# 服务端需求

服务端主要实现以下几个方面的功能模块：

1. 后台数据库查询模块；
2. 网络通信模块，主动发送彩票销售、开奖等动态消息；
3. 监测客户端是否在线；
4. 权限验证模块（需求文档没有提出，但是最后应该实现，以保证信息安全）

# 服务端设计

## 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

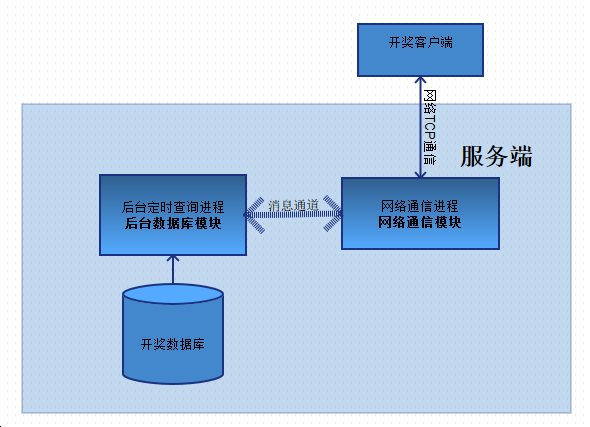


图 1 服务端框架设计

# 服务端接口设计

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。

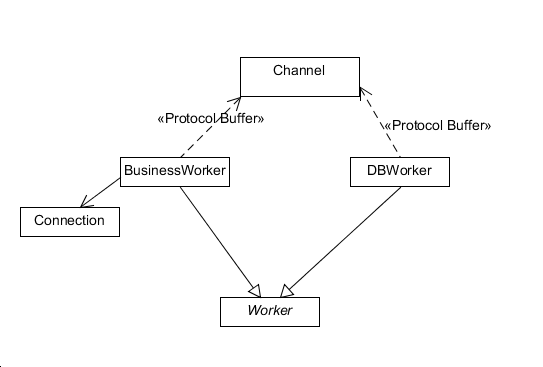


图 2 服务端类关系

# 开发环境

操作系统：

运行系统： Windows 7.0及以上 ; CentOS 6.0及以上

开发系统：Windows7.0

测试系统：CentOS 6.0

开发环境：

PHP运行时: 要求PHP版本>=5.3.3

PHP Socket服务器框架： workerman

PHP PostgreSQL 数据库连接插件：libpq

# 参考资料

Protocol buffer的资料参考：

<http://www.cnblogs.com/stephen-liu74/archive/2013/01/02/2841485.html>

<https://github.com/bramp/protoc-gen-php>

<https://github.com/drslump/Protobuf-PHP>

PHP下载页面：<http://windows.php.net/download>

PHP Socket服务器框架：<http://www.workerman.net/>

PHP数据库连接插件：<http://php.net/manual/en/pgsql.setup.php>