|  |  |
| --- | --- |
| 文档号 |  |
| 密级 |  |

**电脑彩票系统（信息显示）**

**设计说明书**

|  |  |  |
| --- | --- | --- |
| **动作** | **签名** | **日期** |
| 创建人 |  |  |
| 审核人 |  |  |

文档变更历史

|  |  |  |  |
| --- | --- | --- | --- |
| **修订人** | **版本号** | **日期** | **修改内容与原因** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**目 录**

[1 引言 1](#_Toc476585721)

[1.1 版权申明 1](#_Toc476585722)

[1.2 编写目的 1](#_Toc476585723)

[1.3 预期读者 1](#_Toc476585724)

[1.4 参考资料 1](#_Toc476585725)

[2 方案简述 2](#_Toc476585726)

[2.1 设计原则 2](#_Toc476585727)

[2.2 需求概述 2](#_Toc476585728)

[2.2.1 功能性需求概述 2](#_Toc476585729)

[2.2.2 非功能性功能概述 2](#_Toc476585730)

[2.3 系统整体架构 2](#_Toc476585731)

[3 服务端架构设计 3](#_Toc476585732)

[3.1 逻辑架构 3](#_Toc476585733)

[3.2 关键技术 3](#_Toc476585734)

[3.3 接口设计 3](#_Toc476585735)

[4 客户端架构设计 3](#_Toc476585736)

[4.1 逻辑架构 3](#_Toc476585737)

[4.2 关键技术 3](#_Toc476585738)

[4.3 接口设计 3](#_Toc476585739)

# 引言

## 版权申明

本文档版权归北京戈德利邦科技有限公司所有，未经北京戈德利邦科技有限公司书面许可，任何人、任何组织不得出于任何目的，对本文档的任何部分，以任何形式进行翻印、翻译、节选、改编、仿制，北京戈德利邦科技有限公司保留对本文档的最终解释权并可随时进行更改，此类更改恕不另行通知。

## 编写目的

本文从电脑彩票业务特点、业务架构和现有IT设施和开发能力现实情况为出发点，制定电脑彩票系统应用架构设计原则，描述系统应用架构范围与内容，作为系统设计人员、程序开发人员、测试人员的参考资料。

## 预期读者

|  |  |
| --- | --- |
| **角色** | **职责及权限** |
| 应用架构设计人员 | 负责维护本文档 |
| 基础设施架构设计人员 | 结合应用架构设计基础设施架构 |
| 安全架构设计人员 | 结合应用架构设计安全架构 |
| 运维架构设计人员 | 结合应用架构设计运维架构 |
| 概要设计人员，详细设计人员 | 根据本文档做概要设计和详细设计 |

## 参考资料

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **资料名称** | **版本** | **作者** | **日期** | **出版单位/资料来源** | **备注** |
| 《电脑彩票信息显示系统需求说明书》 | 1.0 | 王克学 |  |  |  |

# 方案简述

## 设计原则

* 架构简洁，易于维护，易于继承已有的开奖动画开发成果。
* 高内聚，低耦合，分层设计；
* 尽量保持异步、无状态设计；
* 提供灵活的水平扩展能力；
* 系统性价比高；

## 需求概述

### 功能性需求概述

实现电脑彩票快开游戏的站点信息显示功能，包括但不限于开奖过程动画、开奖结果信息、报表、广告、滚动信息调等信息的配置与显示。

### 非功能性功能概述

电脑彩票信息显示系统应包括但不限于以下的非功能性需求：

* 性能要求

电脑彩票系统需支持一个省福彩热线游戏和快开游戏的交易处理，所以信息显示系统在常规硬件前提下（一台16核主流PC server），需实现对5000台信息显示客户端的支持。

* 系统可扩展性

系统的扩展能力要强，在出现业务量变化或增长时，能够快速水平扩充系统功能和性能；

## 系统整体架构

系统采用C/S模式进行设计，服务端主要采用PHPSocket技术，客户端主要采用Unity技术。系统整体结构如下图：

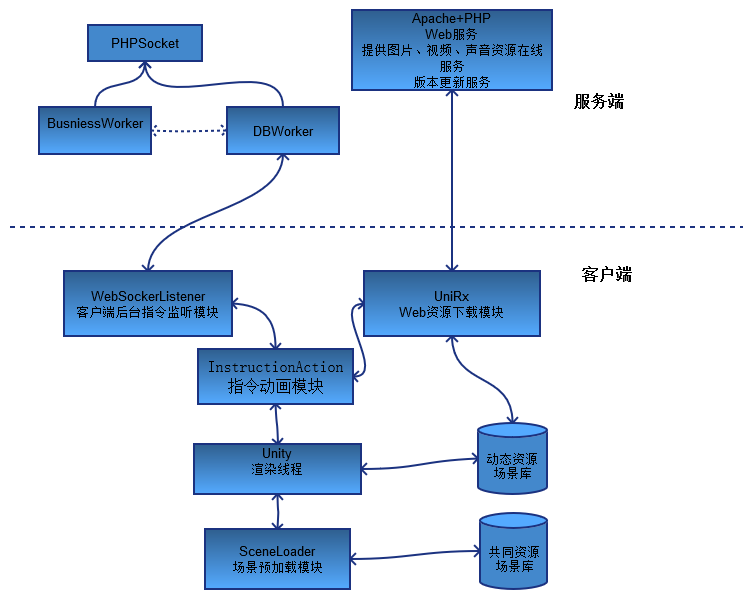


图 1 系统整体框架

# 服务端架构设计

## 逻辑架构

### 通信协议

通信协议格式采用**protocol buffer**格式，protocol buffer是Google开发的数据交换格式，它独立于语言、独立于平台，非常适合消息报文的格式载体。protocol buffer使用proto格式描述报文的格式，如下例所示：

**定义第一个Protocol Buffer消息。**      创建扩展名为.proto的文件，如：MyMessage.proto，并将以下内容存入该文件中。  
      message LogonReqMessage {  
          required int64 acctID = 1;  
          required string passwd = 2;  
      }  
      这里将给出以上消息定义的关键性说明。  
      1. message是消息定义的关键字，等同于C++中的struct/class，或是Java、PHP中的class。  
      2. LogonReqMessage为消息的名字，等同于结构体名或类名。  
      3. required前缀表示该字段为必要字段，既在序列化和反序列化之前该字段必须已经被赋值。与此同时，在Protocol Buffer中还存在另外两个类似的关键字，optional和repeated，带有这两种限定符的消息字段则没有required字段这样的限制。相比于optional，repeated主要用于表示数组字段。具体的使用方式在后面的用例中均会一一列出。  
      4. int64和string分别表示长整型和字符串型的消息字段，在Protocol Buffer中存在一张类型对照表，既Protocol Buffer中的数据类型与其他编程语言(C++/Java)中所用类型的对照。该对照表中还将给出在不同的数据场景下，哪种类型更为高效。该对照表将在后面给出。  
      5. acctID和passwd分别表示消息字段名，等同于Java中的域变量名，或是C++中的成员变量名。  
      6. 标签数字**1**和**2**则表示不同的字段在序列化后的二进制数据中的布局位置。在该例中，passwd字段编码后的数据一定位于acctID之后。需要注意的是该值在同一message中不能重复。另外，对于Protocol Buffer而言，标签值为1到15的字段在编码时可以得到优化，既标签值和类型信息仅占有一个byte，标签范围是16到2047的将占有两个bytes，而Protocol Buffer可以支持的字段数量则为2的29次方减一。有鉴于此，我们在设计消息结构时，可以尽可能考虑让repeated类型的字段标签位于1到15之间，这样便可以有效的节省编码后的字节数量。

使用protoc-gen-php 可以将.proto描述文件生成php对应的类型，从而可以解析和访问protocol buffer的数据流信息。

使用protobuf-net 可以将.proto描述文件生成C#对应的类型，从而可以在.NET环境解析和访问protocol buffer的数据流信息。

|  |  |  |
| --- | --- | --- |
| **属性** | **类型** | **描述** |
| 指令名称 | 字符串 |  |
| 要求版本 | 字符串 | 信息显示程序每次执行指令时，都检查自己的版本是否符合要求版本，如果不符合，执行版本更新动作。 |
| 发送时间 | 字符串 | yyyy-mm-dd hh24:mi:ss |
| 游戏ID | 字符串 |  |
| 游戏名称 | 字符串 |  |
| 奖期编号 | 字符串 |  |
| 参数 | 字符串 | 参见接口说明文档 |
| 开始时间 | 字符串 | yyyy-mm-dd hh24:mi:ss |
| 显示时长 | 字符串 | 秒 |
| 结束时间 | 字符串 | yyyy-mm-dd hh24:mi:ss |
| 指令描述 | 字符串 |  |

指令命令文件InstructionMessage.proto 内容如下：

message InstructionMessage {

string messageName = 1 ;

string version = 2 ;

string sendDate = 3 ;

int32 sendIndex = 4 ;

int32 gameId = 5 ;

string gameName = 6 ;

string awardNumber = 7 ;

string paraments = 8 ;

string startDate = 9 ;

string showTime= 10 ;

string endDate = 11 ;

optional string descript = 12 ;

}

通过protoforPHP, protoforNet可以生成对应语言的协议编码解析的类。如何生成和解析，将会在“服务端开发文档.docx”提供。

### 框架设计

服务端框架主要包含两个模块：网络通信模块和后台数据库模块。

网络通信模块主要负责以下几个方面：

1. 负责客户端连接管理，管理所有连接上客户端基本信息；
2. 负责客户端是否在线心跳测试，释放断线客户端的连接资源信息；
3. 负责与后台数据进程通信，缓存后台数据库更新指令信息；
4. 获得更新指令信息后，负责发送指令信息到客户端；

后台数据库模块主要负责以下几个方面：

1. 负责数据库连接管理；
2. 负责定时读取数据库信息，并判断是否有信息更新；
3. 负责将更新信息通过消息队列(Channel分布式通讯组件），发送到网络通信模块；

实现方案的特点：

1. 数据库端的负载比较低，单进程更新和缓存指令信息；
2. 服务端的负载比较小，只有更新指令信息，才会有网络通信；
3. 数据库进程与网络服务进程独立，有利于提高系统稳定性；
4. 单机可以支持上万连接，减少服务器设备和网络投资;
5. 有利于建立模拟仿真环境

服务端主要由网络通信模块和后台数据库模块组成，后台数据库查询进程与网络通信模块之间，采用消息队列的方式进行进程间通信。其中，后台数据库查询进程执行轮询数据库的操作，在整个服务端体系中属于消息生产者角色，网络通信进程在整个服务端体系中属于消息消费者角色。网络通信进程不断接收后台数据库进程的指令消息，并将指令消息转发给所有连接开奖客户端。基础逻辑架构如下图所示：

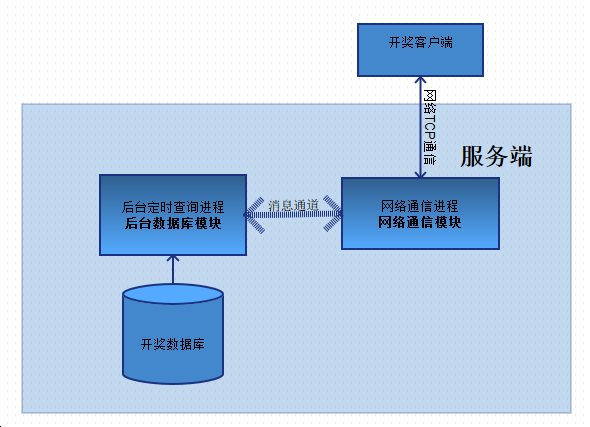


图 2 服务端逻辑架构

## 关键技术

### 多进程守护技术

服务端的服务需要保证稳定和可靠，需要保证后台服务进程不能被中断，或者进程崩溃后，然后自动重启新的进程来提供服务，目前后台服务进程充分利用Linux服务端守护进程的方式实现服务高可靠性。

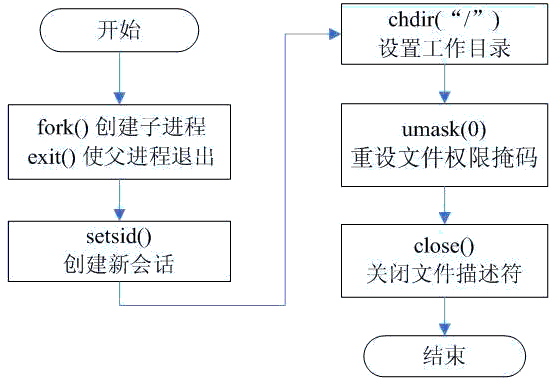


图 3 后台守护进程

### 高并发技术

采用高并发技术，可以尽可能利用单机的硬件能力，提高更多客户端TCP/IP连接数量。从而减少硬件投入和系统开发设计难度。

采用libevent的高性能网络库核心，支持单机高并发连接，目前支持单机5000左右的并发连接。

## 接口设计

服务端主要包括以下几个接口和类模块：

**Worker** 类，提供后台进程模块，为后台定时器进程和网络监听进程，提供后台循环和事件监听等。

**BusinessWorker** 类，提供开奖业务的网络通信模块，基类为Worker，该模块通过Channel组件与后台数据库查询进程通讯。

**Connection** 类，BusinessWork类通过该类管理所有客户端连接。

**DBWorker**类，提供从数据库获得信息定时更新开奖信息，通过Channel组件与BusinessWorkder通讯。

**Channel** 组件，分布式通讯组件，用于完成进程间或者服务器间通讯。其特点有: 基于订阅发布模型和非阻塞式IO。

**InstructionMessage**类，通过protoForPHP生成的protocol buffer协议解析类，用于在DBWorker和BusinessWorkder，进行消息通信。

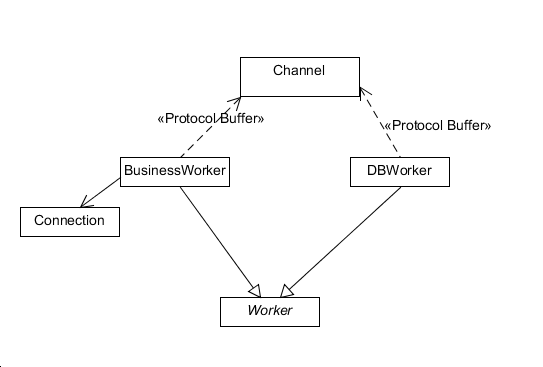


图 4 服务端类关系

# 客户端架构设计

## 逻辑架构

客户端主要由资源管理和加载、后台指令通信模块、指令动画模块三个主要模块组成。 资源管理和加载模块，主要是合理组织客户端渲染显示需要的各种图片、脚本、模型动画等资源，并采用预加载和后台加载的方法，提高客户端的用户体验效果。

后台指令通信模块，后台指令通信模块主要采用事件机制触发机制，接收到指令信息后就执行相关的指令动画。

指令动画模块，指令动画模块采用可扩展的方式，可以基于指令动画模块的扩展方法，增加或者修改指令动画内容。接收到一个指令动画就执行一个指令动画内容。在没有更新指令前，窗口动画停留在上个指令动画结束画面。

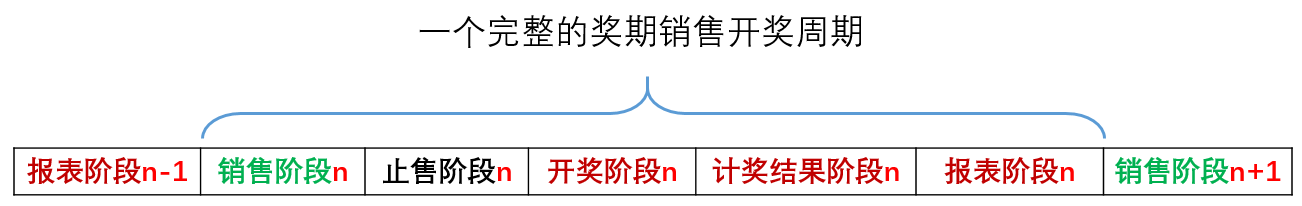


图 5 指令完整周期

## 关键技术

### 渲染数据预加载技术

将渲染数据进行预加载可以避免程序运行过程中的卡顿和黑屏问题。Unity默认在切换场景的过程中，如果需要加载大场景数据就会需要等待较长时间，给用户不好体验。根据目前的项目需求，本系统是重复销售、止售、开奖、报表的循环过程，因此将共用的并且循环使用的资源进行预加载，可以有效避免运行过程中卡顿、黑屏等问题。

### 渲染数据后台加载技术

对于非预加载的渲染数据，为了避免影响主线程的运行，需要实现渲染数据的后台加载技术。将渲染数据通过后台资源加载线程，将资源从磁盘加载到内存，然后再加载到显存中。这种技术通常也称为资源异步加载技术。

### 动态资源加载技术

动态更新数据比如从网络下载的数据，需要加载到场景中进行显示，那么就需要使用使用动态资源加载技术。比如销售阶段的宣传广告资料，这类资料经常需要更新，从服务器下载完宣传广告资料，就需要使用动态资源加载技术，同时结合渲染数据后台加载技术，可以实现平滑的数据加载效果。

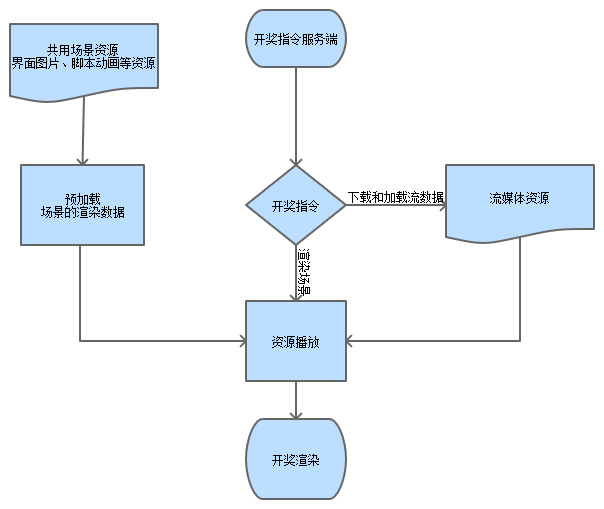
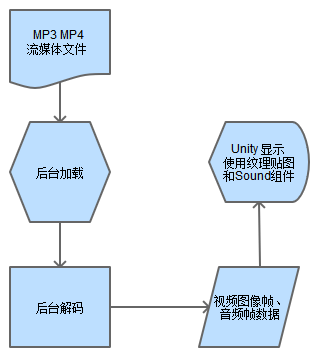


图 6 后台资源加载流程

### 流媒体渲染技术

采用流媒体渲染技术支持视频、音频渲染和播放，从而实现客户端更好的展示效果。流媒体资源通常是指视频数据流和音频数据流。大部分流媒体的数据都比较大，如果全部加载到内存中，将会消耗比较大的内存，资源加载等待的时间也会比较长，用户体验和程序性能都不好。但是流媒体资源的有着显著的特征，流媒体数据在时间上连续的，其在单位时间内传输的数据，可以进行解码播放。

该技术关键在于结合后台渐进加载、音频和视频解码后，将解码后每帧的图像和声音数据，加载到纹理贴图和声音缓冲SoundBuffer中，并不断更新贴图和SoundBuffer，从而实现流媒体播放。



表格 1 流媒体播放

### 基于WebSocket通信技术

基于WebSocket通信技术利用HTTP和Socket连接的特点，利用HTTP发起构建连接通信的请求，请求成功完成后，客户端和服务端建立起Socket长连接。基于WebSocket通信技术可以避免HTTP轮询技术的一些明显缺点，也可以避免自己基于Socket自己开发通信接口和通信协议。

### 版本更新技术

版本更新，主要分为程序版本更新和资源版本更新。通用版本更新方法，将每个文件包都打包版本号，将两者都看成一种普通资源来看待。

因此服务端需要构建一个简单的版本管理和发布系统，可以发布最新的程序版本，提供版本内每个文件的版本信息和下载路径。

客户端需要提供单独的启动程序，每次启动时检查版本是否要更新，如果需要更新那么首先更新程序版本和资源版本，然后再启动主程序。类似于游戏中首先启动Launch.exe检查版本更新，然后再由Launch.exe引导启动Game.exe主程序。

## 接口设计

根据客户端三个主要模块：资源管理和加载、后台指令通信模块、指令动画模块。系统的接口实现主要围绕这三个模块实现展开。

**SceneLoader** : 场景资源加载基类，负责预加载一类场景资源。通常在进程启动时，加载基础共用资源。

**SceneAsyncLoader** ：场景资源异步加载类，目前我们一般不会采用，这里预留给以后如果有更大的场景需要加载时使用。不如以后有多个开奖动画，需要根据不同游戏类型切换不同的开奖动画，这里就需要此技术。

**ResourceAsyncLoader** : 资源异步加载类，主要是完成资源在后台异步加载到场景中。比如销售阶段的多媒体动画信息就需要采用此类来实现资源的异步加载。

**MovieTexture** : 封装Texture和Sound实现视频资源播放。

**WebSocketLisenter** : 指令消息后台监听类，用来接收后台指令消息。并调用消息解码类进行指令信息解码。

**InstructionMessage** : 指令消息解码类，主要是解码指令消息。

**InstructionAction** : 指令动画模块基类，用来提供指令动画模块的扩展。