****



**本 科 毕 业 设 计**

院 系 软件学院

专 业 软件工程

题 目 火影手游同步服务设计与实现

年 级 2018级 学 号 141250115

学生姓名 孙 浩

指导教师 张瑾玉 职 称 讲师

提交日期

**南京大学本科生毕业论文（设计）中文摘要**

毕业论文题目： 火影手游同步服务设计与实现

软件学院 院系 软件工程 专业 2014 级本科生姓名： 孙浩

指导教师（姓名、职称）： 张瑾玉 讲师

摘要：

\*\*项目背景(为什么要做这个项目)。

\*\*技术简介(项目开发中使用了哪些技术，有何特点)。

\*\*项目组整体完成了项目中的哪些功能。

\*\*本人在项目中承担了哪些工作。

关键词：

**南京大学本科生毕业论文（设计）英文摘要**

THESIS：

DEPARTMENT：

SPECIALIZATION:

UNDERGRADUATE:

MENTOR:

ABSTRACT：

KEY WORDS: \*\*项目，\*\*技术(关键名词)1，\*\*技术(关键名词)2，…..

**目 录**

图目录 II

表目录 III

第一章 引言 1

1.1 项目背景 1

1.2 国内(外)工作流建模技术研究现状 2

1.X可补充二级标题 2

1.X.1 可补充三级标题 2

1.X.2 可补充三级标题 2

1.Z 论文的主要工作和组织结构 2

第二章 \*\*技术概述 4

2.1 \*\*技术 4

2.1.1 \*\* 4

2.1.2 \*\* 4

2.2 \*\*技术 5

2.X \*\*本章小结 5

第三章 \*\*系统需求分析与概要设计 6

3.1 \*\*项目整体概述 6

3.1.1 \*\*可补充三级标题 6

3.1.2 \*\*可补充三级标题 6

3.2 \*\*系统的需求分析 6

3.2.1 \*\*可补充三级标题 6

3.2.2 \*\*可补充三级标题 6

3.3 \*\*系统的概要设计 6

3.3.1 \*\*可补充三级标题 6

3.3.2 \*\*可补充三级标题 6

3.X \*\*本章小结 7

第四章 \*\*项目\*\*模块的详细设计与实现 8

4.1 \*\*模块概述 8

4.2\*\*模块的详细设计 8

4.2.1 \*\*可补充三级标题 8

4.2.2 \*\*可补充三级标题 8

4.3 \*\*模块的实现 8

4.3.1 \*\*可补充三级标题 8

4.3.2 \*\*可补充三级标题 9

4.X \*\*本章小结 9

第五章 总结与展望 10

5.1 总结 10

5.2 展望 10

参考文献 11

致谢 12

# 图目录

[图2.1 \*\*图 4](file:///C:\Users\st0001\Desktop\bishe2018\孙浩毕设.doc#_Toc323067900)

[图2.2 \*\*图 4](file:///C:\Users\st0001\Desktop\bishe2018\孙浩毕设.doc#_Toc323067901)

[图4.1 MyWorkController类代码 9](file:///C:\Users\st0001\Desktop\bishe2018\孙浩毕设.doc#_Toc323067902)

# 

# 表目录

[表2.1 \*\*表 5](file:///C:\Users\st0001\Desktop\bishe2018\孙浩毕设.doc#_Toc323067881)

# 第一章 引言

## 1.1 项目背景

火影手游是腾讯魔方工作室旗下的一款横板格斗类手游。它在2016年1月5日开启不删档测试，取得了非常大的成功，曾获App Store畅销榜头部推荐。迄今为止，已累计注册超过3700w的玩家，日活量超过200w，最高同时在线42w玩家。游戏中包括大量玩法，例如pvp、角斗场、忍者赛事、本服/跨服组织战和组队副本战等。每赛季会举行线下赛，有超过300w的直播观看人气。

火影手游是一个典型的移动网络多人在线游戏，也面临着客户端之间的同步问题。同步是影响玩家的游戏体验的重要因素，因此如何在移动网络环境下设计出比较好的同步方案，是一个至关重要的问题。本文将会阐述火影手游的后端同步服务实现。

在游戏中，一个对局或者一个场景都是一个虚拟世界。多人网络游戏往往需要玩家之间进行行为的交互，这就需要玩家直接的客户端之间保持一致性。否则玩家之间的交互体验将会不够逼真。比如在一个FPS游戏中，大家的延迟都很高，A、B两个玩家同时发现了对方，并向对方射击，如果没有很好的同步机制，那么A的屏幕上显示B还没有开枪就被击杀，而B的屏幕上显示A还没有开枪就被击杀，这就出现了不一致的问题。

目前主流的游戏同步方式分为帧同步和状态同步。火影手游后台采用帧同步作为同步方案。本文将会介绍帧同步和状态同步的具体实现，以及它们的优缺点，以及火影手游采用的帧同步所做的改进和具体实现。

## 1.2 工业界主流同步方案

目前主流的网络游戏同步方案分为状态同步和帧同步。用户的操作在客户端和服务器角度来看是控制指令，这些指令将会能够改变虚拟世界的状态。如果这些控制指令被发送到服务器，由服务器根据指令计算虚拟世界的状态，并将这个状态广播给所有关联的客户端，就叫做状态同步。而帧同步则不会同步虚拟世界的状态，而是同步客户端之间的指令，由客户端自己计算虚拟世界的状态。

### **1.2.1 帧锁定(LockStep)同步**

一般来说，帧锁定同步由客户端之间广播控制指令。如果一场对局中有N个玩家，如果每个客户端产生一个控制指令，那么网络上将会有(N-1)2个控制指令。因此帧锁定同步算法要求网络上不应该有超过10个玩家。客户端之间往往是直接同步指令的，并且客户端根据收到的指令计算状态，往往并不会有预表现，所以该算法也要求网络平均延迟(RTT)在100ms以内。所有客户端在任意时刻逻辑都是统一的，缺点是一个人卡机，所有人等待。

帧锁定同步算法将游戏的状态划分为帧。这里的帧不是指图形学概念中的显示帧，而是状态的一个切片，是逻辑帧。后续的算法介绍将会用到逻辑帧的概念。

该算法的通用流程如下：

1. 客户端每隔固定的帧数或时间上传控制操作，例如每过5帧或者每30ms
2. 客户端广播给所有其他客户端或者由服务器统一发送给所有客户端
3. 客户端在收到控制信息后计算状态更新游戏
4. 如果客户端在运行到下个关键帧的时候没有收到服务器发来的控制信息，则等待，否则进行游戏，并收集所有玩家的输入
5. 重复1)

下图给出了客户端和服务器端的帧锁定同步算法顺序图。

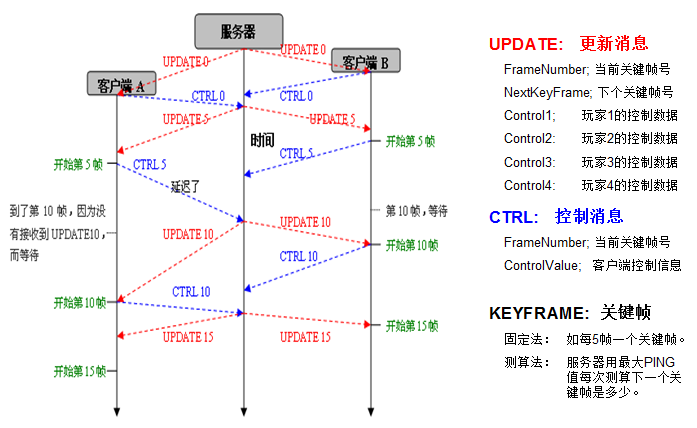


图 1.2.1 帧锁定同步算法顺序图

客户端的具体算法流程如下：

1. 判断当前的逻辑帧是否是关键帧K1，如果不是，则跳转到7)
2. 查看关键帧中是否包含应用于K1的控制指令，如果没有的话，则重复等待
3. 采集所有输入，封装为控制指令，并附带上K1的编号，发送给服务器
4. 从K1的控制指令中获得下个关键帧的编号，并获取下个关键帧编号之前的所有控制指令
5. 运用控制指令计算K1到K2
6. 令K1=K2
7. 执行该逻辑帧的逻辑
8. 重复1)

服务器端算法逻辑如下：

1. 等待所有客户端的当前关键帧K1的控制指令，在收集到所有客户端的手机之前，将一直阻塞
2. 根据所有控制指令计算下个关键帧的更新，并给出下个关键帧的编号
3. 将更新广播给所有客户端
4. 令K1=K2
5. 跳转到1)

可以看到，帧锁定同步的锁定二字体现在，服务器端将会锁定逻辑帧的推进，直到所有客户端的关键帧信息都被发送到服务器端。这是为了保证所有客户端在任意时刻完全一致而作的限制。显然在移动网络，高延迟和高丢包率的情况下，这样的锁定逻辑会严重破坏玩家的游戏体验。

### **1.2.2 状态同步**

状态同步的算法逻辑就要简单的多。首先，状态同步不需要对游戏状态进行分片，也就是不需要帧的概念。所有玩家一旦产生控制指令，就将会发送给服务器。服务器端往往有完整的游戏逻辑，在接收到客户端的控制指令后，立刻或者稍后按照顺序根据指令计算游戏状态。



图1.2.2 状态同步算法顺序图

一般来说，状态同步的算法流程如下：

**客户端:**

1. 接受到玩家输入的操作，转化为控制指令，对本玩家控制的角色进行预表现
2. 对于其他玩家控制的角色，根据以往的动作路径进行预表现
3. 接收到服务器发送来的状态，如果某个角色和服务器状态差距明显，则将该角色立刻拉扯到与服务器状态一致的状态
4. 重复1

**服务器端：**

1. 每隔33ms将当前状态广播给所有玩家
2. 接收到某个客户端的控制指令，将控制指令push到缓存队列中
3. 不断从缓存队列中pop指令并计算状态

1.2.3 状态同步和帧同步比较

下表对状态同步和帧同步的优缺点进行了讨论。

|  |  |  |
| --- | --- | --- |
|  | 状态同步 | 帧同步 |
| 同步数据量 | 同步量大，因为服务器需要发送的是整个虚拟世界的状态。即使根据视野和关联性做了剔除，还是有比较大的数据量。 | 同步量比较小。因为客户端之间只需要交换控制指令。 |
| 同步程度 | 状态同步在客户端发送指令到获得服务器段返回的状态往往需要经过服务器端的计算。因此状态同步往往需要通过预表现来解决各个客户端之间的显示不一致问题。 | 帧同步只需要同步指令，因此客户端的变化能够快速在其他客户端上得到体现。 |
| 开发难度 | 状态同步需要服务器端有完整的游戏逻辑，服务器端开发难度比较大；客户端需要有预表现逻辑，还需要根据服务器端的状态调整现有状态，非常复杂。总体而言开发难度比较大。 | 显而易见，客户端和服务器端是弱耦合的。客户端只需要开发根据指令更新状态的逻辑，不用区分自己和别人的指令；服务器端最少可以只开发转发指令的逻辑，因此开发非常简单。 |
| 调试难度 | 状态同步调试非常简单，客户端之间也不会出现不同步的问题。 | 帧同步调试很困难。如果出现不同步的问题非常难定位原因。比如别人显示我已经死亡，但是我这里血量还剩1%，这显然是不可接受的。可是导致这种不同步的根源很难被发现，只能通过log模糊定位。 |
| 断线重连方案 | 状态同步非常简单，只需要将断线重连的这一刻状态同步至客户端，客户端进行表现即可。 | 帧同步需要客户端将重连之后的所有指令都重新运行一遍。典型的如王者荣耀，断线重连时需要很长一段时间读条。 |
| 反作弊 | Fps和moba游戏往往有透视和开图的作弊问题，也就是能够知到本不该知道的信息，这种方式通过剔除能够很好地解决，状态同步可以选择同步客户端应该知道的状态。  另外，由于服务器有完整的游戏逻辑，因此服务器可以校验玩家输入的合法性。 | 由于所有游戏逻辑运行在客户端，因此无法绝对根绝透视和开图挂，包括其他很多类型的挂。通过客户端之间的校验可以发现不同步，也可以发现作弊行为。但是如果参与的玩家只有两个，或者多个客户端同时作弊，那么校验的成效就很小。  一个解决方案是在服务器端运行一个gamecore，类似状态同步的服务器端游戏逻辑。这需要前后端都有非常好的设计。 |

表1.2.1 帧同步与状态同步优缺点比较

帧同步主要在同步数据量、同步程度、和开发难度上有比较明显的优势。在移动网络下用户的流量都比较宝贵。这也是火影手游选择帧同步作为同步方案的重要理由之一。

## 1.3 论文的主要工作和组织结构

本文将会主要介绍火影手游服务器端的同步模块的设计与实现。首先本文会对整个火影手游后端的架构和设计思路进行概述。同时，会对如何保证如此大量的日活带来的连接和流量压力，进行架构层面的介绍和讨论。本文的核心在于同步模块的设计和实现，因此本文会对同步模块做详尽的介绍。游戏的同步也涉及到很多其他特性，例如鉴权和断线重连等，因此，本文会对有所涉及的模块的设计和实现都作简要的阐述。

第一章：概述和前言部分，主要介绍了项目背景，当前工业界主流的同步方案和论文的内容概要。

第二章：项目技术概述，主要介绍了项目中运用的同步算法和一些工具及技术。

第三章：功能介绍，会对火影手游的玩法做简要介绍，并对服务器端的职能做阐述。

第四章：概要设计，主要会对火影手游的服务器端从架构层面和设计思想层面做描述。

第五章：详细设计，主要会对同步、断线重连、鉴权、场景的相关模块的详细设计进行描述。

第六章：总结与展望，总结当前的同步方案的优缺点，对将来尝试要做的改进进行讨论。

# 

# 第二章技术概述

## 2.1 Libco协程框架

协程类似于线程，是一个为了应对多任务情况而产生的概念。每个协程有自己的栈空间。协程并不会创建一个新线程，如果需要切换到一个新的协程，当前的上下文将会被暂存，当新协程返回后，将能够继续回到离开的地方，并切换回暂存的上下文继续运行。这带来的好处，是可以用同步的写法来达到异步的性能。例如如果需要等待一个耗时的网络操作，可以开辟一个新的协程去做，在新的协程返回时，再返回到上下文中，代码的组织方式将是同步的，但是在等待期间，cpu并不会像传统同步编程一样不断轮询被占满。

Libco是腾讯开源的一款协程框架，使用汇编实现保存linux系统中x86或amd64指令集下的重要寄存器现场，仅有的几个函数接口 co\_create/co\_resume/co\_yield 再配合 co\_poll， 可以支持同步或者异步的写法，如线程库一样轻松，库里面提供了socket族函数的hook。Libco提供以下特性：

1. 仿照pthread风格的协程接口定义
2. 类似于nodejs的事件循环，并有超时机制
3. 基于libc源码的高性能swapcontext的汇编实现
4. 提供了socket函数的coroutine hook，阻塞的socket调用会创建新的协程转换为异步调用



图2.1.1 简单的协程使用顺序图

## 2.2 Tbus通信管线

Tbus是腾讯互娱使用的进程间通信管线。它提供了类似于ip的寻址方式，使得进程之间能够以数据包的方式进行通信。Tbus是基于共享内存的进程间通信方式。它会将消息保存在一个能够通过源地址和目标地址唯一确定的共享内存中的队列中。这保证了即使发送方或者接收方的进程崩溃，消息也不会丢失。Tbus提供了可靠的tcp通信，因此非常适合作为进程间通信的方式。

、

图2.2.1 tbus本机通信示意图

Tbus sdk提供了类似于socket的发送和接受接口。火影手游的进程间通信就是由tbus实现的。这也决定了进城之间的通信方式是主动发送数据包和接受数据包，这也带来了一定的局限性，后面会有阐述。



图2.2.2 tbus网络通信示意图

## 2.3 自定义可靠udp通信协议

前面我们提到，移动网络的显著特征是不稳定性。不稳定性体现在延迟波动比较大，丢包率明显高于有线网络。

## 2.X \*\*本章小结

# 第三章 \*\*系统需求分析与概要设计

## 3.1 \*\*项目整体概述

### **3.1.1 \*\*可补充三级标题**

### **3.1.2 \*\*可补充三级标题**

## 3.2 \*\*系统的需求分析

### **3.2.1 \*\*可补充三级标题**

### **3.2.2 \*\*可补充三级标题**

## 3.3 \*\*系统的概要设计

### **3.3.1 \*\*可补充三级标题**

### **3.3.2 \*\*可补充三级标题**

## 3.X \*\*本章小结

# 第四章 \*\*项目\*\*模块的详细设计与实现

## 4.1 \*\*模块概述

## 4.2\*\*模块的详细设计

### **4.2.1 \*\*可补充三级标题**

### **4.2.2 \*\*可补充三级标题**

## 4.3 \*\*模块的实现

### **4.3.1 \*\*可补充三级标题**

在实现部分，可以有少量关键性的代码，代码的排版形式，代码贴入表格框(1行\*1列)，设置为可以跨页的形式，且代码中的字体使用“Arial/五号/单倍行距”，对该表格框标记为图x.y，并给出以该段代码的用途给出图名。示例如下：

|  |
| --- |
| public String signAndUnsignList() throws Exception {  String resultStr = SIGN\_UNSIGN\_LIST\_RESULT;  Loginer loginer = this.getLoginer();  Page page = null;  CriteriaSetup criStp = new CriteriaSetup();  try {  //设置界面不分页  this.ecLimit = ExtremeTablePage.getLimit(this.getHttpServletRequest(), 999999);  page = this.myWorkextSvc.FindTasksOfActorId(loginer, new Long(ProcessVariable.JBPM\_FLOW\_TASK\_ONLINE),  this.ecLimit, criStp);  } catch (Exception ex) {  Log.debug(ex);  throw new SysException(SysExcepType.BUG\_UNKNOWN\_RUNTIME\_EXCEPTION, ex);}  List list = page.getResult();  //排序  Comparator comp = new Comparator(){  public int compare(Object o1,Object o2) {  TaskBillVO v1=(TaskBillVO)o1;  TaskBillVO v2=(TaskBillVO)o2;  if((v1.getDuration()==null || v1.getDuration().equals("")  || v1.getDuration().equals("null")) && (v2.getDuration()!=null  && !v2.getDuration().equals("") && !v2.getDuration().equals("null")))  return 1;  else if((v2.getDuration()==null || v2.getDuration().equals("")  || v2.getDuration().equals("null")) && (v1.getDuration()!=null  && !v1.getDuration().equals("") && !v1.getDuration().equals("null")))  return 0;  else  return 1;  }  };  Collections.sort(list,comp);  this.setTaskbillVOs(list);  getCaseNum(list);  this.getHttpServletRequest().setAttribute("totalRows", page.getTotalCount());  return resultStr;} |

图4.1 MyWorkController类代码

### **4.3.2 \*\*可补充三级标题**

## 4.X \*\*本章小结

# 第五章 总结与展望

## 5.1 总结

## 5.2 展望

# 参考文献

[1] 作者,译者,书名(版本),出版地:出版社,出版时间,引用部分起止页.

[2] 作者,译者,文章题目,期刊名,年份,卷号(期数):引用部分起止页.

[3] 作者,学位论文名,本科/硕士/博士论文,大学/机构名,年份.

[4] 网页的主题,URL.

# 致谢

感谢指导老师……

感谢参与本项目的同组的其他同学……

感谢其他……