继教网教学管理平台（V3）

（E－Learning）1.1

架构设计

目 录

修订记录

修订日期	版本号	描述	合作者	修订人
06/09/2009	0.1	Initial draft		Leo Zheng
06/23/2009	0.2	添加内存数据库具体操作部分，分开读写分离和数据库分片		Leo Zheng
07/15/2009	0.3	调整文档结构及增加新平台业务目标、基本架构目标、重要架构考虑等多方面内容		Frank Zou
08/01/2009	0.4	修改读写分离图，添加数据库描述表格，修改部署图，远程调用分为远程接口和内部接口，内存数据库使用说明，缓存的一些说明		Leo Zheng
08/15/2009	0.5	添加软件分层结构图，代码结构图，添加数据库分片的图，		Leo Zheng
08/20/2009	0.6	添加附录，根据 Frank 的建议，做了一些调整	Frank Zou	Leo Zheng
08/24/2009	0.7	开发使用的工具列表	Frank Zou	Leo Zheng
08/28/2009	0.9	一些说明性文字的改动		Leo Zheng
09/10/2009	1.0	根据李晓林的反馈，做一些关于数据库分片未来方案的描述	李晓林 Frank Zou	Leo Zheng

1 前言

1.1 业务背景

全国中小学教师继续教育网（以下简称继教网）成立于 2002 年 12 月，是一家根据国家教育行政主管部门要求提供中小学教师远程培训的互联网服务公司。公司成立 6 年来，随着政府对继续教育投入的加大，公司业务不断发展，已经成为中小学教师远程培训领域的主要服务供应商。

伴随着业务的发展，支撑业务发展的软件系统也在不断进化，目前已经完成了两个主要版本，并根据业务需要在持续改进中，为继教网业务的快速发展提供了决定性的技术基础。

不过，随着继教网业务的进一步发展，新的要求不断涌现，现有业务应用系统的局限性对于业务发展的束缚开始显现，在技术架构方面主要表现在以下几方面：

- 缺乏统一的用户数据库

由于现行系统中，所有项目都需要单独对项目中的用户进行注册，造成大量重复数据输入，并且，无法支持继教网目前提出的“网上大学”及“终生教育”等业务理念。

- 每一个项目都需要作大量定制开发

继教网的业务，具体来看是通过一个个相对独立的项目来完成的，而现行的软件系统也是在若干的针对项目设计的软件上发展而来。所有培训项目，从功能与过程上来说，都大同小异，而在细节方面又会有诸多不同，需要定制开发。

在现行系统上处理这个问题的方法，是在原有系统源代码基础上，通过修改原有代码及增加新代码的方式来扩充功能，由此带来了大量不同版本代码需要维护、开发周期较长等不足。

- 高并发支持

为克服现行业务支撑系统的局限性及满足未来业务发展的要求，继教网希望对现行系统的进行再一次升级开发，推出第三版的业务支撑系统 -- 以下简称继教网 eLearning V3 平台。

1.2 新平台业务目标：统一平台

1.2.1 统一平台含义

不同于现行系统中的为每个项目单独开发或定制的单独的业务软件系统，继教网对新系统的定位是一个全新的统一平台，从业务方面讲，主要有以下几方面含义:

- 统一的业务平台

可以在这个平台上运行专题培训、全员培训的业务，以及其它继教网的类似业务

可以多个项目同时进行培训，数据相互独立又可以相互访问。

- 统一的用户数据

用户注册过程和管理的统一标准化，减轻项目前期和运行期间管理和维护工作量。

- 统一的业务组织管理模式

可以将学习中心、班级与省、市、县的各个项目需要的层级管理模式相统一，提供灵活的班级管理和配置功能，在业务层面体现更多的符合具体项目需要的需求。

- 统一平台、独立部署

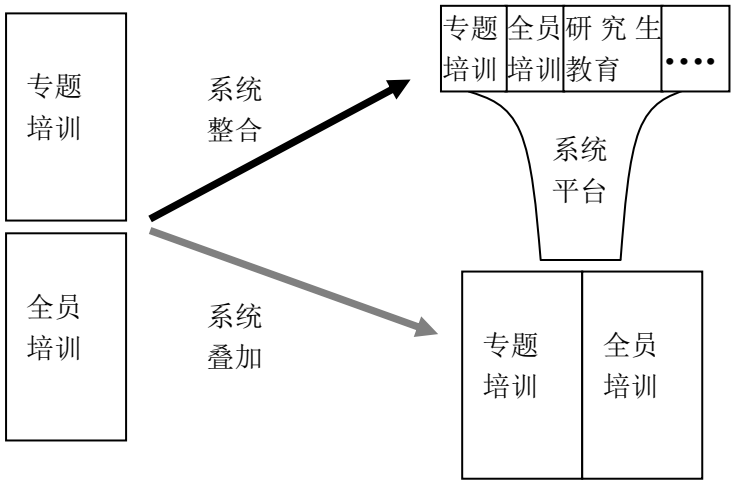
统一的软件平台，但可以独立部署，数据互通。服务器硬件可以独立配置和部署。

- 可定制的门户

提供个人门户、班级门户、专题门户、项目门户，同时可以在门户层面满足每个项目具体的门户页面变化的需求。

1.2.2 统一平台图示

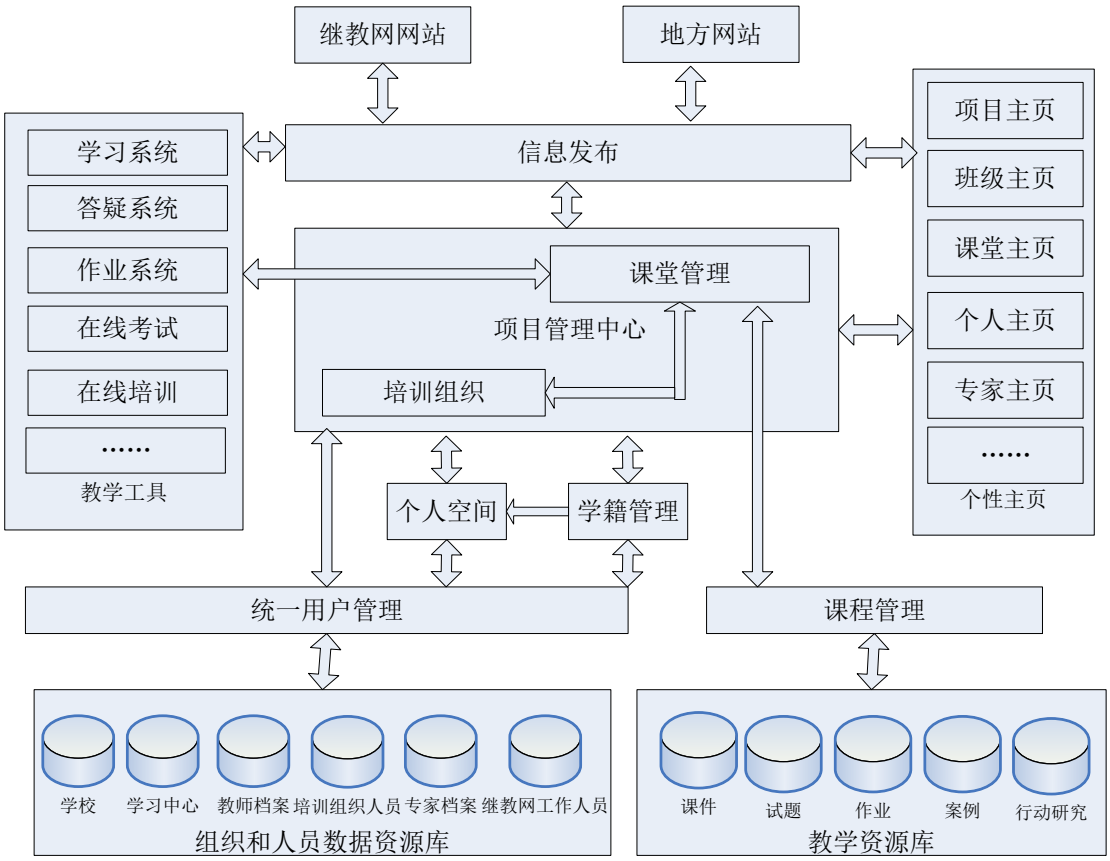
目标平台如何支持继教网现有的以及以后将有的各种形式的业务，如下图所示。



注：图的下半部分是目前的业务支持方式，表现为分离的业务系统；图的上半部分表现出在统一平台支撑下，各种不同的培训业务的开展情况。

1.2.3 新系统的业务模块

新系统建成后，从业务上看，主要的业务模块构成如下图所示：



注：关于业务模块的详细分析，参见业务分析文档。

1.3 本文档目的和范围

此文档主要包含以下几方面内容：

- eLearning V3 平台架构设计的原则与目标
- eLearning V3 平台的总体架构

本文档的具体内容包括系统模块构成、数据库、部署、监控以及性能、伸缩性、安全等架构级的关注要点，目标阅读者是 eLearning V3 平台开发的所有参与者，如客户、PM、PjM、EV、QA 等等。

1.4 术语与缩写解释

编号	术语	解释
1.	数据库分片	把一个项目的数据库分开来存储，可以是不同的物理位置
2.	水平分片	把本来在一个表中的数据，放到不同的表中
3.	垂直分片	把一个项目中不同模块的表，放到不同的地方
4.	Kettle	一个 ETL 工具，在数据仓库应用中，用于数据抽取、转换、装载
5.	AbstractRoutingDataSource	Spring 提供的一个动态数据源选择工具
6.	ReplicationDriver	Mysql 提供的用于读写分离的数据库 JDBC 驱动
7.	Mysql Instance	Mysql Server，里面可以有很多 mysql schema
8.	Mysql Schema	mysql 的一个数据库
9.	DTO	Data Transfer Object 的简称
10.	HTTPServer	提供 http 服务的应用软件
11.	APPServer	提供动态 http 服务的应用软件
12.	Open-session-in-view	这个是 Hibernate 提供的一个功能，可以把 Hibernate 的 Session 在请求开始的时候打开，在请求结束的时候再关闭，这样，程序就可以在请求的任意时候，使用 Hibernate 的 LazyLoad 功能
13.	DataSource	数据源，Java 中定义的一个数据库连接的提供者。
14.	LoadBalance	负载均衡
15.	CAS	一个成熟的开源实现，可以给多种语言，提供多种方式用户统一认证的应用

1.5 参考文档

继教网教学管理平台设计方案（ 洪总）

平台设计建议（昝学新）

eLearning V3 业务分析及架构（李晓林）

需求分析文档（郑贵德）

eLearning V3 业务需求分析报告（姚梁宇等）

2 总体架构

2.1 主要架构挑战

- 高并发访问

由于继教网业务系统面对的潜在用户群十分巨大（中国中小学教师有1000万），参与一个培训项目的人数可能有数十万之巨，所以，如何提高系统的伸缩性（Scalability）以应对短时间内数以万计的访问量是架构设计要考虑的首要问题之一。

为解决这个问题，在架构设计中综合应用了集群、缓存、数据库分片、数据库读写分离、内存数据库等多项专门技术与工具，详情见后文专文描述。

- 平台的扩充性

目前，继教网的主要业务有专题培训、全员培训及研究生培训等等，每一种业务以独立进行的若干个项目的形式来操作。这些不同种类的业务及同一种业务的不同项目之间，有很多相似之处，但也有大大小小的不同的细节。

平台的基本目标是要用一个统一的平台来支持各种现行业务、未来可能的新业务（如对金融行业的培训）以及各种不同的项目。如何在业务分析方面抽象出各种业务与项目的共性，在技术架构方面根据业务共性设计出可扩充的公用组件与基础架构，从而有效支持各种不同的业务及项目，是我们面临的又一个挑战。

2.2 架构设计目标

根据继教网对新平台的业务要求，我们制定平台架构的主要目标如下：

- 可扩充同一平台

对现行业务的支持

新平台建成后，要能够支持现行的所有种类的业务

现行业务的新项目开展时，通常情况通过配置解决一些需求差异，无需开发就能开展业务

新项目的主页等允许定制开发并与平台集成

若新项目差异较大，平台无法满足需求，可在平台提供的API基础上作二次开发

对新业务的支持

新业务系统的开发，能够基于平台提供的API进行，但不能对平台代码产生影响

新的业务纳入平台的时候，能够使用平台上已经积累的资源，包括用户资源及教学资源

- 统一用户管理

继教网平台的所有用户，由一个统一的用户子系统来管理，并中心化存储

用户参加过继教网任何项目后，其身份信息可以用于以后的所有项目

用户在继教网不同系统之间切换时，只需要登录一次

- 高伸缩性

支持单个项目 10 万级别的用户

支持多个项目同时进行

2.3 假设

新平台的建设，我们基于如下一些重要假设。若假设的情况发生变化，或对架构与设计带来较大影响。

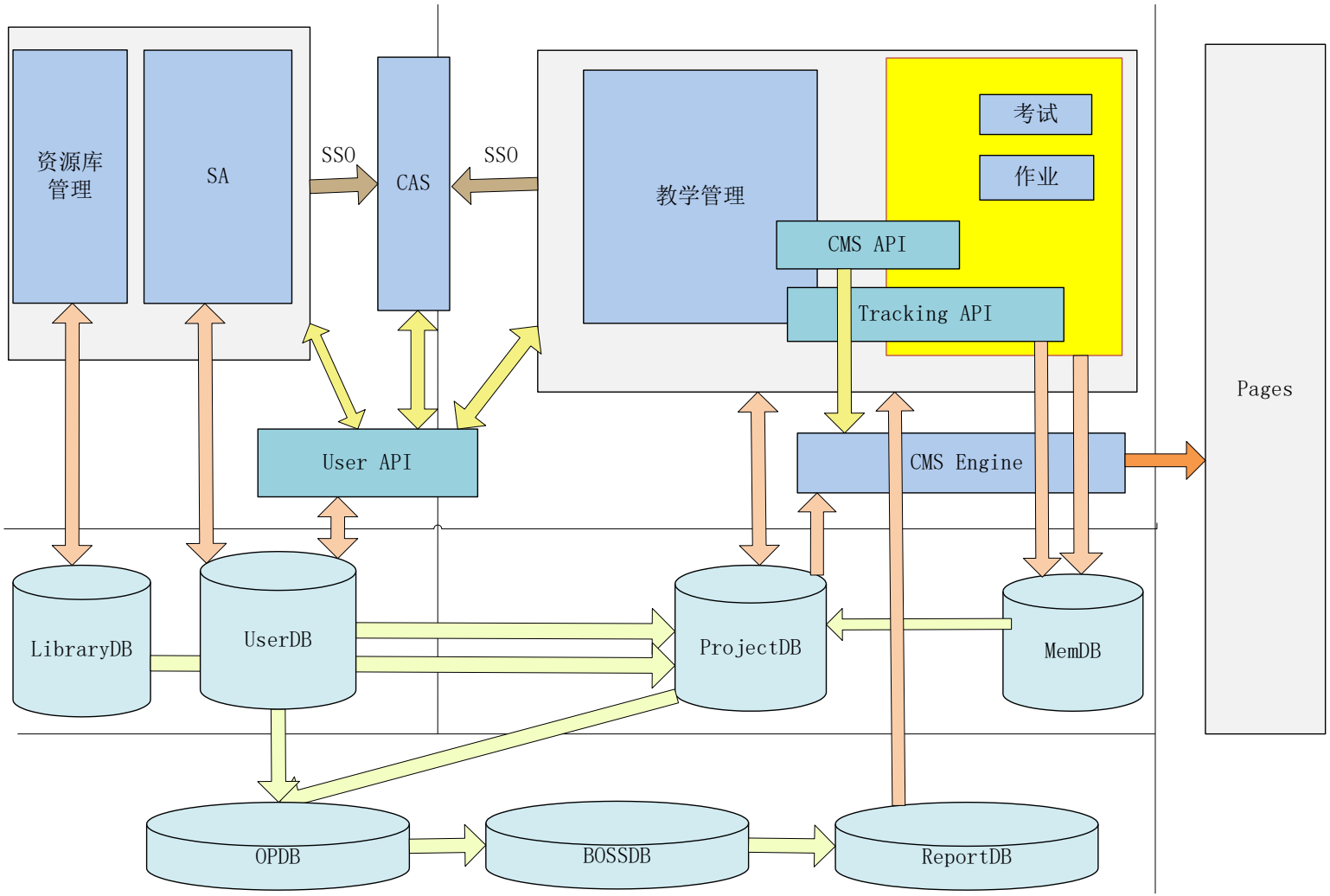
- 单个 IDC（数据中心）

新平台建成后，将和继教网的现行系统部署于同一个 IDC 中，我们目前不支持在多个 IDC 中部署不同模块或项目系统的情况

- 新平台与现行系统的关系

从技术上说，新平台将采用和现行系统完全不同的技术进行开发，涵盖现行系统的所有功能，但是不需要支持与现行系统系统集成与数据迁移。

2.4 总体架构图


资源库
管理
SA
SSO
CAS
SSO
教学管理
考试
作业
CMS API
Tracking API
Pages
User API
CMS Engine
LibraryDB
UserDB
ProjectDB
MemDB
OPDB
BOSSDB
ReportDB

- 系统展现部分

系统展现部分，主要分为管理，认证，教学管理，静态页面 4 个部分。管理部分用于资源的管理，和用户的管理；认证部分做系统用户的统一认证处理；教学管理部分是系统主要业务开展的部分，包含了教学管理，教学工具等所有部分；静态页面，是 CMS Engine 生成的内容，用于对外表现教学进行的情况，以及展现各级学习中心和个人的学习情况。

- 系统数据库部分

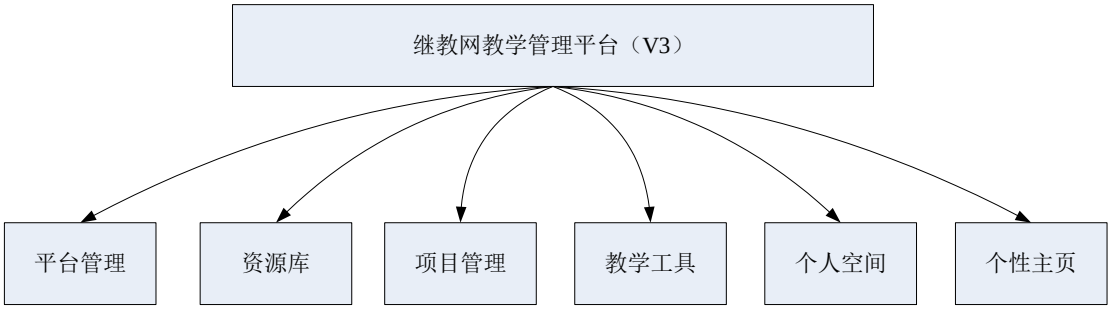
系统数据库部分分为业务数据库和运维数据库。业务数据库是系统日常运行的支撑数据库，主要用于资源、用户、项目运行产生的数据的存储，也有为了解决系统的性能问题，而使用的内存数据库。运维数据库是用户系统长期运行产生的数据的汇总，这些数据可以用于系统信息的综合查询，是“网上大学”以及“终身教育”的有力支撑。

3 架构设计

3.1 应用软件

3.1.1 主要模块/子系统及功能

系统主要实现的模块如下，各个模块又分成很多的子模块，详见各自模块的说明文档。



- 平台管理

平台管理员管理的内容，包括用户管理审核，平台维护，项目创建等功能

- 资源库

积累历史数据的平台，包括教学资源，人力资源的管理，可以在不同项目中实现复用

- 项目管理

设置项目基本信息，包括课程计划，学习中心维护，教师安排等等功能

- 教学工具

教学中的工具，在教学时使用，在项目创建的时候，选择了该项目可以使用的教学工具之后，在项目中就可以使用这些教学工具

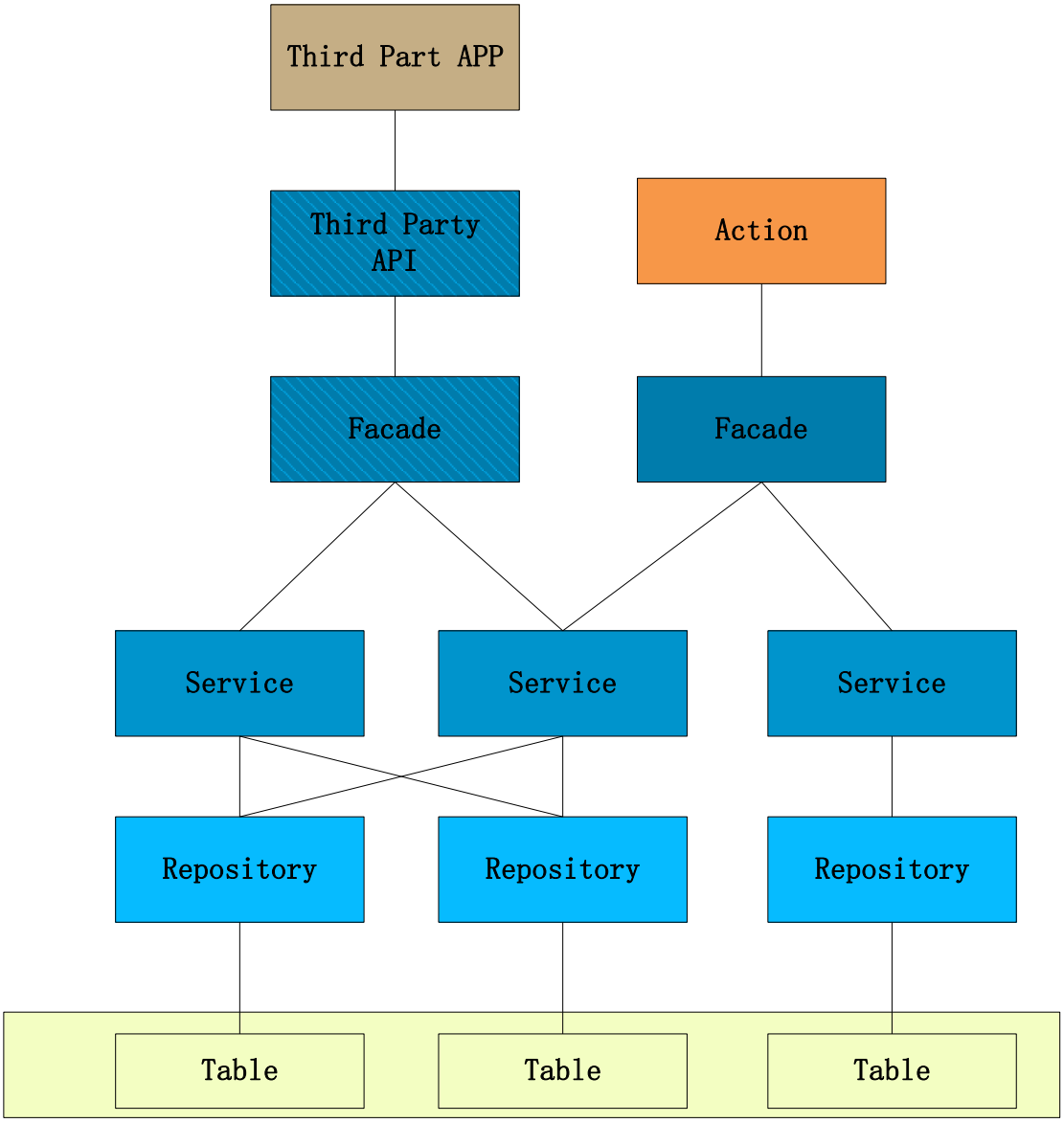
- 个人空间

个人空间，快速进入个人在这个平台中的各个项目，或者操作的门户

- 个性主页

展现项目内容的主页，包括个人，学习中心，课程，以及项目的主页，可以使用系统提供的工具做定制开发

3.1.2 软件分层架构



- 分层结构说明

软件系统分 Repository、Service、Facade、Action 四层

Action 用于收集页面参数,调用 Facade，根据调用结果和相应的业务异常跳转到相应的结果页面 Action 中只能注入一个 Facade，父类 prepare 方法除外，因为这个方法需要处理整体的登录用户需要处理的一些内容

Facade 用于调用 Service，Facade 中可以注入多个 Service，但不可以注入 Repository，Facade 做必要的数据校验，比如 id 值是否为整数，如果校验失败，抛出业务异常

Service 用于调用 Repository，做业务校验，逻辑校验，比如 id 所对应的对象是否存在，校验失败的时候也抛出业务异常，Service 中可以注入多个 Repository

Repository 用于处理数据持久化的相关操作，现在系统数据持久化使用 Hibernate，所以在现行代码中 Hibernate 相关的接口不可以跑到这层之外，包括 Session、Criteria、Query、SQLQuery，如果需要做一些必要的 SQL、HQL、Criteria 组装，也在这层处理

- 分层结构之外的一些代码注意事项

Hibernate 的特性 OpenSessionInView 在系统中是使用的，所以，不必要在任何时候都把 Model 转化成 DTO

DTO 可以在以下情况下使用。包括查询条件 DTO，如果 Model 可以收集所有条件，可以使用 Model；集成 API 的输入输出；Session 中的对象；Struts2 的 JSONResult

3.2 数据库

3.2.1 数据库功能描述

系统使用的数据库，根据项目的情况，现在分成以下几个部分

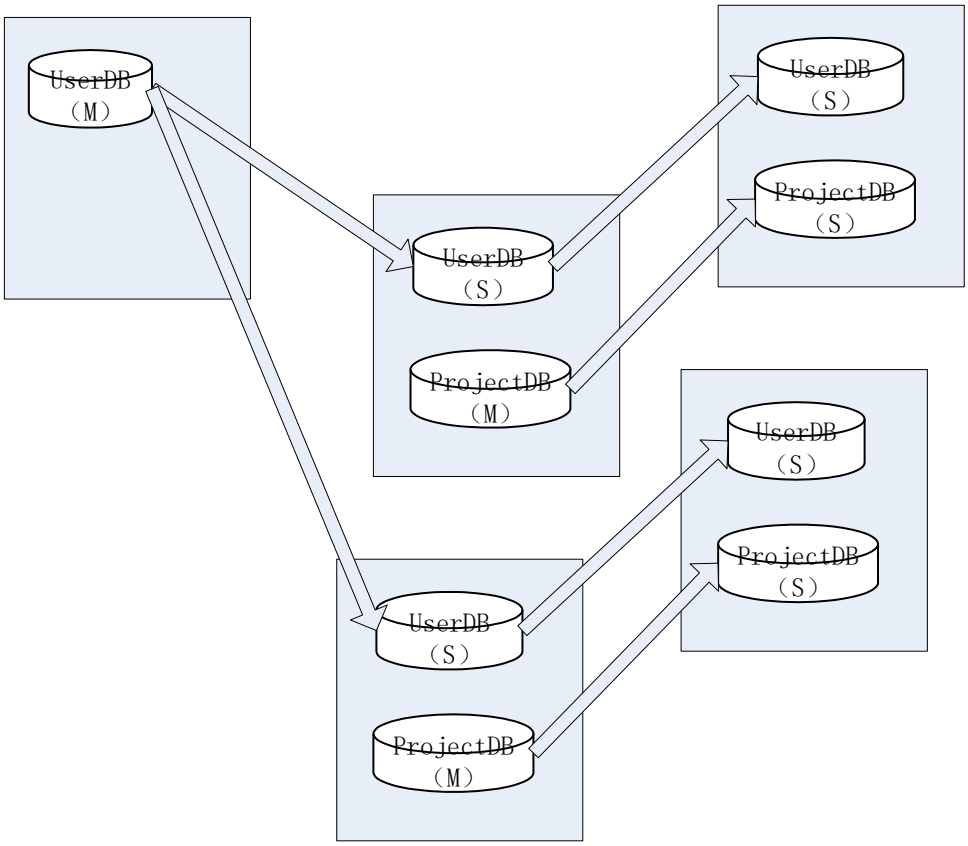
数据库名	功能描述	备注
elearning	LibraryDB，存储资料库信息，如试题库，课程库	全球唯一
elearning	UserDB，存储用户信息，项目概述信息	全球唯一
elearning_portal	UserDB，存储用户的跨项目信息	全球唯一
elearning_prj	ProjectDB，一个项目运行所需要用到的数据，如果有多个项目，就有多个 ProjectDB	根据项目做水平分片
OPDB	OPDB，所有信息的汇总，用于 BOSSDB 的抽取	全球唯一
BOSSDB	BOSSDB，所有信息的统计信息，从 OPDB 加工过来	全球唯一
REPORTDB	ReportDB，需要给项目查询的统计信息，和项目有紧密关系，从 BOSSDB 中抽取出来	全球唯一
无名称	MemDB，内存数据库，用于提高高并发写操作的吞吐能力	各种场景有不同的数据库

3.2.2 数据复制

因为系统中数据库的设计，造成数据的分布式存储，但是在实际业务场景中，不同地方的数据又是有关联的，在处理关联数据的时候，系统可以使用程序接口，也可以把数据复制到需要使用的地方，两种方式各有利弊。在系统中，这两种方式都有存在，根据不同的场景有不同的选择。数据复制方式在系统中主要有以下三种情况，UserDB 到每个 ProjectDB 的复制，UserDB 和 ProjectDB 到 OPDB 的复制，MemDB 到 ProjectDB 的复制。具体详细设计方案如下

- UserDB 到每个 ProjectDB 的复制

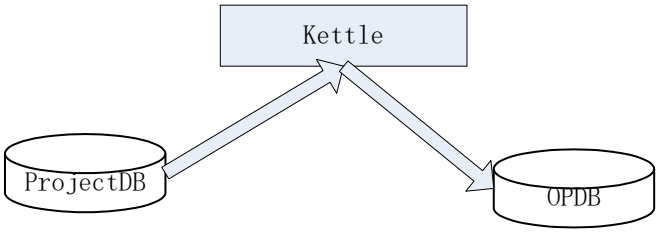
在系统中，资源库管理，用户管理所对应的数据库（UserDB），和项目运行期使用的数据库（ProjectDB），可以不是在同一个 Mysql 的 Instance 上，而 ProjectDB 会频繁的用到 UserDB 中的一些数据，都是做关联查询，如果通过程序接口来做这个关联，效率非常底下，甚至有些业务都无法完成，所以在系统中，通过 Mysql 的 Replication 来把 UserDB 复制到 ProjectDB 相应的 Instance 中，然后在 ProjectDB 中建立 UserDB 的所需要用到的表的 VIEW。另外，由于性能的考虑，需要对数据做读写分离的处理，这里的数据库主备也是通过 Mysql 的 Replication 来实现。具体的复制情况如下图。一个方框表示一个 Mysql 的 Instance，而圆饼框则是 Mysql 的一个数据库。图中 M 和 S 分别表示数据库的主和从，由于查询的需要，UserDB 需要做多级的复制，一来解决数据库复制中，如果从库过多，会过多的消耗主库的性能，而且也解决 Mysql 的数据复制中只能有一个来源的问题。



- UserDB 和 ProjectDB 到 OPDB 的复制

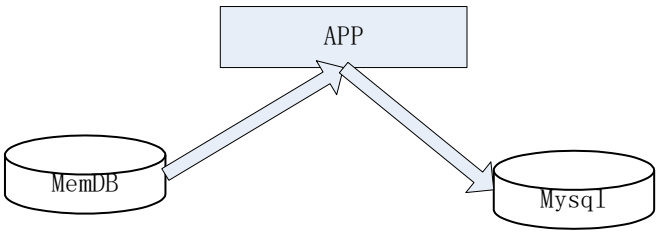
系统的业务数据，因为通过了水平和垂直的切分，分布式存储到不同的地方，虽然在技术上解决了项目的一些性能的问题，但是系统还是需要有一个地方有汇总的数据，给我们做综合查询，做

业务分析支持。但是这个数据不要求是实时的，这里，系统使用 Kettle 来做这个数据迁移的方案，Kettle 是一个成熟的，开源实现的数据抽取、转换、装载的工具，该工具可以通过可视的界面来编辑对数据的抽取和转换过程，并且把这个过程保存成任务，然后通过定时运行这些定义的任务，来完成数据迁移的工作。因为数据传输的过程中，非常依赖网络，我们希望这个时间越短越好，所以不会使用 Kettle 工具来对数据进行转换处理，而使用存储过程来实现，该方案的实现如下图。



- MemDB 到 ProjectDB 的复制

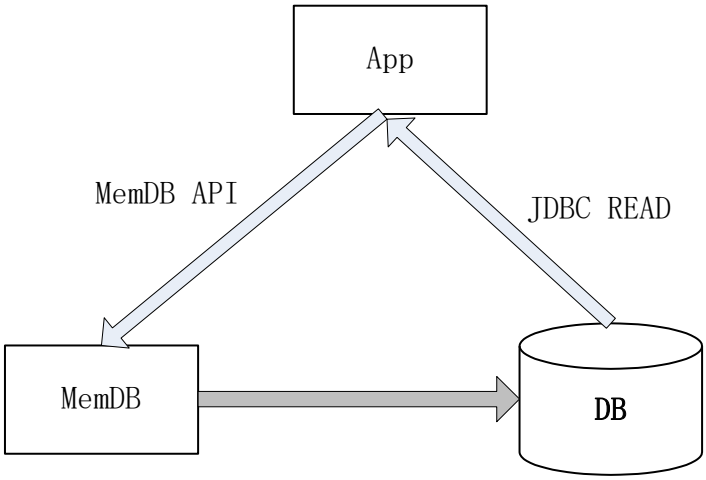
在系统中，因为有些模快有高并发的要求，系统使用 MemDB 来支撑并发的写的要求，内存数据库的具体介绍，请查看内存数据库一节，因为 MemDB 是非结构化存储的问题，在程序做结构化查询的时候，MemDB 不能满足查询的要求，要求程序在把数据写入 MemDB 后，还要把数据迁移到 ProjectDB 中，用于数据后期的查询。MemDB 数据的复制，需要写程序来实现，因为定义的 MemDB 数据都是非实时的，程序中通过 Timer Task 来处理，在复制时，数据是通过批量处理来完成的，比直接单条在数据库中插入，有更好的性能。该复制方案的实现如下图。下图中的 APP 就是我们写的程序， 该程序定时运行，在数据插入 MemDB 之后的一定的时间内，程序会把 MemDB 中的数据迁移到数据库中。



3.2.3 内存数据库

在继教网系统中，存在一些特别高并发的模块，这些模块，使用常规的数据库管理系统，没有办法达到它需要的写的性能要求，这里，系统引入了内存数据库（非结构化存储数据库），来解决这个问题，比如在考试中，大家可能都集中在考前的一小段时间来提交试卷，但是不会马上来查询结果，这个时候我们把数据先存放在内存数据库中，在后期再把数据写入业务数据库。因为存储结构的不同，内存数据库可以有更好的并发能力。

内存数据库的使用模型如下图。在内存数据库的使用上，系统是这么定义的，内存数据库只是用来处理对高并发写的支撑，在后期的结构化查询中，系统还是从业务数据库中。所以对于程序来说，只要处理内存数据库的写，然后系统会根据内存数据库复制方案把数据复制到业务数据库。



通过程序,异步写入数据库

在经过一些比较和试用后，我们使用 Tokyo Cabinet 和 Tokyo Tyrant 来作为系统的内存数据库方案，该选择有以下一些特点，Tokyo Cabinet 是一个高性能的存储，Tokyo Tyrant 是建立在 Tokyo Cabinet 上面的 socket 服务，该方案可以使用主备方式部署，满足我们对 HA 的要求，可以有多种存储方式，让我们在不同的使用场景，使用不同的方案。可以在 64 位机器部署，会有更好的性能表现，有很多成功案例，比如国内的豆瓣网就使用了这个解决方案。

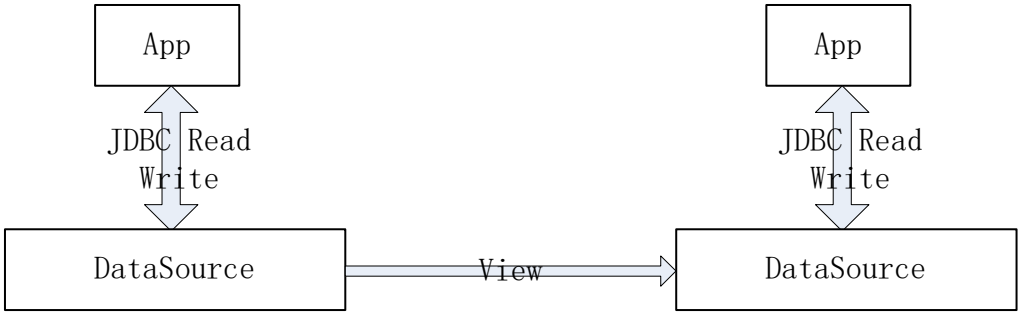
3.2.4 数据库分片

考虑到继教网的用户量，和用户使用习惯，存在大数据量和高并发的要求。并且在不同模块的表现是不一致的，系统采用数据库先垂直分片，后做水平分片的做法。

- 垂直分片

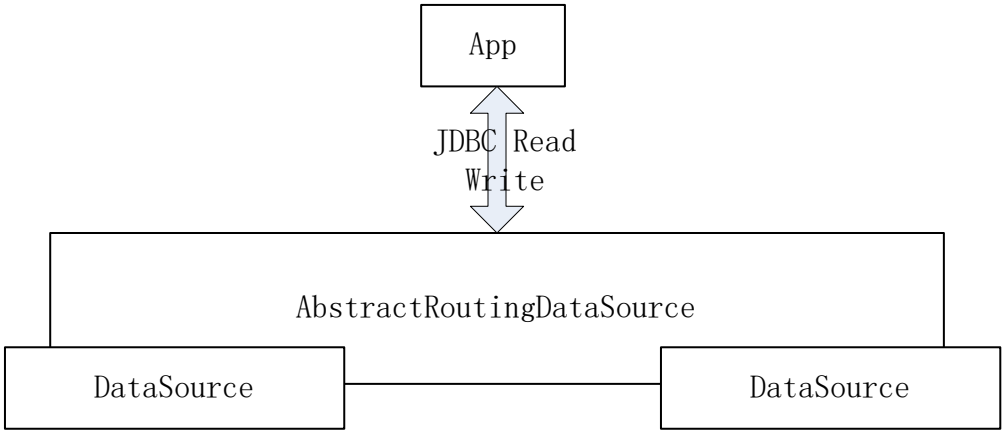
在系统中，平台管理和资源管理模块，使用不频繁，但是做为基础数据，需要给项目模块提供支撑，项目模块，在项目运行期间，使用频繁，于是，我们把平台管理资源管理，和项目做垂直切分，因为在项目中需要用到平台的内容，所以，平台的数据，以视图的方式提供给项目，做只读操作，并且可以减少跨数据库的接口，增加关联查询的速度。

垂直切分，在系统里，就是把平台管理，和资源管理的数据库和项目的数据库放到不同的数据库中，当然，也可以是不同的 Mysql Instance。



- 水平切分

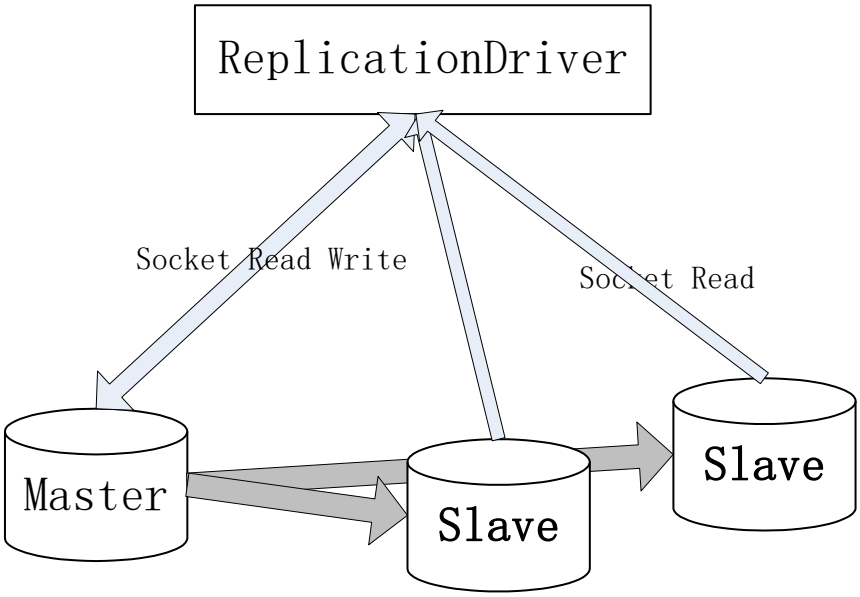
项目模块，由于存在多个项目并发进行，这个时候，一个数据库，不可以很好的支撑多个项目的进行，并且如果一个项目的数据库发生问题，直接影响到多个项目的运行，这个时候，我们把项目相关的内容做水平切分，因为项目之间的数据在项目运行期间基本没有交叉使用，做了以上的处理，可以满足项目的运行。如果一个项目在一个数据库中还是不能支撑，我们可以使用下节中介绍到的读写分离技术。



水平切分的实现，系统采用 Spring 提供的 AbstractRoutingDataSource 来实现，Spring 的 AbstractRoutingDataSource 提供一个虚拟的 DataSource，再根据项目的参数，找到实际的 DataSource，最终得到实际的数据库连接。最终的结果，就是不同的项目使用了不同的 DataSource。

3.2.5 读写分离

考虑到有些项目即使给他一个完全独立的数据库，依然在性能上不能支撑，这个时候系统采用读写分离的方案来解决问题，通过 mysql 的主备复制，在主库中提供写操作，和即时的读操作，而大量的后期的读操作，都放到备用库中，这样可以大大减轻主库的压力。



读写分离的过程是这么实现的，在 Java 的数据库处理中，都会有对数据库操作做事务处理，并且都会设置事务的读写状态，在获取数据库连接的时候，发现如果是只读的事务状态，数据库就会连接到从库，并且会做 LoadBalance。如果发现事务是读写状态，就会取得一个主库的数据库连接，从而连接到主库。

读写分离使用 mysql 驱动自带的 ReplicationDriver, 对最终程序开发基本没有影响,即时的读操作需要和写操作在同一个事务中完成，读写分离的使用，不是每个项目必须的，具体使用与否只要在项目数据源配置的地方，配置不同的数据库参数即可。

3.2.6 特大数据量的考虑

根据我们大家在项目开始之初的共识，我们对数据库按 Project 作了水平切分，目前系统中一个项目中记录数最大的表应该是作业相关的评论，而我们估计一个 10 万人的项目里，这个表的记录的最大数可能在 500 万左右，在 MySQL 的能力之内，所以在现阶段系统暂时没有对特大数据量（单表在 10000w 以上）的存储做特殊的处理，这个问题对于继教网将来的业务是有可能发生的。对于这个问题我们以后可能会有以下一些方式来处理。

- 应用程序处理

通过系统的应用程序，到不同表中读写数据。因为系统的程序结构，把数据访问层做了很好的封装，所以在一些表上实现这个方案不会很复杂，程序的改动也在只发生在数据访问层。不过分表并不能保证查询效率一定好于单表（某些时候可能会更差），取决于一些和分表规则及查询条件等相关的一些具体设计，可能需要 case-by-case 讨论

- 数据库提供的支持

依赖于数据库提供的支持，来做一些处理。具体到我们目前用的 MySQL，可考虑 5.0.X 提供的 Merge Table 及 5.1.X 提供的 Partition 功能来处理。我们现在项目使用 MySQL5 的原因是 MySQL5.1

还没有在我们公司内部做完整的测试，所以使用新版本可能会出现一些问题，如果等我们的测试完成，数据可以无缝升级到新版本。

- 更换数据库

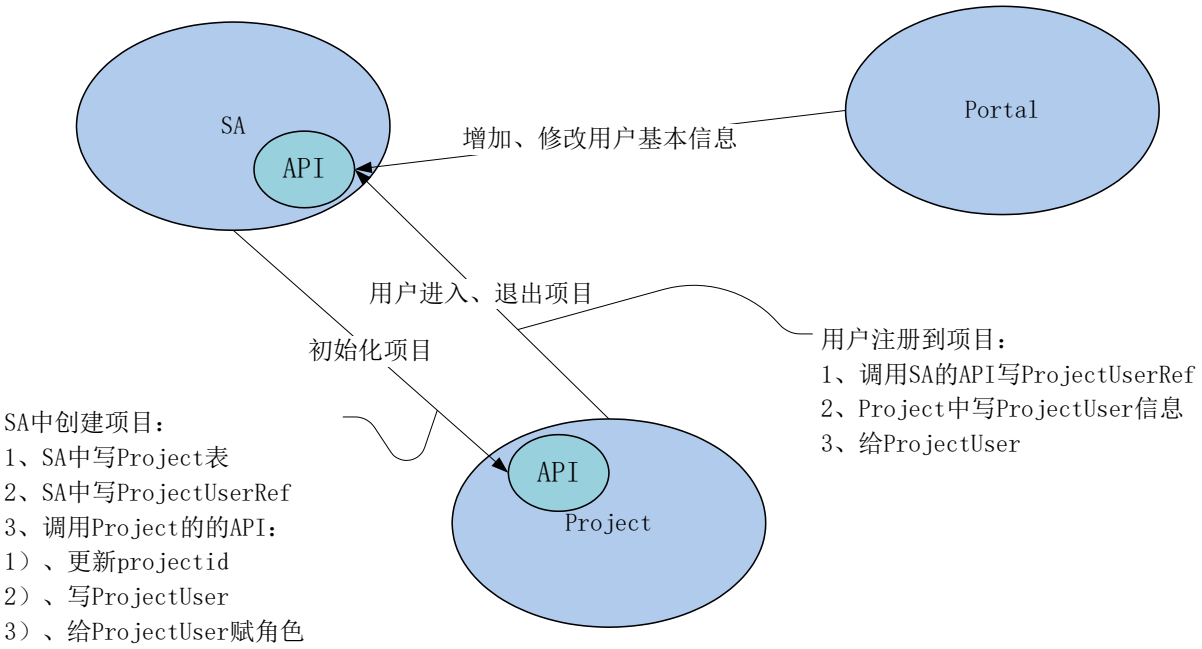
更换成能力更强的 Oracle 等，更强的商业数据库，当然可以更好的解决问题。并且我们现在程序的开发基于 Hibernate，Hibernate 对各种数据库提供无缝支持，我们可以比较方面的迁移应用到 Oracle 等数据库上。

3.3 模块/子系统集成与通讯

因为系统做了分模块的分布式的处理，不可避免的有很多模块间的调用，系统在处理这个内容的时候，使用了程序接口调用，和数据库复制等几种方案。为了提高性能，除了必要的地方，我们使用程序调用，基本都通过数据库复制来实现。

3.3.1 模块间调用

分布式系统的一些耦合部分，由于实时性的要求，系统使用直接的 RPC 调用来完成相关的功能，使用程序调用的方式，系统可以马上知道调用的结果，便于程序马上给客户给出是否成功的标志。在继教网项目中，主要的接口调用，集中在用户和项目这个部分，主要的调用如下图所示。



3.3.2 数据库复制

对于一些非实时要求部分，为了减少远程调用，一些公共的内容（如用户，资源）通过复制的方式复制到项目数据库服务器，再把表通过只读视图的方式，暴露给项目使用。

具体的复制见数据复制部分

3.3.3 远程调用

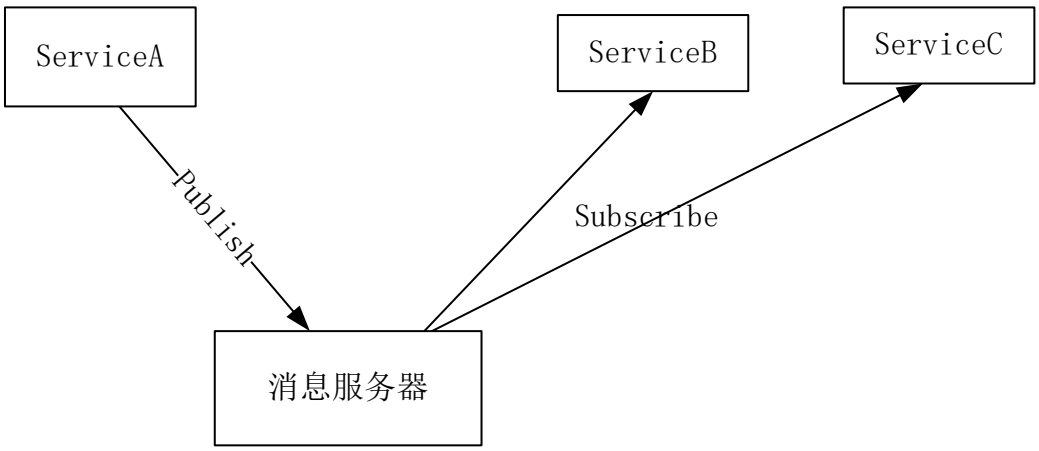
- 对外接口

采用 xmlapi，xml 是中立格式，任何其他语言都可以解析。并且格式和内容都可以通过 dtd 或者 xsd 来定义，是做为第三方接口的不二选择。

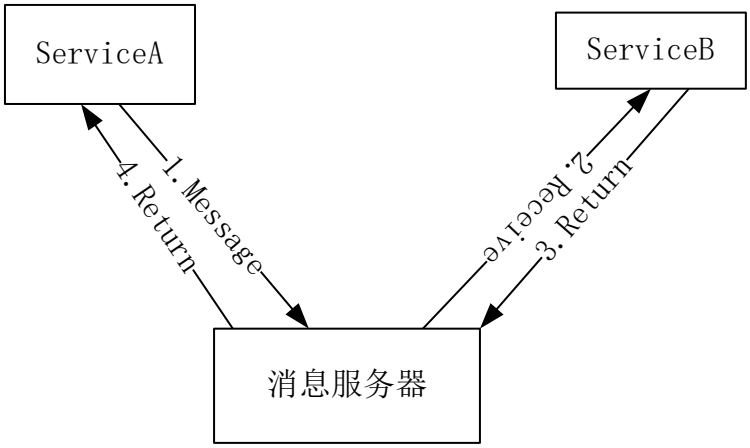
- 内部接口

在系统内部，因为是分模块开发，部署上也可能是分模块的，而整个系统展现是同一的，势必存在一定数量的模块间调用，又因为系统是多项目并存的原因，有些调用，需要多个项目同时响应，内部调用就存在以下两种方式， publish 方式和 rpc 方式，publish 可以处理一对多的发布，rpc 可以实现实时的调用，在接口调用中，除了优化调用的方式之外，减少调用时传递的数据，也是优化的一个非常重要的部分，在系统中，系统选用 google 的 protobuf 作为我们的消息协议，这个协议是 google 内部调用使用的，现在开源出来，给业界使用。这个消息协议在序列化结果长度和序列化消耗时间上都有较大的提升。

publish 方式方法调用结构如下，ServiceA 发起一个调用，注册在消息服务器上可以接收该消息的 ServiceB 和 ServiceC 分别获取该消息，做相应的处理



rpc 方式方法调用结构如下，ServiceA 发送一个请求，ServiceB 是注册在消息服务器上处理该消息的服务，ServiceB 取得该消息后，做相应的处理，然后把结果通过相同的通路返回给 ServiceA，ServiceA 接收到结果后，调用完成



3.4 缓存机制

继教网项目，是一个典型的数据库应用，在这样的应用中，数据库的性能，往往决定了整个应用的性能，而减少数据库的访问，就成了提高应用性能的一个非常重要的手段，在我们的应用中，有多种方法来减少数据库访问，常见方法如下

3.4.1 Hibernate 的二级缓存和查询缓存

Hibernate 的二级缓存和查询缓存是对象缓存，当第一次数据库访问之后，把对象存放在访问速度高于数据库的缓存中，下一次来取相同的数据时候，就直接从缓存中返回，当然，Hibernate 也监听了对象的修改，如果对象修改了就让相关的缓存失效。

3.4.2 静态页面生成

在我们的设计中，很多主页，因为一些内容的时效性要求不是很高，但是访问量又很大，这个时候，系统引入了静态页面生成技术，让大家访问的内容直接从静态内容服务器出来，这样，既可以节省数据库的访问，又可以让应用服务器更加轻松，从实际的测试中可以看到，静态内容服务器的服务能力远远大于应用服务器。

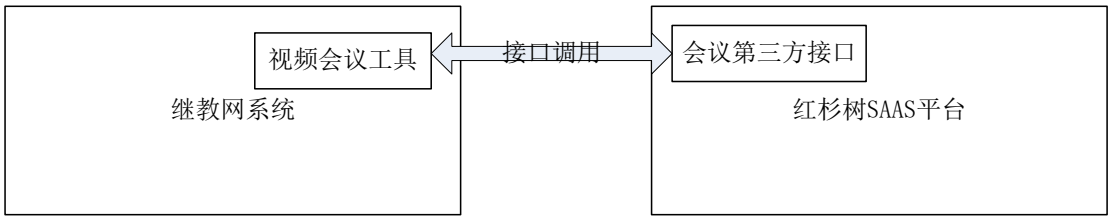
3.5 第三方集成

3.5.1 用户集成

系统提供统一的用户管理，统一的认证管理，对于一些需要做统一登录管理的应用，我们提供认证服务。第三方可以通过统一认证管理集成到我们的系统中来，比如继教网现有的论坛和博客系统。具体接口集成工作的说明，见统一用户管理一节。

3.5.2 接口集成

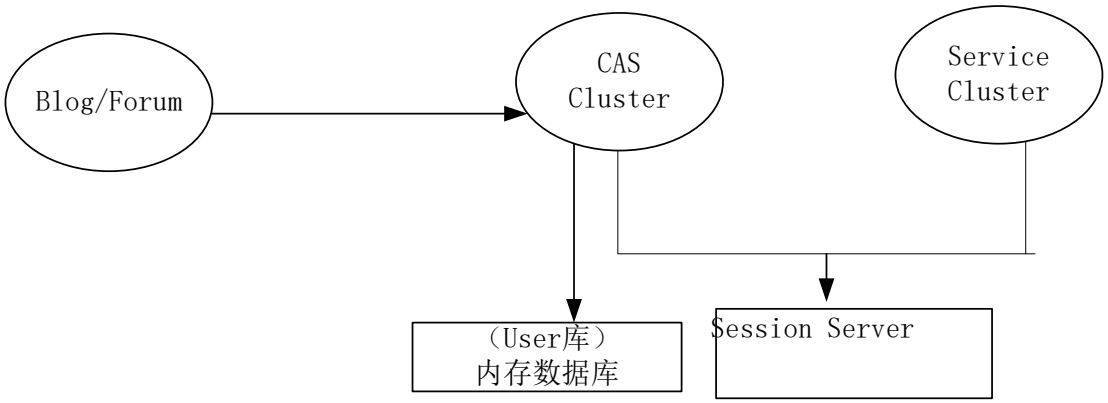
在系统中，还使用了接口的集成方式，比如我们红杉树的会议系统，提供了第三方接口，通过接口集成，可以作为一种工具在继教网的教学工具中直接使用。接口集成的基本结构如下



3.6 统一用户管理

3.6.1 CAS 和 Session Server

CAS 是我们系统中用来做为统一用户认证的工具，该工具可以提供多种方式的，针对多种语言的认证，如 java 和 dot net，该工具很好的给系统用户集成提供了帮助。另外在 session 管理中，我们使用了集中的 Session Server，使用 Session Server，让系统的应用服务器可以支撑更多的在线用户，并且使用户请求可以在不同应用服务器之间的自由的跳转，从而更好的为应用服务器集群扩展做好准备。在系统中，我们的用户集成分成以下两个方式，Java 开发的内部的 War 包，我们提供了单独的 SessionManager 来获取 session，让应用的 session 共享，从而获得统一用户认证的支持，对于第三方的集成，我们使用 CAS 的标准认证方式来做。具体的集成方式如下图所示，具体的集成说明，详见概要设计文档。

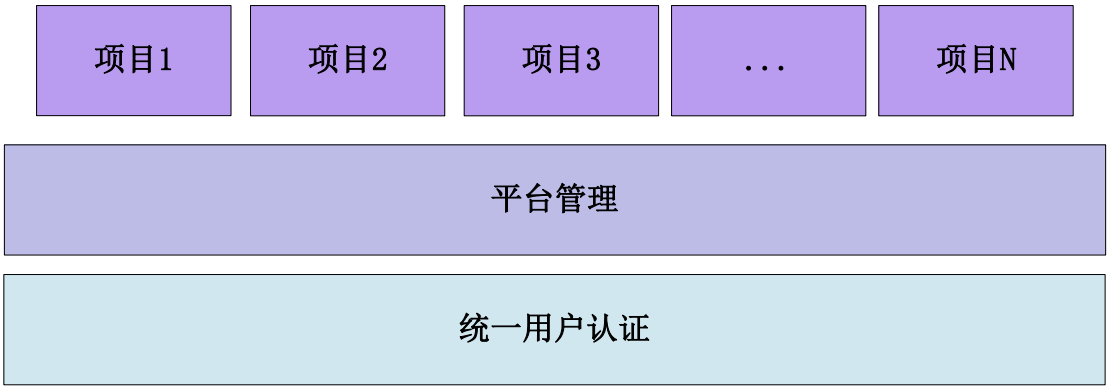


3.7 可扩展性

继教网系统虽然在调研和开发上都做了较大的工作，但是没有任何一个应用是万能的，没有一个系统是可以预见继教网以后的所有业务变化。所以在设计上，我们保留一些可扩展的接口，保证系统可以通过扩展来支撑继教网以后的发展。系统的可扩展性主要包含以下方面。

3.7.1 业务扩展

在系统的设计中，项目的运行是在用户管理和平台管理之上，项目的添加是使用程序配置来完成，简单的说，我们添加一个数据库，然后在程序里添加一些记录，一个新的项目就可以快速开始，项目运行和添加的基本结构如下。



3.7.2 自定义主页

如果项目需要使用不同特色的主页来展现各自项目的风格，系统还提供了 CMS Engine 来做主页的定制，按照一定的规律，可以把主页做成用户想要的样式。

3.7.3 程序扩展

- 工具的基础类库

项目教学工具的变化是不可预见的，但是基础的内容又不会变化很大，在我们开发工具的时候，已经准备了一个工具的基础类库，以后的工具开发，在现有类库的基础上，将变得非常容易和快速，关于工具基础类库的详细说明，详见概要设计文档。

- 二次开发接口

对于某些以后出现的特殊的业务，现有系统可能不能很好的做支撑，这个时候，需要根据现有系统提供的接口做二次开发，从而保证存储的一致，而在展现形式上，可以千变万化。

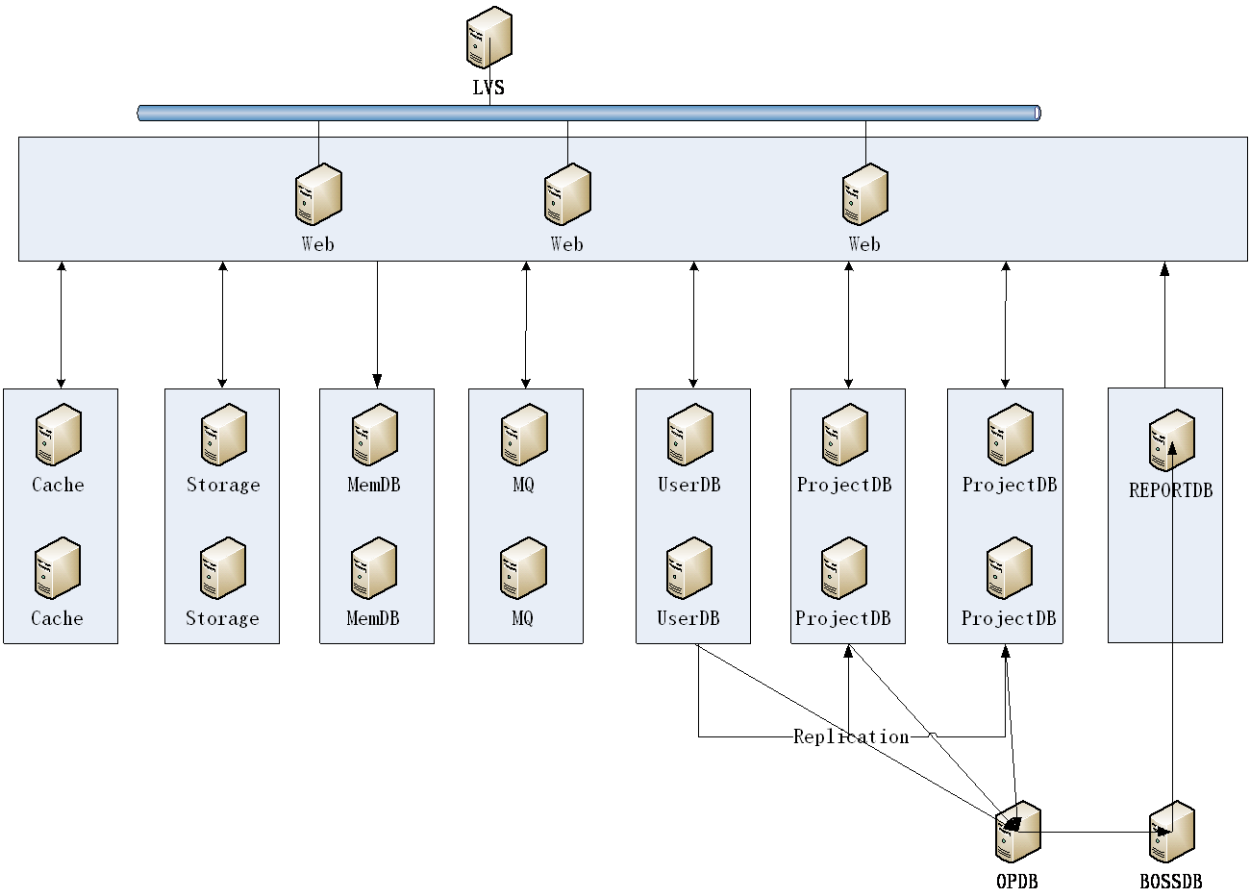
3.7.4 第三方集成支持

对于一些特殊业务，需要通过第三方的系统来支撑，我们也提供了很好的支持，具体内容，详见第三方集成一节

3.8 部署

部署的内容，本文档只提供基本的内容，需要更多，更明确的内容，详见 Hosting 部门提供的部署方案。

3.8.1 部署图



系统中采用 LVS（或者 F5）作为整个系统暴露在最外层，后面连接的是一个 web 集群，提供主要的 web 访问服务，在 web 服务之后，我们有 Cache，Storage，MemDB，MQ，UserDB，ProjectDB 的集群做为后端的支撑，在运维上有有 OPDB，BOSSDB 和 REPORTDB 在运行。在业务支撑方面，我们需要保证没有单点的服务，以保证整个环境的 HA。

3.8.2 集群

- APP 集群

APP 设计成无状态，session 通过 session server 来管理，如果应用服务器在性能上成为瓶颈，我们可以通过添加服务器来处理

- DB 集群

现在我们的 DB 做了水平和垂直切分的操作，在切分之后，又通过主备的方式做了读写分离的支持，在 DB，如果出现很小的项目，我们可以把不同的 ProjectDB 部署在同一个 mysql 的 Instance 中，如果大的项目，我们可以创建主备集群并且使用读写分离来处理。

3.8.3 文件存储

我们通过专用的存储，然后给各个 Server 提供 NFS 的服务

3.8.4 静态与动态信息

静态文件生成之后，或者上传之后，直接通过 httpserver 来处理，不通过 app server，因为 httpserver 的并发能力更好，必要的时候还可以给 httpserver 加上缓存服务，进一步提升其处理能力

动态信息有 httpserver 通过 mod_jk 或者反向代理转交给 appserver 处理。

3.9 监控

通过 Monitor、Slim 等工具实现整个 IDC 中的网络 traffic、Server 状态等资源的监控与管理，同时通过 Alert 系统将信息实时发给相关的工作人员，本文档中不提供详细的监控方案，需要更多内容，请以 Hosting 部门的部署方案为准。

4 几个重要架构考虑综述

4.1 性能（Performance）

在性能上，我们主要考虑了以下一些方面的内容。

- 应用程序

系统使用了 Session Server，可以使应用服务器的扩展更加容易。系统是分模块开发的，在部署的时候，不同的模块可以部署在不同的 server 中，也为我们的部署带来的方便。在程序中，系统大量的使用缓存操作，减少数据库的访问，在类似继教网这种数据库访问频繁的系统中，可以很好的提升系统的性能。在一些访问频繁，但是实时性要求不高的页面，系统使用了页面生成的技术，也为系统性能的提升做了很好的帮助。

- 数据库

系统首先使用了垂直切分，让继教网数据库可以分布在不同的物理机上，而 ProjectDB 我们使用了水平切分来处理，让每个项目都访问自己的数据库，一方面可以相互隔离，另外也可以有更好的性能表现。在数据库切分之后，我们可以对每块做主备的集群，并且采用数据库读写分离的技术来提高整体的性能。当然有些模块，有很高的写并发要求，在这些方面，我们使用内存数据库来提高系统的写并发能力。

4.2 伸缩性（Scalability）

在系统伸缩性方面，我们有以下一些考虑。

- 集群

各个支撑模块，我们都可以提供集群来处理，当然，如果项目小的时候，单独的机器可以很好的完成项目的进行，集群也不是必须的配置。

系统的开发，我们使用了分模块开发，在 web 部署的时候，我们可以根据模块来做不同的部署集群，针对不同模块的不同性能要求，灵活的对集群进行扩展，当然在项目运行前期，我们还可以把不同模块部署在一个集群中，从而节约运行初期的项目投入。

- 垂直和水平分片

系统对数据库做了垂直和水平的分片，可以让数据库部署在不同的物理位置，在做水平分片的时候，我们针对的是 Mysql 数据库，而不是 Mysql Instance。这个时候，在对付小项目的时候，我们可以把很多小项目放在一个 Mysql Instance 的不同数据库中，通过一台物理机器就可以解决多个项目的应用。

- 高并发处理

系统在处理高并发的时候，系统对内存数据库的使用是通过配置来完成，在项目初期，不同的高并发模块可以使用一个内存数据库来处理并发要求，而如果一个内存数据库也不能完成任务的时候，我们可以把不同模块分配到不同的内存数据库中，并且内存数据库还可以做集群处理。

4.3 安全（Security）

继教网是一个互联网应用，在互联网中，可能出现任何形式的入侵，安全的考虑必不可少，在系统中我们会从软件和硬件两个方面来考虑这个问题

- 软件方面

首先系统的安全是系统用户的安全，我们在登录的时候使用 HTTPS 协议，该协议是在 HTTP 协议之上，加上了 SSL 的加密，可以让入侵者无法拦截该请求，也就保证了用户在登录的时候的密码安全。在部署的时候，系统所有服务器均放置在内网，通过 FireWall 或 Route 功能实现 NAT 功能，最前端只放提供转发功能的一个应用，从而从网络上保证了系统的安全，在我们的程序中，我们在做 JDBC 开发的时候，所有的 SQL 请求都使用 PrepareStatement 来处理，从而避免 SQL 注入入侵

- 硬件方面

硬件方面我们会采用防火墙等技术来处理，更多内容详见 Hosting 部门的部署方案

4.4 高可用（High Availability）

继教网是一个需要实时互动的应用，服务应该 24 小时不间断，这个时候我们就要做许多的操作，来实现这个内容，这个保证是从开发方面和部署方面共同来处理的。具体内容有如下处理

- 开发方面

在开发方面，首先我们添加了 SessionServer，让用户的操作可以在不同的 WebServer 上处理，都会得到相同的结果，这样的结果，就是假设有一个 WebServer 当机，用户的请求，还是会通过前端的 LVS 或者 F5 服务器转发到另外的 WebServer，而保证了用户的请求没有断开。

在数据库方面，我们的项目数据库做了水平切分，可以让不同的项目不会产生相互影响。在每个切分的节点，我们又做主备的集群，不管任何一台机器的当机，我们都可以做快速的切换。

- Hosting 方面

在 Hosting 方面，应该有完整的监控方案，可以随时自动的发现问题，并且告知相关人员等等，当然 Hosting 还有更多的详细方案，详见 Hosting 部门的部署方案。

5 附录

5.1 附 1：开发工具选型

以下为本项目开发过程中使用的主要工具的选型表：

使用范围	项目	选型	版本	主要用到的 jar 包
表现层	MVC 框架	Struts2	2.1.6	struts2-core-2.1.6.jar struts2-spring-plugin-2.1.6.jar struts2-testng-plugin-2.1.6.jar xwork-2.1.2.jar
	模板引擎	Freemarker	2.3.15	freemarker-2.3.15.jar
	Ajax 引擎	DWR	2.0.1	dwr-2.0.1.jar
业务层	轻量级容器	Spring	2.5.6	spring-beans-2.5.6.jar spring-core-2.5.6.jar spring-jdbc-2.5.6.jar spring-context-2.5.6.jar spring-tx-2.5.6.jar spring-orm-2.5.6.jar spring-aop-2.5.6.jar spring-web-2.5.6.jar spring-aspects-2.5.6.jar spring-context-support-2.5.6.jar
持久层	持久框架	Hibernate	3.2.5.ga	hibernate-3.2.5.ga.jar hibernate-annotations-3.3.0.ga.jar hibernate-commons-annotations-3.3.0.ga.jar
	数据库连接池	DBCP	1.2.2	commons-dbcp-1.2.2.jar commons-pool-1.3.jar commons-collections-3.2.jar
公用类库	公用类库	ApacheCommons		commons-fileupload-1.1.jar commons-io-1.1.jar commons-collections-3.2.jar

				commons-lang-2.3.jar commons-beanutils-1.7.0.jar
测试框架	Unit	testNG	5.8	spring-test-2.5.6.jar struts2-testng-plugin-2.1.6.jar testng-5.8-jdk15.jar
	Mock	Jmock	2.5.1	jmock-2.5.1.jar jmock-junit4-2.5.1.jar jmock-legacy-2.4.0.jar
	测试基础类库	unitils	1.0	test-utilities-2.0.0.0.jar unitils-1.0.jar dbunit-2.2.jar
日志记录	日志记录	log4j	1.2.14	log4j-1.2.14.jar commons-logging-1.0.4.jar
服务器	应用服务器	JBOSS	4.2.3	
	HTTP 服务器	Nginx	0.7.61	
	数据库服务器	Mysql	5.0.20	mysql-connector-java-5.1.6.jar
	内存数据库	Tokyo Cabinet	1.4.27	amqp-client-1.5.1.jar protobuf-java-2.1.0.jar
	缓存	memcached	1.2.6	java_memcached-release-2.0.1.jar Hibernate-memcached-1.1.0.jar
其它	项目管理	Maven	2.0.9	
	编译服务器	Continuum	1.2.3	
	代码管理	SVN	1.4.5	
IDE	IDE	Eclipse	3.3	

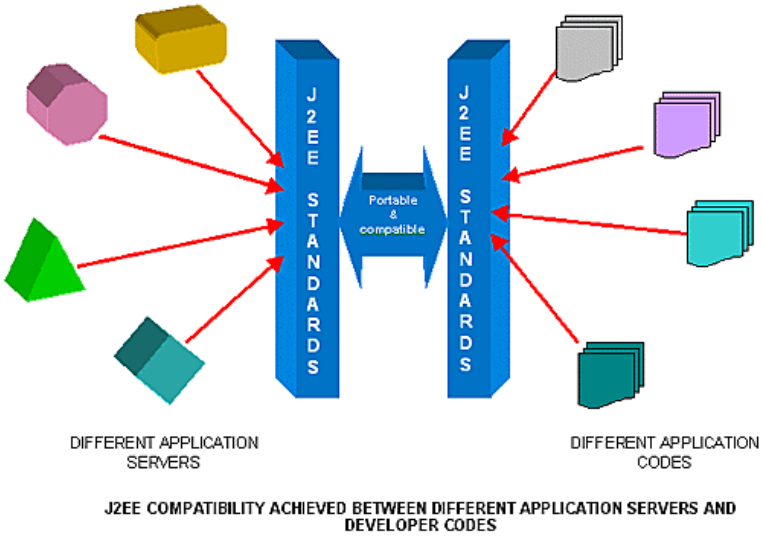
下面对重要对 J2EE 架构及一些重要工具选型做一些说明：

5.1.1 J2EE 构架

J2EE 是一个标准中间件体系结构，旨在简化和规范多层分布式企业应用系统的开发和部署。J2EE 方案的实施可显著地提高系统的可移植性、安全性、可伸缩性、负载平衡和可重用性。

J2EE 技术出现之前，几家主要的中间件开发商的产品各自为阵，彼此之间缺乏兼容性，可移植性差，难以实现互操作，没有一个被普遍认可的行业标准。J2EE 的出现标志着中间件技术在经历了多年的不断摸索和经验总结后，正逐步走向成熟。

J2EE 的核心是一组规范和指南，定义了一个使用 Java 语言开发多层分布式企业应用系统的标准平台。开发人员在这些规范和指南的基础上开发企业级应用，同时由 J2EE 供应商确保不同的 J2EE 平台之间的兼容性。由于基于规范的各 J2EE 平台之间具有良好的兼容性， 因此 J2EE 应用系统可以部署在不同的应用服务器上，无需或只需进行少量的代码修改。



J2EE COMPATIBILITY ACHIEVED BETWEEN DIFFERENT APPLICATION SERVERS AND DEVELOPER CODES

- 多层、分布式中间件语法

采用多层分布式应用模型，J2EE 将应用开发划分为多个不同的层，并在每一个层上定义组件。各个应用组件根据他们所在的层分布在同一个或不同的服务器上，共同组成基于组件的多层分布式系统。典型的 J2EE 四层结构包括客户层、表示逻辑层（Web 层）、商业逻辑层和企业信息系统层。

有了 J2EE，分布式系统的开发变得简单了，部署的速度也可以加快。J2EE 组件的分布与服务器环境无关，所有的资源都可通过分布式目录进行访问。这意味着开发人员不再需要为组件和资源的分布问题耗费精力，从而可以有更多的时间专注于业务逻辑的实现，提高开发效率。

- 企业级应用系统开发平台

J2EE 本身是一个标准，一个为企业分布式应用的开发提供的标准平台。而 J2EE 的实施，则具体表现为诸如 BEA Web logic，Jboss，Tomcat， IBM Web sphere 之类的特定 Web 服务器产品。利用 J2EE 应用-编程模型开发的企业应用系统，可以部署在不同厂商生产的、但相互兼容的 J2EE 应用服务器上。

目前，市场上基于 J2EE 的 Web 服务器品种繁多，性能特点各有千秋，每家厂商的产品都有精心设计的独到之处。但与产品个性无关的是，所有的 J2EE 应用服务器都为企业级应用系统的开发和部署提供了一个共同的基础。

- 电子化应用开发模型

J2EE 应用很容易发布到 Web、掌上电脑或移动电话等手持设备上。换言之，应用组件可以很轻松地实现电子化。J2EE 的应用-编程模型保证组件在向不同类型的客户端移植过程中，商业逻辑和后端系统保持不变。

此外，J2EE 平台的其他主要优点还有：自动负载平衡、可伸缩、容错和具有故障排除等功能。部署在 J2EE 环境中的组件将自动获得上述特性，而不必增加额外的代码开销。J2EE 所有这些特性对于需要构建全天候网络门户的企业来说显得尤为重要。

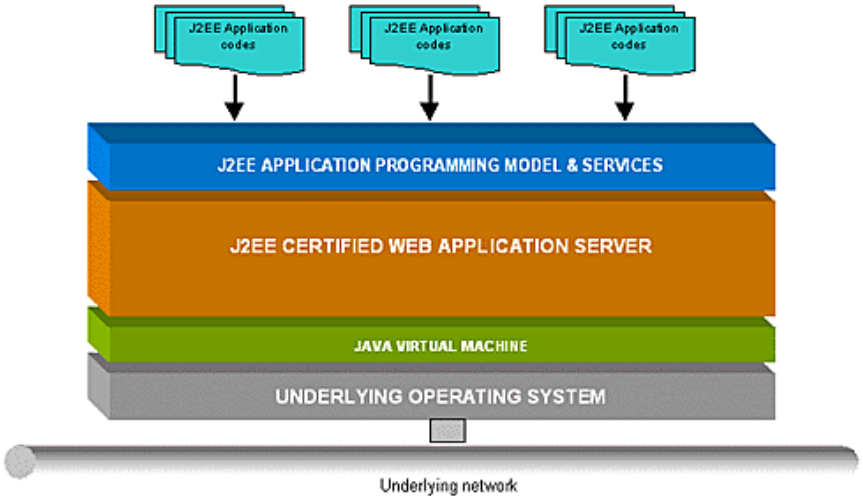
- Web 应用服务器上广泛采用的标准

可以说，J2EE 是首个获得业界广泛认可和采纳的中间件标准。目前几乎所有的一流 Web 应用服务器，如 BEA 的 Web logic、IBM 的 Web sphere、JBoss、Sun 的 iPlanet 和 Macromedia 的 Jrun 等，都是基于 J2EE 的。迄今为止，还没有哪个其他标准能获得如此众多的中间件供应商的一致支持。

而且，有了 J2EE，企业的应用开发对于某个特定的开发商或应用服务供应商的依赖性更小。应用组件只要符合 J2EE 规范，完全可以部署在不同的应用服务器上。为了确保不同厂商的 J2EE 应用服务器的兼容性和一致性，Sun 公司发布了 J2EE 兼容性测试包。

- 独立于硬件配置和操作系统

J2EE 运行在 Java 虚拟机（JVM）上，利用 Java 本身的跨平台特性，独立于硬件配置和操作系统。Java 运行环境（JRE）——JVM 的可安装版本加上其他一些重要组件——几乎可以运行于所有的硬件/OS 组合。因此，通过采用 Java，J2EE 使企业免于高昂的硬件设备和操作系统的再投资，保护已有的 IT 资源。在很多情况下，J2EE 还可以直接运行在 EIS 服务器环境中，从而节约网络带宽，提高性能。



J2EE MIDDLEWARE IMPLEMENTATION IN A SINGLE SERVER

- 坚持面向对象的设计原则

作为一门完全面向对象的语言，Java 几乎支持所有的面向对象的程序设计特征。面向对象和基于组件的设计原则构成了 J2EE 应用编程模型的基础。J2EE 多层结构的每一层都有多种组件模型。因此，开发人员所要做的就是为应用项目选择适当的组件模型组合，灵活地开发和装配组件，

这样不仅有助于提高应用系统的可扩展性，还能有效地提高开发速度，缩短开发周期。此外，基于 J2EE 的应用还具有结构良好，模块化，灵活和高度可重用性等优点。

- 灵活性、可移植性和互操作性

利用 Java 的跨平台特性，J2EE 组件可以很方便地移植到不同的应用服务器环境中。这意味着企业不必再拘泥于单一的开发平台。J2EE 的应用系统可以部署在不同的应用服务器上，在全异构环境下，J2EE 组件仍可彼此协同工作。这一特征使得装配应用组件首次获得空前的互操作性。例如，安装在 IBM Websphere 环境下的 EJB，一方面可以直接与 Websphere 环境下的 CICS 直接交互，另一方面也可以通过安装在别处的 BEA Weblogic 服务器上的 EJB 进行访问。

- 轻松的企业信息系统集成

J2EE 技术出台后不久，很快就将 JDBC、 JMS 和 JCA 等一批标准纳归自身体系之下，这大大简化了企业信息系统整合的工作量，方便企业将诸如 legacy system（早期投资系统），ERP 和数据库等多个不同的信息系统进行无缝集成。

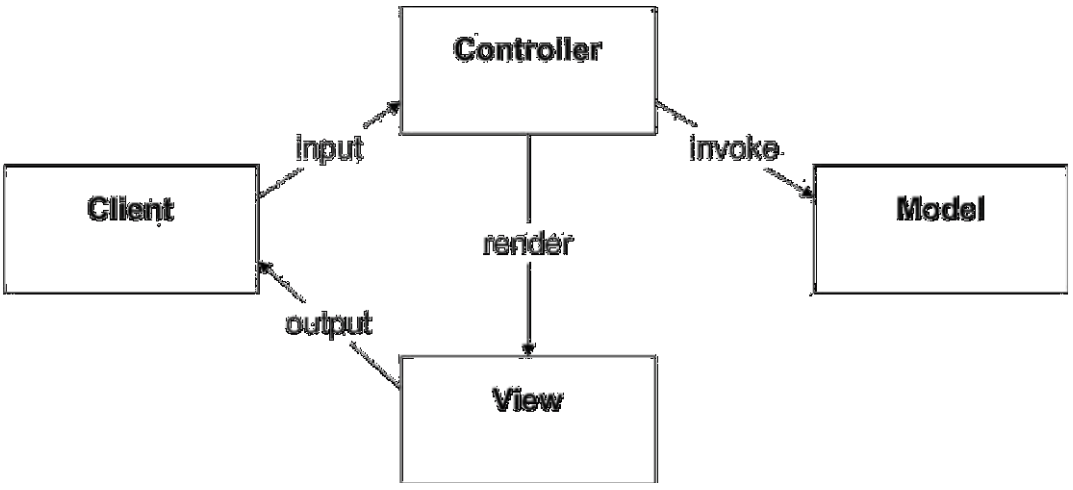
由于几乎所有的关系型数据库系统都支持 JDBC，因此只需借助必要的 JDBC 驱动程序，J2EE 应用就可以和所有主流数据库系统进行通信。类似的，目前业界正冒出一批基于 Java 连接器体系标准的 EI 适配器，也用于提供各类 legacy system 和 ERP/CRM 的无缝集成。

- 引进面向服务的体系结构

随着 Web 服务以及 SOAP 等开放标准的出现，企业异构系统之间的互操作性成为可能。J2EE，作为一个可扩展平台，很自然需要加入 Web 服务特性。为此，Sun 公司发布了一整套称为“JAX 包”的 API，支持从 XML 语法分析、XML 绑定、SOAP 消息发送、注册表查寻、XML RPC 到 XML 消息传递等所有各种 Web 服务需求。

5.1.2 MVC 设计模式

MVC 模式是"Model-View-Controller"的缩写，中文翻译为"模式-视图-控制器"。下图是 MVC 的框架图



MVC 应用程序总是由这三个部分组成。Event(事件)导致 Controller 改变 Model 或 View，或者同时改变两者。只要 Controller 改变了 Models 的数据或者属性，所有依赖的 View 都会自动更新。类似的，只要 Controller 改变了 View，View 会从潜在的 Model 中获取数据来刷新自己。MVC 模式最早是 smalltalk 语言研究团提出的，应用于用户交互应用程序中。smalltalk 语言和 java 语言有很多相似性,都是面向对象语言，很自然的 SUN 在 petstore(宠物店)事例应用程序中就推荐 MVC 模式作为开发 Web 应用的架构模式。MVC 模式是一种架构模式，其实需要其他模式协作完成。在 J2EE 模式目录中，通常采用 service to worker 模式实现，而 service to worker 模式可由集中控制器模式，派遣器模式和 Page Helper 模式组成。

MVC 模式是一个复杂的架构模式，其实现也显得非常复杂。但是，我们已经终结出了很多可靠的设计模式，多种设计模式结合在一起，使 MVC 模式的实现变得相对简单易行。Views 可以看作一棵树，显然可以用 Composite Pattern 来实现。Views 和 Models 之间的关系可以用 Observer Pattern 体现。Controller 控制 Views 的显示，可以用 Strategy Pattern 实现。Model 通常是一个调停者，可采用 Mediator Pattern 来实现。

现在让我们来了解一下 MVC 三个部分在 J2EE 架构中处于什么位置，这样有助于我们理解 MVC 模式的实现。MVC 与 J2EE 架构的对应关系是:View 处于 Web Tier 或者说是 Client Tier，通常是 JSP/Servlet，即页面显示部分。Controller 也处于 Web Tier，通常用 Servlet 来实现，即页面显示的逻辑部分实现。Model 处于 Middle Tier，通常用服务端的 javaBean 或者 EJB 实现，即业务逻辑部分的实现。

- MVC 设计思想

MVC 英文即 Model-View-Controller，即把一个应用的输入、处理、输出流程按照 Model、View、Controller 的方式进行分离，这样一个应用被分成三个层——模型层、视图层、控制层。

视图(View)代表用户交互界面，对于 Web 应用来说，可以概括为 HTML 界面，但有可能为 XHTML、XML 和 Applet。随着应用的复杂性和规模性，界面的处理也变得具有挑战性。一个应用可能有很多不同的视图，MVC 设计模式对于视图的处理仅限于视图上数据的采集和处理，以及用户的请求，而不包括在视图上的业务流程的处理。业务流程的处理交予模型(Model)处理。比如一个订单的视图只接受来自模型的数据并显示给用户，以及将用户界面的输入数据和请求传递给控制和模型。

模型(Model)：就是业务流程/状态的处理以及业务规则的制定。业务流程的处理过程对其它层来说是黑箱操作，模型接受视图请求的数据，并返回最终的处理结果。业务模型的设计可以说是 MVC 最主要的核心。目前流行的 EJB 模型就是一个典型的应用例子，它从应用技术实现的角度对模型做了进一步的划分，以便充分利用现有的组件，但它不能作为应用设计模型的框架。它仅仅告诉你按这种模型设计就可以利用某些技术组件，从而减少了技术上的困难。对一个开发者来说，就可以专注于业务模型的设计。MVC 设计模式告诉我们，把应用的模型按一定的规则抽取出来，抽取的层次很重要，这也是判断开发人员是否优秀的设计依据。抽象与具体不能隔得太远，也不能太近。MVC 并没有提供模型的设计方法，而只告诉你应该组织管理这些模型，以便于模型的重构和提高重用性。我们可以用对象编程来做比喻，MVC 定义了一个顶级类，告诉它的子类你只能做这些，但没法限制你能做这些。这点对编程的开发人员非常重要。

业务模型还有一个很重要的模型那就是数据模型。数据模型主要指实体对象的数据 保存（持续化）。比如将一张订单保存到数据库，从数据库获取订单。我们可以将这个模型单独列出，所有有关数据库的操作只限制在该模型中。

控制(Controller)可以理解为从用户接收请求, 将模型与视图匹配在一起，共同完成用户的请求。划分控制层的作用也很明显，它清楚地告诉你，它就是一个分发器，选择什么样的模型，选择什么样的视图，可以完成什么样的用户请求。控制层并不做任何的数据处理。例如，用户点击一个连接，控制层接受请求后, 并不处理业务信息，它只把用户的信息传递给模型，告诉模型做什么，选择符合要求的视图返回给用户。因此，一个模型可能对应多个视图，一个视图可能对应多个模型。

模型、视图与控制器的分离，使得一个模型可以具有多个显示视图。如果用户通过某个视图的控制器改变了模型的数据，所有其它依赖于这些数据的视图都应反映到这些变化。因此，无论何时发生了何种数据变化，控制器都会将变化通知所有的视图，导致显示的更新。这实际上是一种模型的变化-传播机制。模型、视图、控制器三者之间的关系和各自的主要功能。

- MVC 设计模式的扩展

具有极其良好的可扩展性。它可以轻松实现以下功能，实现一个模型的多个视图；采用多个控制器；当模型改变时，所有视图将自动刷新；所有的控制器将相互独立工作。

这就是 MVC 模式的好处，只需在以前的程序上稍作修改或增加新的类，即可轻松增加许多程序功能。以前开发的许多类可以重用，而程序结构根本不再需要改变，各类之间相互独立，便于团体开发，提高开发效率。下面讨论如何实现一个模型、两个视图和一个控制器的程序。其中模型类及视图类根本不需要改变，与前面的完全一样，这就是面向对象编程的好处。对于控制器中的类，只需要增加另一个视图，并与模型发生关联即可。

同样也可以实现其它形式的 MVC 例如：一个模型、两个视图和两个控制器。从上面可以看出，通过 MVC 模式实现的应用程序具有极其良好的可扩展性。

- MVC 的优点

早期的 Web 应用，初始的开发模板就是混合层的数据编程。例如，直接向数据库发送请求并用 HTML 显示,开发速度往往比较快,但由于数据页面的分离不是很直接,因而很难体现出业务模型的样子或者模型的重用性。产品设计弹性力度很小，很难满足用户的变化性需求。MVC 要求对应用分层，虽然要花费额外的工作，但产品的结构清晰，产品的应用通过模型可以得到更好地体现。

首先，最重要的是应该有多个视图对应一个模型的能力。在目前用户需求的快速变化下，可能有多种方式 访问应用的要求。例如，订单模型可能有本系统的订单，也有网上订单，或者其他系统的订单，但对于订单的处理都是一样，也就是说订单的处理是一致的。按 MVC 设计模式，一个订单模型以及多个视图即可解决问题。这样减少了代码的复制，即减少了代码的维护量，一旦模型发生改变，也易于维护。 其次，由于模型返回的数据不带任何显示格式，因而这些模型也可直接应用于接口的使用。

再次，由于一个应用被分离为三层，因此有时改变其中的一层就能满足应用的改变。一个应用的业务流程或者业务规则的改变只需改动 MVC 的模型层。

控制层的概念也很有效，由于它把不同的模型和不同的视图组合在一起完成不同的请求，因此，控制层可以说是包含了用户请求权限的概念。

最后，它还有利于软件工程化管理。由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化产生管理程序代码。

5.1.3 开发框架

- Spring Framework

即使拥有良好的工具和优秀技术，应用软件开发也是困难重重。应用开发往往牵扯到方方面面，每件事情都难以控制，而且，开发周期也很难把握（除非它的 确是一个重量级的复杂应用，倒也有情可原）。Spring 提供了一种轻量级的解决方案，用于建立“快装式企业应用”。在此基础上，Spring 还提供了包 括声明式事务管理，RMI 或 Web Services 远程访问业务逻辑，以及可以多种方法进行的持久化数据库的解决方案。另外，Spring 还有一个全功能的 MVC 框架，并能透明的把 AOP 集成到你的软件中去。

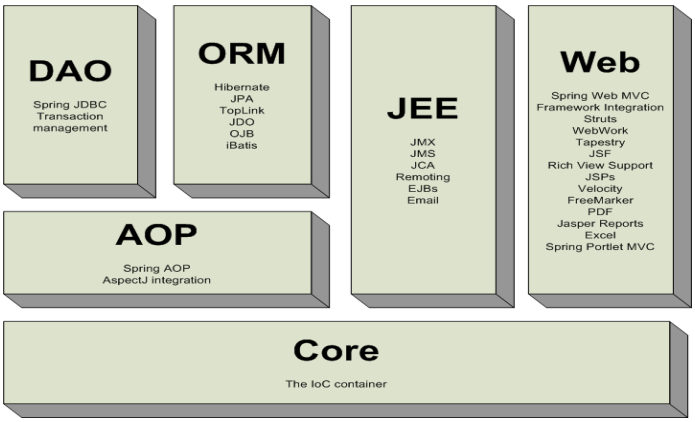
可以把 Spring 当作一个潜在的一站式企业应用。或者，把 Spring 看作一个标准开发组件，根据自己的需要，只取它的部分组件使用而无需涉及其他。例如，你可以利用控制反转容器在前台的展现层使用 Struts 或者 webwork，还可以只使用 Hibernate 集成编码 或是 JDBC 抽象层 去处理数据存储。Spring 被设计成（并将继续保持）无侵入性的方式，意味着应用几乎不需要对框架进行依赖（或根据实际使用的范围，将依赖做到最小）。

Java 应用（从 applets 的小范围到全套 n 层服务端企业应用）是一种典型的依赖型应用，它就是由一些互相适当地协作的对象构成的。因此，我们说这些对象间存在依赖关系。

Java 语言和 java 平台在架构应用与建立应用方面，提供丰富的功能。从非常基础的基本数据类型和 Class（即定义新类）组成的程序块，到建立具有丰富的特 性的应用服务器和 web 框架都有着很多的方法。一方面，可以通过抽象的显著特性让基础的程序块组成在一起成为一个连贯的整体。这样，构建一个应用（或者多个应用）的工作就可以交给架构师或者开发人员去做。因此，我们就可以清晰的知道哪些业务需要哪些 Classes 和对象组成，哪些设计模式可以应用在哪些业务上面。

Spring 的 IoC 控件主要专注于如何利用 classes、对象和服务去组成一个企业级应用，通过规范的方式，将各种不同的控件整合成一个完整的应用。Spring 中使用了很多被实践证明的最佳实践和正规的设计模式，并且进行了编码实现。如果你是一个构架师或者开发人员完全可以取出它们集成到你自己的应用之中。这 对于那些使用了 Spring Framework 的组织和机构来说，在 spring 基础上实现应用不仅可以构建优秀的，可维护的应用并对 Spring 的设计进行验证。

Spring 框架包含许多特性，并被很好地组织在下图所示的七个模块中。



Core 封装包是框架的最基础部分，提供 IoC 和依赖注入特性。这里的基础概念是 BeanFactory，它提供对 Factory 模式的经典实现来消除对程序性单例模式的需要，并真正地允许你从程序逻辑中分离出依赖关系和配置。

构建于 Core 封装包基础上的 Context 封装包，提供了一种框架式的对象访问方法，有些象 JNDI 注册器。Context 封装包的特性得自于 Beans 封装包，并添加了对国际化（I18N）的支持（例如资源绑定），事件传播，资源装载的方式和 Context 的透明创建，比如说通过 Servlet 容器。

DAO 提供了 JDBC 的抽象层，它可消除冗长的 JDBC 编码和解析数据库厂商特有的错误代码。并且，JDBC 封装包还提供了一种比编程性更好的声明性事务管理方法，不仅仅是实现了特定接口，而且对所有的 POJOs（plain old Java objects）都适用。

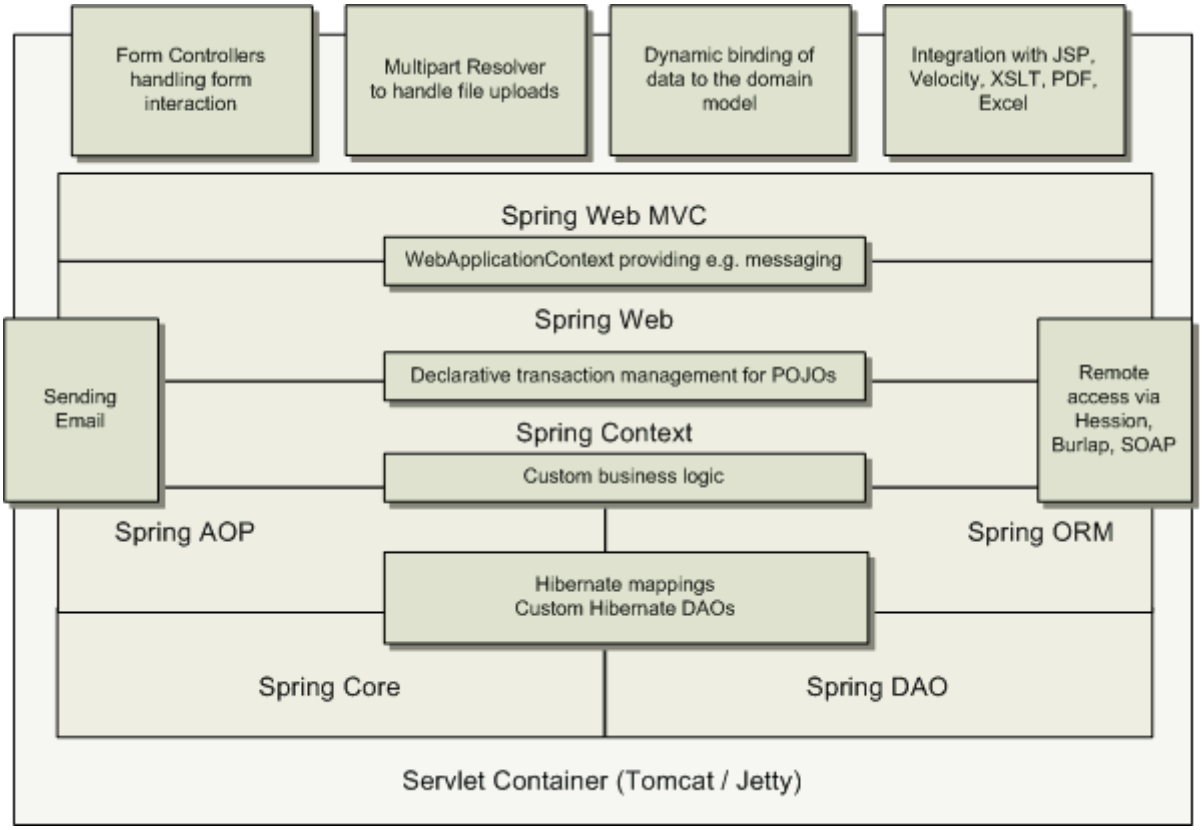
ORM 封装包提供了常用的“对象/关系”映射 APIs 的集成层。 其中包括 JPA、JDO、Hibernate 和 iBatis 。利用 ORM 封装包，可以混合使用所有 Spring 提供的特性进行“对象/关系”映射，如前边提到的简单声明性事务管理。

Spring 的 AOP 封装包提供了符合 AOP Alliance 规 范的面向方面的编程（aspect-oriented programming）实现，让你可以定义，例如方法拦截器（method-interceptors）和切点（pointcuts），从逻辑上讲，从 而减弱代码的功能耦合，清晰的被分离开。而且，利用 source-level 的元数据功能，还可以将各种行为信息合并到你的代码中，这有点象.Net 的 attribute 的概念。

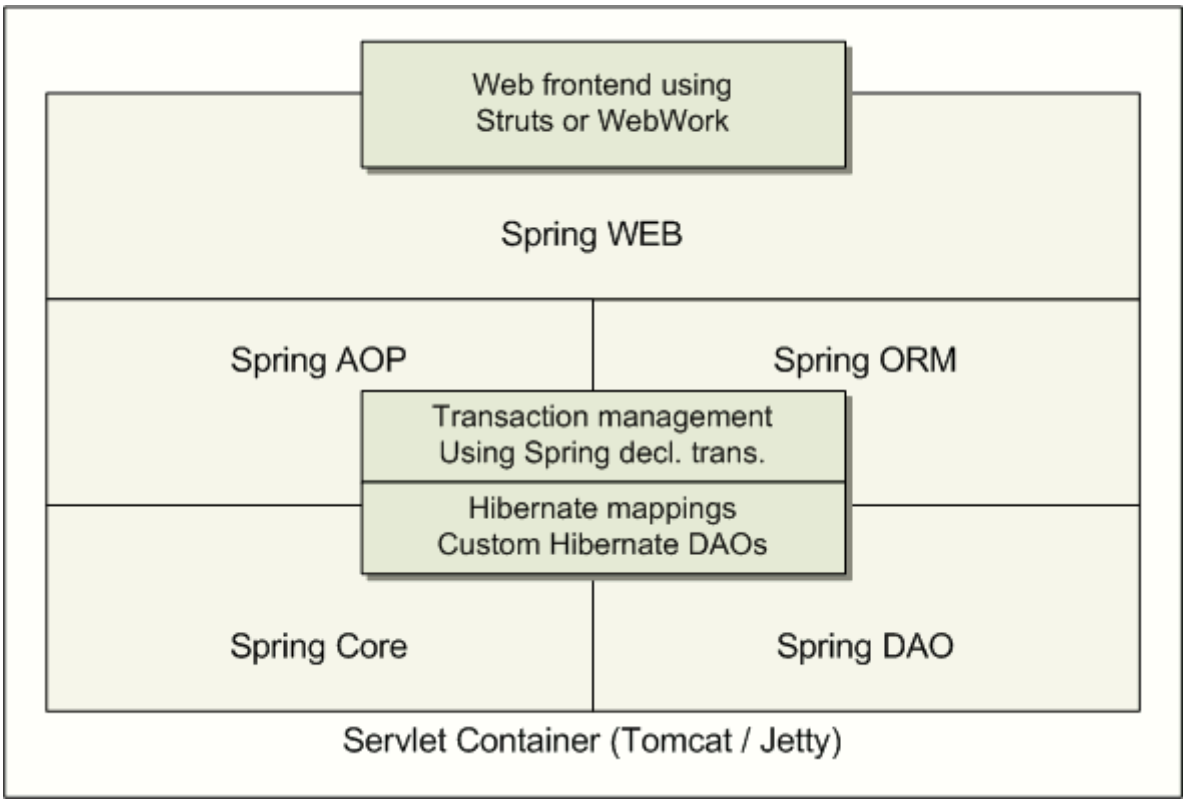
Spring 中的 Web 包提供了基础的针对 Web 开发的集成特性，例如多方文件上传，利用 Servlet listeners 进行 IoC 容器初始化和针对 Web 的 application context。当与 WebWork 或 Struts 一起使用 Spring 时，这个包使 Spring 可与其他框架结合。

Spring 中的 MVC 封装包提供了 Web 应用的 Model-View-Controller（MVC）实现。Spring 的 MVC 框架并不是仅仅提供一种传统的实现，它提供了一种 清晰的 分离模型，在领域模型代码和 web form 之间。并且，还可以借助 Spring 框架的其他特性。

各种情景下使用 Spring 的情况，从简单的 Applet 一直到完整的使用 Spring 的事务管理功能和 Web 框架的企业应用。



通过用 Spring 的 声明事务管理特性，Web 应用可以做到完全事务性，就像使用 EJB 提供的那种容器管理的事务一样。 所有自定义的业务逻辑可以通过简单的 POJO 来实现，并利用 Spring 的 IoC 容器进行管理。对于其他的服务，比如发送 email 和不依赖 web 层的校验信息，还可以让你自己决定在哪里执行校验规则。 Spring 本身的 ORM 支持可以和 JPA、Hibernate、JDO 以及 iBatis 集成起来，例如使用 Hibernate，你可复用已经存在的映射文件与标准的 Hibernate SessionFactory 配置。用控制器去无缝整合 web 层和领域模型，消除对 ActionForms 的依赖，或者避免了其他 class 为领域模型转换 HTTP 参数的需要。



如果，现有的应用使用了WebWork、Struts、Tapestry或其他的UI框架作为前端程序，完全可以只与Spring的事务特性进行集成。只需要使用 ApplicationContext 来挂接你的业务逻辑和通过 WebApplicationContext 来集成你的web层前端程序。

方便的实现通过WebService来访问你的现有代码。

Spring还为EJB提供了数据访问和抽象层，让你可以复用已存在的POJO并将他们包装在无状态SessionBean中。

- Struts2

Struts2作为MVC 2的Web框架，自推出以来不断受到开发者的追捧，得到广泛的应用。作为最成功的Web框架，Struts2自然拥有众多的优点：MVC 2模型的使用、功能齐全的标签库（Tag Library）、开放源代码。

Struts 2开始引入控制反转机制（IOC ： Inversion of Control），提供丰富多样功能齐全的拦截器实现。

Struts2的Action类实现了一个Action接口，连同其他接口一起来实现可选择和自定义的服务。Struts2提供一个名叫ActionSupport的基类来实现一般使用的接口。当然，Action接口不是必须的。任何使用execute方法的 POJO对象可以被当作Struts 2的Action对象来使用。对象为每一个请求都实例化对象，所以没有线程安全的问题。（实践中，servlet容器给每一个请求产生许多丢弃的对象，并且不会导致性能和垃圾回收问题）。

Struts2的Action和容器无关。Servlet上下文被表现为简单的Maps，允许Action被独立的测试。 Struts2的Action可以访问最初的请求(如果需要的话)。但是，尽可能避免或排除其他元素直接访问HttpServletRequest或 HttpServletResponse。Struts2的Action可以通过初始化、设置

属性、调用方法来测试。依赖注入的支持也使测试变得更简单。Struts2 直接使用 Action 属性作为输入属性，消除了对第二个输入对象的需求。输入属性可能是有自己(子)属性的 rich 对象类型。Action 属性能够通过 web 页面上的 taglibs 访问。Struts2 也支持 ActionForm 模式。rich 对象类型，包括 业务对象，能够用作输入/输出对象。这种 ModelDriven 特性简化了 taglib 对 POJO 输入对象的引用。使用 JSTL，但是也支持一个更强大和灵活的表达式语言——"Object Graph Notation Language" (OGNL)。使用“ValueStack”技术，使 taglib 能够访问值而不需要把你的页面（view）和对象绑定起来。ValueStack 策略允许通过一系列名称相同但类型不同的属性重用页面（view）。使用 OGNL 进行类型转换。提供基本和常用对象的转换器。支持通过 validate 方法和 XWork 校验框架来进行校验。XWork 校验框架使用为属性类类型定义的校验和内容校验，来支持 chain 校验子属性。支持通过拦截器堆栈（Interceptor Stacks）为每一个 Action 创建不同的生命周期。堆栈能够根据需要和不同的 Action 一起使用。

5.1.4 数据持久化

系统是一个有别与传统商业网站的一个 Web 应用系统，其业务逻辑相当复杂，牵涉到各种流程的判断以及状态变化的控制与处理。并且针对每个流程都需要在数据库中进行持续化操作。任何 N 层架构归根到底还是会和数据库打交道，良好的数据结构设计是一个系统成功的一半。

在中间层代码中，所有的业务通过对象模型来表现，业务之间的关系组成了一个对象图。而最终数据的表现形式，则体现为 RDBMS 中的记录之间的互相应用。所以持续化技术也称为 O/R Mapping 技术。

对于 Java 有许多持久化对象的方式。在进行架构选型的过程中，我们选择了 Hibernate 做为我们的处理方式：

- Hibernate

Hibernate 是一个轻量级的持续化框架实现，不同于 JDO 技术，Hibernate 对 POJO 的持续化操作不需要前台进行 enhance 的操作，Hibernate 还允许 POJO 对象和持续化引擎动态的 detach/attach，这些都是目前的 JDO 所无法提供的，而正是在真正的项目开发中经常用到的技术。Hibernate 还提供了一种 HQL 语言，开发者可以面向对象的 SQL 语言的形式，提供强大的数据查询功能。此外，Hibernate 还支持直接用标准 SQL 进行数据查询。在设计过程中，考虑到系统的适用范围以及业务需求的特点，决定采用 Hibernate 作为持续化框架。

在整体架构设计上，从核心的 DAO 访问层，到实现具体业务逻辑的 Domain Model 层，到提供统一调用界面的 Façade 层，均严格遵循 Sun 提出的 J2EE 核心设计模式进行设计。层次间功能划分明确，调用流程清晰，所以无论是从结构可伸缩性、可维护性、还是具体性能上，都有可靠的保障。

对一个 Web 应用系统来说，需求可能由于实际应用而瞬息万变，而且这些需求包括对于界面元素的改动和逻辑业务的变化。我们必须采用一种技术来把显示的任务和具体的逻辑处理模块隔离开，从而可以最大限度的减少改动的风险和成本。