
# 贵州大学

计算机科学与信息技术学院

# 基于Web的数字报系统设计


| | |
|---|---|
| 姓名 | 王晓娟 |
| 学号 | 0908060041 |
| 班级 | 计科092 |
| 指导老师 | 吴云 |


April 26, 2013

## 摘要

计算机技术不断发展，互联网技术不断崛起，Web逐渐成为了人们日常生活中不可获取的生命要素。报刊杂志等纸媒作为上一个时代的有利传播工具，将其Web化的步伐已刻不容缓。利用Web快速、精准以及多元化等特性，可以实现一个分众(Focus)、精准(Precise)、互动(Interactive)的多元化平台媒体。

**分众** 即用户差异化，服务方有效地区分受众人群

**精准** 即准确把握目标受众的商业需求

**互动** 即能够实现产业链上下游（上游：厂商客户；下游：终端用户）之间的信息互动、相互沟通、互通了解需求，形成有价值的互动信息回馈链条

而目前亟待解决的问题来自于在思想以及形式上如何从传统媒体平台转换到新媒体平台，现在的解决方案大多分为两类，一类是以电子媒体为主，多以Blog为主，主要战斗力集中在新媒体中与竞争对手进行周旋，在有余力之下，定期地推出某个系列的纸质/电子出版物，其中以SmashingMagzine/iFanr为代表。而另一类则以各大地方报纸、传统媒体行业为主，与前者相反，这些出版社的注意力仍然集中在纸媒体，而在新媒体平台缺乏创新力、创造力的做法，很少能够换得其用户的满堂喝彩。

本文将会以上述这一问题为根本需求依据，结合在实践中的真实经验，介绍了如何通过Express框架等一系列以ECMAScript为核心的

技术体系，进行快速搭建高效、稳定、可迭代的Web应用以及其开发模型。

论文共分为七个章节:

**第一章** 对摘要中的问题进行细致的分析，从企业的角度出发，解决其根本问题

**第二章** 介绍开发平台，包含程序、测试工具、部署平台所使用到的一系列开源工具

**第三章** 从前端出发，说明其架构思路，并提供前端架构图

**第四章** 从后端出发，说明其架构思路，并提供后端架构图

**第五章** 系统的各个组件的设计/实现思路

**第六章** 总结该系统的总体设计思路以及发展路线

**关键字** 数字报刊;NodeJS;MongoDB;系统设计

# 1 背景与意义

## 1.1 背景

如前所述，在传统媒介向新媒体转换、融合的阶段中，现存两种实现方式，其一是以数字媒体为主的网络博客，而另一类则是以传统纸媒为主的数字报格式，本系统旨在为后者解决其Web化的工业化难题。

**问题描述** 目前大多数数字报的Web化形式都仅停留在了形上，并未真正地融入了Internet元素。

那么什么是Internet，什么又是Web呢，简单地说，就是链接，两者都是增加了链接的数量与质量。例如谷歌、百度，是增进了网民获取所需的链接，而社交网络则是增进了人与人之间的链接，无论是何种形态的Web应用都是以人为出发点，去增进两种事物之间链接的应用程序。

而如今的数字报，从链接上仅仅是增加了一个从头版到文章的链接，但缺少了一个重要的因素，这样的链接并未从人、从用户的角度去考虑，无论是从浏览体验，还是其网络属性来说，都仅仅是将报纸铺在电子显示屏而已。

那么真正的Web化是怎样的呢？可以分别从三个方面来逐一体现：

**分众** 细分用户群，对不同的用户群实现不同的运营策略，要做到这点，需要对用户数据进行收集、建模，为后两方面的实现提供基础数据模型。

**精准** 精准是建立在分众之上，没有很好地细分用户，就没办法精准地了解到特定用户模型的刚性需求。

**互动** 你的用户在使用产品时，需要一个快速、实时的平台与你进行交流，共同改进产品，而这种交流可以是显式的自然语言，同样也可以是其他形式来实现。

本章节将会围绕这三个需求，慢慢向你揭开Web化报刊的神秘面纱。

但对于传统报社，因接不暇地业务需求遽增，同时又要分心涉足互联网，无疑是在蚕食其有限的战斗能力，此时该系统就应运而生，论文最后所实现的这一Web应用即是一个针对广大传统报社的SaaS(Software as a Service)云服务。

利用这一服务，注册的报社可以轻松地利用已有的资源，生成电子期刊，并获取在线订阅用户，作为一个纯粹的内容提供商(CP, Content Provider)的同时，我们将来还会为报社提供其电子产品的用户数据，方便其针对数字用户制定相应的商业策略。

## 1.2 需求及系统功能分析

### 1.2.1 目标用户

该服务即一综合服务性平台，既拥有上节中说的企业级用户，同时作为一个报刊阅读平台，也拥有着以阅读为目标的个人用户。前者消费流量，而后者提供流量，从而构成了一个良性的生态循环。

### 1.2.2 需求分析

**企业级用户** 这类用户的需求相对比较复杂，他们需要的是一个集数字报刊发行、数据统计平台为一体的云服务，我们除了提供此类服务之外，还

应该尽量地简化他们发布数字报刊的时间成本，提供阅读质量。

**个人用户** 这是一类需求相对简单的用户群，它们使用我们服务的理由就是以快速获取信息为主。因此我们需要提供一个各终端平台都无障碍的阅读平台，同时尽力地提高其阅读体验。

### 1.2.3 功能分析

**数字报刊发布** 从企业级用户的根本需求出发，以节省其时间代价为根本，极力地简化其发布流程，用户在成功创建一类报刊后，在发布期刊时，仅仅需要上传一份PDF格式文件，我们的服务中心会提供用户分析数据，提供适配所有终端的阅读页面。

**数字报阅读平台** 从个人用户的根本需求出发，为他们提供一个易于阅读，易于分享的阅读平台。

**数据统计平台** 同样为企业级用户量身定做，基于数据可视化技术为企业级用户快速、便捷地筛选出自己想要的信息。

### 1.2.4 性能需求

**高并发** 互联网应用与生俱来的高并发性要求无疑是对程序设计的一大挑战，因此整个系统需要低成本地解决高并发的问题。

**数据安全** 系统应当保证用户能正常、稳定、安全地使用服务。

## 1.3 小结

根据以上需求分析，可以将系统划分为3个子模块，分为：用户/组管理、报刊发布与管理平台、阅读平台，如图1(Figure 1)。

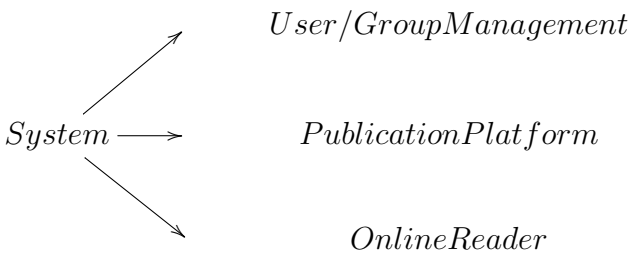


Figure 1: 系统模块分支

用户/组管理包括用户认证、用户分组两个功能项目。一是用于保证登陆系统以及操作的安全性，二是用于差异化用户，分别针对企业级用户与个人用户，设置相应的操作权限。相对应的：企业级用户有权使用报刊发布管理平台的资源，对其发布的报刊进行发布、添加、删除、修改等管理，而个人用户作为流量提供者，仅仅能使用阅读平台，来满足自己的阅读需求，同时贡献流量。

报刊发布与管理平台关系较为复杂，首先需要详细说明一下报刊与用户之间的关系。

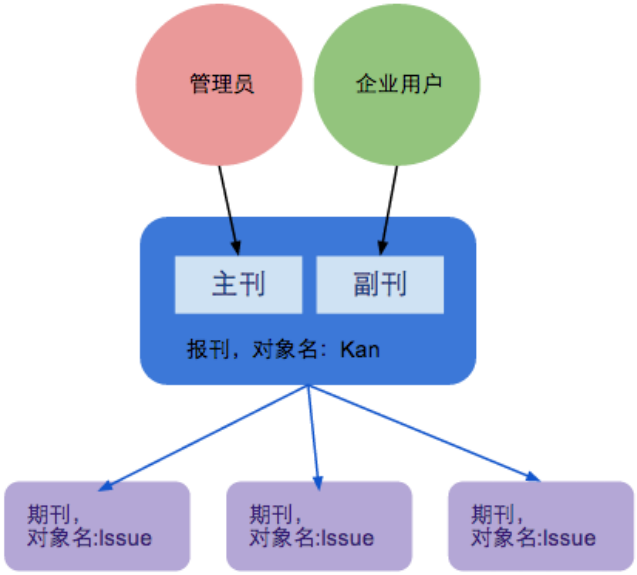


Figure 2: 报刊/用户关系图

对象Kan包含如下属性(Property)：

**name** 报刊名字

**user** 创建该报刊的企业或高级用户

**group** 该报刊所属的分类，如：综合、科技等

**tags** 创刊人为报刊添加的标签，用于更多元化的搜索

**cover** 报刊的封面图片

**description** 创刊人对该报刊的描述

对象Issue包含如下属性(Property)

**title** 期刊的标题

**date** 期刊发布时间

**content** 期刊的内容

**kan** 所属的报刊

主刊由系统管理员进行维护，副刊由我们的高级用户自主维护，主刊更多是作为一综合平台，起到索引的目的，而副刊则可专注于一类或几类媒体。而对于报刊发行系统而言，管理员与高级用户的界面有所区别，但并不是在完全不同的区域进行，甚至可以说只是他们权限不同，因此看到的界面也不同而已。我们将会使用模版嵌套技术来降低这部分代码的冗余。

在线阅读平台对服务器的查询需求相当庞大，需要快速地将数据服务器中的PDF文件或html代码即时地返回到浏览器端。在文件到达浏览器端后，将选择Mozilla实验室的pdf.js来完成pdf文件的渲染工作。

# 2 开发平台介绍

## 2.1 B/S体系结构

B/S体系结构与C/S体系结构相比，不仅具有其大部分优点，还有其不具备的优势

**开放的标准** B/S所采用的标准都是开放的、非专用的，是经过标准化组织所确定而非单一厂商所制定，保证了应用的通用性与跨平台性。

**开发/维护成本低** B/S应用只需在客户端安装通用的浏览器，维护和升级的工作都在服务端完成，不需要对客户端进行任何改变。

由以上分析比较可看出，B/S模式具有更遥远的应用前景，它简化开发与维护流程，并适合在网络中发布信息、传播信息。

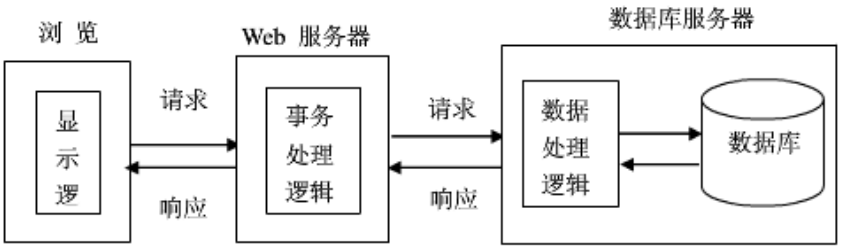


Figure 3: B/S三层结构图

## 2.2 开发语言

本系统采用了一些相对比较新潮的开发语言，如SeaJS、NodeJS以及MongoDB。其中NodeJS作为HTTP服务器平台，其优点是异步编程模型，使用了异步非阻塞(noblocking I/O)来处理复杂的并发Web请求。为

了快速搭建web服务平台上的相关功能，我们选择使用了NodeJS平台上最流行的web开发框架Express，之所以选择它的原因是由于其灵巧的微内核设计方式与插件机制，可以很好地对框架本身的功能进行拓展。

数据库我们使用了新兴的mongoDB作为数据库服务器平台，它被誉为最接近SQL的NoSQL数据库，既不完全舍弃SQL的一些优秀设计理念，又能够使用NoSQL所带来的高效优势，并且在选择NodeJS客户端驱动时，我们选择了带有模式(Schema)设计的mongoose为主要客户端驱动，也是为了利用SQL的基本设计理论帮助我们保证数据与数据库的可维护性。

随着前端开发逐步走向工业化的道路，前端模块化会是影响web应用一个重要的因素，而我们使用SeaJS作为前端工业化平台，SeaJS所遵循的CMD规范与NodeJS的模块(Module)兼容性良好，关于前端模块化的详细内容将在后续章节详细说明。

### 2.2.1 系统开发环境及工具

| | |
|---|---|
| 硬盘空间 | 8G及以上 |
| CPU | 1.7 GHz Intel Core i7 |
| 内存 | 4G |
| 操作系统 | MacOS X |
| 开发平台 | NodeJS/Express |
| 数据库 | MongoDB |

## 2.3 NodeJS

### 2.3.1 NodeJS的前世今生

NodeJS是一个可以快速构建网络服务及应用的平台。该平台的构建是基于Chrome's JavaScript runtime，也就是说，实际上它是对GoogleV8引擎进行了封装。

V8引擎执行Javascript的速度非常快，性能非常好。Node对一些特殊用例进行了优化，提供了替代的API，使得V8在非浏览器环境下运行得更好。

V8引擎本身使用了一些最新的编译技术。这使得用Javascript这类脚本语言编写出来的代码与用C这类高级语言写出来的代码性能相差无几，却节省了开发成本。对性能的苛求是Node的一个关键因素。Javascript是一个事件驱动语言，Node利用了这个优点，编写出可扩展性高的服务器。Node采用了一个称为"事件循环(event loop）"的架构，使得编写可扩展性高的服务器变得既容易又安全。提高服务器性能的技巧有多种多样。Node选择了一种既能提高性能，又能减低开发复杂度的架构。这是一个非常重要的特性。并发编程通常很复杂且布满地雷。Node绕过了这些，但仍提供很好的性能。

Node采用一系列"非阻塞"库来支持事件循环的方式。本质上就是为文件系统、数据库之类的资源提供接口。向文件系统发送一个请求时，无需等待硬盘（寻址并检索文件），硬盘准备好的时候非阻塞接口会通知Node。该模型以可扩展的方式简化了对慢资源的访问，直观，易懂。尤其是对于熟悉onmouseover、onclick等DOM事件的用户，更有一种似曾相

识的感觉。

虽然让Javascript运行于服务器端不是Node的独特之处，但却是其一强大功能。不得不承认，浏览器环境限制了我们选择编程语言的自由。任何服务器与日益复杂的浏览器客户端应用程序间共享代码的愿望只能通过Javascript来实现。虽然还存在其他一些支持Javascript在服务器端运行的平台，但因为上述特性，Node发展迅猛，成为事实上的平台。

在Node启动的很短时间内，社区就已经贡献了大量的扩展库（模块）。其中很多是连接数据库或是其他软件的驱动，但还有很多是凭他们的实力制作出来的非常有用的软件。

不得不提到的是Node社区。虽然Node项目还非常年轻，但很少看到对一个项目如此狂热的社区。不管是新手，还是专家，大家都围绕着项目，使用并贡献自己的能力，致力于打造一个探索、支持、分享、听取建议的乐土。

### 2.3.2 NodeJS的单线程

NodeJS中的JavaScript在单线程上执行，但是作为宿主的NodeJS，它本身并非是单线程的，NodeJS在I/O方面动用到一小部分额外的线程协助实现异步。程序员没有机会直接创建线程，因此有人想当然的认为NodeJS的单线程无法很好的利用多核CPU。

NodeJS封装了内部的异步实现后，导致程序员无法直接操作线程，也就造成所有的业务逻辑运算都会丢到JavaScript的执行线程上，这也就意味着，在高并发请求的时候，I/O的问题是很好的解决了，但是所有的业务逻辑运算积少成多地都运行在JavaScript线程上，形成了一条拥挤

的JavaScript运算线程。NodeJS的弱点在这个时候会暴露出来，单线程执行运算形成的瓶颈，拖慢了I/O的效率。这大概可以算得上是密集运算情况下无法很好利用多核CPU的缺点。这条拥挤的JavaScript线程，给I/O形成了性能上限。

事情又并非绝对的，NodeJS提供了child_process.fork来创建Node的子进程。在一个Node进程就能很好的解决密集I/O的情况下，fork出来的其余Node子进程可以当作常驻服务来解决运算阻塞的问题（将运算分发到多个Node子进程中上去，与Apache创建多个子进程类似）。当然child_process机制永远只能解决单台机器的问题，大的Web应用是不可能一台服务器就能完成所有的请求服务的。拜NodeJS在I/O上的优势，跨OS的多Node之间通信的是不算什么问题的。解决NodeJS的运算密集问题的答案其实也是非常简单的，就是将运算分发到多个CPU上。

### 2.3.3 基于事件的编程

延续上一节的讨论。我们知道NodeJS具有异步的特性，从NodeJS的API设计中可以看出来，任何涉及I/O的操作，几乎都被设计成事件回调的形式，且大多数的类都继承自EventEmitter。这么做的好处有两个，一个是充分利用无阻塞I/O的特性，提高性能；另一个好处则是封装了底层的线程细节，通过事件消息留出业务的关注点给编程者，从而不用关注多线程编程里牵扯到的诸多技术细节。

事件式编程更贴近于现实生活，是更自然的，所以这种编程风格也导致你的代码跟你的生活一样，是一件复杂的事情。幸运的是，自己的生活要自己去面对，对于一个项目而言，并不需要每个人都去设计整个大业务逻

辑，对于架构师而言，业务逻辑是明了的，借助事件式编程带来的业务逻辑松耦合的好处，在设定大框架后，将业务逻辑划分为适当的粒度，对每一个实现业务点的程序员而言，并没有这个痛苦存在。二八原则在这个地方非常有效。

## 2.4 Express.js——NodeJS平台上的Web框架

Express.js是基于NodeJS，高性能、一流的web开发框架，它通过对NodeJS的部分接口进行重新封装，使其更易于适用在web的世界中，并且提供了良好的插件机制，可以灵活地向应用中添加模版引擎，中间件等，因而使其具有一定的生态活力。

## 2.5 NoSQL与MongoDB

随着互联网web2.0网站的兴起，非关系型的数据库成了一个极其热门的新领域，非关系数据库产品的发展非常迅速。而传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，例如：

**对数据库高并发读写的需求** web2.0网站要根据用户个性化信息来实时生成动态页面和提供动态信息，所以基本上无法使用动态页面静态化技术，因此数据库并发负载非常高，往往要达到每秒上万次读写请求。关系数据库应付上万次SQL查询还勉强顶得住，但是应付上万次SQL写数据请求，硬盘IO就已经无法承受了。其实对于普通的BBS网站，往往也存在对高并发写请求的需求。

**对海量数据的高效率存储和访问的需求** 对于大型的SNS网站，每天用户产生海量的用户动态，以国外的Friendfeed为例，一个月就达到了2.5亿条用户动态，对于关系数据库来说，在一张2.5亿条记录的表里面进行SQL查询，效率是极其低下乃至不可忍受的。再例如大型web网站的用户登录系统，例如腾讯，盛大，动辄数以亿计的帐号，关系数据库也很难应付。

**对数据库的高可扩展性和高可用性的需求** 在基于web的架构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，你的数据库却没有办法像web server和app server那样简单的通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供24小时不间断服务的网站来说，对数据库系统进行升级和扩展是非常痛苦的事情，往往需要停机维护和数据迁移，为什么数据库不能通过不断的添加服务器节点来实现扩展呢?

在上面提到的“三高”需求面前，关系数据库遇到了难以克服的障碍，而对于web2.0网站来说，关系数据库的很多主要特性却往往无用武之地，例如:

**数据库事务一致性需求** 很多web实时系统并不要求严格的数据库事务，对读一致性的要求很低，有些场合对写一致性要求也不高。因此数据库事务管理成了数据库高负载下一个沉重的负担。

**数据库的写实时性和读实时性需求** 对关系数据库来说，插入一条数据之后立刻查询，是肯定可以读出来这条数据的，但是对于很多web应用来说，并不要求这么高的实时性。

**对复杂的SQL查询，特别是多表关联查询的需求** 任何大数据量的web系统，都非常忌讳多个大表的关联查询，以及复杂的数据分析类型的复杂SQL报表查询，特别是SNS类型的网站，从需求以及产品设计角度，就避免了这种情况的产生。往往更多的只是单表的主键查询，以及单表的简单条件分页查询，SQL的功能被极大的弱化了。

因此，关系数据库在这些越来越多的应用场景下显得不那么合适了，为了解决这类问题的非关系数据库应运而生。

NoSQL 是非关系型数据存储的广义定义。它打破了长久以来关系型数据库与ACID理论大一统的局面。NoSQL 数据存储不需要固定的表结构，通常也不存在连接操作。在大数据存取上具备关系型数据库无法比拟的性能优势。该术语在2009 年初得到了广泛认同。

**MongoDB** 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。它的特点是高性能、易部署、易使用，存储数据非常方便。

主要功能特性有:

- 面向集合存储，易存储对象类型的数据。
- 模式自由。

- 支持动态查询。
- 支持完全索引，包含内部对象。
- 支持查询。
- 支持复制和故障恢复。
- 使用高效的二进制数据存储，包括大型对象（如视频等）。
- 自动处理碎片，以支持云计算层次的扩展性。
- 支持RUBY，PYTHON，JAVA及NodeJS等多种语言。
- 文件存储格式为BSON（一种JSON的扩展）。
- 可通过网络访问。

所谓“面向集合”(Collection-Oriented)，意思是数据被分组存储在数据集中，被称为一个集合(Collection)。每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似关系型数据库(RDBMS)里的表(table)，不同的是它不需要定义任何模式(schema)。

模式自由(schema-free)，意味着对于存储在mongodb数据库中的文件，我们不需要知道它的任何结构定义。如果需要的话，你完全可以把不同结构的文件存储在同一个数据库里。

存储在集合中的文档，被存储为键-值对的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是各种复杂的文件类型。我们称这种存储形式为BSON(Binary Serialized dOcument Format)。

## 2.6 SeaJS前端平台

SeaJS是一个遵循CommonJS规范的JavaScript模块加载框架，可以实现其的模块化开发及加载机制。与jQuery等框架不同，SeaJS不会扩展封装语言特性，而只是实现JavaScript的模块化及按模块加载。SeaJS的主要目的是令JavaScript开发模块化并可以轻松愉悦进行加载，将前端工程师从繁重的JavaScript文件及对象依赖处理中解放出来，可以专注于代码本身的逻辑。SeaJS可以与jQuery这类框架完美集成。使用SeaJS可以提高JavaScript代码的可读性和清晰度，解决目前JavaScript编程中普遍存在的依赖关系混乱和代码纠缠等问题，方便代码的编写和维护。

SeaJS的作者是淘宝前端工程师玉伯。

SeaJS本身遵循KISS（Keep It Simple, Stupid）理念进行开发，其本身仅有个位数的API，因此学习起来毫无压力。在学习SeaJS的过程中，处处能感受到KISS原则的精髓——仅做一件事，做好一件事。

**RequireJS与SeaJS**

RequireJS作为Javascript模块化开发中的前辈，与SeaJS都是模块加载器，倡导模块化开发理念，核心价值是让JavaScript的模块化开发变得简单自然。而二者的区别是:

**定位有差异** RequireJS 想成为浏览器端的模块加载器，同时也想成为Rhino / Node 等环境的模块加载器。Sea.js 则专注于Web 浏览器端，同时通过Node 扩展的方式可以很方便跑在Node 环境中

**遵循的规范不同** RequireJS 遵循AMD（异步模块定义）规范，Sea.js 遵循CMD （通用模块定义）规范。规范的不同，导致了两者API 不

同。Sea.js 更贴近CommonJS Modules/1.1 和Node Modules 规范。

**推广理念有差异** RequireJS 在尝试让第三方类库修改自身来支持RequireJS，目前只有少数社区采纳。Sea.js 不强推，采用自主封装的方式来“海纳百川”，目前已有较成熟的封装策略

**对开发调试的支持有差异** Sea.js 非常关注代码的开发调试，有nocache、debug 等用于调试的插件。RequireJS 无这方面的明显支持

**插件机制不同** RequireJS 采取的是在源码中预留接口的形式，插件类型比较单一。Sea.js 采取的是通用事件机制，插件类型更丰富。

# 3 服务端架构细节

## 3.1 模块对象设计

由于使用的是基于ECMAScript262的NodeJS为基本开发语言，因此并没有传统面向对象中的类，但并不能说其不支持面向对象的特性。我基于Express.js开发了一个数据库连接工具——express-model，另外它也是一个建模工具，系统中所有的模型都被放在根目录下一个叫models的文件夹中，下面分别对其中的文件进行描述。

### 3.1.1 express-model

在描述对象模型之前，先来看看如何使用express-model，首先是更加仔细地对它进行一段描述：它为Express框架提供Model支持，可集成多种数据库(SQL & NoSQL)，灵活性高的轻量级工具。由于Express官方并未显式地提供Model支持，express-model不仅能有效嵌入Express Mvc中，还支持多种需要Model的场景。模型作为MongoDB与Express的接口，实现了两端数据的同步功能。

创建一个`Model`:

```
Models.define('yourModelName', function(exports) {
        exports.name = 'express-model 1.0'
        exports.getName = function() {
                return this.name
        }
})
```

使用Model:

```
var model = Models.use('yourModelName', function() {
        this.someVariable_1 = 1;
        this.someVariable_2 = 2;
})
response.render('yourViewName', model)
```

### 3.1.2 user.js

其模型定义代码如下:

```
var UserSchema = new Db.Schema({
        name: String,
        email: String,
        password: String,
        role: String
})
UserSchema.index({ name: 1, role: 1 })
```

user.js模型主要用于管理用户与组相关的操作。它提供了一些方法供http服务端请求调用:

**toJSON** 将users对象重新组合，转换为更适合前端请求的json形式;

**add** 用于注册新用户时，将请求的新用户数据同步到数据库服务器中;

**view_homepage** 用于视图页homepage，为其提取相关的信息;

### 3.1.3 groups.js

其模型定义代码如下:

```
var GroupSchema = new Db.Schema({
        content: [ String ]
}, {
        collection: 'groups'
})
```

groups.js模型作为一个独立的集合(Collection)存储在数据库服务器中，提供了一个只读字符串数组。详细的数据库细节将在下一小节给出。

### 3.1.4 kan.js

其模型定义代码如下:

```
var KanSchema = new Db.Schema({
        user: String,
        name: String,
        group: String,
        tags: String,
        description: String,
        cover: String
})
```

```
KanSchema.index({name: 1, group: 1})
```

### 3.1.5 issue.js

其模型定义代码如下:

```
var IssueSchema = new Db.Schema({
        title: String,
        content: String,
        date: Date,
        kan: String,
})
IssueSchema.index({kan: 1})
```

## 3.2 数据库设计

数据结构在计算机中的表示（映像）称为数据的物理（存储）结构，它包括元素表示和关系的表示。

数据库在物理设备上的存储结构与存取方法称为数据库的物理结构，它依赖于给定的计算机系统。为一个给定的逻辑数据模型选取一个最符合应用要求的物理结构的过程就是数据库物理结构的设计。

| 字段名称 | 数据类型 | 描述 |
| --- | --- | --- |
| _id | Objectid | 唯一标示 |
| email | String | 同时作为登陆名 |
| name | String | 用户昵称 |
| password | String | 用户密码，暗文保存 |
| role | Int32 | 用户所属分组 |

**collection 1: users**

| 字段名称 | 数据类型 | 描述 |
| --- | --- | --- |
| _id | Objectid | 唯一标示 |
| group | Array | |

**collection 2: groups**

| 字段名称 | 数据类型 | 描述 |
| --- | --- | --- |
| _id | Objectid | 唯一标示 |
| user | String | 创刊用户 |
| name | String | 报刊名字 |
| group | String | 所属分类 |
| tags | String | 附加标签 |
| description | String | 描述 |

**collection 3: kans**

| 字段名称 | 数据类型 | 描述 |
| --- | --- | --- |
| _id | Objectid | 唯一标示 |
| title | String | 标题 |
| content | String | 期刊内容 |
| date | Date | 发布时间 |
| kan | String | 所属报刊 |

**collection 4: issues**

## 3.3 服务端架构图

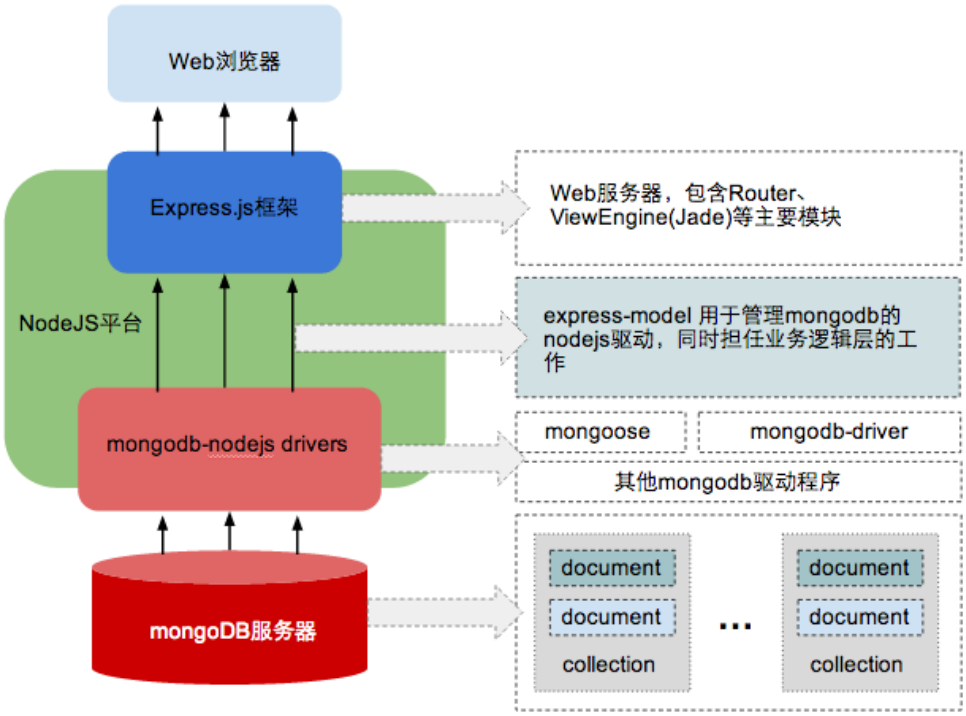


Figure 4: 服务端架构图

# 4 前端架构细节

## 4.1 模块化的前端开发

说起模块化，也许首先想到的就是编程中的模块设计，以功能块为单位进行程序设计，最后通过模块的选择和组合构成最终产品。把这种思想运用到页面构建中，也已经不是什么新鲜事。很大一部分页面架构师都经历了这样几个阶段：第一阶段是在一个css文件中把多个页面按自己的习惯顺序从上往下编写样式，基本不考虑有无公用样式，以完成设计呈现为首要目的；第二阶段是提取不同页面中的通用样式，如公用颜色、图标、按钮等，实现一些基本元素的复用；第三阶段是提取公用功能模块，如导航、版权信息等，实现部分公用模块的复用。

刚才描述的第三阶段的方法已经包含了模块化思想，不少团队也都有一套成熟的模块化开发方案。某些产品中要求使用一种称为UIO方式，模块化通用的功能模块或组件，以达到最大程度的模块独立性与复用性，过去很多工程师认为这种工作方式约束了编码的自由性，过多的结构约束反而降低了工作效率，加之产品之间也存在不统一，最后并没有运用到整个团队。

有一篇关于面向对象CSS的文章中指出，面向对象的CSS有两个主要原则：separate the structure from the skin，separate the container from the content。第一个原则体现在模块化思想可以理解为，模块的设计制作和布局框架本身相分离，意味着模块不能只为某个布局而编写样式，像存在换肤功能的产品更是如此，如果模块在不同的皮肤样式下需要另写很多样式甚至是修改结构的时候，这个模块的制作就是失败的；第二个原则说的布

局与内容的分离，布局中某个位置不必只能放置某种内容，反过来可以理解为模块的灵活性和复用性。

另外，随着web应用不断发展和对JavaScript依赖的进一步加深，出现了使用模块（Modules）来组织代码和依赖性。模块使得我们创建明确清晰的组件和接口，这些组件和接口能够很容易的加载并连接到其依赖组件。AMD模块系统提供了使用JavaScript模块来构建Web应用的完美方式，并且这种方式具有形式简单，异步加载和广泛采用的特点。

**模块系统**

异步模块定义（AMD）格式是一套API，它用于定义可重用的并能在多种框架使用的模块。开发AMD是为了提供一种定义模块的方式，这种方式可以使用原生的浏览器脚本元素机制来实现模块的异步加载。AMD API由2009年Dojo 社区的讨论中产生，然后移动到讨论CommonJS如何更好的为浏览器适应CommonJS模块格式（被NodeJS使用）。CommonJS已经发展成为单独的一个标准并有其专门的社区。

模块化系统的基础前提是:

- 允许创建被封装的代码片段，也就是所谓的模块
- 定义本模块与其他模块之间的依赖
- 定义可以被其他模块使用的输出的功能
- 谨慎的使用这些模块提供的功能

ADM与CommonJS都满足以上需求，并将依赖模块设置为其回调函数的参数从而实现在模块代码被执行前异步的加载这些依赖模块。

AMD格式提供了几个关键的好处。首先，它提供了一种紧凑的声明依赖的方式。通过简单的字符串数组来定义模块依赖，使得开发者能够花很小的代价轻松列举大量模块依赖性。

AMD帮助消除对全局变量的需求。每个模块都通过局部变量引用或者返回对象来定义其依赖模块以及输出功能。因此，模块不需要引入全局变量就能够定义其功能并实现与其他模块的交互。AMD同时是“匿名的”，意味着模块不需要硬编码指向其路径的引用，模块名仅依赖其文件名和目录路径，极大的降低了重构的工作量。

通过将依赖性映射为局部变量，AMD鼓励高效能的编码实践。如果没有AMD模块加载器，传统的JavaScript代码必须依赖层层嵌套的对象来“命名”给定的脚本或者模块。如果使用这种方式，通常需要通过一组属性来访问某个功能，这会造成全局变量的查找和众多属性的查找，增加了额外的开发工作同时降低了程序的性能。通过将模块依赖性映射为局部变量，只需要一个简单的局部变量就能访问某个功能，这是极其快速的并且能够被JavaScript引擎优化。

**ADM的局限**

AMD为我们提供了一个模块加载并协同工作的重要层面。然而，AMD仅仅是模块定义。它并不能为模块创建的API开出任何通用的“特别处方”。比如，你不能指望模块加载器给你提供查询引擎，并期望它从一堆可替换的查询模块中给你返回一个通用的API。当然定义这样的API更利于模块交互，但这不在AMD的范畴内。大多数模块加载器不支持将模块标识映射到不同的路径，因此如果你有可替换的模块，你最好自己定义一个模块标识到不同目标路径的映射来解决这一点。

## 4.2 前端UI库——bootstrap.js

Web前端开发者每天都与HTML、CSS、JavaScript打交道，然而不少人都是周而复始地写模板、样式和交互效果，并没有想过如何将这些重复的工作整合在一起。Twitter推出的Bootstrap能够帮助Web前端开发者摆脱这种重复劳动。

为了应对复杂的需求，早期的Twitter前端工程师在开发网站时几乎采用了所有自己熟悉的前端库。造成了网站维护困难、扩展性不强、开发成本高等问题。此时BootStrap被提上了日程。Twitter要求前端工程师完全依靠这一单一框架进行前端开发。

Twitter 在2011年8月将其开源，并在2012年2月3日发布了2.0版。这个项目已有拥超过2万位关注者和4000个分支。Bootstrap的设计者、著名前端工程师Mark Otto这样写道：“Bootstrap是我和Jacob Thornton编写的一个前端工具箱，目的是为了帮助设计师和Web前端开发人员快速有效地创建一个结构简单、性能优良、页面精致的Web应用。它使用了最新的浏览器技术，可以提供精致的网页排版方式以及表单、按钮、表格、网格栅格化、导航等诸多元素。”Bootstrap的内置样式继承了Mark Otto简洁亮丽的设计风格，任何开发团队都能使用它提供的HTML模板、CSS样式和jQuery组件来布署或者重建一个外观漂亮的页面应用。

BootStrap 2在原有特性的基础上着重改进了用户的体验和交互性，比如新增加的媒体展示功能，适用于智能手机上多钟屏幕规格的响应式布局，另外新增了12款jQuery插件，可以满足Web页面常用的用户体验和交互功能。

如今的Bootstrap已包括了几十个组件，每个组件都自然地结合了设计与开发，具有完整的实例文档，定义了真正的组件和模板。无论处在何种技术水平的开发者，也无论处在哪个工作流程中，都可以使用Bootstrap快速、方便地构建开发者喜欢的应用。难能可贵的是，Bootstrap依旧本着“并行开发”、“作为产品的风格指南”和“迎合所有的技能水平”的原则帮助开发者解决实际问题，不断完善自己，吸引更多人选择Bootstrap应用于自己的项目中。

然而古人云“万物相生相克”，有好就有坏，Bootstrap也是一样。对于在国内的开发者来说，最可怕的就是IE兼容问题。目前Bootstrap对IE6到IE8的支持都不友好。另一个缺点是，采用Bootstrap的模板，网站结构时常会显得臃肿。此外，覆盖一些样式时会造成代码冗余。但与其他前端框架相比，我个人觉得Bootstrap的缺点仅此而以，至于其他方面希望有机会与大家一起探讨和学习。

Bootstrap是一套前端开发利器。它可以帮助我们加速项目开发，让我们身处在一个完备的系统中，拥有一致的设计和实现方法。不需要在外观上花费过多时间，使开发者能将精力集中于更重要的功能。

Bootstrap将改变我们的合作方式与开发进程，任何人都可以基于Bootstrap建立可扩展的前端工具包，或者在它的基础上启动属于自己的框架。

## 4.3 前端模块架构

在开始解释前端模块之前，先通过几段简单的代码来熟悉一下SeaJS的使用方法:

```
define(function(require, exports, module) {
        var moduleA = require('A');
        exports.ret = moduleA;
}
```

上面的代码用于定义一个模块，其中require是一个方法，用来获取其他的模块，如：require('A')，exports是一个输出对象，相当于return关键字。

```
seajs.use(['moduleA'], function(A) {
        A;
})
```

这段代码演示了如何使用定义好的模块，但它一般在初始化才被使用，其他方式都是通过在其他定义(define)模块中使用require方法来实现。

### 4.3.1 初始化Javascript模块

在请求一个页面后，默认载入位于根目录下assets/scripts/index.js模块，不过可以通过在视图页如下声明来改变初始化的Javascript模块：

```
block Module
        script(src='/scripts/pages/issue.js')
```

下面列出assets/scripts下主要文件或文件夹的作用：

**sea-config.js** seajs的配置模块

**index.js** 默认的初始化模块

**lib** 浏览器端需要用到的Javascript类库

**utils** 工具模块组，多用于兼容浏览器、独立不依赖于大量css的UI模块等

**widgets** 大型UI组件，依赖大量的css样式以及模版(html/jade/...)

**pages** 与视图(view)所对应的初始化模块

### 4.3.2 视图

所有的视图都位于根目录下的views下，views/shared用于存放公用的视图。系统内所有的模版都是使用一个名为Jade的模版语法书写的。

**Jade/Haml**

Jade是一款高性能简洁易懂的模板引擎，Jade是Haml的Javascript实现，在服务端（NodeJS）及客户端均有支持。

Haml是一种用来描述XHTML web document的标记语言，它是干净，简单的。而且也不用内嵌代码。Haml的职能就是替代那些内嵌代码的page page templating systems，比如PHP，ERB(Rails的模板系统)，ASP。不过，haml避免了直接书写XHTML代码到模板，因为它实际上是一个xhtml的抽象描述，内部使用一些代码来生成动态内容。Haml 是一种简洁优美的模板语言，可以应用于Ruby on Rails、PHP等Web开发平台，可以大大缩减模板代码，减少冗余，提高可读性。并且Haml是一种完备的模板语言，没有牺牲当前模板语言的任何特性。Haml由Hampton Catlin发明并且开发了Ruby on Rails上的实现。

**模版依赖**

Jade支持模版嵌套，通常是通过block标记来实现这一功能，这样可以大大地减少模版代码的冗余与数量，不过也相应会出现模版依赖的问题，往往在一个存在复杂视图关系的web应用中，视图会越来越复杂，也越来越难以维护。

Web开发过程中，视图通常需要经常更改，或者增加、删除。这样就需要工程师除了写好视图文件内的模版代码外，还需要考虑清楚视图文件夹(views)内的文件架构，即如何命名，如何分类视图文件、模版文件等，后者与路由(Router)设计、控制器(Controller)设计是相互依赖、相互影响的关系。

根模版views/shared/skeleton.jade代码如下：

```
doctype 5
html
  head
    block Head
      meta(name='viewport',
        content='width=device-width,
        initial-scale=1.5')
      link(rel='stylesheet',
        href='//stk.ikanbao.fm:3000/css/bootstrap.css')
      title login
      script
        var ENV = {}
  body
```

```
    block Content
      include header
    script(src='/scripts/lib/sea.js')
    script(src='/scripts/sea-config.js')
    block Module
      script(src='/scripts/index.js')
```

在views/shared还能发现一个名为sidebar-and-content.jade的视图文件，虽然该文件也是基于skeleton模版，不过它们的用处各不相同，sidebar-and-content用于应用中的分栏布局，比如视图文件welcomAuthenicated部分代码如下：

```
// Welcome Authenticated
extend shared/sidebar-and-content
block append head
block append content
```

## 4.4 前端架构图

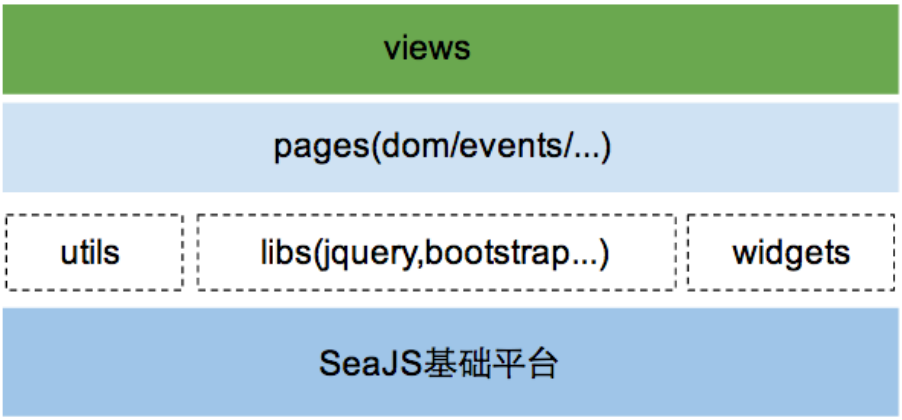


Figure 5: 前端架构图

# 5 系统功能实现

## 5.1 用户/组

### 5.1.1 用户认证

**PBKDF2(Password-Based Key Derivation Function)加密算法**

PBKDF2简单而言就是将salted hash进行多次重复计算，这个次数是可选择的。如果计算一次所需要的时间是1微秒，那么计算1百万次就需要1秒钟。假如攻击一个密码所需的彩虹表有1千万条，建立所对应的表所需要的时间就是115天。这个代价足以让大部分的攻击者忘而生畏。

而该加密算法在NodeJS标准模块中的crypto模块，方法接受5个参数，如crypto.pbkdf2(pwd, salt, iterations, len, fn)，分别代表：明文字符串、随机码salt、迭代次数、字节长度、回调函数。在fn中将会得到随机码值和hash值(暗文)。

加密算法的核心代码被封装在了lib/auth/pass模块内，代码如下：

```
/**
 * Module dependencies
 */
var crypto = require('crypto')


/**
 * Bytesize
 */
```

```
var len = 128


/**
 * Iterations. ~300ms
 */
var iterations = 12000


/**
 * Hashes a password with optional 'salt', otherwise
 * generate a salt for 'pass' and
 *   invoke `fn(err, salt, hash)`.
 *
 * @param {String} password to hash
 * @param {String} optional salt
 * @param {Function} callback
 * @api public
 */
exports.hash = function (pwd, salt, fn) {
  if ( 3 == arguments.length )
    crypto.pbkdf2(pwd, salt, iterations, len, fn)
  else {
    fn = salt
```

```
    crypto.randomBytes(len, function(err, salt){
      if (err) return fn(err)
      salt = salt.toString('base64')
      crypto.pbkdf2(pwd, salt, iterations, len,
       function(err, hash){
        if (err) return fn(err)
        fn(null, salt, hash)
      })
    })
  }
}
```

用户信息编码/解码(Encode/Uncode)的核心代码如下:

```
/**
 * encode a user
 */
function encode(user, fn) {
  hash(user.password, function(err, salt, hash) {
    if (err) {
      return fn(err, null)
    }
    user.salt = salt
    user.hash = hash
```

```
    fn(err, user)
  })
}
```

```
/**
 * uncode a user
 */
function uncode(user, fn) {
  var __u_ = isExisted(user)
  var pass = encodePassword(user)
  if(!__u_) {
    fn(new Error('cannot find user'))
  }
  hash(pass, __u_.salt, function(err, hash) {
    if (err) return fn(err)
    if (hash == __u_.hash) return fn(null, __u_)
    fn(new Error('invalid password'))
  })
}
```

在uncode(user, fn)内的实现代码中，使用到了两个未显示的函数：isExisted(user)和encodePassword(user)，其用途分别是判断一个用户是否存在，与对密码进行编码。

**用户认证机制**

系统在启动服务器后，优先从数据库中读取所有的用户信息，并通过flashDB这么一个缓存管理器，写入到内存中。

```
/**
 * ready workers
 */
function ready() {
  db.collection('users').find().forEach(
   function(user) {
    utils.encode(user, function(err, user) {
      users.add(user)
    })
  })
}
```

这么做有效利用了局部性原理（缓存），以提高在进行用户认证时的用户体验，在提速的同时，也大大简化了用户认证的业务逻辑，如下代码：

```
utils.uncode(user, function(err, user) {
  if (!err) {
    // Authentication success !
  }
})
```

另外，这种采用in-memory方式实现用户认证的机制将适时地增加其安全性，它在向服务器中写入用户数据时，将会对每一个对象进行过滤以及增添，例如添加salt值、转换hash值等。

## 5.2 报刊发布平台

报刊发布平台模块用于提供给高级（企业级）用户定期或不定期发布其电子刊物（副刊）的平台，同时也作为系统管理员/操作员发布主刊的主要用户接口。其功能的特殊性，使得这一平台会频繁地修改服务器数据，相对业务逻辑也比较复杂，因此对前端和后端的性能要求都相对较高。

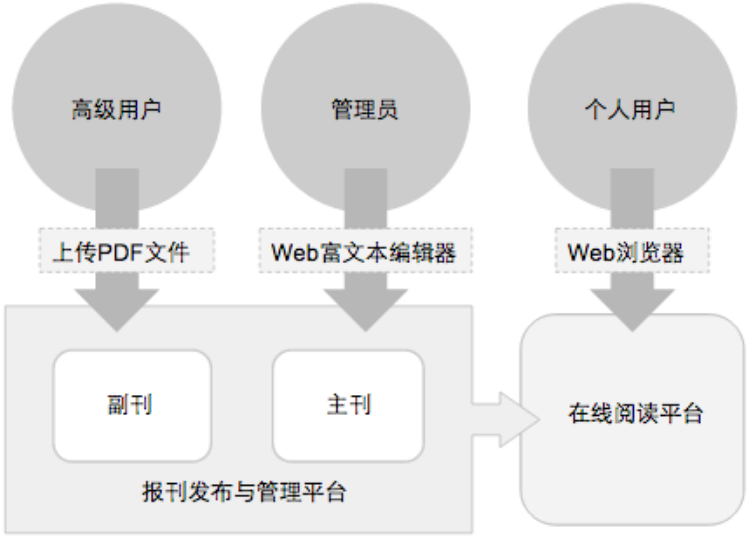


Figure 6: 报刊发布与管理平台

为了简化该平台用户发布的操作流程:

**高级用户** 上传期刊(Issue)的pdf格式的方式来简化流程，通过这种方式来对相应副刊进行管理，可大大简化了高级用户的使用成本;

**管理员** 通过系统基于百度的开源项目Ueditor的Web富文本编辑器来对主刊进行管理;

**个人用户** 我们提供的在线阅读器兼容主流的现代浏览器，相关内容在下一小节中详细说明；

### 5.2.1 PDF上传组件

我们使用了著名开源框架swfupload来为我们提供客户端上传功能，其特点是：多队列上传、类Ajax的用户体验、兼容性良好、灵活自由。它不同于其他基于Flash构建的上传工具，它有着优雅的代码设计，开发者可以利用XHTML、CSS和JavaScript来随心所欲的定制它在浏览器下的外观；它还提供了一组简明的JavaScript事件，借助它们开发者可以方便的在文件上传过程中更新页面内容来营造各种动态效果。

**与seajs兼容** 与swfupload官方示例所不同的是，在我们的系统中，需要使用seajs来对swfupload.js文件进行异步加载，不过这也并不困难，只需像下面这样：

```
seajs.use(['lib/swfupload'], function() {
        var swfu = new SWFUpload({
                'upload_url': 'url_replace'
        })
})
```

**客户端实现** swfupload提供了一组简明的Javascript事件，下面分别看看常用的几个事件：

**swfupload_loaded_handler()** 在swfupload载入后触发，通常用来对UI进行一些初始化的工作

**file_dialog_start_handler()** 在每次打开选择文件对话框时触发

**file_queued_handler(file)** 在成功将一个文件添加到上传等待队列后触发

**file_queue_error_handler(file, err, msg)** 在添加一个文件到队列失败时触发

**file_dialog_complete_handler(fileId, queneId, length)** 确认选择文件对话框后触发，若点击取消并不会触发该事件

**upload_start_handler(file)** 上传开始时触发

**upload_progress_handler(file, curr, total)** 实时监听上传进度，直到上传完成，其中file为文件对象，curr为已上传的大小，total为总共需要上传的大小，在使用进度条时，会使用到这一事件

**upload_success_handler(file, json, res)** 上传成功后触发

首先需要在file_dialog_complete_handler事件中执行如下代码:

```
...
dialogComplete: function(fileId, queuedId, length) {
        this.startUpload()
},
...
```

只有在调用了this.startUpload()这个方法后，才会触发那些与上传相关的事件。

**服务端实现　我**们在NodeJS服务器上开放上传功能的接口相对简单，借助于express的bodyParser插件，我们轻松地可以将上传地文件都先缓存到assets/photos/tmp内，然后再根据请求头或请求参数决定如何处理这些文件，并且之后的处理并不阻塞当前请求的返回，从而提高web用户体验。

## 5.3　在线阅读平台

该平台提供给个人用户，进行报刊浏览、检索等功能，目标是兼容个大主流的浏览器平台(IE8+,chrome,firefox,safri,opera)，由于主刊与副刊格式与显示方式不同，我们将其分开进行说明。

### 5.3.1　主刊

主刊刊主作为系统管理员，是作为所有副刊的索引，也是综合类型刊物最大的报刊，主刊文章是基于HTML编写，因此可以很轻松地展现在阅读器中。

### 5.3.2　副刊

为了简化副刊的发布流程，特意使用了通过上传PDF文件的格式来对其进行最简化，而对于副刊的阅读来说，一个前端性能良好，体验绝佳的PDF阅读器（解析器）是系统所需要的。项目使用了Mozilla的开源项

目pdf.js来提供这一功能模块，而将它无缝地嵌入到系统中，也就成为了一个设计重点，好在pdf.js提供了一系列的Javascript接口来供开发者使用，下面就先介绍一下这个开源项目——pdf.js：

**pdf.js** 一款基于HTML5的高效PDF解析器，并且完全不依赖于浏览器的本地援助。它是在Mozilla实验室带动下的社区驱动的开源产品，旨在创建一个基于web标准化技术平台下的PDF解析器和渲染器。

# 6 总结

本电子报刊发行系统整体上采用了SeaJS+NodeJS+MongoDB系统架构，为了更好地复用其业务逻辑，使得整体系统结构具有强内聚，弱耦合，将整个系统分为了三层：视图表现层、业务逻辑层，数据持久层。各层分别运用了最新潮的开源框架技术：如ExpressJS、moogoose，同时，为了紧贴系统本身业务需求，还开发了2个开源工具：express-model与flashDB。

SeaJS是一个遵循CommonJS规范的JavaScript模块加载框架，可以实现其的模块化开发及加载机制。与jQuery等框架不同，SeaJS不会扩展封装语言特性，而只是实现JavaScript的模块化及按模块加载。SeaJS的主要目的是令JavaScript开发模块化并可以轻松愉悦进行加载，将前端工程师从繁重的JavaScript文件及对象依赖处理中解放出来，可以专注于代码本身的逻辑。SeaJS可以与jQuery这类框架完美集成。使用SeaJS可以提高JavaScript代码的可读性和清晰度，解决目前JavaScript编程中普遍存在的依赖关系混乱和代码纠缠等问题，方便代码的编写和维护。

Express.js是基于NodeJS，高性能、一流的web开发框架，它通过对NodeJS的部分接口进行重新封装，使其更易于适用在web的世界中，并且提供了良好的插件机制，可以灵活地向应用中添加模版引擎，中间件等，因而使其具有一定的生态活力。

express-model是基于express.js的对象建模工具，通过它可以轻松地连接到各种数据库服务器，并对它们进行操作，express-model的使用，解放了原来绝大部分位于控制器内的代码，在models内的对象中，包含有大量的业务逻辑与数据集合，而这些都是架构在书写对象模型时严谨的模式内

的，在应用过程中，我不断优化其内部代码实现以及性能，并不断地简化其API，虽然相较于其他成熟的建模工具，相形见绌，但凭借着以上优点，也一定有其用武之地。

MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。它的特点是高性能、易部署、易使用，存储数据非常方便。

这些开源技术的结合，极大地提高了整个系统的开发效率，同时也提供了系统的可测试性，与易于维护的方便性。作者通过研究各种技术的特点与实现，将其运用到电子报刊发行系统中，取得了一定的成果。不过这些使用的技术都还处于发展阶段，其版本号与相关文档也在不断地修改、升级，技术不断地被完善，很值得我们去进一步研究。最后，对于该电子报刊发行系统的拓展，也值得进一步研究，由于制作比较仓促，部分方面如数据统计平台、在线阅读平台移动化等重要的方面都未能顾及到，不过这些工作将会是什么有意义的。

# 7 致谢

首先，我要感谢我的指导老师——吴云老师，本文的研究工作是在他的悉心指导下完成的，在本科学习期间，吴老师付出了大量的心血，和宝贵的时间，在学业上、工作上给予了我莫大的帮助。他严谨的治学态度，勤奋朴实的工作作风，广博的知识，使我于耳濡目染中获益，这些使我今后人生道路上一笔宝贵的财富。

感谢与我朝夕相处的同学们、好友们，四年的学习生活离不开他们的关心与照顾，愿我们友谊长存。

最后，对在百忙之中审阅论文的各位老师和专家表示衷心地感谢，恳请各位老师多多批评指正，并多多提出宝贵的意见与建议。