SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：IOS平台网络游戏服务端的设计与实现

学生姓名：李自勉

学生学号：5090379027

专　　业：软件工程

指导教师：肖凯

学院(系)：软件学院

上海交通大学

本科生毕业设计（论文）任务书

课题名称：IOS 平台网络游戏服务端的设计与实现

执行时间：2012 年 12 月 至 2013 年 06 月

教师姓名：肖凯 职称：讲师

学生姓名：李自勉 学号：5090379027

专业名称：软件工程

学院(系)：软件学院

毕业设计（论文）基本内容和要求：

本课题主要内容是学习与使用 Go 语言，研究服务端架构的设计与编写，实践网络消息的收发与解析。研究过程中具体会涉及到的内容主要有以下几个方面：

1. 游戏服务器架构设计。所谓服务器结构，也就是如何将服务器各部分合理地安排，以实现最初的功能需求。所以，结构本无所谓正确与错误；但是优秀的结构更有助于系统的搭建，对系统的可扩展性及可维护性也有更大的帮助。

2. 游戏通信协议设计。游戏通信协议包含两种不同的部分：客户端和服务器（C-S）之间的交互协议，游戏内部服务器（S-S）之间的交互协议。前者为了降低延迟，应该尽可能减少报文长度。同时，为了防止外挂，必须作加密处理。相反，后者在服务器之间，通信协议就可以比较灵活。

毕业设计（论文）进度安排：

序号	毕业设计（论文）各阶段内容	时间安排	备 注
1	背景研究、技术学习、资料积累	2012.12-2013.02	
2	根据游戏的基本需求完成游戏服务端的原型设计	2013.02-2013.03	
3	根据具体需求添加与修改服务端	2013.03-2013.04	
4	服务端基本编写完成后，编写毕业设计论文	2013.04-2013.05	
5	准备毕业设计答辩	2013.06	

课题信息：

课题性质：设计■　　论文□

课题来源：国家级□　省部级□　校级□　横向□　预研□

项目编号 _____

其他_____　导师_____

指导教师签名：_____

年　　月　　日

学院（系）意见：

院长（系主任）签名：_____

年　　月　　日

学生签名：_____

年　　月　　日

上海交通大学
毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：

日期：　　年　　月　　日

上海交通大学
毕业设计（论文）版权使用授权书

本毕业设计（论文）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

保密□，在___年解密后适用本授权书。

本论文属于

不保密□。

（请在以上方框内打“√”）

作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

IOS 平台网络游戏服务端的设计与实现

摘要

移动平台开发是当今软件开发最热门的方向之一，苹果公司提供的 IOS 平台是当今最流行的移动平台之一。根据官方数据统计，IOS 平台的应用程序数量早已突破 50 万，而在这些应用当中，比重居于首位的就是游戏应用。不难看出，随着智能手机和平板电脑的普及，在人们手中它们的用途除了最基本的通讯，娱乐休闲也成了更加重要而不容忽视的一部分。基于市场的需求，本项目所描述的关于制作一款卡牌题材的 IOS 平台网络游戏的想法应运而生。本项目的一大特色就是采用了较为新潮的 Go 语言来编写项目的服务端。Go 语言是由 Google 推出的一种简洁、高效的编程语言，这个语言在网络编程方面具有自己的特色与优点。本文会具体描述使用 Go 语言来开发这个服务端所遇到的问题与解决方案，并讲述在项目开发过程中的种种想法与体会。该服务端实现了对客户端连接与登录的管理，客户端连接维持，客户端游戏排队和具体的游戏逻辑等功能。

关键词：IOS 平台，网络游戏，服务端，Go 语言

DESIGN AND REALIZATION ABOUT SERVER OF ONLINE GAME ON IOS

ABSTRACT

Developing for mobile platform is one of the most popular direction of software developing. And IOS released by Apple is one of the most popular mobile platform. According to official statistics, the number of application downloads on IOS is greater than 500000, and game software holds the largest market share. So we can see that entertainment is becoming a more and more important use of mobile device expect communication as long as smart phones and tablet PC spread among the people. So we prepare to design and develop an card online game on IOS based on market demand. One important feature of this project is that we use the new programming language Go to develop a server. Go is a simple and efficient programming language developed by Google, and it is good at network programming. We will describe the problem and solution that we find in the process of developing, and we will tell some thought and experience that we got. This server can manage the connection and log-in of client, maintain the connections, manage the game queue, and realize the logic of this game.

Key words: IOS, online game, server, Go,

目 录

第一章 绪论

1.1 开发背景

1.1.1 开发语言

Go 语言是 Google 开发的一种编译型，并发型，并具有垃圾回收功能的编程语言。于 2009 年 11 月由 Google 正式宣布推出[1] [2]。

Go 语言是一个开源项目，其目的是提高开发人员的生产效率。Go 语言的特点是表达力强、简明、整洁和高效。可以使用它的并发机制轻易地编写运行在多核或网络计算机上的程序，其新型的类型系统使程序的构建变得更加灵活和模块化。Go 程序能快速地被编译为机器码，并且具有垃圾回收和运行时反射功能。它是一个快速的、静态类型的、编译型的语言，但使用起来却像一个动态类型的、解释性的语言[3]。

Go 语言被 Google 宣传为 web 时代的 C，因为它兼具 C 语言的高效和 Python 等解释性语言的简洁，并且线程轻量级，并行架构友好，易于进行网络级编程[4]。Go 语言的语法接近 C 语言，但是对于变量的声明是不同的，其他语法不同之处是 for 循环和 if 判断式没有括号围绕。Go 语言支持垃圾回收功能。Go 语言的并行模型是以 Tony Hoare 的 CSP 为基础，采取类似模型的其他语言包括 occam 和 Limbo，但它也具有 Pi 运算的特征，比如通道传输。与 C++相比，Go 语言并不包括如异常处理、继承、泛型、断言、虚函数等功能，但增加了 slice 型、并发、管道、垃圾回收、接口（interface）等特性的语言级支持[5]。

Go 的前身是 Limbo。Limbo 是用于开发运行在小型计算机上的分布式应用的编程语言。它支持模块化编程，编译期和运行时的强类型检查，进程内基于具有类型的 channel 通讯，原子性垃圾收集，和简单的抽象数据类型。它被设计用于即便是没有硬件内存保护的小型设备上，也能安全的运行。Go 从 limbo 中集成的一个重要的特性是 channel。channel 是用于向系统中其他代理发送和接收特定类型对象的通讯机制。channel 可以用于本地进程间通讯；用于连接到命名的目的地的库方法。两种情况都是直接发送和接收操作的。在 limbo 之前，channel 早在 Erlang 和 Newsqueak 中就得到了应用。它们的时间关系如下图[6]：

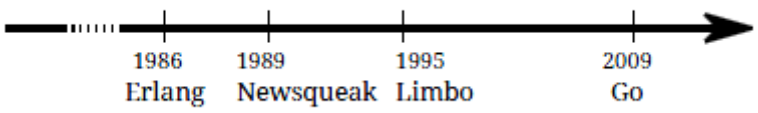

图 1-1 Go 语言的历史

1.1.2 项目开发背景

卡牌游戏是一种种类繁多、受众面广的游戏类型，也是桌面游戏的一种。卡牌游戏主要分为两类，即集换式卡牌游戏与非集换式卡牌游戏。非集换式卡牌游戏包括常见的扑克牌和现在非常流行的三国杀等，集换式卡牌游戏则包括万智牌、游戏王等。

卡牌游戏在最近的时间里，在 IOS 平台等移动平台非常常见与流行。这些卡牌游戏大

多是集换式卡牌游戏，原因大致有以下几点：第一点是集换式卡牌游戏具有收集要素，这一点可以提高玩家的沉迷度，并且创造稳定的盈利点；第二点是在虚拟的平台上集换式卡牌游戏的卡牌间的互动规则设计的开放程度更大，甚至可以抛开卡牌的要素去设计交互规则，此类的代表有最近比较流行的 IOS 平台卡牌游戏《我叫 MT》；第三点是集换式卡牌游戏玩家间除去基于游戏规则的对抗或者合作，还有交易要素，这一点也可以作为一个盈利点。总结来说，就是集换式卡牌游戏可以预期带来较高的收益，并且相对来说设计较为简单和开放。

图 1-2 集换式卡牌游戏[7]

1.2 项目目标与主要工作

基于集换式卡牌游戏的交互特性与数据安全性的考虑，这个类型游戏必然需要做成需要服务端参与的网络游戏。服务端在项目中主要负责用户数据和游戏数据的存储，以及游戏逻辑的运算。

本项目的目标就是实现一个能够持续正常运行的服务端，服务端可以不断接收新的连接，完成用户创建与登录工作。玩家进行对战要建立玩家的排队队列，然后分配每一局游戏的玩家与对手，每局对战过程中的逻辑运算要由服务端计算完成，并且把计算结果用消息发送给客户端，并接受客户端的反馈。

1.3 本文的组织结构

本文共有 8 章，分别是：

第一章绪论，介绍了项目的开发背景，项目目标与主要工作；

第二章相关技术介绍，介绍了项目中主要应用到的 3 个技术，有 goroutine、interface 和 JSON，前两个都是 Go 语言的最主要的特性；

第三章项目需求分析，包括了系统整体分析，分模块的系统功能与用例分析和非功能性需求。

第四章架构设计与数据库设计，包括了分模块的用例实现，进程设计与数据库设计。相关设计都配有相关视图。

第五章用户管理子系统的实现，包括了文件目录结构，服务端的基础构建，系统相关

类的设计与实现。分析结合了相关代码与编写思路。

第六章游戏逻辑子系统的实现，包括了相关文件结构和系统相关类的设计与实现。该系统的实现需要对游戏规则的理解，在文档里也进行了详细的说明。

第七章系统测试，，包括了测试概述和功能测试。该章节会列出测试用例和测试结果。

第八章总结与展望，分析了整个项目的开发过程，找出了不足，并提出了未来目标。

第二章 相关技术介绍

2.1 Go/Goroutine

2.1.1 Goroutine 的介绍与实现

Goroutine 是 Go 语言在语言级别支持并发的一种特性，Go 语言最大的特点就是线程轻量级的实现，goroutine 配合 channel 形成了 Go 语言处理并发的基础。但 goroutine 的实现并不是真正的并发，而是类似 coroutine(协程)。

Goroutine 的调度实现的代码主要位于 src/pkg/runtime/proc.c。从代码中，我们可以清楚地看到调度的过程：

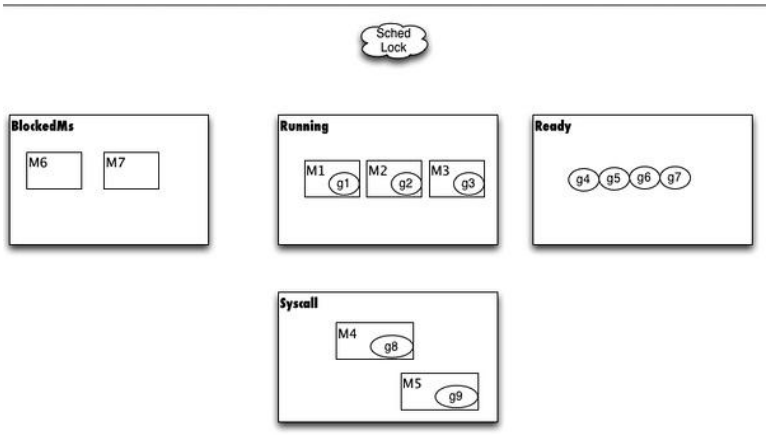

图 2-1 goroutine 运行时的状态与对象

上图中，M 是 Go 中对 CPU 的抽象，即一种可以执行代码的对象，实际是用线程来实现的；g 是 Go 对 goroutine 的抽象，是 Go 中的调度单位；Running 方框代表 runtime.GOMAXPROCS()函数所控制的同时运行的 goroutine 的最大数量。

g 平时是在 M 中运行的，在一定的时机，M 会对身上放着的 g 进行切换，换成一个 Ready 中的 g 执行，这个时机一般是 goroutine 进入 Syscall 或者用户主动调用 runtime.Gosched()函数进行切换。

下面以主动切换为例，说明调度的过程：

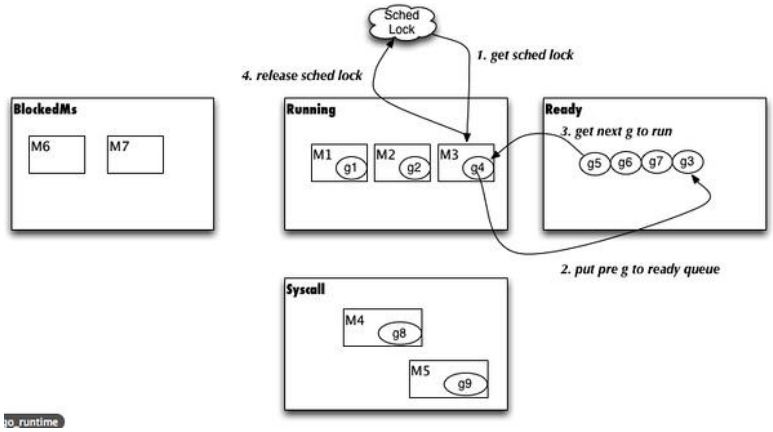

图 2-2 goroutine 调度过程

如上图所示，整个过程有以下几个步骤：

1)获取全局对象 sched lock，锁住调度，确保同时只有一个调度在发生；

2)把当前 M 上的 g 放到 Ready 队列里；

3)从 Ready 状态的 g 中选出一个来执行；

4)释放 sched lock。

以上就是 goroutine 具体的调度实现过程[8]。

2.1.2 Goroutine 的具体使用

在实际代码中，只需要在函数前加上关键词 go，就可以使该函数并行运行。例如：

```
func main() {
    go funcA()
    funcB()
}
```

上面一段代码中，funcA()与 funcB()就可以“并行”的执行。但实际上这并不是真正的并行，因为 funcA()与 funcB()实际上还是在一个线程上的，只不过 goroutine 通过自身的调度，让这段代码看上去是在并行运行而已。但实际上，funcA()和 funcB()很有可能就无法正常并行运行。比如，funcA()中有一个循环，让该线程一直饱和运行，这时就会发现，funcB()完全没有运行。这就是调度的问题，也是当我们要用 goroutine 时必须要考虑的问题。

那当我们的代码无法避免上述的情况时，我们应该怎样来解决这个无法并行的问题呢？现在来说，有 3 个解决的办法。第一种是，在 main 函数里调用 runtime.GOMAXPROCS()函数，将参数设置为 2 或者更大。这个函数的作用是手动分配几个线程给 goroutine 用于调度。调用这个函数之后，上述例子中的 funcA()与 funcB()会在 2 个线程中并行运行。第二种是制造阻塞，比如在 funcA()中的循环体里面加入 time.Sleep()函数，这样子当 funcA()阻塞住之后，funcB()就得以执行。第三种方法是手动切换，可以在函数循环体中调用函数 runtime.Gosched()，该函数的作用就是手动切换 goroutine。

2.2 Go/interface

Go 语言中的 interface 被赋予了多种不同的含义，每个类型都有接口，意味着对那个类型定义了方法集合。Go 语言无须明确一个类型是否实现了一个接口，这实际上实现了鸭子

类型(duck typing)的模式。鸭子类型是动态类型的一种风格。在这种风格中，一个对象有效的语义，不是由继承自特定的类或实现特定的接口，而是由当前方法和属性的集合决定[9]。

在 Go 中的接口有着与许多其他编程语言类似的思路：C++ 中的纯抽象虚基类，Haskell 中的 typeclasses 或者 Python 中的 duck typing。然而没有其他任何一个语言联合了接口值、静态类型检查、运行时动态转换，以及无须明确定义类型适配一个接口。这些给 Go 带来的结果是，强大、灵活、高效和容易编写的。任何数据结构，只要实现了 interface 所定义的函数，自动就 implement 了这个 interface，没有像 Java 那样冗长的 class 申明，提供了非常灵活的设计度和 OO 抽象度，让你的代码非常干净。

而且基于 interface 的设计，空 interface 在 Go 语言里可以表示任何类型，有一种类似泛型的用法。

在本项目中，interface 还起到了一个用于模仿实现继承的作用。因为 Go 语言本质上不是面向对象语言，而是面向过程化的，本身不支持继承。但是这个具体项目设计过程中是有没法避开集成的地方的，于是为了模仿继承的实现，本项目采取了用一个拥有子类所有方法接口的 interface 来代替父类的方法。

2.3 JSON

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，采用完全独立于语言的文本格式，是理想的数据交换格式[10]。 在 JSON 中，有两种结构：对象和数组。

对象的格式：以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值’ 对”之间运用 “,”（逗号）分隔。 名称用引号括起来；值如果是字符串则必须用括号，数值型则不须要。例如：

{ "first": 1, "secend":"what" }

数组的格式：数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间运用 “,”（逗号）分隔。例如：

{"a":[{ "first": 1, "secend":"what"},{ "first": 2, "secend":"where"}]}

在本项目中，JSON 用于封装网络消息，只采用了对象的结构。项目中每一个消息都是一个 struct，例如：

```
type MatchingInfo struct {
    InfoType int
    SelfName string
    SelfHp   int
    OppoName string
    OppoHp   int
    IsRed    bool
    DeckNum  int
}
```

每个消息都有一个类型标志位，JSON 的封装和解析都以此为准。Go 语言对 JSON 是原生支持的，可以用自带的函数对数据进行封装与解析。

2.4 本章小结

在本章节中，我们主要对该服务端编写需要用到的技术进行了解释与阐述。本章节介绍的技术前两个，goroutine 与 interface 其实是 Go 语言二个最重要的特性，对于用 Go 语言编写的本项目来说，这两个技术是必须要阐释清楚的 2 个。JSON 则是服务端与客户端通信传送消息所应用到的封装手段。在有了对用到的技术的基本了解之后，我们会继续先从项

目的需求分析入手进一步分析整个项目。

第三章 项目需求分析

3.1 系统整体分析

3.1.1 整体功能分析

该项目作为一个卡牌游戏的服务端，首先我们先考虑整体的需求。从玩家的操作流程来看，需要先创建角色，然后登陆游戏，管理卡牌，进行匹配游戏。当然这是玩家的操作流程，主要是客户端需要考虑的交互过程。服务端需要完成的任务是配合客户端的请求完成玩家的操作。所以从相应的操作可以看出，我们需要这么几个系统，第一个我们可以称为用户管理系统，该系统负责响应用户的注册、登录、退出操作以及玩家的卡牌管理和游戏匹配；第二个我们可以称为游戏逻辑系统，该系统负责处理玩家在一局游戏中游戏中所有操作的响应。

现在我们分好了项目的子系统。对于用户管理系统来说，所有的需求基本都已经确定，这个系统一旦完成后，后面应该基本不会有大的改动。而对于游戏逻辑系统来说，这个系统负责所有游戏方面的管理，对于游戏项目来说，游戏整体的设计需求是不会马上全部确定的，后期的修改与添加是会非常频繁的，所以这一系统是先完成现在的最基本的需求。

3.1.2 用户特征

该产品面向用户主要为 IOS 平台使用者。主要的使用者一般会有喜欢卡牌游戏，具有收集的爱好，喜欢与别人合作与比拼等特点。

3.1.3 设计约束

该项目作为一个网络游戏项目，必须要考虑的设计约束有 2 点：

第一点，网络延迟。在游戏操作的过程中需要考虑网络延迟的影响，设置相应的响应时间。极端情况下，可能会出现网络断线的问题，针对这一点，应该考虑断线重连的解决方案。

第二点，多用户并发的效率与稳定性。该网络游戏一定会有很多玩家的并发操作与游戏，设计上要考虑怎样保证运行效率，而且服务器的稳定性也是要必须保证的。

3.1.4 项目开发环境

开发语言：Go 语言 1.1
开发系统：MacOS 10.8
数据库软件：MySQL 5.5.16 for Mac

3.2 系统功能与用例分析

3.2.1 用户管理子系统

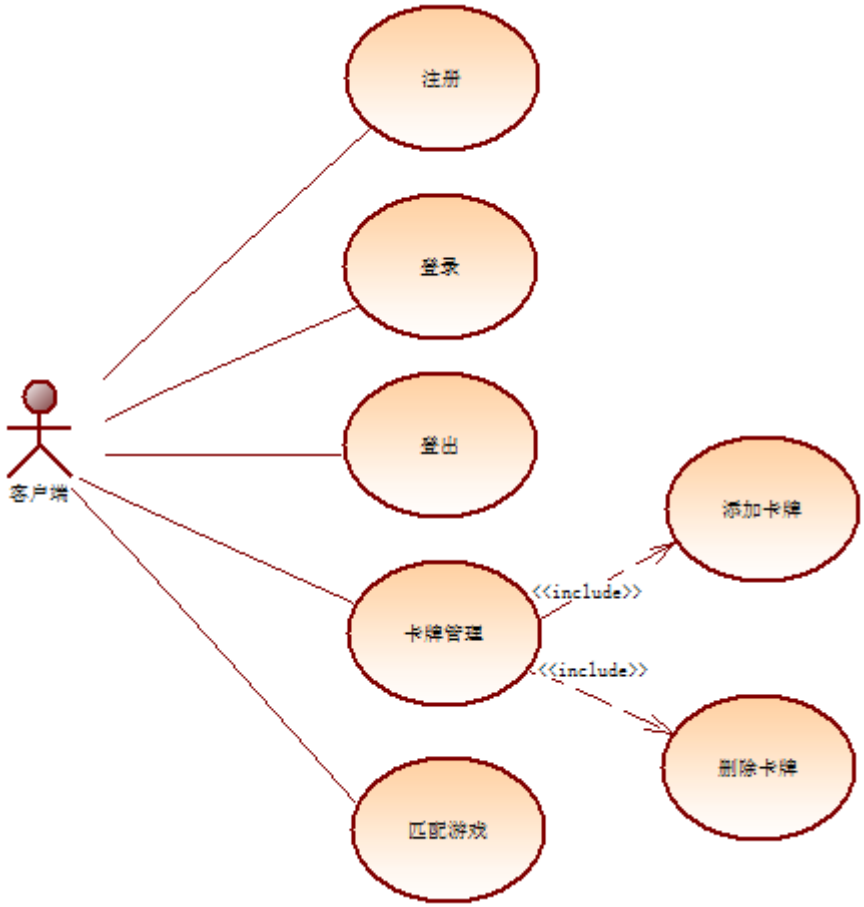

图 3-1 用户管理子系统用例图

整体说明：

用户管理子系统是用户通过客户端与服务端进行交互的平台，负责除了游戏逻辑外其他所有的功能。

用例规约：

注册、登录与登出是最基本的用户管理用例，在此略过用例规约。

卡牌管理：

简要说明：客户端发送管理卡组的请求，包含添加卡牌或者删除卡牌。添加卡牌可能是购买或者奖励获得，删除卡牌可能是出售或其他。

事件流：

(a) 客户端传递一条添加或者删除某卡牌的消息；

(b) 若是添加，则服务端在玩家背包添加该卡牌，并且执行添加条件，比如购买需要减少相应的金钱；若是删除，则服务端在玩家背包搜索该卡牌，找到后删除，并且执行删除条件，比如出售需要增加响应金钱；

(c) 反馈给客户端执行结果成功的消息；

备选流：

(a) 如果是购买卡牌，没有足够的金钱，不满足添加条件的话，则直接反馈给客户

端执行结果失败的消息；

(b) 如果是删除卡牌，玩家背包里不存在该卡牌的话，则直接反馈给客户端执行结果失败的消息。

匹配游戏：

简要说明：客户端发送玩家匹配游戏的请求。服务端会为该玩家找到一名对手，并开启一局游戏。

事件流：

(a) 客户端传递一条玩家匹配游戏的消息；

(b) 服务端判断该玩家当前状态，若为准备排队的状态，则将该玩家加入匹配队列，匹配队列会为 2 个正在排队的玩家开启一局游戏；

(c) 服务端反馈给客户端玩家进入一局游戏的消息；

备选流：

(a) 该玩家当前状态不为准备排队状态，则反馈给客户端匹配失败的消息；

3.2.2 游戏逻辑子系统

整体说明：

游戏逻辑子系统是负责游戏流程的系统，该系统负责按照给定的游戏规则，处理在玩家的操作下游戏的运行过程与结果，游戏的过程都会通过消息发送给客户端以显示。

	客户端	数据传输	服务端
1	加入（登陆）	“join”:“user”:xxx, “password”:xxx	
2			检验，匹配。
3		“info”:“name”:xxx, “hp”:xx “name”:xxx, “hp”:xx	匹配成功。
4	界面初始化		随机先手
5		isTurn 为 0,则没有后面的信息 “turn”:“isTurn”:1/0 ,“number”:x, “cards”:“id”:x ,“id”:x…	提示回合（2 个玩家）。发牌。
6	界面更新，计时开始		
7	放牌（每次）	“place”:“which”:x, ”position”:102	
8			更新并通知每个玩家。
9	结束回合	“over”:1/0	
10			开始另一个玩家的回合，重复 5-9 一次。
11		“move”:“from”:204, “to”:210 “attack”:“from”:301,“to”:303, “hurt”:x“win”:1/0	战斗模拟（每次）。判断胜利。
12	更新动画		

图 3-2 游戏逻辑子系统流程简单说明

游戏规则简要说明：

(a) 每位玩家具有初始给定的血量和一副套牌，套牌中暂定由士兵卡牌和技能卡牌构成。

(b) 游戏按照每个玩家一回合，轮回执行。

(c) 当有任意一位玩家血量降为 0 时，则该玩家该局游戏失败，对方玩家胜利。

(d) 每个玩家回合分为 4 个阶段，分别为抽牌阶段、放置阶段、战斗阶段和结束阶段：
抽牌阶段，无特殊情况玩家抽取 1 张卡牌。在游戏刚开始时双方玩家都要先抽取 2 张卡牌才进行后面的回合。
放置阶段，玩家将可以使用的士兵卡牌放置在战场上，或者使用技能卡牌。
战斗阶段，玩家放置好的卡牌进行行动，包括移动和攻击以及卡牌技能的发动与使用。
结束阶段，更新玩家手牌的状态，并检查手牌数是否超过上限，超过则弃掉超过数目的牌。

3.3 非功能性需求分析

3.3.1 可靠性

能够满足 24*7 小时无故障运行，每周能有 2 小时时间停机维护；代码不能出现严重的逻辑错误。

3.3.2 性能

系统能同时容纳的用户数大于 1000 人，支持高并发操作；尽可能多的缩减消息通信次数；对消息的响应时间要快速而稳定。

3.3.3 可支持性

系统可以发行多种平台的编译版本；系统严格按照 Go 语言的编码标准。

3.4 本章小结

在本章节中，我们主要对本项目的需求进行了细致的分析。通过需求说明，我们应该逐渐了解本项目所做的工作。用户管理子系统是对整体用户进行管理的系统，游戏逻辑子系统则主要负责游戏逻辑的运算。在了解了这些之后，我们会对系统的架构设计进行相应的说明。

第四章 架构设计与数据库设计

4.1 用例实现

4.1.1 用户管理子系统

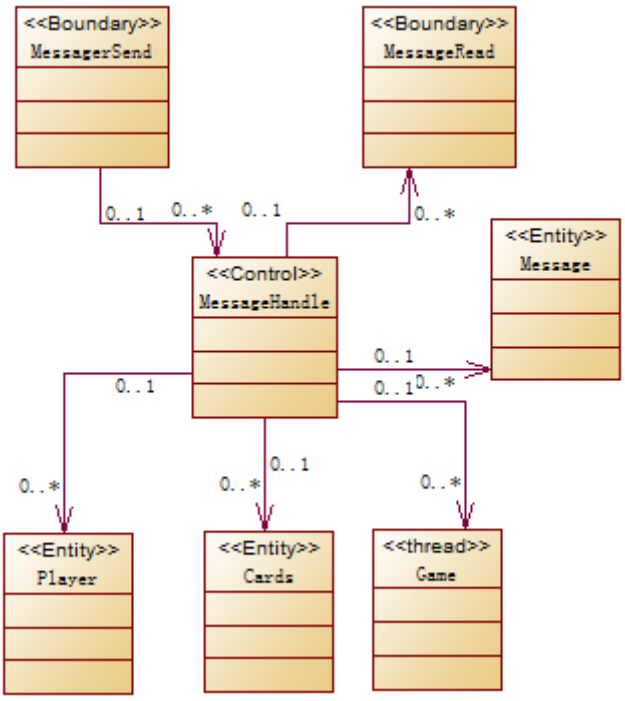

图 4-1 用户管理子系统用例实现类图

上图中的设计类中，MessageSend 与 MessageRead 负责与客户端消息的收发，Message 记录各种消息类型，Player 记录玩家信息，Cards 记录卡牌信息，Game 表示一局游戏，MessageHandle 负责通过消息交互来运行相关函数。

MessageHandle 控制消息的收发，通过相应的消息来执行相应的接口，Player 和 Cards 都是对相应的数据的管理，Game 则是对游戏逻辑系统对外部的接口，也负责玩家匹配游戏排队队列。整个系统通过消息与外部交互。

4.1.2 游戏逻辑子系统

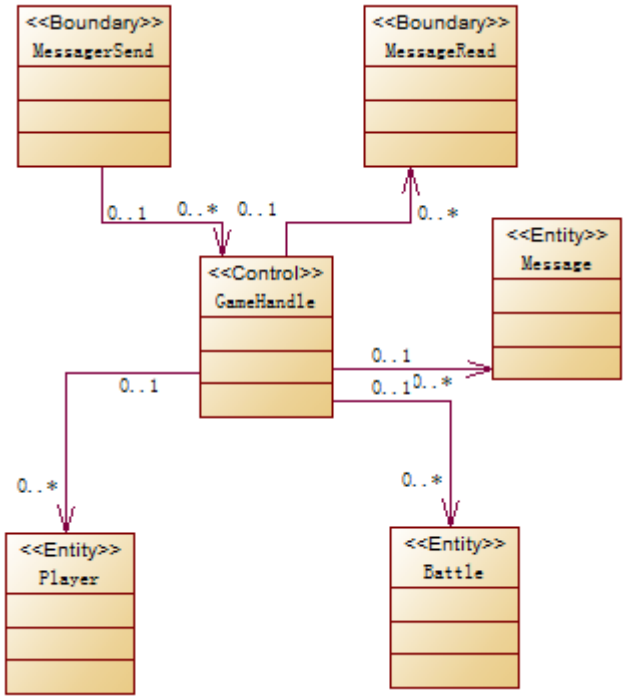

图 4-2 游戏逻辑子系统用例实现类图

上图中的设计类中，MessageSend 与 MessageRead 负责与客户端消息的收发，Message 记录各种消息类型，Player 记录玩家信息，Battle 负责记录对战战场信息，GameHandle 负责消息交互并且进行逻辑运算。

游戏逻辑子系统依然是通过消息与外部交互的。Player 依然是存储的玩家数据，Battle 则是存储战场容器以及拥有战场操作的接口。

4.2 进程视图

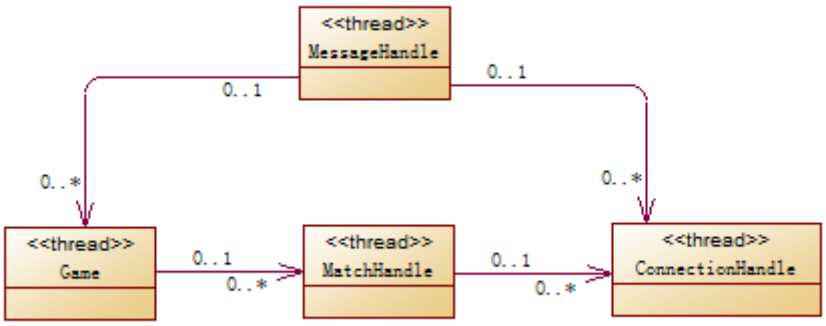

图 4-3 整个系统的进程视图主框架

该系统整体设计基于 Go 语言的 goroutine，整个系统是多线程运行的。上图中：

线程 MessageHandle 负责与客户端消息的收发；

线程 ConnectionHandle 负责客户端的连接与维持，管理用户的登录等活动；

线程 MatchHandle 负责匹配排队游戏的玩家；

线程 Game 负责进行一局游戏；

4.3 数据库设计

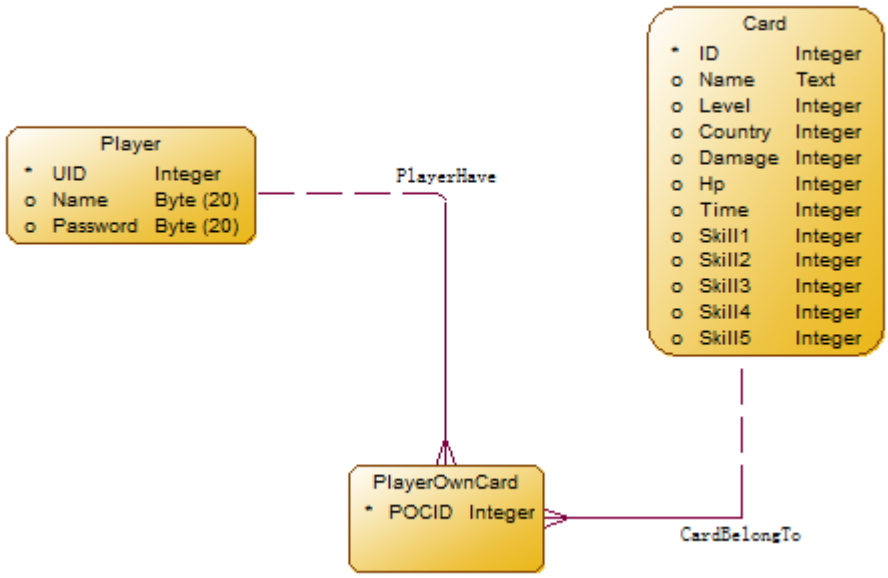

图 4-3 整个系统的数据库 LDM

该系统的数据库设计较为简单。实体表有 2 个，分别是存储用户信息的 Player 表，以及存储卡牌信息的 Card 表。这两个表的关系是多对多的，因为一个玩家会有多张卡牌，而一张卡牌可能属于多个玩家，所以又建立一个关系表 PlayerOwnCard。

4.4 本章小结

在本章节中，我们主要对本项目的架构设计以及数据库设计进行了细致的分析。通过架构设计的描述，我们大概了解了本项目编写的大概框架。作为一个服务端项目，自然本项目要考虑到多线程的实现，也一定会应用到数据库的操作。下面的章节中，我们会对整个项目的细节进行详细的描述。

第五章 用户管理子系统的实现

5.1 文件目录结构

图 5-1 文件目录结构

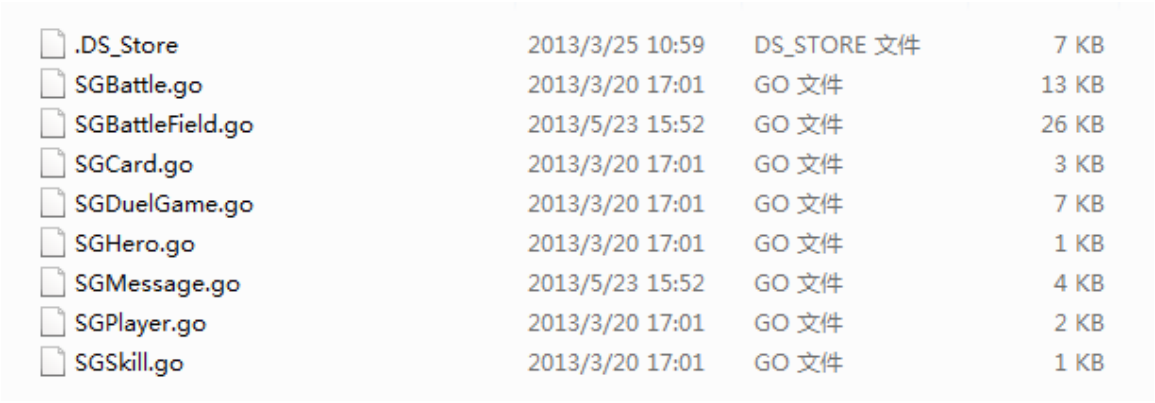

图 5-2 SGGame 文件夹内文件

因为 Go 语言简洁的风格，以及 Go 语言实质是一个面向过程的语言，所以 Go 语言编写的项目的文件结构是很简洁的。

最外层的 SGGameServer.go 是调用其他文件的执行文件。SGGame 包内是定义的 struct。SGGameServer.go 通过 import ("./SGGame")，就可以调用包内的 struct。

5.2 服务端的基础构建

用户管理子系统是实现服务端与客户端通信的基础。构建这个系统的第一部就是搭建一个可以与客户端建立连接并且通过 Socket 建立通信的渠道。要实现这个基本功能，用 Go 语言是非常简单的。Go 语言自带的 net 库中对建立监听端口已经进行好了封装，我们只需要简单的调用就可以完成，具体的实现如下所示：

```
lis, err := net.Listen("tcp", remote)
defer lis.Close()
if err != nil {
    fmt.Printf("Start Server Error: %s\n", err)
}
fmt.Println("Initiating Server... (Ctrl-C to stop)")
```

实现代码非常的简洁明了，这也充分的体现出了 Go 语言在此方面的优势。

在接收连接的同时，同时建立一个线程对接收到的连接进行登录验证。可以通过关键词

go 来建立一个新的 goroutine，go acceptConnection(lis)。

对于我们接收到的连接，也需要一个建立一个容器来存储。该项目中，我使用了一个 Map 来存储连接，并且用一个 int 来记录连接的状态，如下所示：

```
var allConn map[net.Conn]int
```

记录的状态有 0(代表不在游戏中)，1(代表正在游戏中)，2(代表正在排队中)，3(代表断开连接)。

对于每一个连接，我们需要有维持连接的方式，并且能够判断每个连接当前是否可用。在本项目中，我使用了心跳维持的方式。服务端每间隔 1 秒，会往所有的连接发送一次心跳(心跳为一个主体内容为空的消息)，如果发送不成功，那么就会判定该连接断开。之后会将断开的连接从所有容器中删除。

最后，还需要一个 goroutine 来负责排队游戏玩家的匹配。按照现阶段的需求设计，匹配遵循先来先服务原则。该线程会将排队队列中的玩家两两匹配，匹配好的玩家会被分配到一个新的 goroutine 来进行一局游戏。

至此，用户管理子系统的构建已说明完毕。服务端在此构建下就可以不间断的运行来响应客户端的请求。

5.3 系统相关类的设计与实现

5.3.1 玩家类 SGPlayer 的设计与实现

该类存在于文件 SGPlayer.go 中，具体的内容如下：

```
type SGPlayer struct {
    playerID     int
    playerName string
    playerHp     int
    playerDeck []int
}
```

PlayerID 是玩家的 ID，playerName 是玩家的用户名，playerHp 是玩家的游戏中生命值，playerDeck 是玩家所使用的套牌。

这个类的设计非常简单。该类负责保存玩家当前的状态信息，信息是由从数据库中读取的信息来初始化的，也会反馈到数据库中来保存。基于 Go 语言的特性，struct 的成员函数是无需声明的，只需要在定义的时候标示，具体如下：

```
func (player *SGPlayer) GetName() string {
    return player.playerName
}
```

该类的数据初始化需要从数据库中读取。从之前的需求分析可知，本项目使用的是 MySQL 数据库，如下图：

id	name	lv	country	damage	hp	time	skill1	skill2	skill3	skill4	skill5
131001	??	1	1	1	4	1	0	0	0	0	0
132001	??	2	1	2	4	1	0	0	0	0	0
131002	??	1	1	1	6	2	1002	0	0	0	0
131003	????	1	1	1	4	1	1002	0	0	0	0
131004	???	1	1	2	2	1	1006	0	0	0	0
132004	???	2	1	2	3	1	1006	0	0	0	0
131005	????	1	1	2	4	2	0	0	0	0	0
132005	????	2	1	2	6	2	0	0	0	0	0
231001	???	1	2	1	4	2	1009	0	0	0	0
232001	???	2	2	1	5	2	1009	0	0	0	0
231002	????	1	2	1	5	2	1002	1009	0	0	0
231003	??	1	2	1	3	2	1015	0	0	0	0
232003	??	2	2	1	4	2	1015	0	0	0	0
231004	??	1	2	2	4	3	1001	1021	0	0	0

图 5-3 MySQL 数据库

Go 语言并没有对数据库的原生支持，于是我从 google code 上下载了一个开源的 Go 连接 MySQL 的驱动。安装完后，头文件需要添加如下声明：

```
import (
    _ "code.google.com/p/go-mysql-driver/mysql"
)
```

这个驱动的功能并不算特别全面，现在并没有特别成熟的用于 Go 的 MySQL 的驱动，但是基本的功能还是都支持的。该项目的数据库是被公司部署在阿里云远程服务器上的，经过测试该驱动是可以正常使用的。

5.3.2 卡牌类 SGCard 的设计与实现

该类存在于文件 SGCard.go 中，具体的内容如下：

```
type SGCard struct {
    cardID      int
    cardName    string
    cardLevel int
    roundMax    int
    roundLeft int
    playerID    int
}
```

以上是卡牌父类的内容。里面的成员变量都是卡牌的基本属性。CardID 是卡牌的 ID，cardName 是卡牌的名称，cardLevel 是卡牌的等级，roundMax 是卡牌使用所需等待的回合数，roundLeft 是卡牌使用还需等待的回合数，playerID 是该卡牌主人的玩家 ID。游戏中的卡牌是不能随时使用的，每张卡牌必须按照上面的标示等待相应的回合数才能够使用，这个是遵循了游戏策划方案里的设定。

该类的实现较为麻烦。因为按照游戏策划方案，卡牌分为 2 种，分为士兵牌和技能牌，这两种卡牌有部分属性是相同的。按照面向对象的设计方案，首先要设计一个父类 SGCard，然后有 2 个子类 SGSoldier 和 SGSkill 继承父类 SGCard。这个设计是非常自然的。但是又因为 Go 语言本身不是面向对象语言，所以不支持继承。但是 Go 语言有一个非常灵活的 interface 特性，于是可以从这个特性入手来模拟集成的实现。

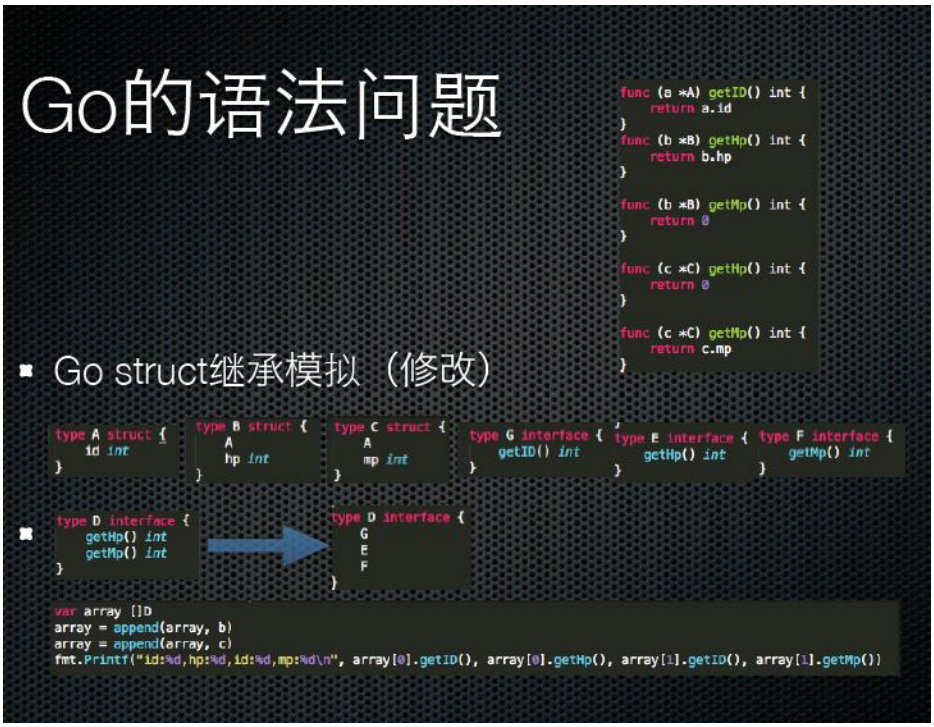

图 5-3 Go 语言继承的模拟

如同上图所示，图中 A 是父类，B 和 C 是继承 A 的子类。通过让 A 成为 B 与 C 的成员变量，可以通过 B 和 C 直接访问 A 的成员变量，这样实现了继承的一部分功能。然后我们要实现的是子类对于父类成员函数的继承。于是我们声明 3 个 interface G E F，它们的接口分别是 A B C 的成员函数，然后声明 1 个 interface D，它包含上面 3 个 interface。当然这里 G E F 的存在是为了区分 3 个类的成员函数。这样声明之后，D 实际上就可以被我们当做一个通用的接口来使用。比如，我们声明一个 D 的数组，然后可以在里面存入 B 和 C 的实例，然后我们可以从 D 的数组中取出任意一个成员，然后访问 B 或者 C 的成员函数。

在实际项目中，我也是如此编写的。父类是 SGCard，子类是 SGSoldier 和 SGSkill，通用的接口如下：

```
type cardOpt interface {
    SGCardOpt
    SGSoldierOpt
    SGSkillOpt
}
```

在后面的应用中，我们只需要声明 cardOpt 的容器，就可以存储所有类型的卡牌，并且正常的取出使用了。

5.3.3 通信消息的设计与实现

所有的通信消息存在于文件 SGMessage.go 中。

消息类的结构：

```
type DataInfo struct {
    infoType int32
}
```

根据设计，服务端与客户端的通信内容的结构为:（前 4 位）长度 +（1 位）信息类型 + 信息内容。其中，消息长度是在计算后将结果的整数值转换为 4 个字节的字符类型，具体实现方法如下：

```
func intToByteArray(i int) [4]byte {
    var result [4]byte
    result[3] = byte((i >> 24) & 0xFF)
    result[2] = byte((i >> 16) & 0xFF)
    result[1] = byte((i >> 8) & 0xFF)
    result[0] = byte(i & 0xFF)
    return result
}
```

算法很简单，就是将 int 的 4 个字节的内容的每个字节都存储在一个字符类型的变量里。信息类型是根据约定好的 int 来代表该消息具体是哪一个，用于接收端对消息包的解析。

现在定义的消息从消息的收发源来分可以分为 2 种，就是由客户端发给服务端的和由服务端发给客户端的，根据功能需求划分的话大概可以分为登录消息、匹配消息、游戏过程消息等几类，现在大部分的消息都是用于游戏进行过程中交互用的。

消息的内容是用 JSON 来封装的。因为 Go 语言对 JSON 的原生支持，可以非常方便的的封装与解析。

5.4 本章小结

在本章节中，我们主要对用户管理子系统的所有细节实现进行了较为详细的描述。用户管理子系统包括了整个服务端大体框架的构建，通过本章的描述之后，我们应该对服务端的大体框架有了一个较为清晰的认识。下面的章节中，我们会对本项目另一个重要的子系统游戏逻辑系统进行详细的讲解。

第六章 游戏逻辑子系统的实现

6.1 相关文件结构

与游戏逻辑子系统相关的文件主要有 SGBattle.go SGBattleField.go SGDuelGame.go SGSkill.go 等 4 个，其中前三个是与游戏逻辑实现相关的。

SGBattle.go 是针对每一个玩家的，拥有存放套牌(cardDeck []cardOpt)、手牌(cardHand []cardOpt)和卡牌坟墓(cardTomb []cardOpt)的容器，还有负责记录玩家状态比如游戏中的生命值、最大手牌数等的相关变量。SGBattle.go 中的函数主要是几个容器之间变量的传递，因为从游戏逻辑上来说，这就是卡牌在几个容器中位置的变化。

SGBattleField.go 是最核心的文件。其中定义了每局游戏的两个玩家(player [2]SGBattle)，以及战场(field [3][10]cardOpt)。所有的具体的游戏逻辑都是在这里实现的。

SGDuelGame.go 中主要是对游戏逻辑的调用。它向外部提供了进行一局游戏的接口，接口内会初始化所有的变量，然后调用游戏逻辑。

SGSkill.go 则是用类似枚举变量的形式记录了游戏中所有的技能。之所以说是类似枚举变量，是因为 Go 语言没有提供对枚举变量的支持，然而存在关键字 iota，可以用它来代替实现，具体如下：

```
const (
    SkillRun = iota + 1001
    SkillRide
    SkillSlow
    SkillStop
)
```

这样定义之后，SkillRun 就会被赋值为 1001，下面的会依次叠加，从而类似实现了枚举变量，当然缺点是没法定义一个枚举变量类型。

6.2 系统相关类的设计与实现

6.2.1 游戏中玩家状态类 SGBattle 的设计与实现

该类存在于文件 SGBattle.go 中，具体的内容如下：

```
type SGBattle struct {
    hero        SGHero
    maxHand    int
    cardDeck []cardOpt
    cardHand []cardOpt
    cardTomb []cardOpt
}
```

其中 hero 是玩家所扮演的角色，maxHand 是玩家最大手牌数，cardDeck 是玩家套牌，cardHand 是玩家手牌，cardTomb 是玩家卡牌墓地。

类成员函数有：

```
func (battle *SGBattle) shuffleCard() bool
func (battle *SGBattle) drawCard(cardNum int, conn net.Conn) bool
func (battle *SGBattle) placeCard(hand int, pos Position, cardBattle *SGBattleField, conn
```

net.Conn) bool

func (battle *SGBattle) removeCard(pos Position, cardBattle *SGBattleField) bool

func (battle *SGBattle) dropCard(hand int) bool

func (battle *SGBattle) cardMove(from Position, to Position, cardBattle *SGBattleField, self net.Conn, oppo net.Conn) (int, bool)

func (battle *SGBattle) cardHeal(to Position, hp int, cardBattle *SGBattleField, self net.Conn, oppo net.Conn) (int, bool)

func (battle *SGBattle) cardAttack(from Position, to Position, cardBattle *SGBattleField, self net.Conn, oppo net.Conn) (int, bool)

func (battle *SGBattle) cardAttackPlayer(from Position, isRed bool, cardBattle *SGBattleField, self net.Conn, oppo net.Conn) (int, bool, int)

func (battle *SGBattle) loseGame(self net.Conn, oppo net.Conn) bool

所实现的函数大部分都是围绕着 3 个容器展开的。shuffleCard()是洗牌函数，作用于玩家套牌容器。drawCard() placeCard() removeCard() dropCard()等 4 个函数都是卡牌在不同容器中的位置改变，分别实现的是抽牌，即将卡牌从套牌容器移到手牌容器，放牌，即将卡牌从手牌容器移到战场容器，移除卡牌，即将卡牌从战场容器移到卡牌墓地容器，弃牌，即将卡牌从手牌容器移到卡牌墓地容器。

后面几个函数都跟游戏逻辑有一定的相关性。cardMove 是卡牌在战场上移动。cardHeal 涉及到卡牌的一种技能“治疗”，是对该技能的实现。cardAttack 与 cardAttackPlayer 都是某卡牌进行攻击，不过攻击对象不同，前者是另一张卡牌，后者是玩家。loseGame 则是对胜利条件的判断。

上面函数实现的是对某一张卡牌所能进行的所有的操作，后面游戏逻辑的实现会以这里的实现为基础。上述函数中，参数带有 net.Conn 类型的是表示该函数给客户端发送消息，因为很多操作都是要告知客户端，然后客户端用动画的形式来表现。

6.2.2 游戏战场逻辑类 SGBattleField 的设计与实现

该类存在于文件 SGBattleField.go 中，具体的内容如下：

```
type SGBattleField struct {
    winner int              //-1 for left one, 1 for right one
    player [2]SGBattle    //0 for left one, 1 for right one
    field    [3][10]cardOpt
}
```

其中 winner 是该场游戏的胜者，用于判断游戏结束的条件。Player 是 2 个 SGBattle，分别用来进行 2 个玩家在游戏中的操作。Field 则是用于存储战场的容器。

类成员函数有：

func (battleField *SGBattleField) oneTurn(currentPlayer int, self net.Conn, oppo net.Conn) (int, bool)

func (battleField *SGBattleField) initPhase(currentPlayer int, num int, conn net.Conn) bool

func (battleField *SGBattleField) drawPhase(currentPlayer int, conn net.Conn) bool

func (battleField *SGBattleField) placePhase(currentPlayer int, self net.Conn, oppo net.Conn) (int, bool)

func (battleField *SGBattleField) fightPhase(currentPlayer int, self net.Conn, oppo net.Conn) (int, bool)

func (battleField *SGBattleField) roundPhase(currentPlayer int) bool

func (battleField *SGBattleField) healPhase(pos Position, currentPlayer int, self net.Conn, oppo net.Conn) (int, bool)

func (battleField *SGBattleField) searchForGuard(pos Position, currentPlayer int, faceToRight bool) (bool, Position)

func (battleField *SGBattleField) searchForEnemy(pos Position, attRange int, currentPlayer int) (bool, bool, Position)

func (battleField *SGBattleField) searchForEmpty(pos Position, movStep int, currentPlayer int) (bool, Position)

func (battleField *SGBattleField) longRangeRule(pos Position, currentPlayer int, self net.Conn, oppo net.Conn) (int, bool)

func (battleField *SGBattleField) shortRangeRule(pos Position, currentPlayer int, self net.Conn, oppo net.Conn) (int, bool)

该卡牌游戏的执行时回合制的，每个玩家按照回合内的阶段顺序轮番行动。

OneTurn 函数是对游戏中一回合的所执行操作的完整的描述。按照当前的游戏策划方案的设定，玩家在自己的回合会依次执行 roundPhase，该回合阶段负责对玩家手牌的等待回合数按照规则进行更新调整，然后是 placePhase，该回合阶段负责接收客户端玩家的操作消息在战场上防止卡牌，再后面是 fightPhase，该回合阶段负责按照规则使当前玩家场上的卡牌进行行动，最后是 drawPhase，该回合阶段负责给玩家发放相应数量的手牌。

InitPhase 函数实现了一个特殊的回合阶段，它不被包含在 OneTurn 函数中，它只会在游戏起始的回合由双方玩家都执行一次。该函数完成的任务是，双方玩家分别进行洗牌，然后发放相应数量的起始手牌。

DrawPhase 函数是实现的抽牌阶段。只要满足条件，当前套牌内卡牌数量大于 0，和玩家当前手牌数小于规定的最大手牌数，则玩家抽取相应数目的牌，按照当前的游戏策划方案，该阶段抽取的数目是固定的，为 1。

PlacePhase 函数是实现的放牌阶段。放牌阶段是有规定好的 30 秒的时间限制，当 30 秒钟消耗完，或者玩家发送结束放牌的消息时，放牌阶段立即结束。具体实现过程中，在这里建立了一个新线程用于计时，线程由该阶段开始时计时，30 秒后通过 channel 传递循环终止参数，而主线程则一直等待读取客户端的消息，读到放牌消息则按照消息来放牌，读到结束消息则结束等待循环，或者通过 channel 改变终止参数来结束等待循环。

FightPhase 函数是实现的战斗阶段。战斗阶段是游戏最复杂的一个阶段。后面的 healPhase、searchForGuard、searchForEnemy、searchForEmpty、longRangeRule、shortRangeRule 等函数都是被 fightPhase 函数所引用，用于战斗阶段实现的函数。因为技能的存在，战斗系统中有很多额外的判定阶段，但如果暂时先把这些要素刨除，战斗阶段的核心部分就是士兵牌的行动规则的实现。在该函数中，我们会在战场上搜寻该玩家的卡牌，然后按照卡牌的类型来执行不同的行动规则。

在该游戏的策划方案的设定中，士兵牌是有很多种类型的，他们的行动方式都有一定的区别。但是如果从一个大概的行动准则来划分的话，可以暂时把士兵牌分为 2 种类型：近战士兵和远程士兵。这两种基本类型的士兵可以用一句话来简单描述他们的行动准则，近战士兵是靠近敌人后攻击，远程士兵是在最远距离攻击敌人。下图是对近战士兵逻辑具体描述的流程图：

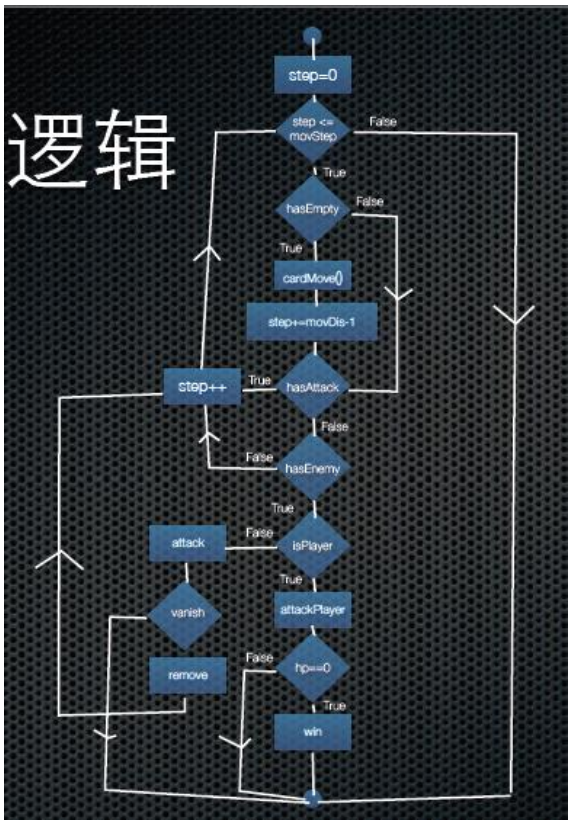

图 6-1 近战士兵逻辑

在基本的行动逻辑确立之后，具体的逻辑的确立还需要继续考虑技能对士兵行动流程的影响。按照目前的游戏策划方案，游戏中的技能主要有这几种，如下图：

	SKILLID	名字	效果
2	S1001	奔跑	移动＋1
3	S1002	骑兵	移动＋2
4	S1003	钝步	移动－1
5	S1004	锁足	移动－2
6	S1005	短弓	射程＋2
7	S1006	弓箭	射程＋3
8	S1007	长弓	射程＋4
9	S1008	飞行	移动时可穿越敌方单位
10	S1009	铁甲1	减免受到的物理伤害1点
11	S1010	铁甲2	减免受到的物理伤害2点
12	S1011	铁甲3	减免受到的物理伤害3点
13	S1012	法抗1	减免受到的法术伤害1点
14	S1013	法抗2	减免受到的法术伤害2点
15	S1014	法抗3	减免受到的法术伤害3点
16	S1015	治疗1	每回合为受到最多伤害的友方士兵回复1点生命
17	S1016	治疗2	每回合为受到最多伤害的友方士兵回复2点生命
18	S1017	治疗3	每回合为受到最多伤害的友方士兵回复3点生命
19	S1018	治疗4	每回合为受到最多伤害的友方士兵回复4点生命
20	S1019	吸血	造成伤害时，对自己治疗等量的伤害
21	S1020	反击	受到射程内的攻击时，反击攻击者
22	S1021	破甲	造成的伤害无法被减免
23	S1022	警戒	优先攻击身后的单位

图 6-2 卡牌游戏士兵技能

这些技能可以被分为不同的类型，会在游戏系统中不同的模块中被实现。具体来说，前 7 个技能是对卡牌属性的影响，会在对卡牌初始化赋值的时候实现；第 8 个技能“飞行”则是一个会影响游戏逻辑的技能，它的影响体现在对移动位置的选择不同，而 searchForEmpty 函数是返回卡牌下一步移动位置的函数，该技能会在这个函数里实现；技能 9-14 以及 19、21 都是对卡牌收到伤害计算过程的影响，会在 SGBattle 类里在 cardAttack 函数里进行判断

与实现。技能 15-18 实际上是在战斗阶段直接添加了一个新的阶段，于是我们通过 healPhase 函数来实现治疗阶段，这个函数会在 fightPhase 函数的一开始被调用。技能 20“反击”是与攻击成对出现的判定，于是在 fightPhase 函数里 cardAttack 被调用后会加入相关的判定流程。技能 22“警戒”是近战士兵特有的一个技能，拥有“警戒”的近战士兵的搜索敌人的方法会完全改变，于是我们通过 searchForGuard 函数实现了这个技能。

6.2.3 游戏类 SGDuelGame 的设计与实现

该类存在于文件 SGDuelGame.go 中，具体的内容如下：

```
type SGDuelGame struct {
    battleField SGBattleField
    turn        int
}
```

还有一个表示房间的 struct，如下：

```
type SGRoom struct {
    Red, Blue *SGPlayer
}
```

该文件中只有一个函数，如下：

```
func NewGame(red *SGPlayer, blue *SGPlayer, redCon net.Conn, blueCon net.Conn, roomNum int) (net.Conn, net.Conn, bool)
```

该函数是被外部调用的接口，参数只要给出 2 个对战的玩家类，2 个玩家对应的连接以及对战房间的编号，该函数就可以自动生成一局游戏。在实际项目中，每一局游戏都会开启一个 goroutine 来运行。

该函数内部实现过程也是非常明确的。首先会连接数据库，将玩家信息进行赋值，然后玩家的卡牌信息也要进行赋值。还会随机生成玩家在该局游戏中的 ID，该做法的目的是为了保证玩家信息的安全，随机的 ID 在一局游戏结束后也会消失，防止了外部可能的得知 ID 来操作游戏的可能性。该函数中，随机 ID 的生成方法如下：

```
redID := roomNum*1000 + r.Intn(1000)
blueID := roomNum*1000 + redID + r.Intn(100) + 1
```

即是一方 ID 是房间号码乘以 1000 再加上 1000 内的随机数，另一方 ID 是房间号码乘以 1000 再加上上一个 ID 再加 100 内的随机石最后再加 1，这样也避免了房间内 2 名玩家 ID 相同的情况。

在一切数值都赋值完毕后，该函数就会调用 SGBattleField 里的函数来进行游戏。游戏运行过程中也有一些调试输出来方便测试，如下图：

```
Game Message : Turn 1
Game Message : Player tony Action, Hp:20
Game Message : Turn 2
Game Message : Player lizimian Action, Hp:20
Testing Message: Place Card Information:{"CardIndex":0,"Position":108}
Testing Message: Card index 0, place to (0,8)
Testing Message: Card (0,8) move to (0,5)
Game Message : Turn 3
Game Message : Player tony Action, Hp:20
Testing Message: Place Card Information:{"CardIndex":3,"Position":101}
Testing Message: Card index 3, place to (0,1)
Testing Message: Place Card Information:{"CardIndex":2,"Position":201}
Testing Message: Card index 2, place to (1,1)
Testing Message: Place Card Information:{"CardIndex":1,"Position":200}
Testing Message: Card index 1, place to (1,0)
Testing Message: Place Card Information:{"CardIndex":0,"Position":100}
Testing Message: Card index 0, place to (0,0)
Testing Message: Card (0,1) move to (0,4)
Testing Message: Card (0,4) attack to (0,5)
Testing Message: Card (0,5) removed
Testing Message: Card (0,0) move to (0,2)
Testing Message: Card (1,1) move to (1,3)
Testing Message: Card (1,0) move to (1,2)
```

图 6-3 卡牌游戏调试输出

6.3 本章小结

在本章节中，我们主要对游戏逻辑子系统的所有细节实现进行了较为详细的描述。游戏管理子系统是对游戏策划方案中指定的游戏规则的具体实现。在描述的过程中，我们通过将代码设计与实现的分析和对策划方案的游戏规则进行相应的说明和对比，较为清楚了讲述了该系统的编写细节与编写思路。下面的章节中，我们会对针对该项目的系统测试进行详细的说明。

第七章 系统测试

7.1 测试概述

该项目是一个网络游戏项目，整个项目是由客户端与服务端联合实现的。网络游戏的测试方法和一般软件会有很大的区别的，实际测试流程会是十分漫长而复杂的。网络游戏初始的测试阶段一般是功能测试，因为游戏软件的特殊性，它的功能点的划分很难做到特别明确，特别是游戏项目的规模特别庞大的时候。一般功能测试模块的划分会有游戏资源模块、游戏 AI 逻辑模块、玩家互动模块等部分，实际的划分还会根据具体项目来确定。功能测试会随着项目的开发与版本升级来持续进行。每次版本更新后还会进行回归测试。功能测试之后还会有性能测试，性能测试一般是针对服务端承载能力、运行效率的测试，性能测试一般是大规模测试，会通过实机运行等手段来进行测试。

对于本具体项目，我们只进行了功能测试。

7.2 功能测试

测试环境：

客户端：运行设备为 iphone 5，设备操作系统为 IOS 6.1.3；

服务端：运行设备为阿里云远程服务器，设备操作系统为 Linux。

功能模块：

针对功能测试，我们首先对该项目的功能模块进行了划分，分为用户登录模块、游戏模块。

用户登录模块测试：

针对该模块设计了几个测试用例，如下：

(a) 用户名和密码为空。测试结果为无法成功登陆，客户端判定登陆错误。

(b) 用户名和密码超过最大长度限制。测试结果为无法成功登陆，客户端判定登陆错误。

(c) 采用数据库里存在的用户名和密码登陆。测试结果为成功登陆。

(d) 采用数据库里不存在的用户名和密码登陆，测试结果为无法成功登陆，服务端判定登陆错误。

游戏模块测试：

整个游戏模块的关键点较多，测试用例的根据相关流程进行了设计，如下：

(a) 针对手牌上限限制测试。玩家摸牌到手牌上限，然后等待到下一回合的摸牌阶段。测试结果为手牌到达上限后，摸牌阶段不会分配新的手牌。

(b) 针对手牌牌源测试。玩家摸牌直至套牌被完全摸空，然后等待到下一回合的摸牌阶段。测试结果为套牌摸空后，摸牌阶段不会分配新的手牌。

(c) 针对手牌等待回合数更新测试。玩家新摸一张牌，等到下一回合，观察该卡牌的等待回合数数。测试结果为该卡牌的等待回合数减 1.

(d) 针对手牌等待回合数惩罚测试。玩家等待一张手牌的等待回合数为 0，不使用它，等到下一回合，观察该卡牌的等待回合数的变化。测试结果为该卡牌的等待回合数变为最大等待回合数的一半取下整。

(e) 针对游戏胜利条件测试。玩家等待自己被攻击至生命值为 0，观察结果。测试结果为该玩家游戏失败，对手玩家游戏胜利。

(f) 针对近战士兵逻辑测试。玩家将一张近战士兵卡牌放置在距离对手一张卡牌 2 格以上距离的位置，观察卡牌活动的次序。测试结果为近战士兵卡牌先移动至对手卡牌

的面前一格，然后攻击，若距离对手卡牌的距离超过该卡牌的最大移动距离，则移动至可移动的最远处，然后如果对手卡牌在攻击范围之内则进行攻击，若在攻击范围之外则结束行动。

(g) 针对远程士兵逻辑测试。玩家将一张远程士兵卡牌放置在距离对手一张卡牌 2 格以上距离的位置，观察卡牌活动的次序。测试结果为如果对手卡牌在攻击范围内，则远程士兵攻击对手卡牌，然后结束行动，如果对手卡牌在攻击范围外，则远程士兵移动到可移动的最大距离，然后结束行动。

(h) 针对技能“飞行”测试。玩家将一张具有“飞行”技能的士兵卡牌放置在一张对手卡牌的面前，观察卡牌活动的次序。测试结果为如果对手卡牌不具备“飞行“技能，则飞行士兵直接穿过对手卡牌移动到课移动的最大距离，如果对手卡牌具备”飞行“技能，则活动模式与近战士兵相同。

(i) 针对技能“治疗“测试。玩家放置一张具有”治疗“技能的士兵卡牌，观察卡牌活动的次序。测试结果为己方场上生命值减少最多的卡牌会恢复技能说明的生命值，如果有多个生命值减少一样的卡牌，则按照从上到下，从左到右的搜索顺序，选择第一个。

(j) 针对技能“反击“测试。玩家放置一张近战士兵卡牌在对手具有”反击“技能的卡牌面前，观察卡牌活动的次序。测试结果为在己方卡牌进行攻击之后，对手具有”反击“技能的卡牌也会进行一次攻击。

(k) 针对技能“警戒“测试。玩家放置一张具有”警戒“技能的士兵卡牌在对手卡牌的背后，观察卡牌活动的次序。测试结果为具有”警戒“技能的士兵会攻击身后的对手卡牌。

总体测试结果：

按照设计的测试用例进行测试后，服务端的测试结果完全符合预期的正常结果。对现有的功能，本系统都得以正确的实现。

7.3 本章小结

在本章节中，我们对本项目的测试的设计、实现以及结果进行了详细的描述。本次的测试只有功能测试，由测试的结果，我们可以看出本项目在功能实现方面基本符合预期的要求。下面的章节中，我们会对整个项目进行总结与展望。

第八章 总结与展望

本次毕业设计项目的题目是由公司提供的，是一个网络卡牌游戏的初始策划阶段的产物。在该策划案中，本游戏基本的规则与功能框架已基本描述清楚。我们的项目算是对该方案实现的一个 demo 版本。本次项目的实现选择了 Go 语言，虽然 Go 语言作为一个新语言，还有很多未解决的问题，但是它的定位是准确的，它对网络相关的编程确实提供了很多的便利。本项目的编写过程也是一个学习的过程，首先选择了 Go 语言，就需要对 Go 语言先进行相关的熟悉与编写练习，在逐渐熟悉了 Go 语言的语法之后，对于服务端编写非常重要的服务端架构的设计和通讯协议的设计，也需要阅读很多相关方面的文章，而这些也都是在完成本次毕业设计项目的过程中的重要收获。

本次项目的实际完成虽然从大体上符合了最初的设计要求，但是依然存在不少缺陷和需要改进的地方。比如对系统线程的设计需要进一步的优化，因为多线程的存在同时也带来了很多隐患，所以现在的很多服务端设计都会尽量减少线程的数量，将很多操作优化成顺序执行的。再比如服务端运行效率的问题在本次设计中没有被重点考虑，而实际代码中也许会存在很多影响运行效率的地方。

因为该项目在公司的安排下可能会作为正式项目发布，所以该项目的开发还会继续下去。在未来的版本中，我们会对之前不合适的设计与实现进行改正与优化，并且根据后面的策划方案实现更多新的功能。

参考文献

[1] Google. The Go Programming Language FAQ[EB/OL]. http://golang.org/doc/faq

[2] Jason Kincaid. Google's Go: A New Programming Language That's Python Meets C++[EB/OL]. http://techcrunch.com/2009/11/10/google-go-language/

[3] chaishus...@gmail.com. GettingStarted Go 语言简介[EB/OL]. Golang-china, 2010, http://code.google.com/p/golang-china/wiki/GettingStarted

[4] 51CTO. Google Go！融合 Python 速度与 C 性能的新语言[J/OL]. 51CTO, 2009-11-11, http://developer.51cto.com/art/200911/162227.htm

[5] 51CTO. Google Go：新兴语言的代表[J/OL]. 51CTO,2011-1-5, http://doc.chinaunix.net/web/201101/1188349.shtml

[6] Miek Gieben. Learning Go[EB/OL]. http://miek.nl/cgi-bin/gitweb.cgi?p=gobook.git;a=summary

[7] 邱伟. 趣味盎然的万智牌[J]. 新体育,1999(2).

[8] 芝麻大盗. 关于 goroutine 的调度[EB/OL]. Douban, 2012-12-06. http://www.douban.com/note/251142022/

[9] Davis Robin S. Who’s Sitting on Your Nest Egg?[M]. 7-8.

[10] JSON.org. Introducing JSON[EB/OL]. http://www.json.org/

谢辞

感谢肖凯老师对本课题的耐心指导与审核！

感谢品志文化公司和公司的同事对于本课题方案的提供，课题资源的制作以及课题项目的管理！

感谢共同在公司做项目的同学的支持与鼓励！

DESIGN AND REALIZATION ABOUT SERVER OF ONLINE GAME ON IOS

IOS(iPhone Operating System) is one of the most popular mobile OS in the world. According to some official statistics, game software holds the largest market share in the apple store. So it is a good opportunity to develop a kind of game software. But the next question is what kind of game is the most suitable to attract the eyes of IOS users. After some time to compare and analyze, we choose to develop a card game or exactly trading card game.

When we talk about the trading card game, I think many people know some kinds of these product like MTG(Magic The Gathering), Yu-Gi-Oh etc. The features of trading card game are collecting many cards and competing with other players. If people play trading card game on the mobile phone, they will find it's so convenient because they can play with other player at anytime and anywhere.

We choose the new programming language Go to develop the server of this game. Go is an open source programming language. It is designed and developed by Google. The features of this language are type-safe, memory-safe, intuitive concurrency, efficient garbage collection, and high-speed compilation. The net library of Go is powerful and useful, so it is so easy to build a game server by using the net library of Go. Goroutine and interface are also very important functions that Go provides. Goroutines are small lightweight threads that Go provides. We can use the go statement to create a goroutine when we want to allocate a thread to run a function. Goroutines are executed concurrently with other goroutines, including their caller. They do not necessarily run in separate threads, but a group of goroutines can be multiplexed onto multiple threads, allowing parallel execution. Execution control is moved between them by blocking them when sending or receiving messages over channels. Channels are bounded buffers, not network connections.Goroutines can share data with other goroutines. Race conditions can occur in concurrent Go programs. Concurrent Go programs are not memory safe. Unlike Java, the interfaces which a type supports do not need to be specified at the point at which the type is defined, and Go interfaces do not participate in a type hierarchy. A Go interface is best described as a set of methods, each identified by a name and signature. A type is considered to implement an interface if all the required methods have been defined for that type. An interface can be declared to "embed" other interfaces, meaning the declared interface includes the methods defined in the other interfaces.

To complete this project, at first we should make a requirements analysis. We can divide this whole server into two subsystem, user management subsystem and game logic subsystem. The user subsystem is in charge of some users' operating like registering, login, logout and card management and game matching. Then I draw some use cases pictures to show the design. I will explain two use case specification in detail. The first one is card management. The mainly operations of card management are adding cards and deleting cards. The process is like this: a client passed a message to server to add or delete any card; if adding a card, system should delete

the cost before adding the card; if deleting a card, system should search this card in the player's bag, then delete it after finding it; at last, server will send a message to client to tell the result. The second one is game matching. The mainly process is like this: a client sends a message to server to start the game matching; the system will add the player to the matching queue after judging the status of this player; then the system will choose two players in the queue to start a game; at last, server will send a message to client to tell the result.

The other subsystem is game logic subsystem. Game logic subsystem is responsible for the process of the whole game, which is responsible in accordance with the rules of the game given to deal with the results of running the game, the game will be through the process of sending the message to the client to display. Then I will describe the simple rules of this card game to under what this subsystem does. Rule one: every player has health point and a deck which is composed by soldier card and skill card. Rule two: both players play in turn. Rule three: when health point of one player turns into zero, he loses and his opponent wins. Rule four: one turn has four phase, called draw phase, place phase, fight phase, end phase. In the draw phase, every player should draw one card as hand card, but at the starting turn, every player should draw two cards. In the place phase, every player should place the soldier one the battle field or use skill card. In the fight phase, all cards on the battle field will move and attack enemies. In the end phase, hand cards of every player will update their status, and then check the number of hand cards.

After the requirements analysis, I should do architecture design and database design. In the part, I drew some class diagrams and logic data model diagram. At first, let's talk about the architecture design. For user management subsystem, we have four entity classes name as Player, Cards, Game and Message. Class Player records the data of player, Class Cards records the data of cards, and Class Game can use the interface of game logic subsystem. Class Message just records all kind of messages. And there are two boundary classes name as MessageRead and MessageSend. They read or send the messages of client. The last class is the control class MessageHandle. It will run the right function which message calls. And for game logic subsystem, there are three entity classes name as Player, Battle and Message. Class Player also records the data of player. Class Message also records all kind of messages. Class Battle records the status of the battle field in game. There are two boundary classes name as MessageRead and MessageSend. They also read or send the messages of client. The last control class is GameHandle which can use all interface of battle operating. I also drew the process diagram of this system. The whole system has four threads, they are MessageHandle, ConnectionHandle, MatchHandle and Game. MessageHandle read and send messages from and to Clients. ConnectionHandle receives the connections and listen them. MatchHandle manages the game matching. Game manages the game playing. The last topic is database design. There are two entity tables and one relationship table. The entity tables are Player and Card. As we can see, Player table stores the information of players, and Card table stores the information of card. The relationship table PlayerOwnCard represents a many to many relationship.

After all the designs, we can start to program using Go. To build a database, I choose to use MySQL, and to use MySQL in Go, I download an open-source drive of MySQL from Google Code. The net library of Go is really useful to me to build a server. We can use the interface that library provides to build a socket listening port. And we can let the program running on a goroutine. The file structure of Go code is very simple. We just create one package named SGGame, and there are a file out of the package and eight files in the package. In the file that out

of the package named SGGameServer.go, there is the whole structure of this project. It used the interface of SGGame. And we implement the interface in the files in package SGGame.

At last we test all the system by functional test. Game testing is not like other software testing especially online game testing. Online game testing methods are very different from general software, and the actual testing process will be very long and complex. Online games generally will do functional testing first. Because of the particularity of game software, it is difficult to devide function points clearly, especially game project is particularly large. General function test module will be game resources module, game logic module, player interaction module and other parts, the actual division will be determined according to the specific project. Each version after updating will do regression testing. After the functional test there will be performance testing. performance testing is generally for the carrying capacity of server, efficiency testing.And performance testing is generally large-scale testing running by real machines.

This project topic is provided by the company which is a online card game in the initial planning. In the planning document, the basic rules of the game and functional framework has been described clearly. Our project is just a demo version. We choose Go to implement this project. Although Go is a new language, and there are many unresolved issues, but its positioning is accurate, and it provides a lot of convenience to do network-related programming. I learn a lot by pragramming with Go. When selecting Go, i need to read many materials and practice a lot. After being familiar with the syntax of Go,I need to learning more about architecture design and communication design for server. When finishing this project, i found that i learning many new knowledge and technology. At last, thanks to everyone who helped me.