國立交通大學

資訊科學系

碩 士 論 文

MMOG 中介軟體之儲存元件與服務

A Research of Persistence Component on MMOG Middleware

研 究 生：葉倫武

指導教授：袁賢銘　教授

中華民國 九十四 年六月

MMOG 中介軟體之儲存元件與服務
A Research of Persistence Component on MMOG Middleware

研 究 生：葉倫武　　Student：Lun-Wu Yeh

指導教授：袁賢銘　　Advisor：Shyan-Ming Yuan

國 立 交 通 大 學
資 訊 科 學系
碩 士 論 文

A Thesis

Submitted to Institute of Computer and Information Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

MMOG 中介軟體之儲存元件與服務

學生：葉倫武　　　　　　　指導教授：袁賢銘

國立交通大學資訊科學系

摘要

MMOG (Massively Multiplayer Online Game)是一種提供多人同時在一個虛擬遊戲世界互動的遊戲類型。但是開發一個 MMOG 的難度是非常高的，開發者會遭遇到問題，例如高效能的網路、分散式的技術、延展性、容錯機制、負載平衡…等，而這些問題大多是一般遊戲所不會遭遇到的問題，所以開發一個 MMOG 是非常耗費時間和人力的。DOIT (Distributed Organized Information Terra)是一套中介軟體平台，來讓開發者能夠減少開發時所遇到的問題與負擔。然而 DOIT 並沒有提供資料儲存方面的元件與服務，所以開發者必須自行處理各式各樣的 MMOG 物件與資料傳輸，以及在伺服器主機和資料庫之間的交易問題。

而我們提出的這套儲存元件與服務可以減少系統開發時的負擔，並提高系統效能，它包含了一些基本且常見的 MMOG 物件型態，讓開發者可以方便的來開發與使用，而 in-memory 方面的 operation，可以提升系統效能，而此系統會管理這些物件，並處理其資料同步和交易方面的問題，來讓開發者可以更快速、更方便的來開發一套 MMOG。

A Research of Persistence Component on MMOG Middleware

Student: Lun-Wu Yeh Advisor: Shyan-Ming Yuan

Department of Computer and Information Science

National Chiao Tung University

Abstract

MMOG (Massively Multiplayer Online Game) is a type of computer game that enables hundreds or thousands of players to simultaneously interact in a game world via the Internet. But developing a MMOG is very hard. Developer will encounter many problems, such as high-performance network, distributed technology, extensibility, fault tolerance, load balancing…etc. Most of these issues are not existed in traditional single-player games. Therefore, development a MMOG will consume a lot of time and plenty efforts. DOIT (Distributed Organized Information Terra) is the middleware platform to reduce development effort. However, DOIT does not provide data persistence mechanism. Developers have to handle varied MMOG objects, data transport between servers and database transaction.

Persistence components can reduce system development effort and enhanced system performance. It includes MMOG objects for designing and implementing a game. There are in-memory operation to enhance system performance and manager to handle data synchronization and transaction. With persistence components, developers will construct MMOG more conveniently and quickly.

Acknowledgement

首先我要感謝袁賢銘教授給我的指導，在我的研究領域裡給予我很多的意見，並且給予我最大的空間來發揮我的創意。也要感謝所有幫助過和指導過我的學長，蕭存喻和吳瑞祥學長給我很多意見與指導，並給予我許多的建議，讓我能完成這篇論文，而鄭明俊和邱繼弘學長則是讓我學到了些做研究的方法。還要感謝郭予中和陳冠傑，幫助我在英文寫作方面的問題，另外還要感謝陳俊元和林慧雯，常常大家一起在實驗室努力到深夜，最後我要感謝我的爸媽，給予我這個良好的環境讓我求學生涯毫無後顧之憂，專心於學業，謹以這篇小小的學術成就來感謝您們的養育之恩。

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1. MMOG

MMOG (Massively Multiplayer Online Game) is a type of computer game that enables hundreds or thousands of players to simultaneously interact in a game world they are connected to via the Internet. But developing a MMOG is very hard. Developer will encounter plenty of problems, such as high-performance network to exchange thousands of messages in a short time, distributed technology to deploy many servers, extensibility, fault tolerance, load balancing…etc. Most of these issues are not existed in usual game. Therefore, development a MMOG will consume a lot of time and plenty efforts.

MMOGs are the main stream of many other game types at present. Several companies devote themselves to this field, including large companies, such as Microsoft and SONY, etc. And, many companies provide plenty of hardware and software to support game development. All large-scale game companies develop MMOG as an important policy without exception. But, adjudging one game is good or bad that have many considerations. Not only determine by game content but also the server processing power and stability are important issues. By means of the MMOG middleware platform which developed can promote server processing power and stability. And we can develop MMOG more quickly by means of some existence functionality in this platform. The opportunity will be taken beforehand in the present environment in which we need to strive for time.

DOIT [1] (Distributed Organized Information Terra) is a middleware platform of

MMOGs. It provides many technologies such as distribute architecture, load balancing, avatar migration…etc. It also provides plenty of simple APIs to programmer to develop MMOGs.

1.2. Motivation

Although DOIT provides many features for developer, it does not handle the MMOG data well. Game data is the key to a success game. Besides networking, another important issue is persistent on MMOGs middleware platform. Because it's plenty of the game data, they can't be place to main memory at a time. When users logout the game, user's data must be the same when they login next time. Relational database is the most people to use and performance fits in with the requirements of large scale system.

Nowadays, there are plenty of commercial products of MMOGs middleware platform. For example, Butterfly.net [2], Terazona [3] are famous commercial solutions. But, we usually can't know the detail and internal design of the commercial products. The similar type of the software, MUD (multi-user dungeon), has some open sources software such as JAdventure [4], TigerMUD [5]. Take these for example, these software are aware of that data to deal with on persistent component. So, it can hard code to write or to add some mechanism of error handling. This kind of way is not a general solution. It is not convenient to programmer of MMOGs to modify or develop.

In addition to design the roles in the game, the MMOG developer also need to design the fields of the table and the relations of the tables in the relational database.

When programming, they even use the complicate SQL (Structured Query Language) command to manipulate game data. It takes them too much efforts and waste a lot of time. For the reason, we wish to overcome all issues to simplify game development.

1.3. Research Objectives

In order to reduce the developing effort and provide a better MMOG environment, persistence components are necessary. They will not only handle manipulate relation-database but also provide a better method to use. For the reason, those components should fulfill the following requirement: easy-to-use, high-performance and transparent persistent.

- Easy-to-use

It takes much time and efforts to develop a MMOG. If we can provide simple, easy-to-use and useful API to developer of MMOGs, it can save much more time to develop a MMOG.

- High-performance

Generally speaking, there are always thousands of people, even ten thousands of people on line at the same time in a MMOG. So, the performance issue is very important to us. But, how can we response these plenty of uses in a short time? It is a very important issue for us to solve.

- Transparent persistent

Programmers can be in object points of view to design and write the MMOG, but can't be in table points of view in relational database. Through simple API, the framework will save user's data to relational database automatically. Programmers will not feel that they use relational database on the backend. It achieve to transparent in persistent.

According to our research objectives, we design a framework to process all storage issues. This framework will let programmers use simple way to write MMOG. Let programmers concentrate on designing and writing game logic, not to care about the data on the bottom layer what kind of databases to store data.

DOIT middleware platform doesn't have any system components or services about storage. Programmers must be use JDBC [6] to hard code to write the persistent parts. Although, this way is more free to programmers. However, it can not restrict to programmers to write the code of persistent parts only. It will have big problems to maintain the system when the scales of MMOGs become larger and larger. If the system is written by our component and architecture, it will be more easy to maintain and avoid the code relative to use the JDBC to write the persistent pares directly.

1.4. Organization

In Chapter 2, we enumerate the persistent framework on the market and discuss there pros and cons. And we discuss some research about in memory operation. In Chapter 3, we introduce the components in our system and their responsibility. In

Chapter 4, we discuss the implementation details for each component in our system. In Chapter 5, we build up some experiments and evaluations to our system. In Chapter 6, we give the conclusion and future works for our system.

Chapter 2 Related Works

Objects views of designing MMOG will let programmers develop MMOG characters and logics more conveniently. Programmers will concentrate on developing MMOG if they only design every character in the MMOG and various attributes of characters.

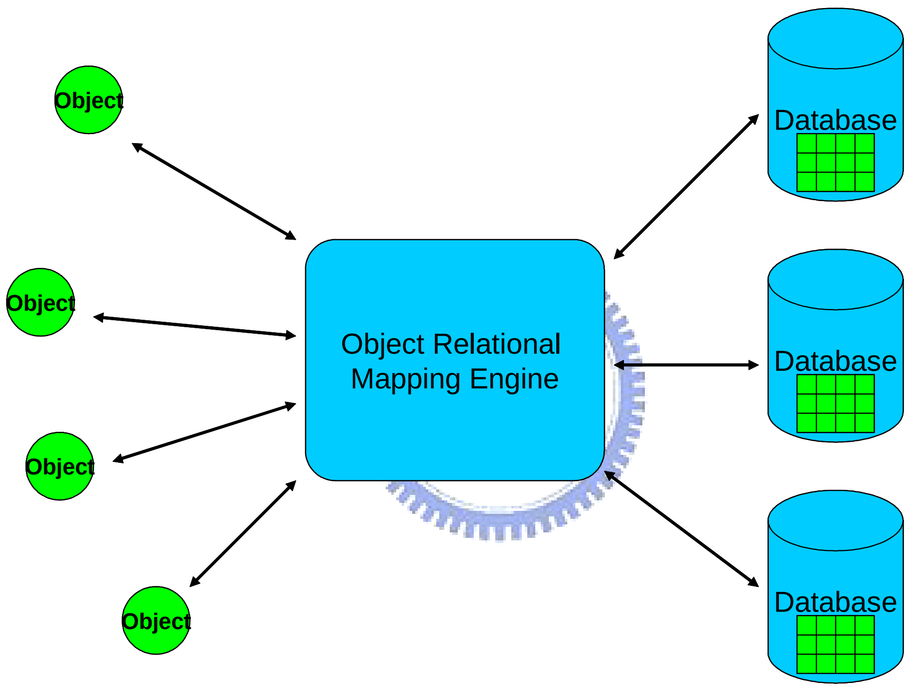


Figure 2-1: Object Relational Mapping Engine

Relational database is the main stream of the market. It is the most people to use on the storage aspect. It also has better performance than other database, such as network database or object-oriented database. However, programmers must design the game objects of the table fields when programmers write the persistent code. Programmers even use the complicate SQL command to write the storage code. Most programmers use object-oriented language to develop system because it is easier to write and maintain program code. The gap between object-oriented language and relation database is large. In order to solve this problem, there are many ORM tools to turn up. ORM is so-called Object Relational Mapping tools. It maps the objects in program to the tables in relational database. Although we use the ORM tools to develop the system, it does not have better performance to use the JDBC directly. When the scale of the system is more and more large, it will take us much time to maintain the system. Hence, if we can choose a kind of ORM tools to develop system, it can reduce the system development and maintenance time. The research topic about ORM tools were lasted for several years. There are a lot of open source software and complete commercial products about ORM tools. This chapter will list some ORM tools which most people are using and compare them in detail.

2.1. DOIT middleware platform

DOIT (Distributed Organized Information Terra) is a MMOG middleware platform. It uses the three tier architecture that is client-gateway-server architecture. It has the load balancing and simple API for programmer easy to develop the MMOG. And it also provides the user define protocol that it can prevent game cheating. It has many characteristics, such as easy to develop MMOG for example, scalability, flexibility, easy-to-use and high-performance … etc. However, there are no complete and good mechanisms about persistence in DOIT middleware platform. It just let programmer to use JDBC to develop storage part of code.

2.2. Existing persistence handling of MMOG middleware platform

Nowadays, even though there are some existence MMOG middleware platforms such as Butterfly.net, Terazona …etc. But there are all commercial products. We can not to know the inner component and detail design. The similar type of the software, MUD (multi-user dungeon), has some open sources software such as JAdventure and TigerMUD. This software is already aware of that format of storage component. So it writes hard code and adds some error handling mechanism. This is not general solution for programming. It is not convenient for programmers to develop MMOG and modifying it to use.

2.3. EJB

EJB [7] (Enterprise JavaBeans) is a part component of J2EE architecture. Entity Bean uses to communication with database. CMP (Container Managed Persistence) uses the J2EE container to manage the storage. It is more convenient for programmer using it to develop storage program. But when we use EJB to develop storage program, it must have J2EE container. J2EE container requires consuming a lot of memory in average. Programmers must be according to the specification of EJB to develop EJB program. Programmers must overwrite some method. Its specification is very complicate. You will want to an expert understanding of this API before coding with EJB. Also, you need to be aware that each J2EE container requires proprietary deployment descriptors outside of ejb-jar.xml. Therefore, it costs very much to develop program using EJB.

2.4. JDO

JDO [8] (Java Data Object) is a standard persistent framework which proposed by Sun Microsystems Inc.. Its specification is provided by JSR 12 [9]. JDO is an architecture that provides a standard way to transparently persist plain java object. It is designed to work in multiple tiers of enterprise architecture, including J2SE, Web tier, and Application Server. JDO dose not itself provide some functionality in EJB such as distributed objects, distributed transactions, or security services. If those functionalities are needed, programmer can integrate JDO with a J2EE container that provides these services to achieve that. However, for the time being, it uses byte code enhancement for the JDO reference implementation for the most part on the market. It doesn't seems a good choose for the java developer that using post byte code compiler. Hence, there are really not so many developers using the JDO technique now.

2.5. Hibernate

Hibernate [10] is a famous persistent framework. Programmer just writes plain java object and an xml mapping file. And programmer can use its simple API to access persistent object. Hibernate also provides primary key generation algorithm for us to choose. It has better performance to the same kind of products. It uses many technologies to promote better performance such as lazy initiation, outer join fetching and batch fetching …etc. It is open source software. The development team maintains it constantly and updates new version time after time. There are many software frameworks using it on persistent part such as Spring Framework [11], XPlanner [12] …etc. Presently, Hibernate is the most popular ORM tools which used to develop persistent code. There are many detail documents on network, so it can solve problem easily.

2.6. Other related research about persistent framework

There are many other ORM tools. For example, Apache Project provides a useful persistent framework called OJB [13] (ObjectRelationalBridge). It is also an OR Mapping tools. It provides JDO reference implementation but not already release completely now. It is a newer open source ORM tool. So, there are a little people used it to develop persistent code. There are less related documents to introduce them.

In addition to using ORM tools, we use some operation about in-memory to promote performance. And there are some researches in this field. TimesTen [14] bring up the Front-Tier which takes some operation in backend database to front end application server. Although they achieve better performance, it takes plenty of memory. On the other hand, MMDB (main memory database) is the similar to in-memory operation. XSol [15] bring up the system to achieve this way and some functionality resembling ORM tools. Nevertheless, their system is mainly used to integrate their application server.

2.7. Summary

Even through there are many kinds of ORM tool on the market, there are advantage and shortcoming individually. Entity Beans provide much additional functionalities such as transaction service, security services …etc. But when we want to execute the Entity Beans, it must have a J2EE container. The J2EE container has much overhead. When we don't want the other features about J2EE container, it wastes so much memory to the system. JDO is a standard specification proposing by Sun. But JDO use the byte code enhancement technology, there are few developer to use it to develop storage code. On the other hand, Hibernate can build up on any application or any platform, it can't have problem of EJB which must be used to J2EE container. Moreover, Hibernate supports the ORM and transaction service. It can access the data in database through Hibernate and also can be a persistent foundation management of MMOG middleware platform.

Chapter 3 System Architecture

3.1. MMOG platform objects

There are many kinds of characters in a MMOG, for example, player's data, level, equipments … etc. These are persistent data which must be stored in database. Player's data should be existed in game world when they login at next time. These are basic kinds of persistent data. When system boot at begin, it loads data into memory such as the map of the game world. These kind of persistent data just load once from database. Afterward, these data can not be altered definitely any more. And some game objects may be used by two players at the same time. Two or more players may alter one object simultaneously. Even through there are so many kinds of game objects in a game world. To sum up, all game objects merely can be classified into the following types. We introduce briefly these types in our system.

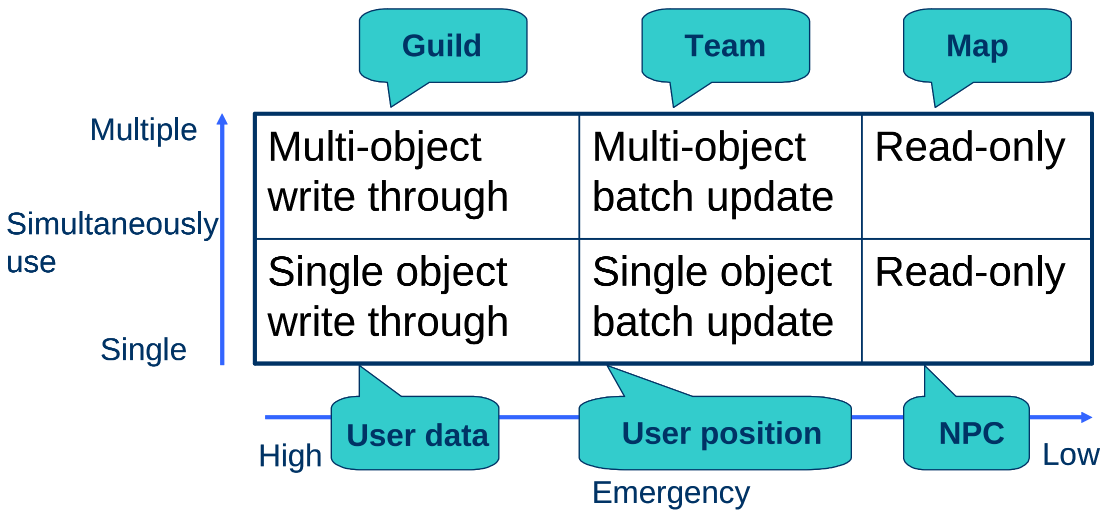

Guild
Team
Map
Multiple
Simultaneously use
Single
Multi-object write through
Multi-object batch update
Read-only
Single object write through
Single object batch update
Read-only
High
User data
User position
NPC
Low
Emergency

Figure 3-1: MMOG Object Types

We survey many object types in MMOGs. We analysis its many object types then conclude these to some kinds of object type. The results are in Figure 3-1. According to horizontal axis, it indicates emergency level for storing data to database. The emergency level high to low is from left to right. Write through means the very emergency data which must be written to backend database as soon as possible. And some data are not emergency which can be stored in `MemoryManager`. After a while, it can write a batch of data to backend database. And, the most less emergency data, it means the read only data. It must not write to database. According to vertical axis, it indicates how many users access this object at the same time.

For example, guild is an example for multi-object write through. In MMOG, many people may join to the same guild. This data are very important. If some data has been modified, it must be written to backend data quickly. Team is an example for multi-object batch update. In MMOG, few players can make up of a team. But this information is temporary. So it can be stored to `MemoryManager` for the time being. After a while, it updates a batch of data to database. The players data in MMOG are belong to single object write through. Every player's data are very important, so it must be written to backend database quickly. But the player's position is not important. It can not update to backend database immediately. And some data like game world map and NPC are all read only data. So these kinds of data must not update any information to backend database.

3.2. System Architecture

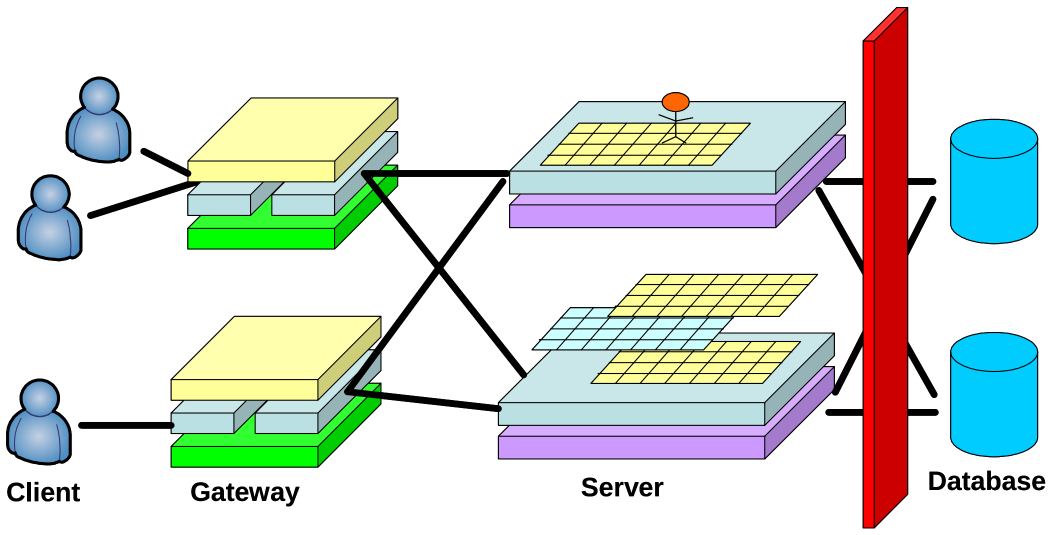


Figure 3-2: DOIT System Architecture

There are three tiers in DOIT platform that is client-gateway-server architecture. Our system is established on front end of database. It is the red part which is front of the database in figure 3-2. We add some components front of database to let programmers more conveniently. It is convenient for programmers to develop programs related to persistence.

In general, programmer can use Hibernate to communicate with database. Figure 3-3 indicates this architecture. Programmers must not write the bottom layer programs which using JDBC and complicated SQL command. They just use the simple API which is provided by Hibernate then they can access the data in database easily.

In the application programs, some objects must be stored in database. Another object is transient which just store in memory. Even if the system reboot doesn't matter with the transient object.

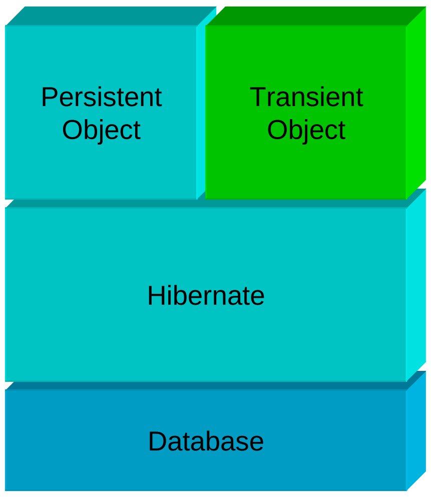


Figure 3-3: Hibernate System Architecture

Figure 3-4 indicates our high level system architecture. As Figure 3-4 indicated, Hibernate can divide into two parts in detail. One is used to communication with database at the bottom layer. Another is used to get the persistent object. Programmer uses the `Session` object to get persistent object and uses it to access data object. On top of the system architecture, there are persistent objects and transient objects. We add one layer called `MemoryManager` to handle user's objects in memory between Hibernate and programmer's object.

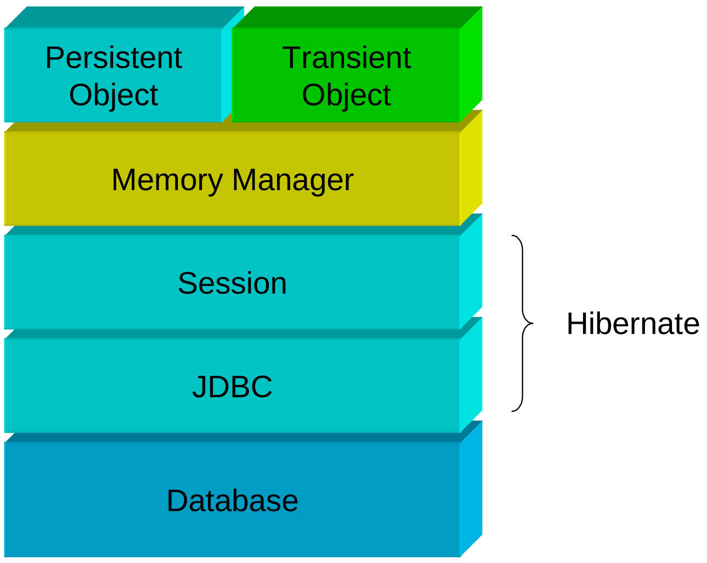


Figure 3-4: High-level System Architecture

3.3. Memory Component

- **MemoryManager**

"`MemoryManager`" is the layer between Hibernate and application layer. The purpose of the component is to manage the objects in memory which developer used. If programmers want to access the persistent objects, it must communication with `MemoryManager`. `MemoryManager` will control the life cycle of persistent objects.

If programmers want to persistent some object to backend database, it must rely on `MemoryManager`. They can not take objects to Hibernate immediately. It can improve performance and efficiency to persist object by means of `MemoryManager`.

- **MemoryObject**

The basic units which are stored in `MemoryManager` are `MemoryObject`. Excluding the data of the `MemoryManager`, there is much more information in `MemoryObject`. This information can be used to manage life cycle of `MemoryObject`.

3.4. System Object

To help programmers develop MMOG more convenient and quicker, we define some kinds of system objects. Developers just think about characters in game.

Programmers consider the behavior of the character in the game to decide what kinds of system object. Programmers can use it by inheriting the system object. We can be in object points of view to design the game. It must not consider the table field points of view in relational database Programmers take viewpoint of object to design and develop the MMOG. Let programmer develop the MMOG more convenient.

- **BaseObject**

  It is the basic object type of system object. Every type of object must inherit this object.

- **TransientObject**

  It is transient and temporary object in the system. As far as developing MMOG are concerned it is used to NPC (non-player character) in general. We can produce it again even if the system reboot. It needs not to access any data in database.

- **PersistentObject**

  Persistent object means that this kind of object must store in database. As far as developing MMOG are concerned it is used to player's data, treasure and equipments … etc. `PersistentObject` class is the basic type of persistent class. All kinds of persistent object must inherit it.

I. ImmutablePersistentObject

It only needs to fetch data from database at the first time for this kind of persistent object. After that, this object needs not to update its record in database. It will not alter the data itself. As far as developing MMOG are concerned it is used to map data. In the beginning, map just only load to memory from database. Map doesn't alter its data any more.

II. GeneralPersistentObject

It is general persistent object. At the same time, only one player can access this object. For example, player's personal data, basic equipments and its attribute values all belong to this object type.

III. SyncPersistentObject

This persistent type object can be accessed by many players at the same time opposite to `GeneralPersistentObject` class. So, our system can deal with this object some synchronization control. If multiple threads access this object at the same time, it also can execute normally. Developers just use it by inheriting the `SyncPersistentObject` class. Our system will guarantee the thread safe.

In section 3-1, we analysis many MMOG object types and conclude to six type.

These six types of MMOG objects can map to our system object. Single object write through and single object batch update map to `GeneralPersistentObject`. Since only one player can access this object at the same time. Multi-object write through and multi-object batch update map to `SyncPersistentObject`. Because multi player may access these object at the same time. And read only data map to `ImmutablePersistentObject`. `MemoryManager` can handle write through and batch update. It can let programmer to determine object behavior.

Chapter 4 Implementation Detail & issue

In this chapter, we begin to dive into the implementation details for our system. Our system's architecture is depicted in figure 4-1. Developers interact with the application layer which is the front-end of the system. The middle tier is `MemoryManager` which communications with Hibernate. Hibernate communications with the database directly at the backend. A rational database is used to store the actual data.

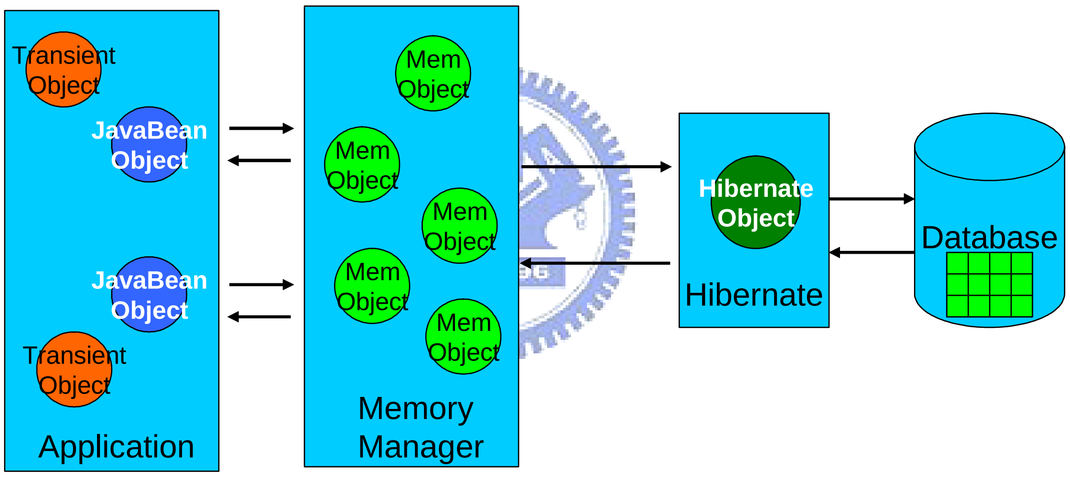


Figure 4-1: System Architecture

Programmers develop the objects which inherit different type of system objects and programmers write additional methods according the game characters by the getter and setter model. Programmers write that methods to the getter and setter model. Programmers don't have to overwrite some complicated methods like EJB, they just have to write some simple JavaBean [16]. When these JavaBean objects are stored into `MemoryManager`, they'll be encapsulated into `MemoryObject`. When

`MemoryManager` wants to store `MemoryObject`'s to the backend, it returns original JavaBean objects and gives them to Hibernate which will then be stored into database.

4.1. Application Tier

Programmers develop the game content and game logic in the application tier. There are many types of character in the MMOG. They can be classified into several types of objects based upon the behavior of persistent objects. We define some basic kinds of object type in our system. Programmers can choose which object type to use according to the object's attributes and corresponding game character. The following will introduce every kinds of object in detail in our system and its relationship between each other.

- **BaseObject**

The `BaseObject` class is the base class of all system objects. All system objects must inherit from this object. It defines an identity attribute called id and overwrites the `equals(obj:Object)` method of `java.lang.Object`. It differentiates every object by object's id.

`BaseObject` class inherits `mmog.doit.GameObject` class in the DOIT middleware platform. When programmers use DOIT to develop MMOG, every character must inherit from `GameObject` class. Then the `GameObjChannel` delegate some methods call to its associating game object.

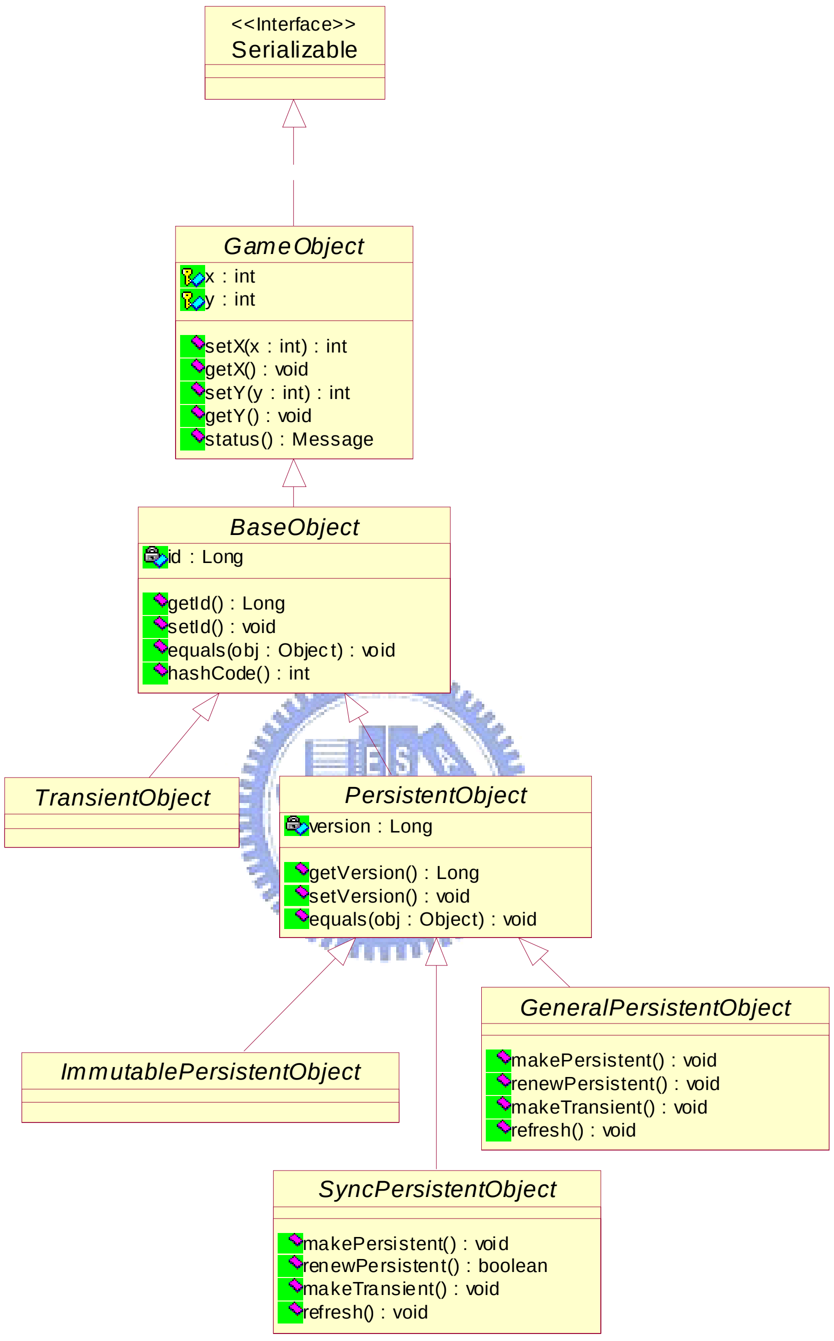


Figure 4-2: Class diagram of system objects

- **TransientObject**

The `TransientObject` class extends `BaseObject` class. It is a temporary object in our system. The `TransientObject` is stored in main memory. It can’t be stored in database.

- **PersistentObject**

`PersistentObjet` class also inherits `BaseObject` class. Classes which extend from `PersistentObject` class can be stored into the database. Beside the attributes inherited from `BaseObject` class, it also adds a version attribute which can be used to keep track of when it was stored. The value is a timestamp which that time was stored in database.

- **ImmutablePersistentObject**

This type of class is mainly used in objects where data is immutable. It means that we can not modify any of its attributes after loading from database. So we don’t have any action about saving or loading data from database.

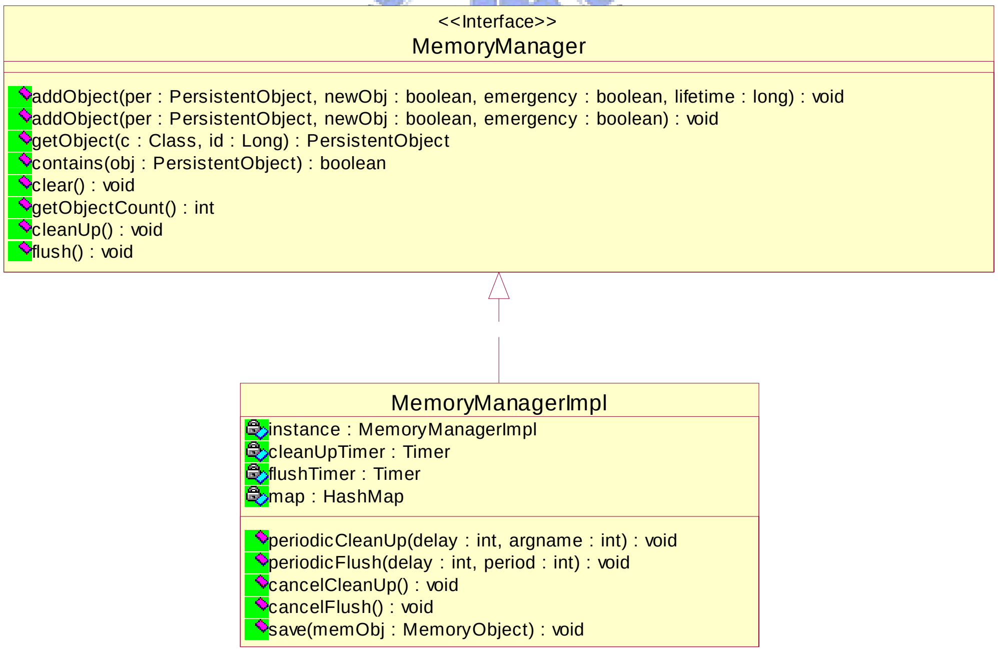
- **GeneralPersistentObject**

There are three main methods in this kind of persistent class. They are `makePersistent()`, `makeTransient()` and `renewPersistent()` which are used to insert, delete and update. Besides these, there is a method called `refresh()` which can renew the state of the object and keep it concurrent with the database. This class can be used by only one player at any given time.

- **SyncPersistentObject**

This class is similar to `GeneralPersistentObject` class. But this class can be accessed by many players at the same time. The system decides whether to store this object according to the version attribute, thus achieving data consistency. The `renewPersistent()` will return a boolean value to determine if the data has been stored to the database successfully or not.

4.2. Manager Tier



<<Interface>>
MemoryManager
addObject(per : PersistentObject, newObj : boolean, emergency : boolean, lifetime : long) : void
addObject(per : PersistentObject, newObj : boolean, emergency : boolean) : void
getObject(c : Class, id : Long) : PersistentObject
contains(obj : PersistentObject) : boolean
clear() : void
getObjectCount() : int
cleanUp() : void
flush() : void
MemoryManagerImpl
instance : MemoryManagerImpl
cleanUpTimer : Timer
flushTimer : Timer
map : HashMap
periodicCleanUp(delay : int, argname : int) : void
periodicFlush(delay : int, period : int) : void
cancelCleanUp() : void
cancelFlush() : void
save(memObj : MemoryObject) : void

Figure 4-3: Interface and implementation of MemoryManager

`MemoryManager` is used to manage persistent objects. It provides a basic and simple API to programmers developing MMOG's. Figure 4-3 indicates the interface and the implementation of `MemoryManager`. `addObject` can add objects developer created to `MemoryManager`, and the objects can chose whether or not to store to the database directly by setting the attributes. We can also set the objects' life cycles. Then `MemoryManager` will flush the objects that exist in `MemoryManager` to the database periodically. And `cleanUp` will take expired object to garbage collection periodically. Because a system's main memory is not infinite, it can not store all objects in main memory all the time. For this reason, we implemented the LRU [17] (Least-Recently-Used) algorithm. When system's main memory is not enough, it will store the object to database which longest time since last been used. And it will delete that object from main memory.

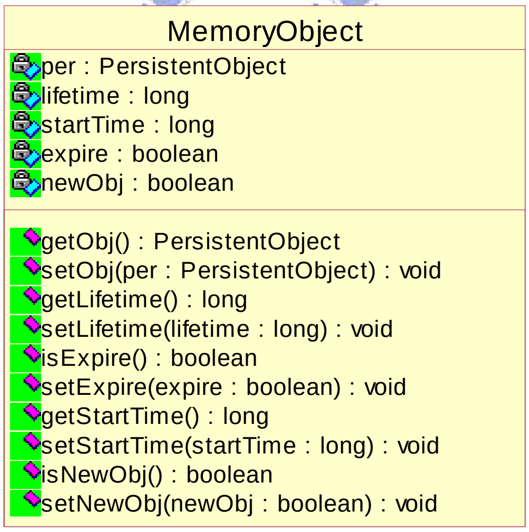

MemoryObject
per : PersistentObject
lifetime : long
startTime : long
expire : boolean
newObj : boolean
getObj() : PersistentObject
setObj(per : PersistentObject) : void
getLifetime() : long
setLifetime(lifetime : long) : void
isExpire() : boolean
setExpire(expire : boolean) : void
getStartTime() : long
setStartTime(startTime : long) : void
isNewObj() : boolean
setNewObj(newObj : boolean) : void

Figure 4-4: MemoryObject

All objects which are stored in `MemoryManager` are encapsulated in `MemoryObject`'s. Figure 4-4 indicates the member functions of `MemoryObject`. `MemoryObject` can keep track of some information about itself, such as the lifetime, whether or not it's expired, whether or not to create a new object, etc.

The implementation of `MemoryManager` class was done by hash. The identification key of this hash is the id and class name combined. Every pair is unique, so every single object can identified in `MemoryManager`.

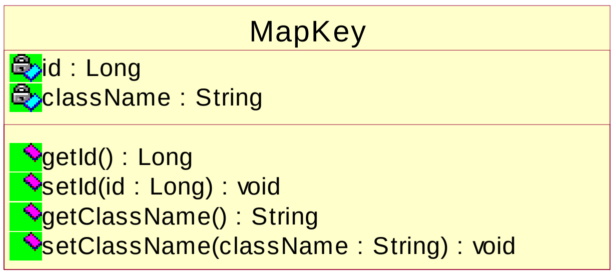


Figure 4-5: MapKey

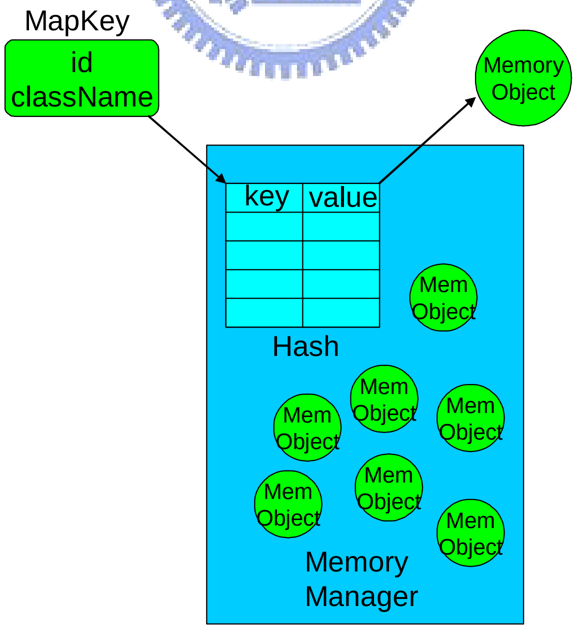


Figure 4-6: MemoryManager

4.3. Persistent Tier

Our system actually used Hibernate to communicate with the backend database. In Hibernate, we used `Session` class to handle almost all actions about accessing the database. For every thread in our system, we use one `ThreadLocal` class to persist a `Session` object, so every thread can access only its local value and can not the others'. The transaction in Hibernate also works in the same way. The `Session` object in Hibernate is obtained using `SessionFactory` class. The `SessionFactory` class is a heavyweight class, so a system better has only one `SessionFactory` in its application. We set that attribute to static, and encapsulate in `HibernateUtil` class for developers to use.

4.4. Implementation issues

In the following, we list some issues about this system. And we will discuss about these issues.

- POJO

  Information technology progresses at a great speed these days. Every company wants to quickly produce new games and new contents to players which use and play them. Under this demand, programmer must develop games more quickly. In order to develop more quickly, programmers can not spend too much time on surveying and understanding complicate specifications and API’s.

For example, programmers who want to use EJB to develop a system must understand that all system architecture and its complicated specification before actually develop the system. During the implantation process, they must overwrite many EJB methods. In comparison, when using our system, programmer only needs to write POJO (Plain Old Java Object) Java programs. It greatly decreases the time of development. The product can ship faster gaining an advantage on the market.

- In-memory

With regard to a MMOG, user data changes and updates happen very often. Facing thousands of connection in a MMOG, how we can achieve high performance and short response time is a very important issue. Due to fast hardware advancement in recent year, main memory price has become lower and lower gradually. There used to be only few data which can be stored in main memory, but now, much more data can be stored in main memory. It can promote system performance in evidence. In-memory data management has become an issue for many more applications that are in dire need of high performance. Therefore, in-memory data management and operation will become an important topic.

- Develop Cost

EJB is a persistence framework standard which Sun Microsystems Inc. brought up. But, when programmers use EJB to develop programs, there must be J2EE containers. However, J2EE containers cost a great deal to companies. Although there are some free and open source J2EE containers on the market, their performances are not enough to support large scale system such as a MMOG. Because there are many players playing the game on line at the same time, there must be some commercial product such as J2EE container to sustain this kind of application. Our system uses Hibernate to store objects to the backend database. It is an open source persistence framework. We can modify its code depending on our requirements. Hibernate is not in the J2EE architecture, so it need not have any J2EE container to execute. It can run standalone without J2EE environment. Company need not consume any capital on commercial persistence framework and J2EE container.

- Maintain cost

With regard to a complete MMOG, if programmers used JDBC to hard code the storage program, when their system becomes larger in scale, the will become much harder to maintain. But if programmers used our system to develop their system, their source code can be more modular and clear. Programmers only have to take an object view on designing their system. Their system becomes more clear and modular thus more convenient on maintenance later.

Chapter 5 Experiment and Discussion

We start to experiment and evaluation our system in this chapter. We experiment in two rounds. In the first round, we enumerate the most often four table relationship in database. It experiment our system by using this four kinds of relationship in the first round. And the second round, we adopt general test case to evaluation the database system. This experiment is 80% read operation and 20% write operation to experiment.

5.1. Round1: Database relationship

In this round, we use usually four cases about table relationship in database. We use these four cases to experiment our system. There are single table, many-to-one, one-to-one and many-to-many.

- **Hardware configuration**

Usage	**Number**	**Configuration**
Database	1	P4 2.4GHz CPU with 512MB ram
Application	1	P4 2.4GHz CPU with 1GB ram

Table 5-1: Hardware configuration of round1

Table 5-1 indicates the hardware configure in this experiment. In order to other packets to interfere with this experiment, it uses one switch hub to connect these two computers. And it uses 100 MB Ethernet network to connect each other.

- **Software configuration**

We use the MS SQL Server 2000 to our persistent storage. In application tier, we use JDK 5.0 for our experiments. It compares with using our system component or not in all environment. We experiment difference between using our system component and doesn't use our system component.

Figure 5-1 indicates our system architecture. We flush objects in MemoryManager every 1000 millisecond. And figure 5-2 doesn't use our system component. It simply uses Hibernate alone. We take 1000、2000、3000、4000 objects to experiment. We perform insert and update operation to our experiments and keep track of that time.

Because this experiment need many objects in memory, the Java virtual machine parameter can not use default value. We set initial heap size to 384MB and set maximum heap size to 512MB.

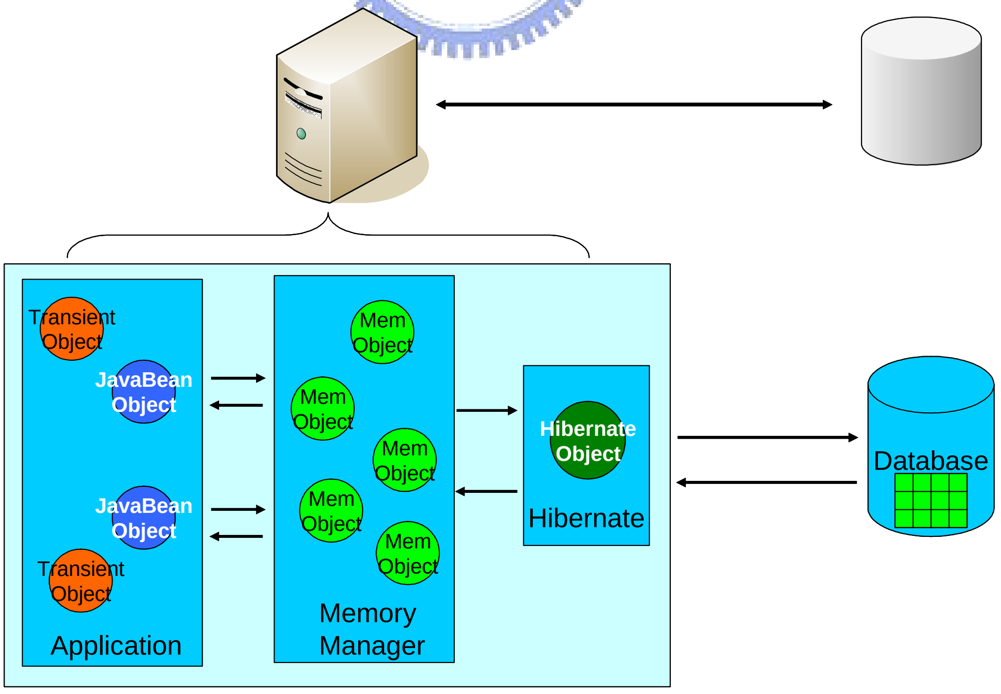


Figure 5-1: Experiment with our system architecture

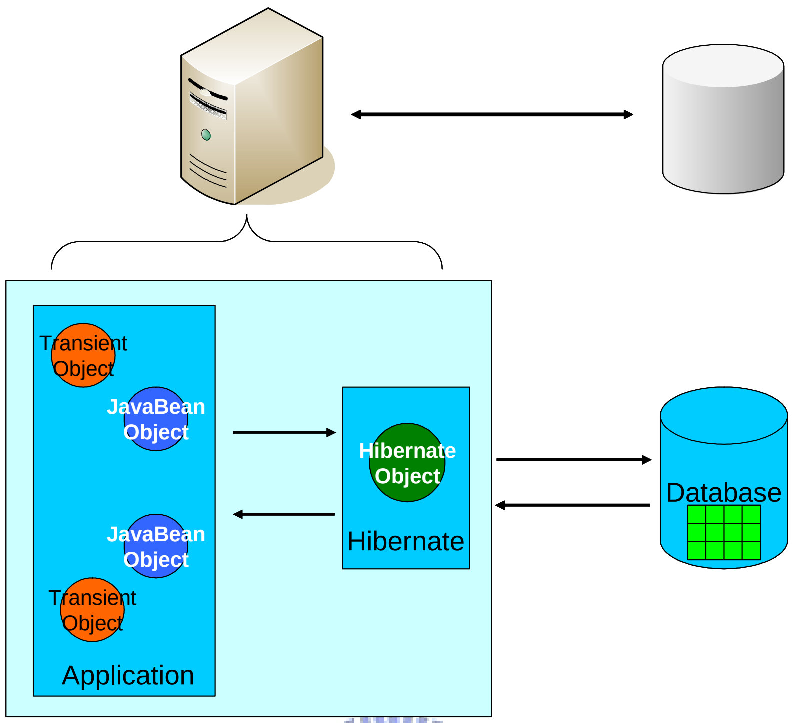


Figure 5-2: Experiment without our system architecture

- **Single table**

In this experiment, we use one class called `GameRole` which have eight attributes and inherits `GeneralPersistentObject`. And we write an xml mapping file to description this object map to one table in database. The results are as follows, it takes insert and update operation.

Single table average time taken (millisecond)				
Object number	Insert		Update	
	without MemMgr	with MemMgr	without MemMgr	with MemMgr
1000	3148.25	992	5144.75	1257.75
2000	8179.75	1910.25	16566.5	2503.5
3000	15050.75	2894.5	58648.75	3496.25
4000	29183.5	3801	207211	4707

Table 5-2: Experiment result of round1 single table

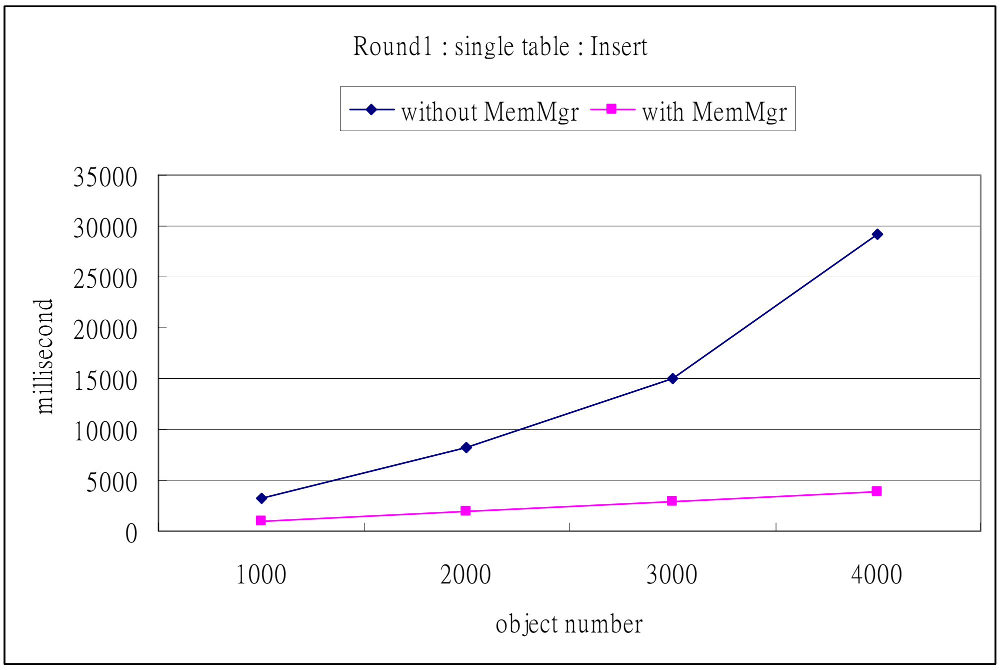


Figure 5-3: Experiment result of round1 single table insert operation

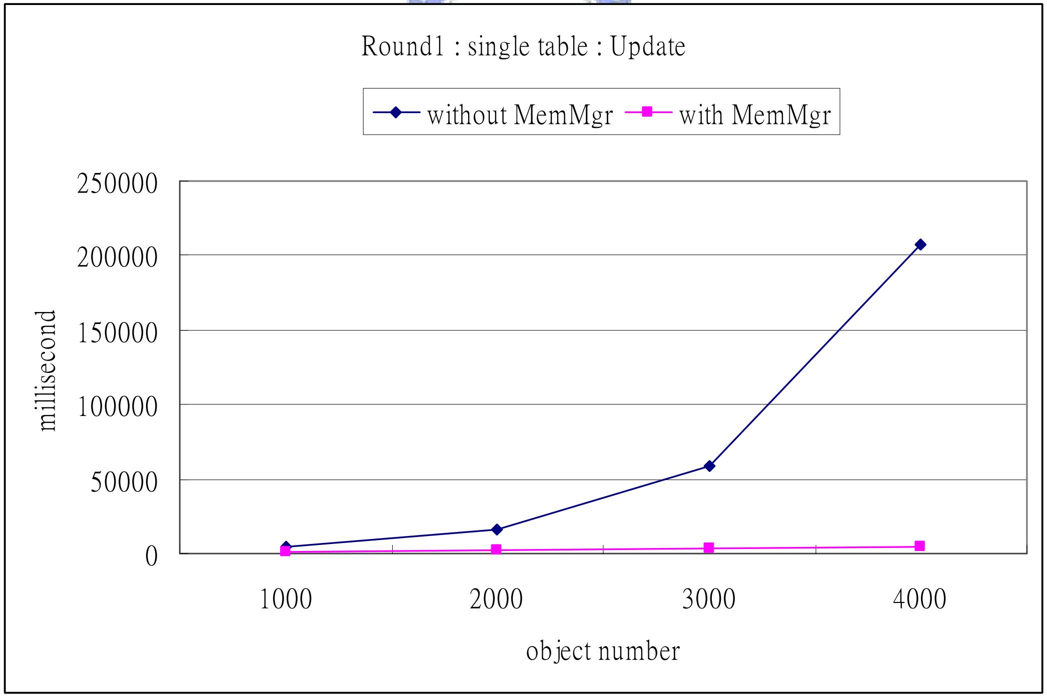


Figure 5-4: Experiment result of round1 single table update operation

- Many-to-One relationship

In this experiment, we use two classes which `GameRole` and `Equipment`.

These two classes both inherit `GeneralPersistentObject`. `GameRole` have eight attributes and `Equipment` have four attributes. Every `GameRole` has one Equipment attribute. But one `Equipment` can be owned by many `GameRole`. The results are as follows, it takes insert and update operation.

Many-to-One relationship average time taken (millisecond)				
Object number	Insert		Update	
	without MemMgr	with MemMgr	without MemMgr	with MemMgr
1000	4160.25	1199	7980.5	1699.5
2000	11019.5	2336.25	27629	3273.25
3000	22195.5	3562.5	93004	4758
4000	36066.5	4722.5	482898.5	6453

Table 5-3: Experiment result of round1 Many-to-One relationship

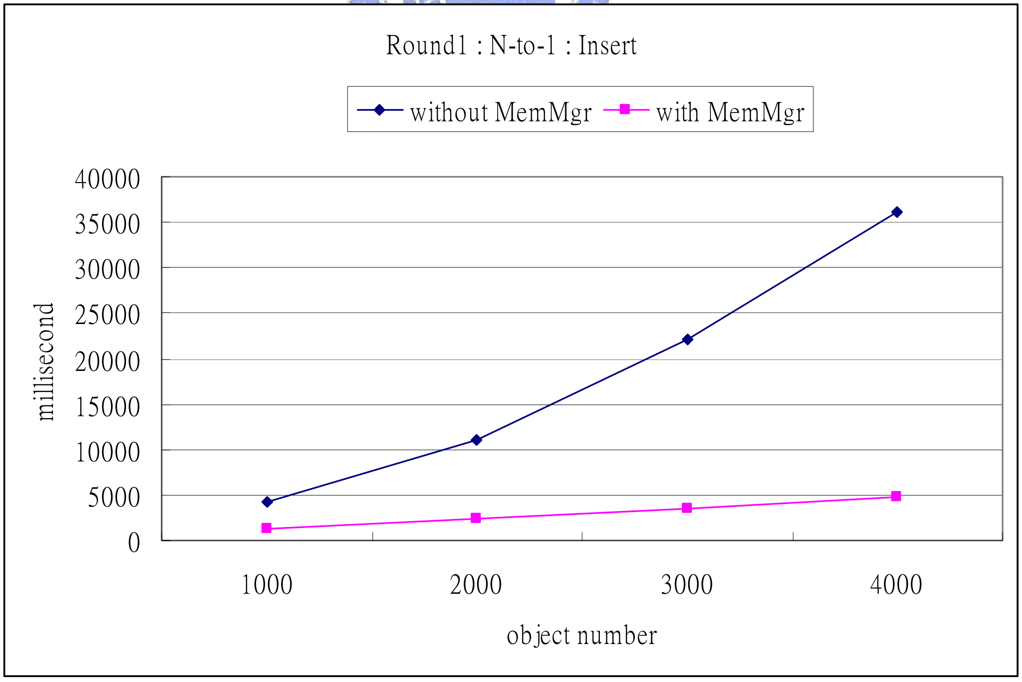


Figure 5-5: Experiment result of round1 Many-to-One insert operation

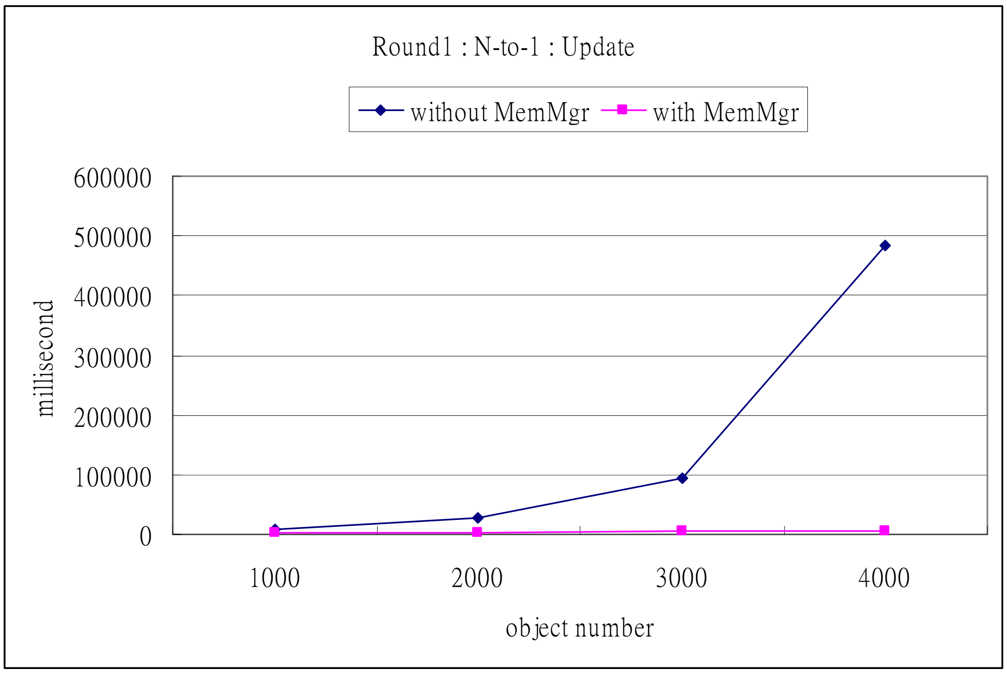


Figure 5-6: Experiment result of round1 Many-to-One update operation

- **One-to-One relationship**

In this experiment, we use two classes which `GameRole` and `Equipment`. These two classes both inherit `GeneralPersistentObject`. `GameRole` have nine attributes and `Equipment` have five attributes. Every `GameRole` has one Equipment attribute. One `Equipment` can be owned by only one `GameRole`. The results are as follows, it takes insert and update operation.

One-to-One relationship average time taken (millisecond)				
Object number	Insert		Update	
	without MemMgr	with MemMgr	without MemMgr	with MemMgr
1000	7898.5	1870.75	33781	3999.75
2000	26437.5	3824.25	124394.5	6429.25
3000	51062.5	5679.75	183726.75	9101.5
4000	100957	10128.75	278801	11859.25

Table 5-4: Experiment result of round1 One-to-One relationship

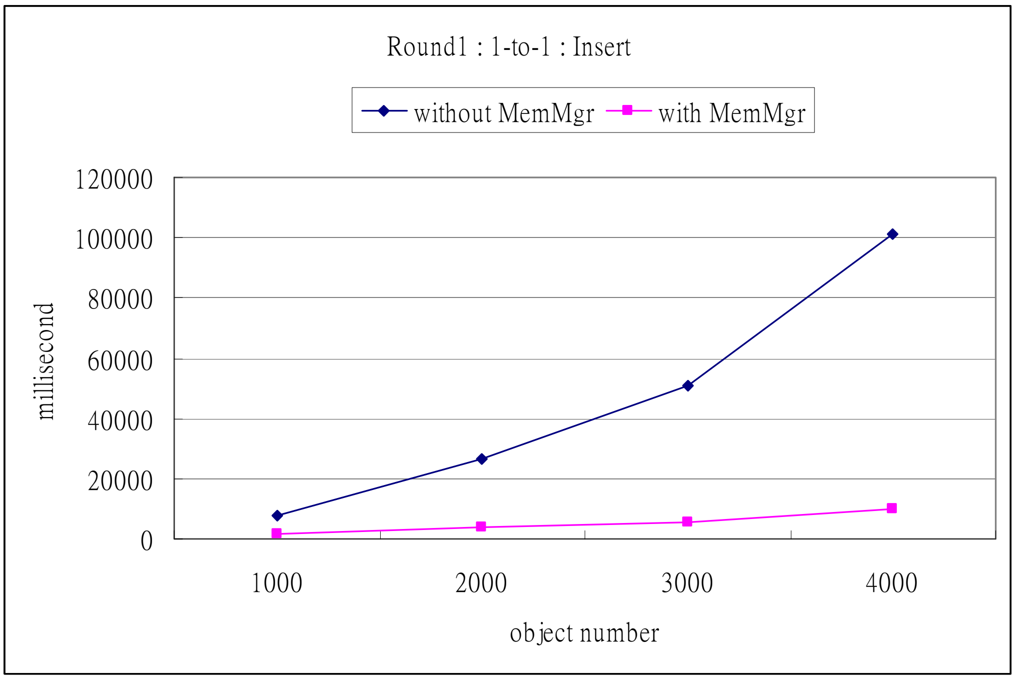


Figure 5-7: Experiment result of round1 One-to-One insert operation

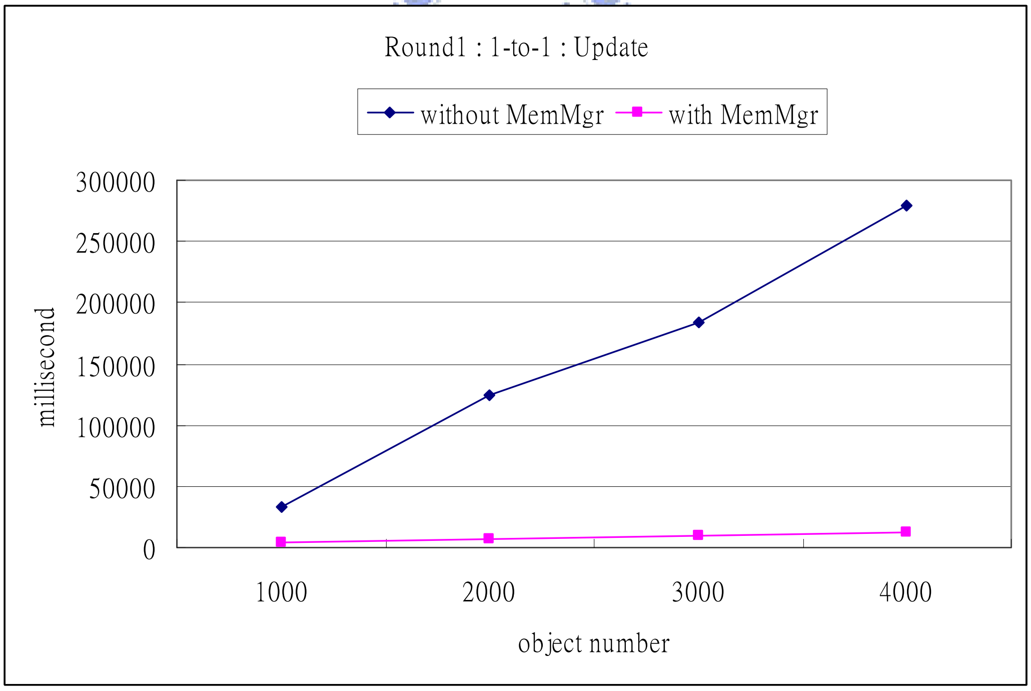


Figure 5-8: Experiment result of round1 One-to-One update operation

- **Many-to-Many relationship**

In this experiment, we use two classes which `GameRole` and `Equipment`. These two classes both inherit `GeneralPersistentObject`. `GameRole` have nine attributes and `Equipment` have five attributes. Every `GameRole` has many `Equipment` attribute and its type is `java.util.Set`. One `Equipment` can be owned by many `GameRole` and its type is also `java.util.Set`. The results are as follows, it takes insert and update operation.

Many-to-Many relationship average time taken (millisecond)				
Object number	Insert		Update	
	without MemMgr	with MemMgr	without MemMgr	with MemMgr
1000	60164	2683.75	180132.75	4398.5
2000	216660.25	9851.75	630894.5	18707
3000	318129	7797.25	1030347.25	25867.5
4000	920843.75	8941.5	1830486.75	29734.25

Table 5-5: Experiment result of round1 Many-to-Many relationship

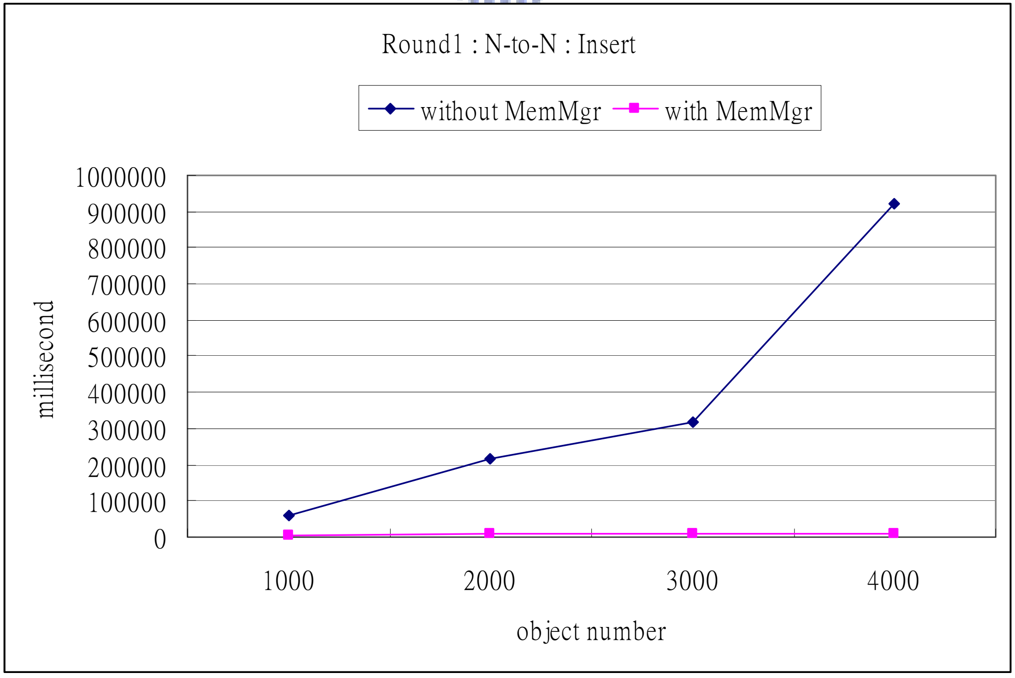


Figure 5-9: Experiment result of round1 Many-to-Many insert operation

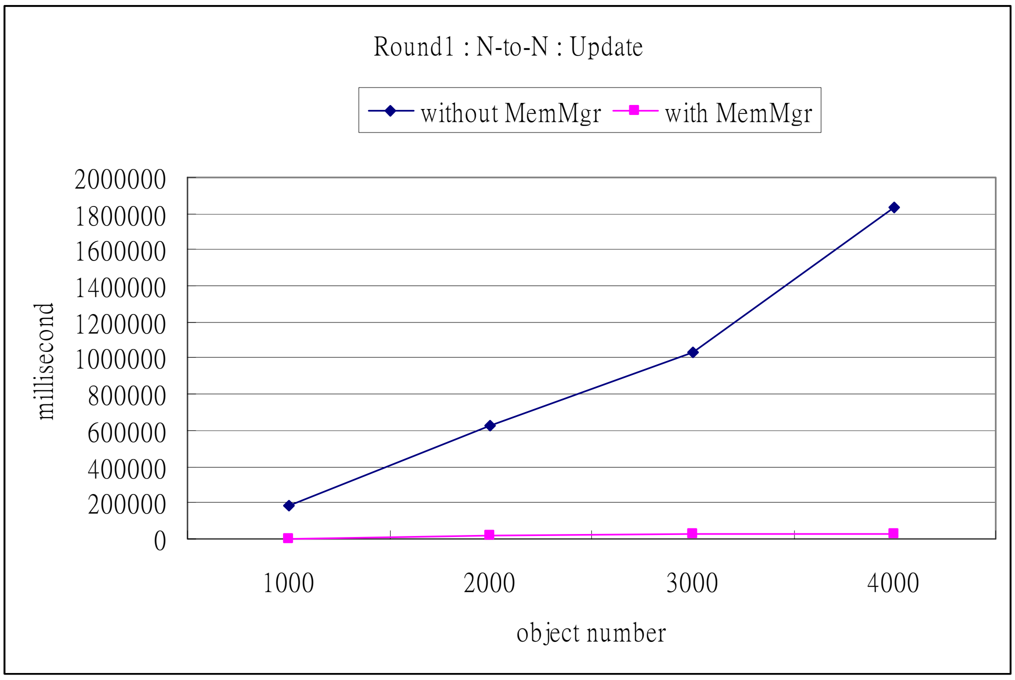


Figure 5-10: Experiment result of round1 Many-to-Many update operation

To sum up these four cases, when object numbers have fewer such as 1000 or 2000 objects, there is little difference between our system architecture and simple to use Hibernate. But when there are plenty of object numbers such as 3000 or 4000, our system architecture needs only less time to do this operation. Because it simply uses Hibernate to do these operations that directly write data to database every time. But it uses our system components that write data to memory first and these objects can be stored to `MemoryManager`. Our `MemoryManager` can manage this object and its life cycle in memory. After a period of time, data can be committed to database. So it takes less time and performance will be better.

5.2. Round2: 80% read and 20% update

In this round, it takes the general rule for evaluation database. It uses 80% read operation and 20% write operation to experiment. We experiment that using our system architecture or not. Table 5-6 indicates this result that keep track of average time taken using 500、1000、2000、3000 and 4000 objects.

- **Hardware configuration**

Usage	Number	Configuration
Database	1	P4 2.4GHz CPU with 512MB ram
Application	1	P4 2.4GHz CPU with 1GB ram

Table 5-6: Hardware configuration of round2

Table 5-6 indicates the hardware configure in this experiment. In order to other packets to interfere with this experiment, it uses one switch hub to connect these two computers. And it uses 100 MB Ethernet network to connect each other.

- **Software configuration**

We use the MS SQL Server 2000 to our persistent storage. In application tier, we use JDK 5.0 for our experiments. It compares with using our system component or not in all environment. We experiment difference between using our system component and doesn't use our system component.

Because this experiment need many objects in memory, the Java virtual machine parameter can not use default value. We set initial heap size to 384MB and set maximum heap size to 512MB.

Average time taken (millisecond)		
Object number	without MemoryManager	with MemoryManager
1000	5318.8	53
2000	13503.2	62.6
3000	29265.6	75
4000	54890.6	93.8

Table 5-7: Experiment result of round2

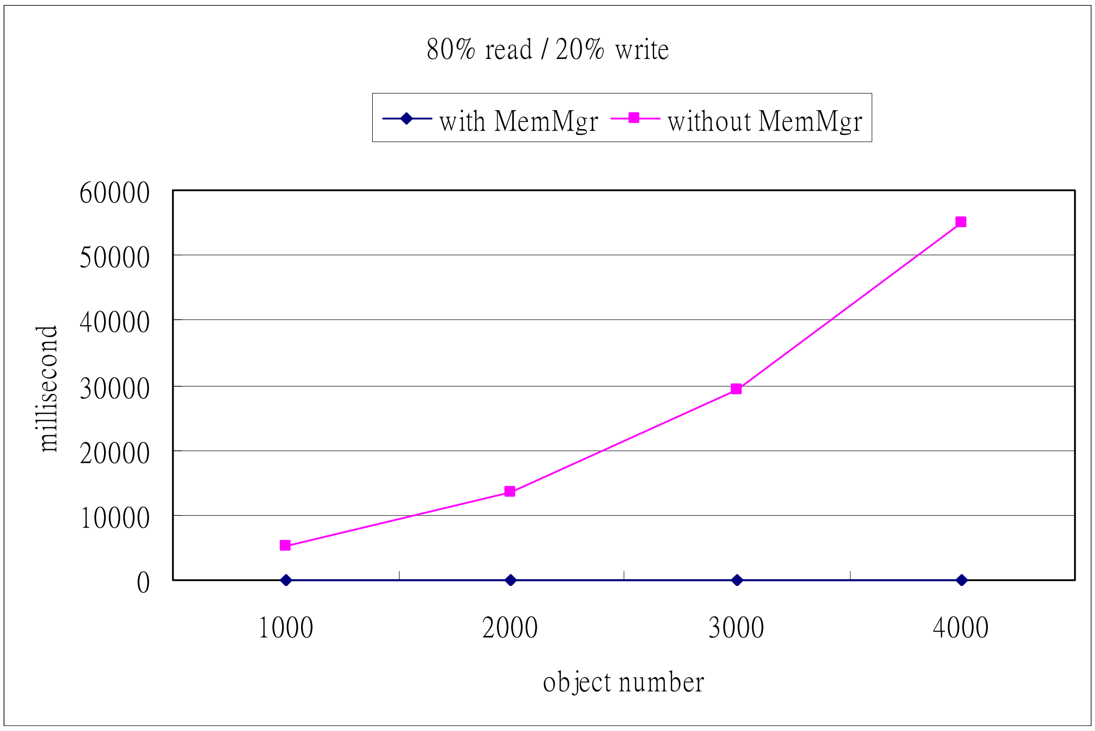


Figure 5-11: Experiment result of round2

These results lead us these conclusion. When using ours system component, the time taken will not increase very much which object number from 1000 increase to 4000. But it increases much time in evidence when it only uses Hibernate to do this operation. If it only execute one time, the result can not be so much difference. Because we use what kinds of system architecture whether or not, it must load data from database when it executes read operation at first time. It will be faster that using our system component after the first time to do read operation. Because our system will be cache data in main memory, the read operation will be faster than does not use our system in evidence.

Chapter 6 Future Works and Conclusions

6.1. Conclusions

MMOG is very hot game type at present. But, it needs much strength and many efforts to develop a complete MMOG. So, how to use the least time to develop a complete MMOG is very important.

We accomplish persistent system architecture and services which base on DOIT middleware platform. At the bottom layer, we use an open source persistent framework called Hibernate to communication with database. We define some basic type class for programmer to use. It can develop MMOG characters simply according to inherit these classes. Programmers even can use these components to develop other kinds of application.

The following part, we take two points of view to discuss and analyze our system.

- Design time

  In observing the perspective on design time, if programmers can use this system to develop MMOG, it will save much time and many efforts. Our system already defines many classes and provides some simple used API. Developers need not write the complicated SQL command but they still can achieve persistent functionality. So, developers can concentrate his mind on designing game content and logic. They need not take efforts to deal with problem about design system. It can decrease developers overhead.

● Run time

In observing the perspective on run time, it is efficiency that we store object to main memory before data are actually stored to database. After that, we store in-memory object data to database periodically. Our system also supports transaction management. It can guarantee data integrity and consistency. When many players access the same data, we can use the mechanism of synchronization. Don’t worry about data inconsistency problem.

By means of the platform and its services, not only can reduce designing program and developing time for game but also can improve performance to the server.

6.2. Future Works

In the thesis, these experiments are all single operation. These are quite different from actual game. After all, the actual online game can not fix insert and update operation like our experiment. If we can design and implement an actuality game and let many users to play this game to experiment. It will more fit with the situation which actual game happened. And we can take this result to tuning performance with our system.

MMOG is more hotter and hotter in recently year. So there are many kinds of games are produced on the market. Our system objects may be not comfortable for this new type of games. How to design the object type to more suitable for new type of games? And let it can promote all system performance. It will be a very important research topic.

Bibliography

[1] Chen-en Lu, Tsun-Yu Hsiao, Shyan-Ming Yuan. Design issues of a Flexible, Scalable, and Easy-to-use MMOG Middleware. In Proceeding of Symposium on Digital Life and Internet Technologies 2004

[2] Butterfly.net, `http://www.butterfly.net/`

[3] Zona Inc., `http://www.zona.net/`

[4] JAdventure, `http://graphmud.sourceforge.net/`

[5] TigerMUD, `http://www.tigermud.com/`

[6] Sun Microsystems Inc. JDBC Technology, `http://java.sun.com/products/jdbc/`

[7] Sun Microsystems Inc. Enterprise JavaBeans Technology, `http://java.sun.com/products/ejb/`

[8] Sun Microsystems Inc. Java Data Objects, `http://java.sun.com/products/jdo/`

[9] JSR-12, `http://jcp.org/aboutJava/communityprocess/final/jsr012/index2.html`

[10] Hibernate, `http://www.hibernate.org/`

[11] Spring Framework , `http://www.springframework.org/`

[12] XPlanner, `http://www.xplanner.org/`

[13] ObJectRelationalBridge, `http://db.apache.org/ojb/`

[14] The TimesTen Team. In-Memory Data Management in the Application Tier. Proceeding of the 16th International conference on Data Engineering, February 2000.

[15] John Grundy, Steve Newby, Thomas Whitmore and Peter Grundeman. Extending a Persistent Object Framework to Enhance Enterprise Application Server Performance. The 13th Australasian Database Conference (ADC2002).

[16] Sun Microsystems Inc. JavaBean, `http://java.sun.com/products/javabeans/`

[17] Abraham Silberschatz, Peter Baer Galvin. Operating System Concepts. Publisher by John Wiley & Sons Inc.