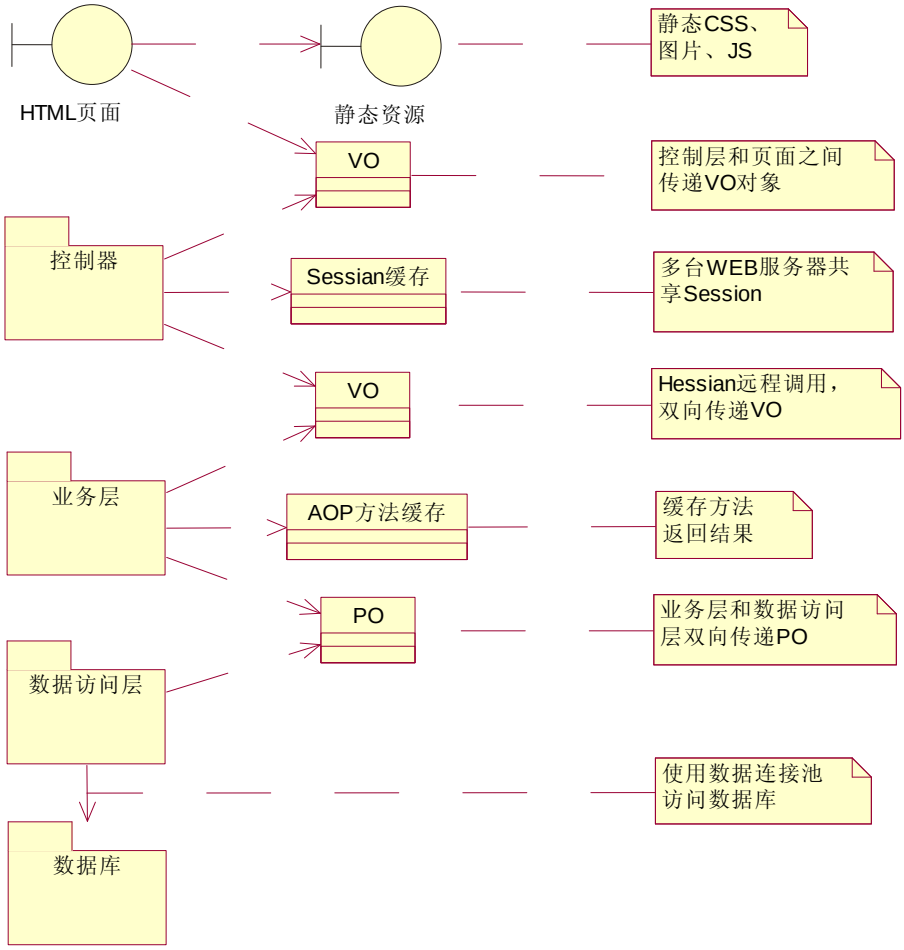
3.2.2 中间层

3.2.2.1中间层概述

系统的中间层是由系统公共组件，公共控制部分组成的，包括开发通用组件，权限组件等。例如：打印组件、报表组件、权限管理、组织结构

3.2.2.2中间层依赖关系

中间层提供了架构基础的各类组件，应用层主要架构在这一层之上，下图展示了中间层各组件之间的依赖关系。



3.2.2.3中间层包以及子系统说明

（一）**打印**

打印组件分为两类，一是对于报表打印采用报表组件的打印功能；第二是对普通的表单或者查询统计信息的打印将直接调用浏览器提供的打印功能。

（二）**用户组织权限管理**

用户组织权限管理采用应用基础框架平台的权限系统，提供用户管理、角色管理、组织管理、权限管理、权限分配管理。

（三）**字典表**

各个模块和公共控制部分使用的字典信息(包括标准化)。

（四）**日志管理**

日志管理包括系统日志管理和业务日志管理，系统日志对系统运行状况进行

记录跟踪；业务日志管理提供对用户使用系统的情况进行记录跟踪。

（五）**异常管理**

架构采用统一的异常管理，便于系统运行期间定位问题。

3.2.3 系统层

由于应用承载着大量的用户访问、业务查询、数据交换等处理，对系统的总体性能要求较高，所以应用对应用服务器软件本身的性能要求同样较高（相应对硬件要求也高）。涉及到第三方系统软件包括：

- 操作系统

  RedHat Linux 5.4 Update3 64bit

- web 服务器

  Nginx、tomcat

- 应用服务器

  Jboss5.1GA

- 缓存

  Memcached、tokyotyrant

- 数据库软件

  Mysql 5

- 消息中间件

  ActiveMQ5.3

- 相关的其他配套软件。

  JDK1.6

3.3 实施视图

实施视图又叫做部件视图，是系统实现的物理模型，实施视图的目的在于将系统的实现逻辑定义成实际的物理单位，实施视图中的模型元素主要包括，包、部件(Component)、部件关系图。

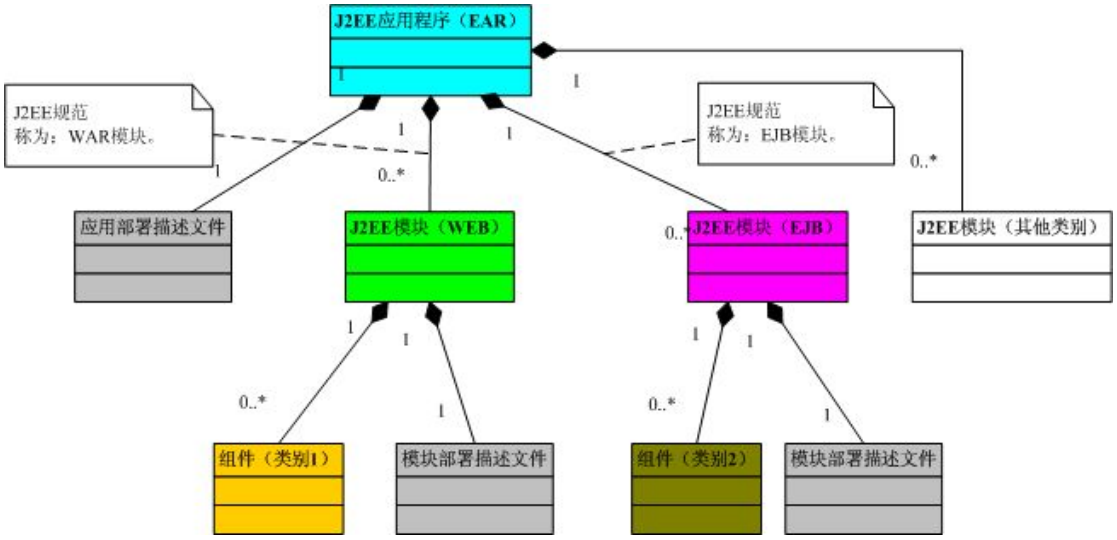
实施视图在描述系统架构时，重点描述对于系统有重要意义的构件。

根据 J2EE 规范的相关规定，整个 J2EE 应用程序的生命周期分为三个阶段：

开发阶段－－组装阶段－－部署（实施）阶段。可见实施是建立在程序组装基础之上的后续工作。

J2EE 应用程序是由一个或多个 J2EE 组件与一个 J2EE 应用部署描述文件组成。部署描述文件以模块的方式列出所包含的组件的说明。

J2EE 模块代表了基本的 J2EE 应用程序的基本组成单元。J2EE 模块包涵了一个或多个 J2EE 组件与一个模块级别的部署描述文件。J2EE 组件是一组类似功能的结构单元，比如：WEB 组件（集中的 JSP、servlet 应用）、EJB 组件、连接器组件等，是开发的基本逻辑结构，他们之间的关系可用下面的图简单表示：



图表 3-1

WEB 模块包含：

1、JSP

2、servlet

3、HTML、CSS 等内容

4、javaBean 等内容

这些内容可以打包成一个扩展名为 war 的文件。

其他类别的模块

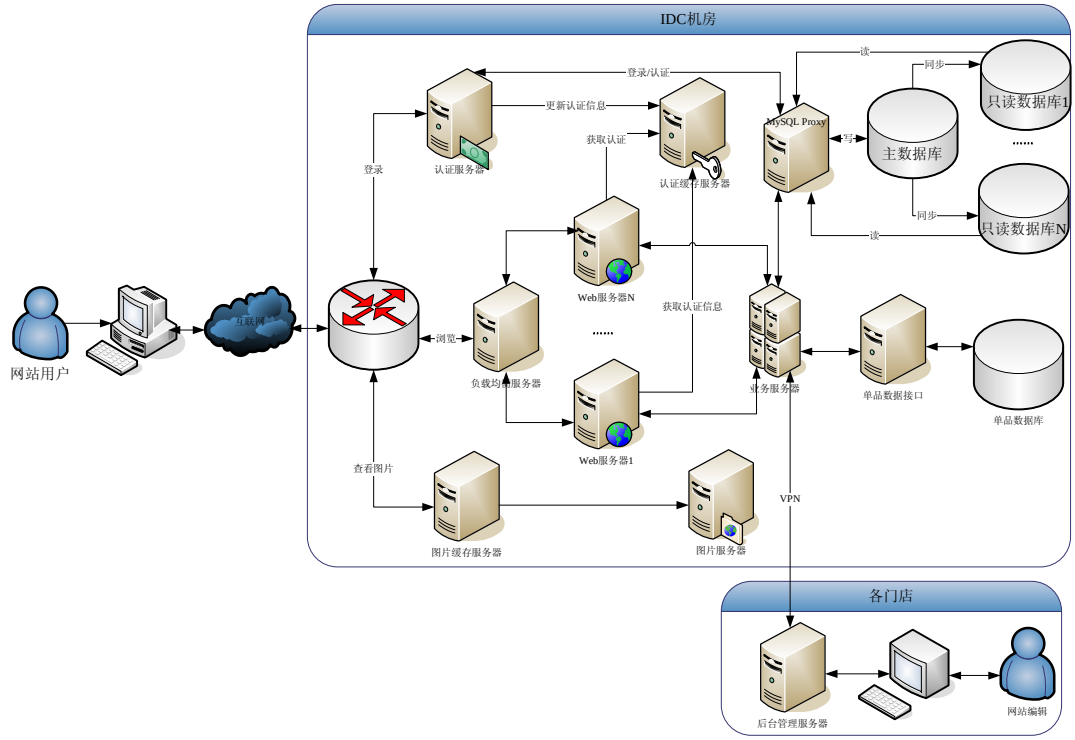
3.4 部署视图

本节说明用来部署和运行该软件的一种或多种物理网络（硬件）配置。对于每种配置，它至少应该指出执行该软件的物理节点（计算机、CPU）及其互连情况（总线连接、LAN 连接、点到点连接等）。另外还要包括进程视图中的各进程

到物理节点的映射。

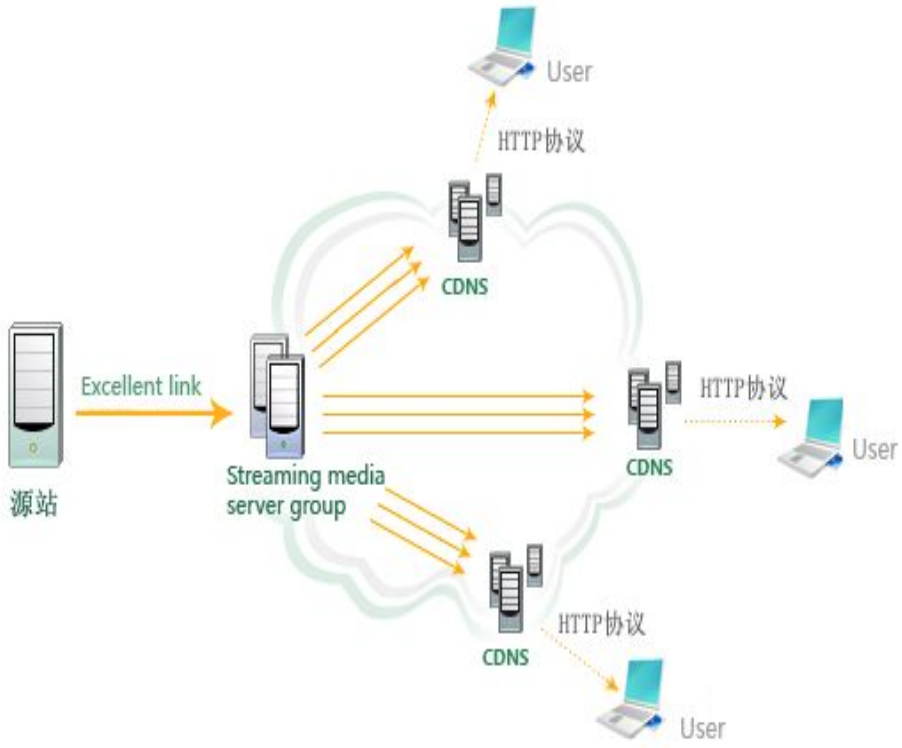
系统部署涉及到系统软件，业务管理信息系统，安全系统、加密系统等诸多方面。

撇开第三方的软件系统（属于系统集成范畴内容），例如操作系统，防火墙部署等因素，可以把整个应用系统的部署简化如下图表示：



注：上述在各个物理节点的应用未必全部列出以实际部署为准。

网站运行一段时间后，流量上升后要考虑 CDN 网络，把静态资源（图片，CSS，JS）等发布到 CDN 网络，降低我们网站本身的压力。CDN 部署如下图：



第 4 章 集成架构

4.1 安全设计

安全设计包括硬件系统和软件系统相关的安全设计方案，其中硬件部分包括加密机设备、防火墙软件硬件设备、软件部分包括防病毒软件等。

（一） 网络安全设计

（二） 应用安全设计

在这里说明上品折扣网本身基于安全方面的考虑。

i. 用户机制

登录控制

权限控制

功能全限

数据权限

密码传输加密处理

ii. 权限管理功能

灵活控制用户的权限。

iii. 在线用户监控

可以实时监控用户的在线情况，具体参见权限设计部分。

（三） 管理安全设计

无论从硬件到软件设计无论如何做，最终决定是否安全是由人的因素决定的。有必要根据实际情况制定相应的安全规范，从规范人的角度加强系统安全性。

4.2 数据存储设计

4.3 负载均衡设计

能进行负载均衡的网络设计结构通常为对称结构，在对称结构中每台服务器都具备等价的地位，都可以单独对外提供服务而无须其他服务器的辅助。然后，可以通过某种技术（包括硬件技术、软件技术），将外部发送来的请求均匀分配到对称结构中的每台服务器上，接收到连接请求的服务器都独立回应客户的请求。在这种结构中，需要建立内容完全一致的 Web 服务器，因此负载均衡技术就成为建立一个高负载 Web 站点的关键性技术，这个技术就是带有均衡策略的服务器集群。

4.4 硬件设计

根据实际情况应用需要进行负载均衡设计。

4.5 软件设计

根据上一节的硬件设计，在每个集群的点上涉及到了第三方的软件产品，明确提出了要求这些软件能够满足集群的应用，这些软件包括：

- 操作系统
- 应用服务器
- 数据库服务器

- 数据传输中间件

要求这些软件能够满足集群的需要。

（一）操作系统集群部署

随机配备的操作系统应该具备支持集群的能力或者配送集群支撑软件。

（二）应用服务器集群部署

集成厂商负责完成部署，提供部署技术支持。

（三）数据库集群部署

集成厂商负责安装配置实现。

（四）上品折扣网集群部署

满足集群的条件：

对于应用服务器的集群支持，要求开发者必须遵从一些规范，如果违反这些规范的约束，那么就可能无法实现集群。

这几类实现方式都不要开发者额外开发应用，只需按照约束开发应用即可，特别是底层架构的支持。

会话状态的复制（session 同步）问题，应用服务器负责完成。

系统没有额外的其他有状态的信息需要同步。

设计上，类不能具有状态静态，除非是一个静态的方法，不能保存类的静态信息。

Web 应用的配置信息，根据 J2EE 相应的规范定义，可以利用 web 容器的特点，将配置信息放置在 web－inf 下供系统使用，这个目录是运行时决定的路径。

通过集群使上品折扣网成为一个高可用性系统。

4.6 容错和灾难恢复设计

目前本期项目中不予考虑灾备的问题，这个问题是个软件与硬件投资的过程，简单的叙述如下：

容错系统通过特殊的硬件、软件设计，如双主机板、冗余磁盘、专有操作系统等方法实现可靠性。容错系统可以达到无停顿的处理效果，缺点是大量的硬件冗余带来高成本，操作系统的专有版本造成系统不够开放。

通过支持集群技术，使系统成为高可用的系统，但是通过集群不能够解决由

于地震，火灾等不可抗拒的外力造成的破坏。最好的办法是建设异地数据备份中心。

（一）在灾难恢复涉及的硬件内容；

网络设备、机器设备、存储设备等。

（二）灾难恢复涉及的软件内容：

文件系统

数据库；

网络地址

应用程序；

系统环境；

用户自定义接管项目。

4.7 容错设计

双机容错系统；

存储采用磁盘阵列；

应用软件支持集群设计；

4.8 灾难恢复设计

灾难恢复实际上是两个问题：

一个是灾难发生后的业务连续性；

一个是灾难恢复；

通常业务连续性是指负责员工具有马上恢复业务的资源（比如另一个网络）；灾难恢复就是把状态恢复到灾难发生前的状态。

如果想保证灾难发生时保证业务的连续性，那么这实际上就成了另一个问题，那就是成本。如果是必须保证业务连续性，额外的硬件软件投入是必须的。

建议：提供集成方面的选择。

4.9 备份设计

4.9.1 数据库备份

定期将文件数据备份到磁带机上。文件数据包括应用的可执行文件，日志文件，配置文件等。

使用数据库本身的功能实现简单的数据备份。

4.9.2 文件系统备份

备份内容：

（一）操作系统

（二）业务管理信息系统

（四）其他必要文件信息

4.10 系统管理

借助已有系统本身的功能进行相应的维护。

日常的维护可以分为几个方面：

- 用户管理
- 权限管理
- 基础数据管理
- 系统参数设置
- 系统日志管理
- 系统接入管理
- 数据备份与恢复
- 远程系统维护

第 5 章 关键性技术设计机制

5.1 技术框架设计

5.1.1 设计思想

面向 Web 应用的 J2EE 应用架构，将应用从逻辑上分为展示层、商业服务层、数据访问层。

把应用从逻辑上分为三层是为了把应用将实现的关注点分离开，以便每层能仅关注某一特定的点，以实现更好的内聚。这三层分别的作用和包含的内容：

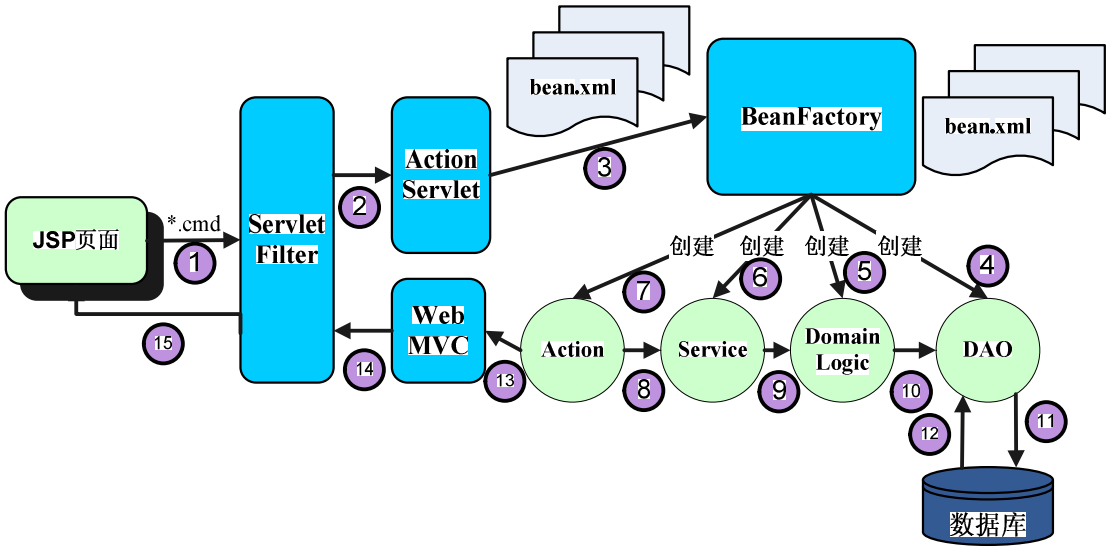
展示层（Presentation layer）：为用户提供良好的界面，提高软件的可操作性，包含用户界面和服务器端对用户请求的提取和页面的转发；

商业逻辑层（Business services layer）：具体的业务逻辑，是整个应用的核心部分。

数据访问层（Data access layer）：访问持久存储（通常是一个或多个关系型数据库）的对象。

这样做了之后，展示层仅关注用户界面，商业逻辑层仅关注某个特定业务该如何实现，数据访问层仅关心如何能使前台数据与数据库中的数据同步。那么显然数据库访问的 SQL 语句是不可能出现在商业逻辑层的，各层的工作就更专了。

对框架做了原理上的介绍后，下面通过图示的方式展示一个完整的处理流程：



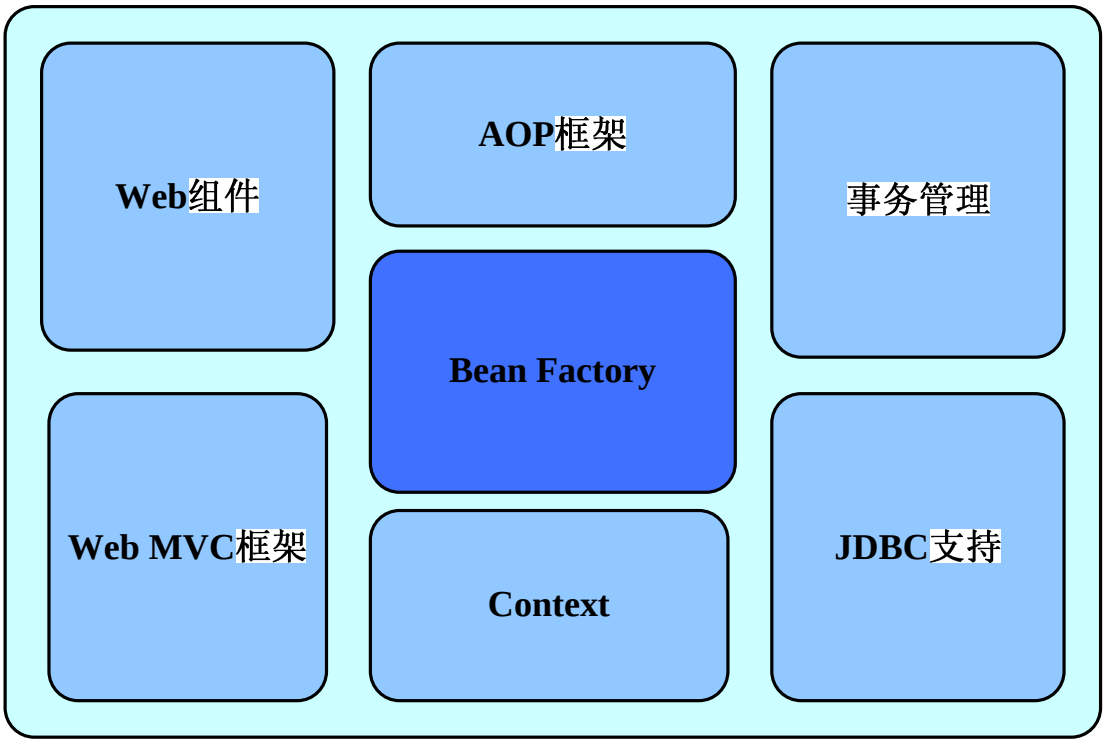
图表 5-1

现在对整个流程简单的描述：用户在页面操作后，向服务器发送必要业务请求，用户的请求发送到服务器上后先通过 Filter（Servlet Filter）对用户请求进行验证，通过验证继续下面的处理，然后发送到 Web 控制器（ActionServlet）上，Web 控制器根据配置信息到 BeanFactory 去请求 Action 处理实例，BeanFactory 又根据配置文件的依赖关系将 DAO、Domain Logic、Service、Action 实例依次创建出来，之后就执行业务处理，将业务处理结果再返回给 JSP 页面。

从逻辑上分，JSP 和 Action 属于展示层，Service 和 Domain Logic 属于商业逻辑层，DAO 属于数据访问层。

5.1.2 设计方案

框架组件结构图如下：



图表 5-2

这套框架以 Bean Factory 为核心，提供了 AOP 框架、Web 组件、Web MVC 框架、Context、事务管理、JDBC 支持等。各种组件之间没有任何依赖关系，就类似搭积木一样任意组合来使用。

Bean Factory 是遵循被公认的 IoC 原则而实现的。IoC（Inversion of Control）的另外一个知名同义词为 DIP(Dependency Inversion Principle)。在企业应用系统中使用 IoC 模式的一个主要目的是为了降低应用系统的耦合，使得系统：

1. 可重用更多的类
2. 类更容易测试
3. 系统更容易组装和配置
4. 对基于接口开发的更好的支持
5. 扩展性更好

所有的在服务器端的程序（Action、Service、Dao 等）都是由 IoC 容器负责管理，具有相互关系的实例之间不需要直接通过某种途径去建立实例关系，而是由 IoC 容器完成，这样就把不属于业务的实例关系从实例中剥离出来，具体开发者仅关心如何去实现需求。

5.1.3 使用方法

技术框架是为了更好的解决业务的问题而抽象出来的，支持业务设计和开发的最理想的模式是在设计和开发过程不关心任何框架的具体实现，整套框架已经实现了这个目标，在设计和开发过程中已经脱离了 API 的依赖。

从 OOA 到 OOD，框架仅提供一些约束，而不给予任何限制。OOA 阶段的实体类、控制类、界面类可以对应到相应的部分，其中界面类的展示部分映射为 JSP 页面，界面类内容的提取部分映射为 Action 对象，实体类的属性被抽取形成 POJO 类，实体类的商业逻辑处理映射为 Domain Logic，其中 Domain Logic 中的数据库操作抽象为 DAO 对象，而控制类则映射到 Service 对象。

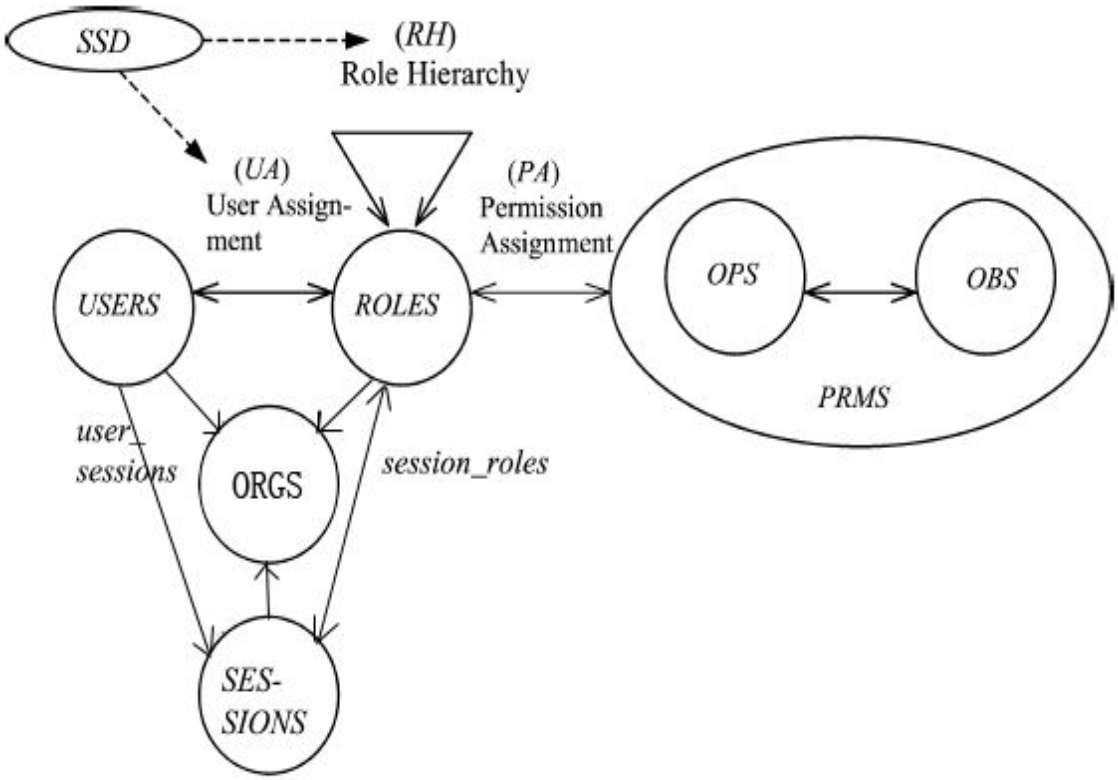
在编码实现时同样只需关注业务实现本身，可以在完全不了解框架的情况下完成实际业务。具体的对象的实例化已经被框架完成，具体对象之间的依赖关系是通过配置文件来配置完成的。这样降低业务和框架的耦合，实现业务组件可复用，可插拔。

5.2 权限管理设计

5.2.1 设计思想

以美国国家信息技术标准委员会制定的 RBAC96（Role Based Access Control，基于角色的访问控制）标准为基础，并根据企业管理特点将组织的概念合理的加入进去，以形成适合企业管理的权限控制模型。并在权限验证的时候使用 OASIS eXtensible Access Control Markup Language (XACML)进行权限验证，以支持访问控制的扩展。

在 RBAC 标准仅是一个通用的权限管理模型，只考虑了用户、角色、权限三者之间的关系。在企业管理应用中，组织是在权限系统中发挥着非常重要的作用，每个用户一定得在某个组织环境下工作，而角色也是存在于一个或多个组织内，因此为适应企业管理类应用，就对 RBAC 参考模型做了稍微的扩展。目前的权限模型如下图：



现在权限模型的内容进行详细的解释。其中：

- OBS：基本对象，权限系统需要保护的资源，例如：入库单；
- OPS：在对象上的操作，例如：修改；
- PRMS：权限集合，单独一个操作是没有任何意义的，只有当操作作用在对象上才有用，操作和对象的组合就形成了权限，例如：修改入库单；
- ROLES：存在的角色；
- Permission Assisgnment：权限分配，为了管理方便权限只能被分配到角色上；
- USERS：系统的用户；
- User Assisgnment：用户要操作系统的功能，那么用户必须具有相应的权限，而权限又不能直接分配给用户，因此必须把角色分配给用户。
- 权限系统中支持的标准有：ISO17799、ISO7498-2、XACML、JAAS。

全面支持 ISO7498-2 定义的各项安全服务，包括：

- 实体鉴别（Entity Authentication）

- 数据保密性（Data Confidentiality）
- 数据完整性（Data Integrity）
- 防抵赖（Non-repudiation）
- 访问控制（Access Control）

在实体鉴别（Entity Authentication）层面支持基于传统关系数据库的认证、基于 LDAP 的身份认证、支持单点登录（SSO）、支持数字证书。

在数据保密性（Data Confidentiality）层面支持 MD5/SHA 加密算法，用户可根据自身安全需要灵活扩展自己的加密算法。

数据完整性（Data Integrity）层面支持数字签名和消息摘要。

对于防抵赖（Non-repudiation），权限系统提供了功能强大的审计功能，捕获安全性相关事件记录的操作，以利于用户发现系统中可能存在的问题，预防和追究非法操作。

权限模型的实现被分为两部分：

- 第一部分是权限设置：权限设置通过先设置角色以及角色权限，然后维护用户。
- 第二部分是访问控制：访问控制是动态验证用户是否有权限。

5.2.2 权限设置

权限管理必须跟组织结构结合，系统管理权限的作用范围是本级组织结构辖区及其所管辖的下级组织结构，业务功能权限的作用范围只限于本级组织结构，向下没有延伸力。

系统中永久存在的角色是超级管理员和系统管理员，超级管理员拥有系统内部全部的功能权限。

超级管理员和系统管理员可以：

1.为公司创建管理员用户。

2. 为公司创建业务角色。

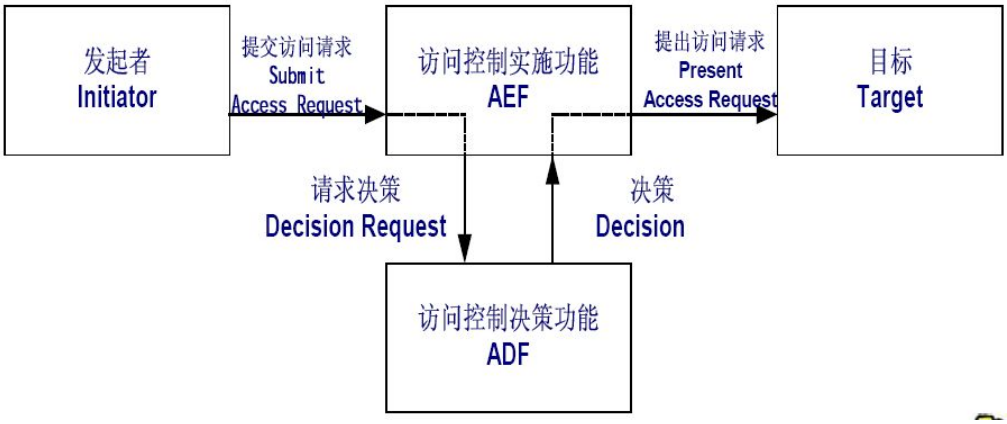
3. 撤消现有用户使用权限。

4. 角色功能权限的灵活分配。

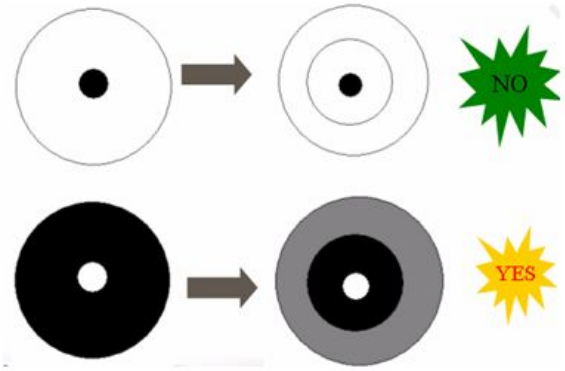
5.2.3 访问控制

访问控制是为了限制访问主体（用户、进程、服务等）对访问客体（文件、系统等）的访问权限，从而使计算机系统在合法范围内使用；决定用户能做什么，也决定代表一定用户利益的程序能做什么。

访问控制的整体模块设计图如下：



根据 ISO-17799 的规定，访问控制规定的建立必须基于如下前提“除非得到明白的许可，否则一般必须禁止此类行为”，而不是更弱的标准“除非被明令禁止，否则所有的行为都是允许的”。



本系统的访问控制遵循如下基本原则：

1) 访问控制基于 NIST-BRAC(基于角色的访问控制模型)规范（静态约束级）。

访问控制的资源分为功能资源和数据资源。功能资源分为业务功能和系统功

能，角色分系统角色和业务角色，系统资源不能授予业务角色，该原则适用于角色继承的情形。根据 ISO17799 的规定，在“没有明确表示禁止的就应当被允许”和“没有明确表示允许的就应当被禁止”两种授权策略中我们选择“没有明确表示禁止的就应当被允许”策略进行权限管理。

5.3 持久化设计

本系统的特点：大量依赖于数据库系统，其中涉及到批量数据的存取，以及多表的关联等，需要出非常复杂的报表。持久化的总体设计：遵循 Domain Model 的思想，采用 DAO 模式加 JDBC 的支持完成整个持久化。

Domain Model（领域模型），合并了行为和数据的领域的对象模型。采用普通的 POJO 方式实现。POJO 在创建 Domain Model 中非常有用。对比其他的 Bean 或特殊的 Java 对象，POJO 不存在任何限制。

DAO 是数据访问对象（Data Access Object）的缩写，DAO 是 J2EE 中非常重要的模式。商业逻辑对象不需要关心数据获取和存储的细节，只需要关注真正的商业逻辑。DAO 必须实现特定的持久策略，并提供通用的 DAO 接口供商业逻辑对象使用。

“JDBC 支持”是对 JDK 提供的 JDBC 的一个对象化的封装，解决了面向对象的语言与关系型数据之间的衔接。

5.4 事务处理设计

在基于数据库的业务系统中事务处理是非常关键的一个环节，几乎每一个操作都会涉及到事务处理。

在 Java 中传统的事务处理是采用 JDBC 所提供的 API 方式。传统的事务一般仅是指的是单个数据库的事务，随着系统的不断完善，各种存储介质都有可能出现事务的问题。传统的事务处理的实现是将事务代码写在每一段需要事务处理的地方，而现在采用声明式事务控制，在配置文件中实现灵活的事务边界划分。

系统提供的新的事务处理模式就是要解决所有的已知的问题，这就需要把事务处理完全抽象出来。考察 JTA、JDBC、Hibernate 数据库层 和 JDO 各自的事务处理模型，提供一致的事务管理的 API，并使用全新的面向方面

（Aspect-oriented）的思想将事务处理理解为一个方面（Aspect），结合 AOP 框架做到声明式事务。这样做就考虑了各种事务处理模式的无关性，而且摆脱了事务代码的编写，也就随即使得事务与业务分离。利用 AOP 实现的事务处理就是声明式事务，这种模式就解决了业务的不断扩展所带了的未知的业务组合的问题，而且这样做还能减少重复的编码。

5.5 日志管理设计

5.5.1 日志分类

日志可以被分为：

- 系统日志：用于记录程序内部的调试信息。
- 操作日志：是用户使用系统时对系统的影响的信息。
- 安全日志：是对影响系统安全方面的信息记录。

系统日志的目的是为了让程序开发工程师容易进行调试，这些日志一般某种特定的模式（例如：输出到文件中、显示在标准输出设备上等），但不管采用何种模式都会让程序多走一些与业务无关的步骤，而且还可能受相应的 IO 的限制。Apache 提供的 commons-logging 加 Log4J 的方式。具体的日志内容采用在程序中写，然后通过配置文件配置是否记录，并可随时控制记录到哪个级别的信息。

操作日志是为了让用户在系统中的操作有依据可查。操作日志同业务相关，架构提供一个 API，供程序开发人员调用，仅需要传入具体的参数。系统提供日志的查询和导出功能。应用基础框架平台在后台提供日志处理功能。

安全日志是为了让发现系统安全的隐患和被攻击的状况。

5.5.2 日志审计

管理日志的目的是为了记录系统运行的轨迹，以便系统管理员发现问题并解决问题。

- 记录日志：系统记录所有的系统日志和操作日志信息。
- 上传日志：日志信息可以自动或手动上传到总、分公司。
- 管理日志：为系统管理员提供日志的分析、查询和导出功能。

为保证业务系统原始数据或作业流程的安全性，系统设计除了在业务流程中提供业务审核外，信息系统日志还提供审计功能，必要的服务有如下几项：

- 在线监控。该服务提供了实时的在线用户监控功能，管理员可以随时监控系统的使用情况，并可在紧急情况下终止用户会话。
- 用户登录历史记录查询。在线用户历史记录提供了追踪用户历史使用系统记录的功能。管理员可根据认证记录追踪用户在登陆系统过程中的所做的具体操作记录。
- 日志查询。为了实施防抵赖，系统对于用户在使用系统期间的关键操作做了审计记录，管理员可以根据需要实时查看审计日志，监控系统资源的使用情况，以防范和化解系统安全风险。
- 针对数据交换过程中的审计日志，系统提供数据恢复功能。

5.6 错误处理设计

错误处理在整个系统中是较为关键的环节，错误也分为两类：业务类和系统类。其中系统类主要是系统级的严重问题，例如数据库无法连接、内存溢出等严重问题等，这些问题大多无法立即解决，而且也不受用户所控制；业务类错误，主要由用户操作所引发的问题，几乎所有问题都能重现。

关于系统类错误没有什么严重级别的区分，都将影响系统的正常运行，只有解决了系统错误才能让系统正常运行，特点是出现一个错误就将影响整个系统，因此这类错误就直接转向一个错误页面，并在页面内提示出现的错误，并提醒由系统管理员解决这个问题。

业务类错误是可控的，这类错误仅影响当前用户，对其他用户无影响。

不管是系统类错误还是业务类错误的捕获过程都是相同的，统一对错误信息码进行管理，以便系统管理员及时了解错误信息并向有关人员报告并解决。错误信息的提示可以通过配置来扩展。

5.7 查询设计

查询组件主要分为两大部分功能：查询设计和结果展示。

查询设计部分主要完成格式、变量以及公式的设计。其中格式是存放结果展示的样式；变量指出单据内各个单元格的变量名，以及各个变量如何取值；公式是对一些不能直接指明如何取值的变量的进一步处理，例如：合计列的取值公式。组件规定了足够详细的属性设置，以适应大多数的情况。

结果展示是将查询出来的结果展示给用户的整个过程。结果的展示需要相应的程序传入初始值，查询组件先取出查询定义信息，然后根据查询定义到数据库中把能直接取出的数据直接取出来，接着把需要计算的数据计算出来，将直接取出的数据和计算数据一起赋值给一个 Java 对象，然后将 Java 对象传给的展示部分，展示部分根据样式然后结合 Java 对象把整个信息完全展示出来。

第 6 章 架构质量

在前面的逻辑视图的描述中，从多个方面阐述架构的基本结构与功能。这些基本的结构与功能最终实现与提交物品的表现形式就是以特定目录存在的物理文件，这些特定的目录结构就是 java 程序代码的基本结构称之为“包”。所以包的结构基本展现了架构的结构。这些包间的关系我们采用松耦合接口设计，因为耦合度的高低决定了架构灵活性。而在每个包内通过适当的抽象与使用接口，又增加了每个包又变得灵活。另外我们遵从简洁的原则设计，简化代码，一个十分复杂的结构势必影响其灵活性与可维护性，从而影响到整个架构的品质。

在开发方法上我们吸取当今最佳的软件开发实践，控制架构质量，遵循以下几个基本原则：

（一）坚持每日构建

每日构建的目的是每日都有一个可发布运行的版本，这个当架构出具雏形能够满足基本运行条件后，即开始这个过程，这是一个完全的过程，非增量的方式对所有以前的代码进行编译发布，代码来源于版本控制系统，这个过程是自动完成的。

（二）测试驱动开发

先写对于功能的测试，根据这个测试在开发相应的代码，所有提交到版本控制系统的代码必须先经过本地的测试。本的代码在每次测试前需要同版本控制系统同步，在执行本的测试，只有所有测试全部通过方可提交到版本控制系统里。

（三）面向组件编程

架构是由一系列的基本的功能组件组成的，功能组件本身的规模不等，这些规模是由具体的功能的负杂度确定的，在这个基础上搭建其他应用。

系统包含：日志组件、报表组件、DAO 组件、基本服务组件等。这些组件相互配合完成更复杂的功能，从而构建一个更加负杂的系统。

6.1 性能

（一）软件设计：

架构能够满足垂直伸缩性的要求。架构支持动态的分配内存，保留一定的物理内存满足机器的基本运行要求，必要时可通过增加系统物理内存满足垂直扩展要求。

架构满足水平伸缩性要求，支持负载均衡。

高并发的环境下，一定的硬件条件下势必要降低系统的响应时间，基于此架构设计考虑到对集群的支持，从技术角度考虑，某些具体实现方式与集群是矛盾的，必须遵循集群的相应约束，比如规范各个物理配置文件都是相对路径，除了 Session 外没有其他缓存信息，除去了在多个服务器之间信息同步的问题，这个同步操作是可以从技术角度实现的，同步机制很好的保证了多个虚拟机的数据一致性，这需要基于广播方式的通讯，但是基于多个 JVM 的同步操作同时也降低了系统的响应性能，这就是性能与数据一致性之间存在矛盾，同集群的横向扩展性能是矛盾的。

对于开销大的操作进行优化，以提高性能，减少各种资源的占用，我门采用可靠的高效的日志记录系统，避免频繁的 I/O 操作降低系统的响应时间。

（二）硬件设计：

在总公司应用部分可以采取如下策略：

- 采用千兆光纤网，解决网络通讯瓶颈。
- 采用 SAN 高速存储架构。
- 采用集群设计。

直属库可以采用如下的策略：

对于直属库可以采用双机的部署策略将压力分担到两台机器上。对于规模较小的直属库可以采用单台服务器部署策略，系统支持这种变化，对原有程序不需要做任何改动，改变配置即可办到。

6.2 可扩展性

架构系统采用松耦合，组件化设计策略，保证架构系统有良好的可扩展性，组件采用可定制策略改变行为，新增功能不会影响到已经存在的原有功能。

架构封装了对数据库的访问操作，在 JDBC 的基础之上进一层的抽象，使得编码量大大减少，这层采用的是称为 DAO 模式实现的，在架构里面表现的就是一个或多个类。在 DAO 前面的 Services 有效的封装了对 DAO 的访问，一个或者多个 Services 组合在一起完成特定的功能，因为已经将数据访问与实际的业务操作有效的分离与组合，大大增加了灵活性。

数据交换、设备接口采用 XML 格式，保持最大的灵活性。

6.3 可移值性

系统软件架构严格遵从 Sun 的 J2EE1.4 规范开发设计，保证在支持 J2EE1.4 规范的主流的应用服务器之间自由移植。系统也充分考虑非商业应用服务器的使用要求，比如与开源的 Jboss，Tomcat 兼容，理论上只要这些应用服务器经过了 Sun 的 J2EE1.4 的规范兼容性测试，我们就可以支持。

表现层实现策略：采用标准的 HTML4.0 规范，Jsp1.2 规范，Javacript1.2。

后台实现策略：连接数据库采用 JDBC3.0 规范。

通过遵循这些规范，保证系统对所有通过 Sun J2EE1.4 认证的应用服务器保

持高度兼容，从而保证了健壮的移植性。

6.4 可靠性

（一）在硬件方面

建议采用 UPS 电源，避免断电造成数据的不可恢复性的物理损坏。

（二）在软件方面

所有的代码都是必须可测试的，支持测试驱动开发是架构的一个显著的特点。

采用成熟的可靠的技术，所有技术都是经过实践检验过的。

保证所有可编译可执行代码每次发布都是一次完全编译，而不是增量编译，坚持每日构建及早发现问题。

另外在总分公司应用集群技术，基于磁盘阵列的数据存储设计。在规模较大的直属库通过硬盘 raid 方式增强数据安全性。

6.5 可维护性

基于 MVC 三层结构设计，三层之间松耦合设计，保证各个层面之间的接口稳定，将变化封装在每层的内部，不会出现牵一发而动全身的问题，对某个组件的修改不会影响到其他的组件。

严格的版本控制策略，建立有效的补丁分发机制，制作安装脚本，减少人为干预出错的概率。

6.6 安全性

在网络部署上有防火墙，加密机等设备。在软件方面采取了相应的策略，在服务器端的密码验证，用户权限管理，动态验证码技术。基于传输的 SSL 加密技术的应用，从各个层面充分考虑了安全的需要。

6.7 易用性

简洁化设计，对外接口简单，通用功能组件化，页面模板化，代码支持自动生成功能。

6.8 设计开发的高效性

提供统一对外接口，通过接口设计，简化应用，各种组件，web 组件，页面模版，使得开发者专心于业务逻辑的开发。

系统架构是基于 MVC 的三层结构设计，各个层之间通过简单的接口实现通讯，一个功能可以按照三层结构的组成由不同的开发人员合作完成，美工人员发挥制作页面方面的特长，Java 程序员发挥开发后台逻辑的优势，优化了人员分工，做到专业高效，同时有效地保证了代码的质量。