计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(2):64−70 [doi: 10.15888/j.cnki.csa.006186]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于有限状态机与 WebSocket 的投票系统①

李　骞, 胡扬帆, 刘　培, 马艳彬

(新华通讯社 通信技术局, 北京 100031)

摘　要: 近年来, 随着企事业单位信息化建设的大力发展, 投票评审成为考评决策的重要手段. 然而传统的投票评审系统普遍存在稳定性和扩展性较差的问题, 并且难以应对投票过程中的各种突发情况. 本文利用有限状态机和 WebSocket 技术, 实现了一个实时高可靠的投票系统. 本系统具有较高的扩展性, 可通过灵活配置, 实现多种形式的投票信息化管理, 并可扩展多种模板以满足不断增长的业务需求. 经过多次实际应用证明, 系统实现了快速稳定的投票评审过程, 极大地提高了评审效率.

关键词: 有限状态机; WebSocket; 实时; 高可靠; Node.js; 投票系统

引用格式: 李骞,胡扬帆,刘培,马艳彬.基于有限状态机与 WebSocket 的投票系统.计算机系统应用,2018,27(2):64–70. http://www.c-s-a.org.cn/1003-3254/6186.html

Voting System Based on Finite State Machine and WebSocket

LI Qian, HU Yang-Fan, LIU Pei, MA Yan-Bin

(Technical and Telecommunications Bureau, Xinhua News Agency, Beijing 100031, China)

Abstract: In recent years, with the development of informationization construction of enterprises and institutions, voting appraisal becomes an important method for appraisal decision. However, the traditional voting system has widespread poor stability and scalability, and it is difficult to cope with various unexpected situations in the voting process. In this study, a real-time and high reliable voting system is realized by using the finite state machine and WebSocket technology. The system has high scalability, and can be flexibly configured to realize a variety of forms of voting information management, and can expand a variety of templates to meet the growing business needs. Several practical uses have proved that the system realizes a rapid and stable voting process, and has greatly improved the appraisal efficiency.

Key words: finite state machine; WebSocket; real-time; high-reliability; Node.js; voting system

近年来, 随着互联网技术的飞速发展, 各企事业单位为了更好的履行职能, 提高办公效率, 提升管理水平, 都在大力发展信息化建设. 投票评审系统作为促进业务发展的一种手段在各单位已成为常态化, 投票表决事项、投票规则、统计方式等也趋于多样化. 传统的投票评审方式准备周期长且复杂, 系统扩展性较差, 难以应对投票过程中产生的各类突发情况, 用户并发操作时, 系统性能下降明显. 如何利用先进的科学技术和手段替代传统低效率的投票评审方式成为了亟待解决的问题.

1　系统设计

1.1　系统总体功能

本系统基于有限状态机原理, 利用 WebSocket 等互联网技术设计并实现了一个实时高可靠的投票评审系统. 本系统具有较高的扩展性, 可通过灵活配置, 满足基于人员、事项等多种对象的投票信息化管理, 涵盖了多种投票形式.

① 收稿时间: 2017-05-02; 修改时间: 2017-05-19; 采用时间: 2017-05-25

1.2 系统功能架构

图 1 所示的功能架构中, 管理员通过管理端进行会议及投票计划的创建, 并对投票流程进行控制. 评委通过评委端进行投票及投票结果的实时查看.

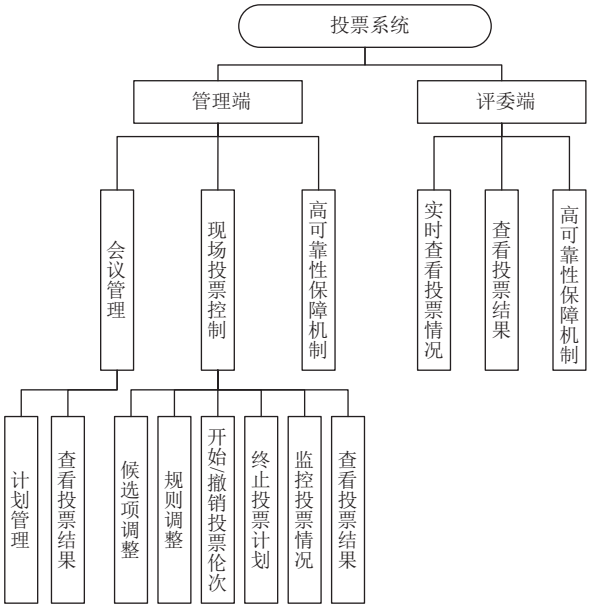

图 1 系统功能架构图

1.3 系统技术架构

1.3.1 整体架构

如图 2 所示, 系统使用了 Nginx+Node.js+MongoDB 的开发框架进行快速迭代开发. 通过 Machina.js 组件实现系统中有关状态机的各种逻辑. 通过 Socket.IO 组件运用 WebSocket 技术实现系统应用前后端实时通信, 并保证了应用系统的高可靠性.

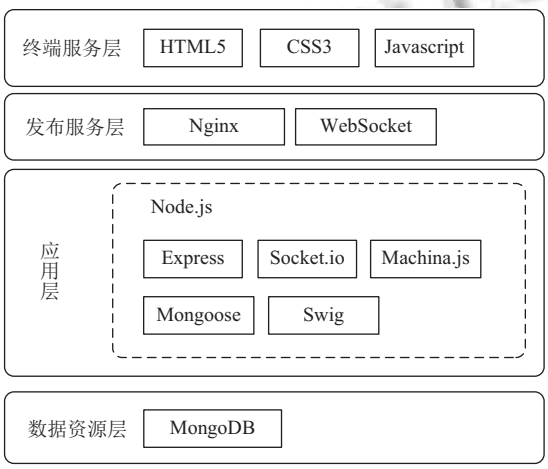

图 2 系统技术架构图

1.3.2 投票处理逻辑的设计

如图 3 所示, 投票模型 (Voting Model, VM) 是投票流程的状态机模型, 主要用于处理计算及流程状态相关逻辑.

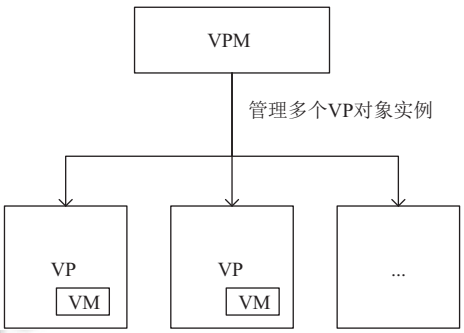

图 3 VPM 和 VP 的关系

投票处理器 (Voting Processor, VP) 主要负责投票流程中的数据预/后处理、存取逻辑并负责具体处理 WebSocket 请求, 一个 VP 对象实例管理并操作对应的一个 VM 对象实例, 一对 VP 与 VM 形成一套投票流程模板, 具体负责一类投票流程.

投票处理器管理器 (Voting Processor Manager, VPM) 主要负责多个 VP 实例的管理, 包括 VP 的加载、同步、切换等等, 并负责处理转发 WebSocket 请求至相应的 VP 实例.

2 关键技术

2.1 有限状态机

有限状态机 (Finite State Machine, FSM) 是计算机领域中一种用来表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型[1].

有限状态机可以用来进行对象行为建模, 其作用主要是描述对象在它的生命周期内所经历的状态序列, 以及如何响应来自外界的各种事件[2].

在本系统中, 使用了 JavaScript 的有限状态机库 Machina.js 来实现状态机相关的逻辑.

2.2 WebSocket

由于 HTTP 协议的请求-响应特性, Web 应用都是以一种单向的方式进行通讯——所有的通信都是由客户端发起和控制的. 于是许多网站为了实现向客户端实时推送数据, 采用了一种轮询的技术, 即每隔一段时间由客户端向服务端请求最新数据[3]. 然而由于服务端需要不断的响应客户端发出 HTTP 请求, 造成了很大

的网络资源浪费. 图 4 是采用轮询技术和 WebSocket 的网络负载对比[4].

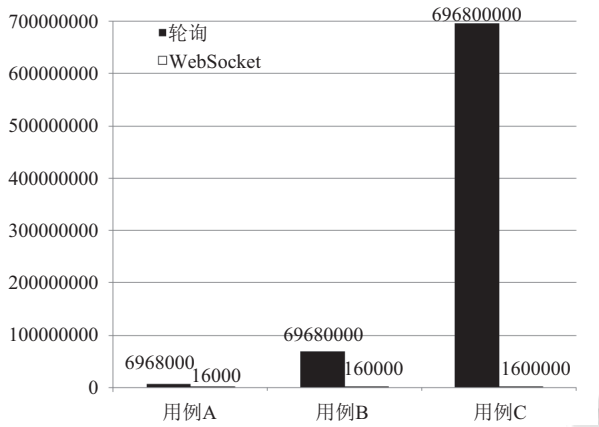

图 4 轮询和 WebSocket 实现方式的网络负载对比图

由此可见, 采用轮询技术的网络负载远远大于 WebSocket 技术的网络负载.

于是, HTML5 标准定义了 WebSocket 协议. 该协议在客户端和服务端之间建立了一个全双工的套接字连接, 客户端和服务端会通过这个持续存在的连接通道自由地传递数据. WebSocket 协议是基于 TCP 的一种通信协议, 所以其工作流程也需要经过发起连接请求、握手、建立连接三个步骤[5]:

1) 客户端发送连接请求;

2) 服务器响应并切换协议;

3) 通信两端建立全双工连接.

最后当通信中的某一方认为可以结束会话时, 便会给对对方发送关闭连接的请求, 这样便结束了此次连接.

WebSocket 协议工作流程如图 5 所示.

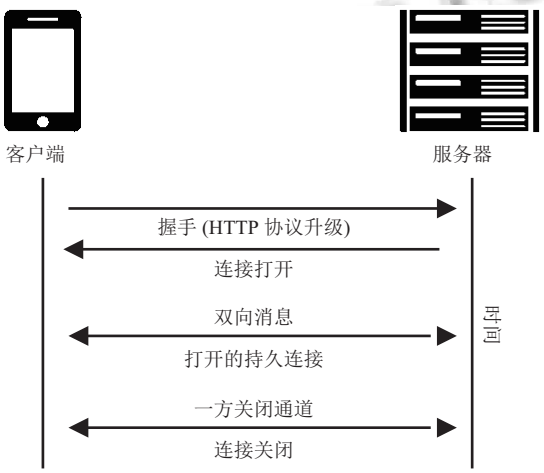

图 5 WebSocket 协议工作流程

作为一种新的 Web 标准, 虽然 WebSocket 协议仍在发展中, 却已经具有一系列优秀的特性. 首先, WebSocket 是一种有状态的协议, 所以通信时可以省略部分状态信息, 提高了数据传输的效率, 具有更强的实时性; 其次, WebSocket 有更好的二进制支持, 相对于 HTTP 可以更好地处理二进制内容; 最后, WebSocket 支持扩展, 用户可以由此实现用户自定义的子协议[6].

本系统中我们采用了开源的跨平台实时通信库 Socket.IO. 它提供了客户端和基于 Node.js 服务器的解决方案.

2.3 Node.js 应用运行环境

Node.js 是一个服务器端的 JavaScript 运行环境, 由于封装了 Google 的 V8 引擎, 使得其执行 JavaScript 代码的速度和性能都有很大提升. Node.js 的事件驱动和非阻塞 I/O 特性, 使得它通常被用于 I/O 密集的实时应用系统[7].

传统的 Web 服务器技术中, 每当新增一个连接请求时便会生成一个新的线程进行处理和响应[8]. 而 Node.js 使用了非阻塞的单线程方式, 利用 JavaScript 的事件机制, 在收到一个查询请求后便将 CPU 的控制权交出, 直到当数据处理完成后触发一个事件, 再取得控制权继续执行.

如图 6 所示, Node.js 的结构与 Chrome 非常相似, 他们都是基于事件驱动的异步架构. 浏览器通过事件来处理界面上的交互, Node.js 通过事件来处理 I/O. 在服务器端, Node.js 虽然不再处理 CSS、与 DOM 和 BOM 打交道, 但是却具有 JavaScript 的一切语言特性——基于原型链的对象继承、事件处理机制、回调函数等. 此外, Node.js 还可以方便地访问本地文件、数据库以及网络等资源[9].

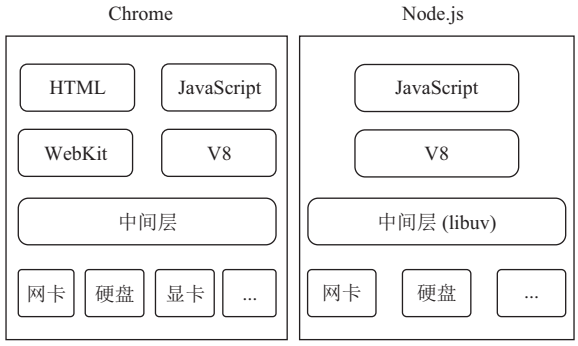

图 6 显卡 Chrome 浏览器和 Node.js 的组件构成

在本系统中, 我们采用了 Express 作为 Node.js Web 框架. Express 是一种简洁易用、功能强大的 Web 应用程序框架, 它提供了一系列丰富的 API 及中间件用于进行快速开发. 我们还使用了基于分布式文档存储的非关系型的 MongoDB 作为底层数据库.

3 重点问题解决方案

下面我们将首先聚焦于有限状态机在 VM(投票模型) 实现中所起到的作用, 然后我们还将具体描述投票流程以及 VM、VP、VPM 的联动关系, 并阐述系统可靠性的实现方法.

3.1 可扩展的投票流程模型的实现

本系统支持多种投票形式, 如差额投票、多轮次投票、候选逐一投票等等, 并支持将来的扩展. 我们整理了十几种不同的投票场景、规则, 将它们进行归纳整理, 并最终使用有限状态机 (FSM) 来对投票流程进行建模.

我们举一个比较简单的投票场景为例, 其规则如下:

1) X 个候选人中选出 Y(目标额度) 个人通过 (X>Y);

2) 根据候选人所得赞成票多少进行统计, 须得到与会评委相应比例 (如 2/3) 的赞成票方有资格通过;

3) 满足规则 2 的候选人超过规定额度时, 按得票多少取满目标额度;

4) 在规则 3 情景下, 如出现目标额度位置前后得票并列的情况, 则对并列的候选人重新投票, 直到选满目标额度为止. 假设我们按票数从高到低对候选人进行排列, 如出现票数$_Y$=票数$_{Y+1}$=票数$_{Y+2}$ 或票数$_{Y-1}$=票数$_Y$=票数$_{Y+1}$ 的情况就需要进行并列逻辑处理, 如出现票数$_{Y-1}$=票数$_Y$ 且票数$_Y$>票数$_{Y+1}$ 的情况, 则不需要进行并列逻辑处理.

针对此投票场景, 我们使用如图 7 所示的状态机来对其投票流程进行建模.

该状态机共有 5 个状态, 各状态及转移情况描述如下:

(1) 初始状态 (uninitialized)

接收 init 事件及相关参数后, 转移至 knockout 状态, 并执行 init 初始化方法对首轮投票参数进行初始化.

(2) 投票状态 (knockout)

接收 init 事件及参数来初始化投票轮次.

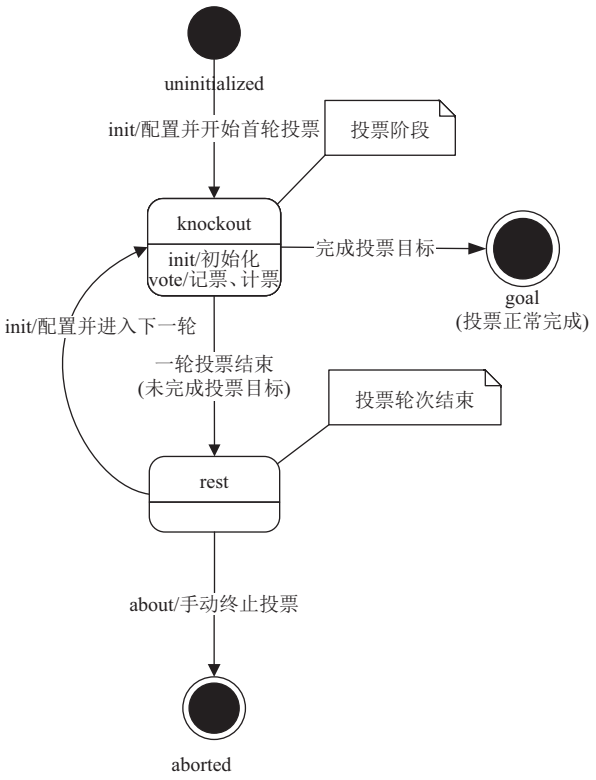

图 7 投票流程模型状态机图

每一张投票触发一次 vote 事件, 记录选票, 直到所有评委均投票完成, 触发计票逻辑, 计算本轮投票结果. 若未满足投票目标则状态机转移至 rest 状态, 计算好下一轮投票可能的参数 (如剩余额度、评委投票权数 (评委可投的票数)、候选项等), 准备进行下一轮投票; 若满足投票目标则状态机转移至 goal 状态, 完成投票计划.

(3) 投票轮次结束状态 (rest)

接收 init 事件及相关参数后, 转移至 knockout 状态, 并执行 init 初始化方法对下一轮投票参数进行初始化.

接收 abort 事件, 转移至 aborted 状态, 终止本次投票计划.

(4) 最终态 (goal)

投票正常完成.

(5) 最终态 (aborted)

投票被手动终止.

根据此状态机设计, 投票规则 1) 至 4) 中所涉及的计算逻辑可在 knockout 状态的 vote 事件的计票逻辑中实现, 从而可完成单轮或多轮的投票场景, 并可人工进行终止.

针对更复杂的投票场景, 如含有多个规则不同的环节的投票计划 (如含有预选、复议、定评等环节), 我们可以建立多个针对不同环节的状态, 如 qualification、knockout、finalreview, 完成各自的初始化和记/计票逻辑, 并复用 uninitialized、rest、goal、aborted 状态的相关逻辑, 便可完成该投票场景的计算模型.

状态机的实现上我们使用了 machina.js 框架, 一个成熟的 javascript 状态机框架. 在该框架中我们完成了状态、状态转移、状态事件的定义, 由于 machina.js 定义的 FSM 是基于 EventEmitter 的, 所以我们方便地进行了输出事件的定义. 关键代码如下:

```
module.exports = machina.Fsm.extend({
initialize: function (options) {}, //初始化 fsm 的方法
initialState: "uninitialized", //初始状态
states: { //各状态定义及状态间转移定义
  uninitialized: {},
  knockout: {
    init: function (param) {
      //初始化本轮投票
    },
    vote: function (ballot) {
      //记录投票
      //计票
      //根据计票结果进入 goal 或 rest 状态
    }
  },
  rest: {},
  aborted: {},
  goal: {
    goal: function (result) {
      this.emit("goal", result); //发出 goal 事件 (由 VP 捕获处理), 触发投票结束后的逻辑
    }
  }
},
//以下为状态机接收的事件
init: function (param) {
  this.handle("init", param);
},
vote: function (ballot) {
  this.handle("vote", ballot);
}
});
```

3.2 投票流程及系统高可靠性的实现

3.2.1 数据 IO(事务性) 与计算分离

系统采取了以数据库为中心的计算与 IO 分离的设计策略, 对于每一步经过确认的数据先存入数据库, 存储成功后再进行下一步处理, 每一次的投票状态变化都先存储或更新数据库再进行下一步数据操作. 以步步为营的方式记录最新的投票状态, 这样保障了数据的一致性, 同时为状态恢复机制提供了依据.

如图 8, 整个投票过程的前后端数据交互都由 WebSocket 协议实时传递, 评委端的所有选票数据由 WebSocket 发送给后端的 VPM, VPM 对每一份收到的数据进行验证、收集、存储和计算, 得出投票结果后通过 WebSocket 对评委端、管理端和投票监控大屏等展示终端进行实时广播, 各端收到投票结果数据立即进行展示.

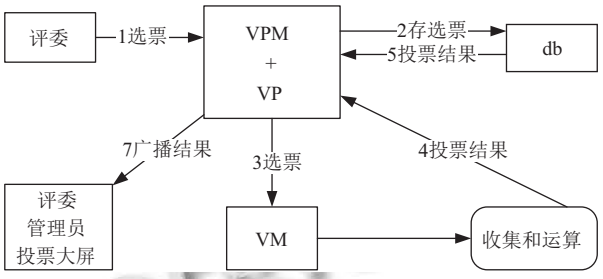

图 8 系统数据流图 (数字标号表示顺序)

具体为: 首先选票数据由评委端发送给 VPM, VPM 交给对应投票类型的 VP, VP 对每一份选票数据进行验证后连同评委信息一起存入数据库, 仅当存库成功后才进行步骤 3, 将选票传入 VM 状态机用于收集和计算, 当所有与会评委投票完成后, VM 自动计算投票结果, 并将结果返回给相应 VP, VP 进行存库, 最后广播结果数据给前端.

投票过程中数据的验证、传递和存储由 VP 负责, 它是基于持久化存储的, 非易失的, 可靠性强; 计算由 VM 负责, VM 的数据运行在内存中, 是易失的, 可靠性弱. 当系统经历意外宕机等情况时, VP 可以从数据库中获取宕机前投票计划的最后状态的数据, 然后用计划状态数据初始化 VM 状态机和前端页面, 即可恢复投票状态.

3.2.2 投票状态恢复与监控

投票过程中服务端和客户端都可能出现意外情况而断电或断网, 因此服务端和客户端都需要完备的状态恢复机制.

对于服务端, 当系统经历意外中断之后, 重启程序时 VPM 会先从数据库中读取尚未结束且处于活动状态的计划, 然后 VP 依据具体的计划状态和票箱 (已投) 数据确定恢复计划的具体数据, 包括当前的轮次信息, 评委提交信息, 组装好具体数据后 VP 将其广播给各前端 (评委端、管理端、投票监控大屏), 主动刷新了前端的状态. 服务端的相关程序, 如应用、数据库、nginx 等程序都部署了开机自启动脚本, 因此整个投票系统可以在无人工干预的状态下自动恢复到投票中断前的状态.

由于前端设备数量较多, 可控性较差, 在投票的任何阶段都可能出现终端连接断开和重新连接的情况, 因此投票过程中一旦有前端发起同步请求, 服务端先检视当前投票状态, 根据当前状态准备前端的恢复数据, 返回给前端进行页面更新.

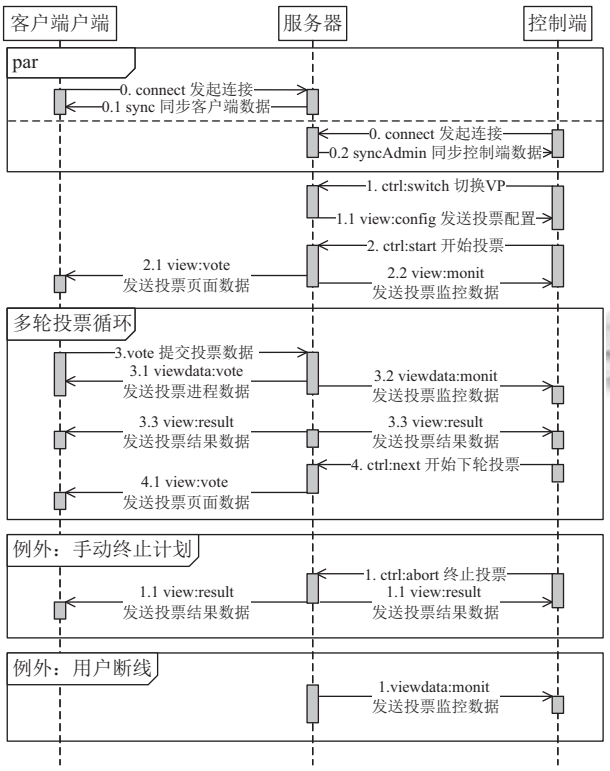

图 9 系统时序图

图 9 所示的时序图主要描述了前端和服务端之间的数据交互. 图中所有消息都是异步发送, 由 socket.emit 事件触发, 然后监听返回事件, 依据返回数据进行页面更新. 以 ctrl 开头的指令表示控制指令, 控制投票状态, 例如投票开始; 以 view 开头的指令表示更新页面. 根据时序图步骤 0, 每个新建立的 WebSocket 连接都会收到服务端发出的 sync 同步指令, 该指令数据中定义了投票状态和渲染页面所需的信息, 前端根据 sync 指令的数据来更新页面内容. 此步骤保证了前端页面随时都可以与服务端通过 WebSocket 接口恢复投票状态.

当投票正在进行时, 前端步骤 3 发送的投票数据在服务端收到后会广播 viewdata:vote 投票进程数据, 各端页面收到后会即时更新当轮投票的进度. 如果当轮投票结束且满足整个投票结束的条件后, 服务端根据步骤 3.3 也会将结果数据广播给各前端页面, 当管理员手动终止投票时也是同样的流程.

这样, 系统以数据库为中心, 采用 WebSocket 实时双工数据传输协议, 以投票状态为依据, 从应用设计和部署层面实现了投票的高可靠性.

4 系统实际应用

本系统目前已经正式投入使用, 完成了本单位多次社领导级别的重要评审会议的技术保障工作, 包括社编务会通报表扬事项表决、采编业务考核非固定加减分事项表决、全社创新工作奖励集中评审表决、年度社级优秀新闻作品评审表决、驻外分社职称评审表决等等.

投票模板的应用使得切换投票方式变得简单易行, 从前端展示到后台统计均可一次性完成, 大大节省了投票准备工作时间. 传统投票系统在轮次间需要人工进行下轮参数的计算和设置, 平均人工准备时间大约需要 2 到 5 分钟. 使用本系统后, 评审过程中系统自动计算出各种参数 (如候选项), 流程衔接顺畅, 没有了人工准备时间, 极大地提高了评审效率. 系统智能程度较高, 学习成本低, 简单易懂的操作界面提升了用户体验. 高可靠性使得系统能灵活应对各种突发事件, 确保数据正确完整.

“界面新颖、体验良好、功能灵活、流程顺畅”的新一代投票评审系统, 投入使用后效果显著, 得到了本单位社领导、各部门领导和组织单位的一致好评.

5 结语

本系统具有较高的可靠性, 满足基于人员、事项

等多种对象的投票信息化管理, 并可扩展多种模板以满足不断增长的业务需求. 系统的使用极大地提高了评审管理效率, 增加了评审工作的标准化与科学化, 互联网思维的用户体验也增加了用户的黏合度. 未来, 计划在现有系统基础上增加基于系统使用数据的大数据分析, 辅助领导决策, 促进业务发展.

参考文献

1 钱忠胜, 邹俊. 正规文法与有限自动机的等价构造. 计算机应用与软件, 2008, 25(6): 110–112.

2 为 Linux 应用构造有限状态机. https://www.ibm.com/developerworks/cn/linux/l-fsmachine/. [2014-10-01].

3 易仁伟. 基于 WebSocket 的实时 Web 应用的研究[硕士学位论文]. 武汉: 武汉理工大学, 2013.

4 WebSockets 简介:将套接字引入网络. https://www.html5rocks.com/zh/tutorials/websockets/basics/. [2010-10-20].

5 李锡辉, 杨丽. 基于 WebSocket 的服务器推送技术研究. 网络安全技术与应用, 2014, (6): 45–46.

6 齐华, 李佳, 刘军. 基于 Websocket 的消息实时推送设计与实现. 微处理机, 2016, 37(3): 36–39, 43.

7 万里晴, 杨浩. 探究基于 V8 引擎的 Node.js 在各应用领域的发展. 通讯世界, 2015, (13): 97. [doi: 10.3969/j.issn.1006-4222.2015.13.066]

8 骆文亮. Node.js 服务器技术初探. 无线互联科技, 2014, (3): 227.

9 朴灵. 深入浅出 Node.js. 北京: 人民邮电出版社, 2013.