게임 포트폴리오
(메이플스토리 모작)

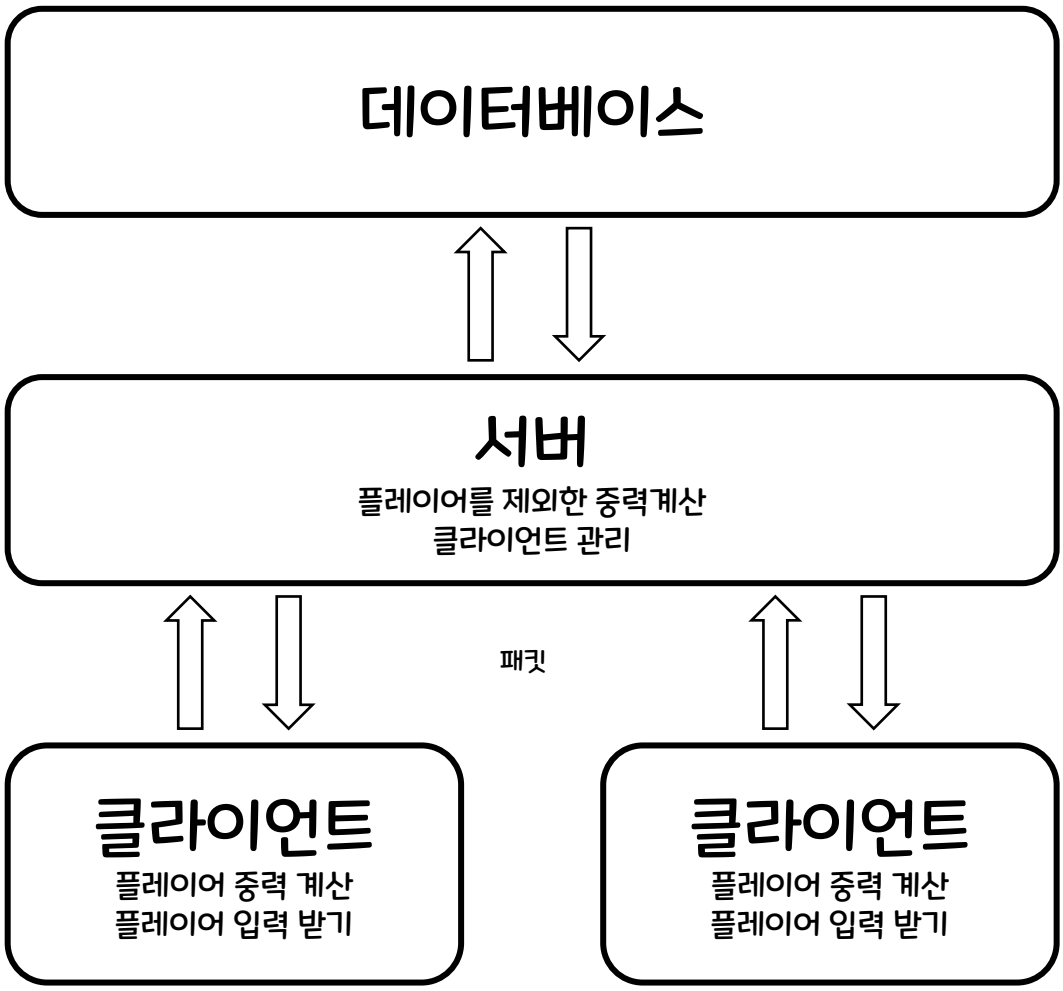
이동헌

목차

1. 소개

- 개발 기간 : 약 4개월
  - 클라이언트 : 2022.06 ~ 2022.08
  - 서버 추가 : 2022.09~2022.11
- 개발 인원 : 1인
- 사용 언어 : C++
- 그래픽 API : Direct2D & WinAPI
- 서버 : IOCP
- 데이터베이스 : MySQL
- 유튜브 : https://youtu.be/pgrcKuHF7sU
- 깃허브 : https://github.com/Lee-Dongheon/Portfolio_Maplestory

2. 게임 구조



1) 서버

- 기본적인 게임 로직 계산(중력 계산 등)
- 데이터베이스와 연동하여 정보 저장

2) 클라이언트

- 플레이어 입력 및 중력계산
- 서버에서 정보 받아 데이터 수정 및 렌더링

온라인 게임이므로 같은 맵에 속한 유저들이 같은 화면을 볼 수 있도록 기본적인 게임 로직은 서버에서 처리합니다. 이 때 클라이언트의 조작감을 위해 클라이언트의 조작 및 충돌계산만 클라이언트에서 처리하도록 구현하였습니다.

3. DB 구조

user_db

user_account		
Key	Id	Pw

ID(유저 id)

status					
level	exp	maxhp	hp	maxmp	mp

skill		
step(스킬 차수)	name	point(투자한 sp)

item				
name	type	count	X	y

Equip	
name	type(장비 타입)

Pos		
num(맵 번호)	x	y

---

user_db라는 데이터베이스에 user_account 테이블이 있습니다. 아이디를 생성하면 이 테이블에 id, pw를 저장함과 동시에 id를 이름으로 하는 데이터베이스를 생성합니다.

유저 id로 생성된 데이터베이스는 각각 status, skill, item, equip, pos 테이블을 가지며, 위와 같은 정보를 저장합니다.

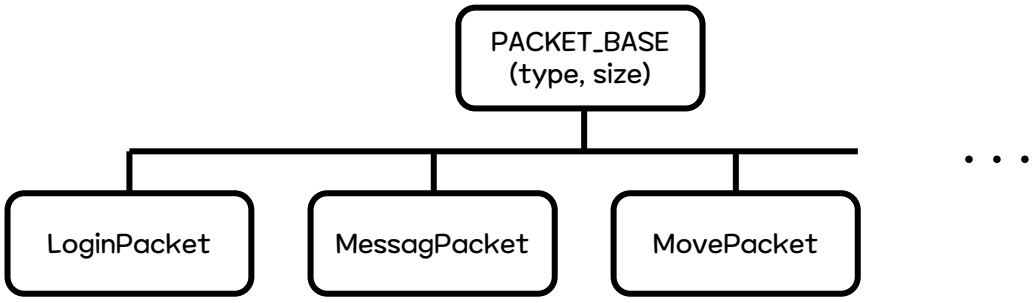
4. 서버와 패킷

```cpp
typedef struct _tagOverlappedEx
{
    WSAOVERLAPPED overlapped;
    WSABUF dataBuf;
    SOCKET socket;

    // send 또는 recv 요청시마다 카운트할 변수
    // 구조체 지울 때 카운트가 0인지 체크할 것.
    int send = 0;
    int recv = 0;

    char buffer[PACKET_LENGTH];
    SOCKET_OPERATION_TYPE eType; // 소켓 정보 확인

    void Init(SOCKET _socket, SOCKET_OPERATION_TYPE _eType)
    {
        memset(&overlapped, 0, sizeof(WSAOVERLAPPED));
        dataBuf.buf = buffer;
        dataBuf.len = PACKET_LENGTH;
```

SOCKETINFO 구조체

```cpp
bool CClientManager::PacketProcess(PSOCKETINFO pSockInfo)
{
    PACKET_TYPE ePacketType;
    ePacketType = (PPACKET_BASE(pSockInfo->dataBuf.buf))->type;

    switch (ePacketType)
    {
    case PT_CREATEACCOUNT:
        return CreateAccount(pSockInfo);
    case PT_LOGIN:
        return Login(pSockInfo);
    case PT_MOVE:
        return MoveCharacter(pSockInfo);
    case PT_MESSAGE:
        return Message(pSockInfo);
    case PT_PORTAL:
        return Portal(pSockInfo);
    case PT_SKILL:
        return UseSkill(pSockInfo);
    case PT_CHANGESKILL:
```

PacketProcess 함수 코드 일부



패킷 구조체 구조

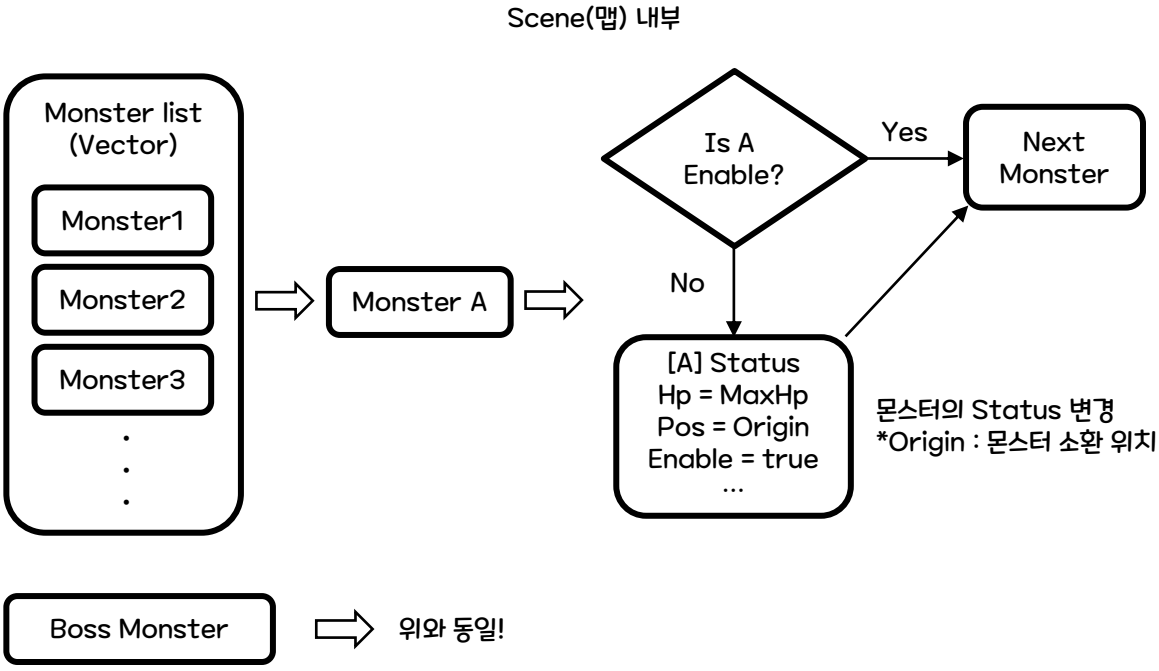
서버프로그램의 로직

1. CPU 코어 개수만큼 쓰레드 생성
2. 쓰레드 수를 가져와 IOCP 객체 생성
3. AcceptEx를 호출하여 IOCP에 등록, 이 때 Completion Key로 SOCKETINFO 구조체 사용
4. GetQueuedCompletionStatus 함수를 통해 완료를 확인할 때 CompletionKey(여기서는 SOCKETINFO)의 eType을 확인
5. eType이 Accept면 클라이언트 소켓 저장하고 AcceptEx 재등록, Recv면 PacketProcess에 따라 처리, Send면 SOCKETINFO 구조체 제거

PacketProcess는 받은 SOCKETINFO의 버퍼를 PPACKET_BASE(PACKET_BASE의 포인터)로 캐스팅합니다. 패킷별로 구조체를 정의했는데, 모든 패킷은 PACKET_BASE를 상속받습니다. 이 구조체는 패킷의 종류와 사이즈를 저장합니다. 따라서 PACKET_BASE로 캐스팅하여 해당 패킷의 종류를 체크하고, 종류에 맞는 함수를 실행해줍니다.

모든 패킷 통신에서 Send를 하는 경우 새로운 SOCKETINFO를 생성하여 전달함으로써 IOCP객체가 버퍼를 전송하는 도중 버퍼를 지우거나 수정하여 데이터가 오염되는 일을 방지합니다. Recv인 경우는 SOCKETINFO 구조체를 초기화하여 재사용하여 메모리 할당/해제 오버헤드를 줄여주었습니다.

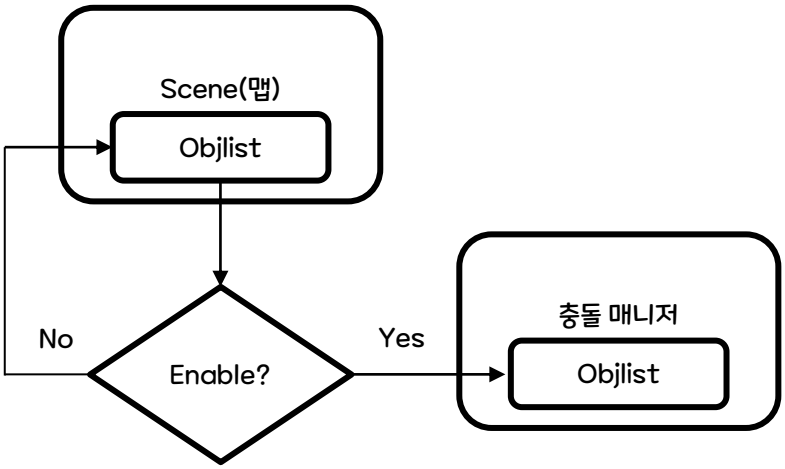
5. 몬스터 관리

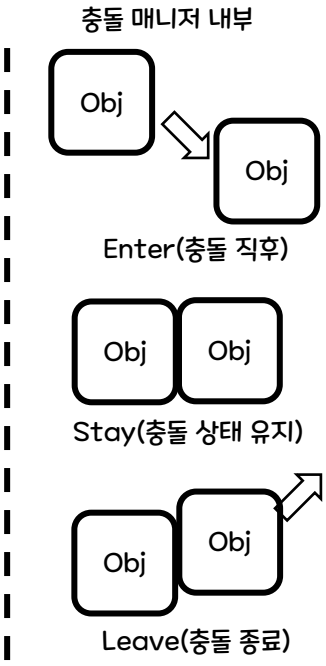


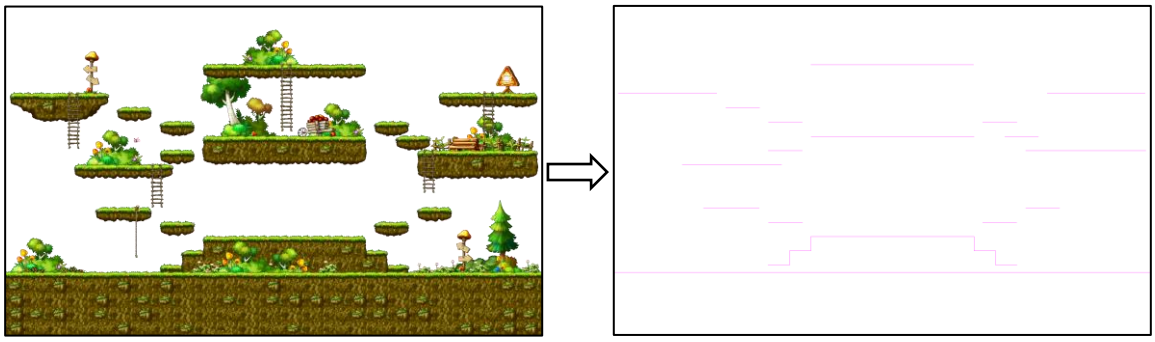
게임 실행 중 몬스터는 죽고, 생성되는 일이 잦습니다. 이 때 항상 정직하게 오브젝트를 소멸시키고 생성시킨다면 비효율적입니다. 따라서 기존 오브젝트를 재활용(메모리 풀) 할 수 있게 다음과 같이 구현하였습니다. 이 때, 보스몬스터는 활성화 여부를 체크하는 시간을 늘려주고 일반 몬스터와 따로 관리하도록 하였습니다.

1. 몬스터가 생성되는 맵은 생성될 수 있는 최대 몬스터 수만큼 몬스터를 미리 생성
2. 몬스터가 죽는 경우 소멸이 아니라 비활성화 상태로 변경(몬스터 처치)
3. 일정 시간마다 맵은 몬스터 목록에서 비활성화 된 몬스터의 기본 상태를 변경해주고 활성화 상태로 변경(재생성)

6. 충돌계산







픽셀 충돌체 예시(오른쪽 파일을 로드하여 충돌체로 사용)

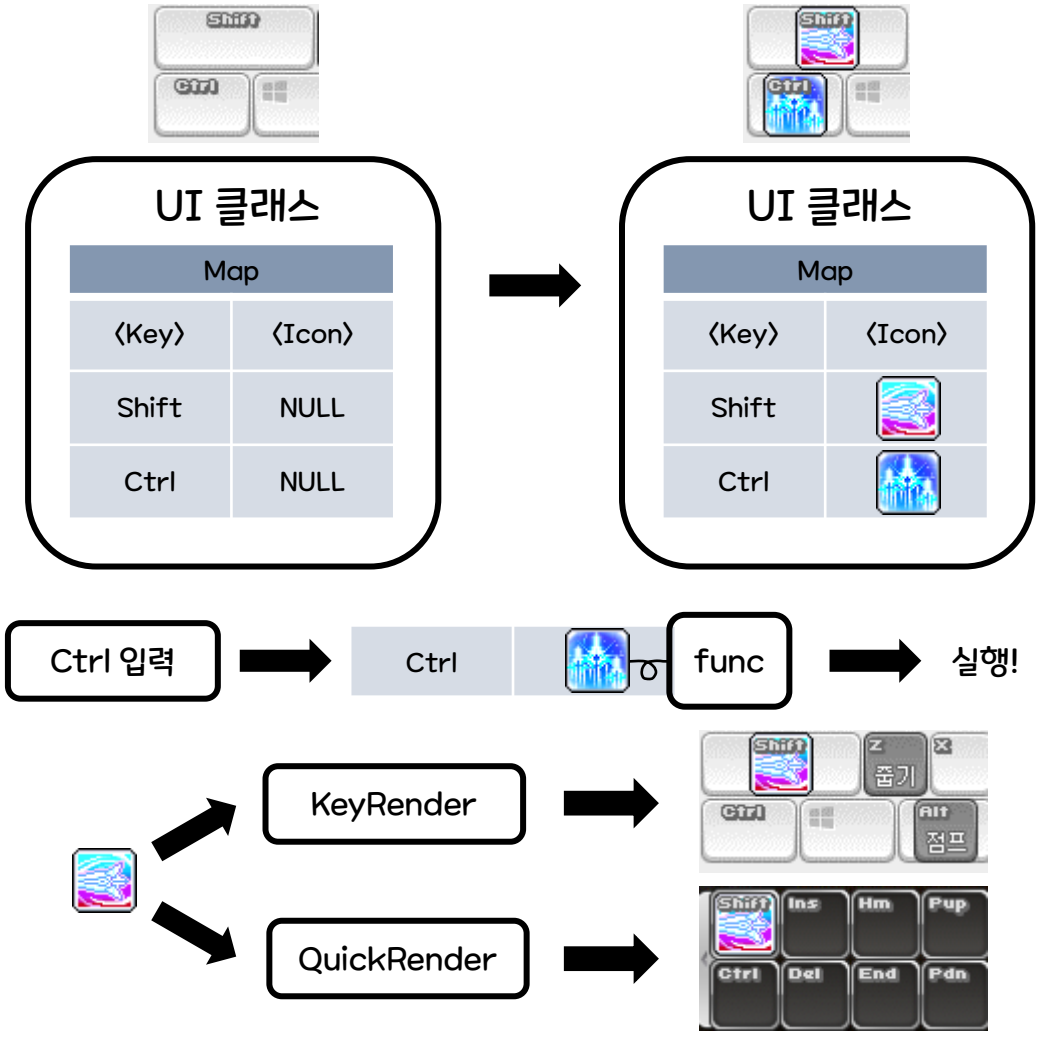
충돌 계산에 사용할 충돌체로는 픽셀, 사각형 등이 있으며 맵의 경우 바닥 정보를 픽셀로 가진 픽셀 충돌체를, 캐릭터나 몬스터의 경우 사각형 충돌체를 정의하여 사용하였습니다. 단, 중력계산에 사용하는 충돌체는 크기가 캐릭터 또는 몬스터의 전체를 감싸지 않고 발 역할을 할 수 있도록 충돌체의 높이를 작게 지정하였습니다.

충돌계산

1. 현재 맵에 활성화 되어있는 오브젝트들을 충돌 매니저(싱글턴)에 추가
2. 충돌 매니저에서 오브젝트 리스트 속 충돌체끼리 충돌 계산
3. 충돌 상태는 Enter, Stay, Leave 세 가지이며 충돌 시 상황에 맞는 바인딩 된 함수 호출
4. 리스트 속 오브젝트 전체에 대해 실행이 끝나면 리스트 비우기

충돌체 초기화 시 <functional> 라이브러리를 이용해 Enter, Stay, Leave마다 각각 다른 함수를 바인딩하였습니다.

7. 키세팅과 아이콘

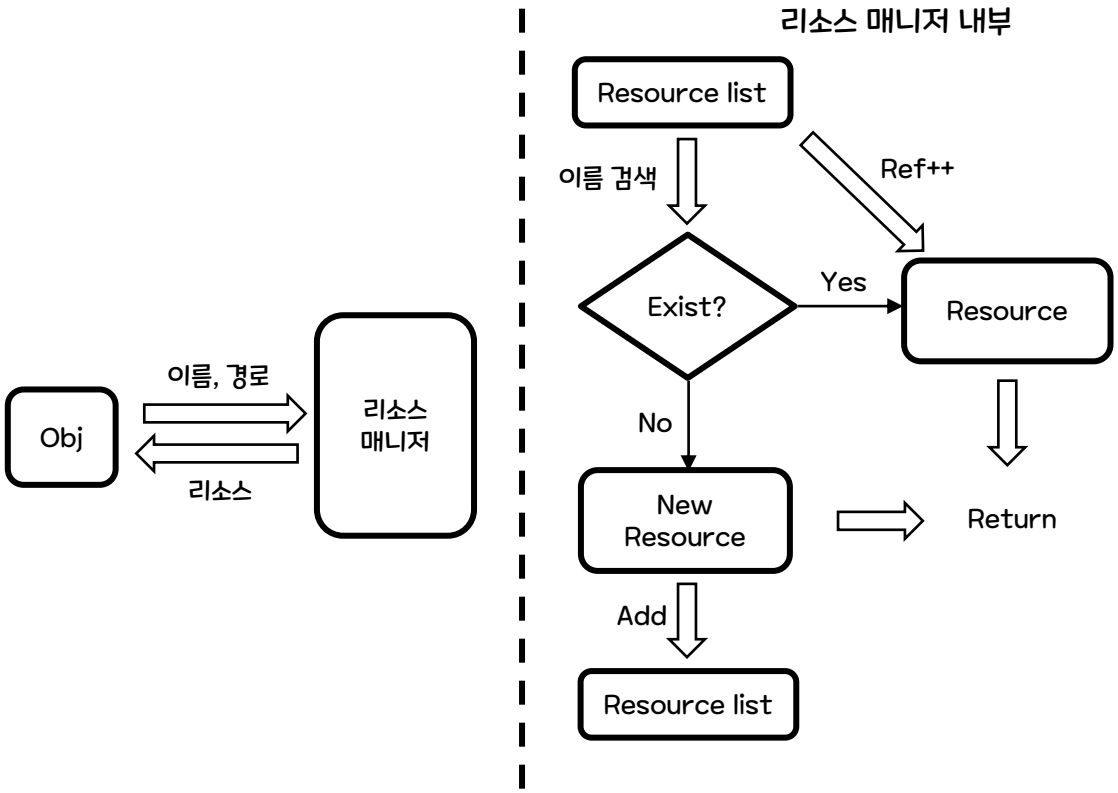


사용자 지정 키세팅

1. 모든 UI종류의 클래스들은 UI 클래스를 상속받는데, 이 UI클래스에 정적 멤버변수로 아이콘 맵이 존재
2. MainUI는 매 프레임마다 입력 매니저(싱글턴)에게 입력받은 키 목록를 전달받음
3. 해당 벡터를 키 값으로 가지는 아이콘에 바인딩 된 함수 실행(함수 바인딩은 〈functional〉 라이브러리를 활용)

또, 아이콘은 QuickRender 함수와 KeyRender 함수가 따로 있습니다. 각각 퀵슬롯UI, 키세팅UI에 아이콘을 그려주는 함수입니다. UI들은 아이콘 맵을 공유하여 퀵슬롯과 키세팅이 연동됩니다.

8. 리소스 관리



같은 리소스를 사용하는 경우 불필요한 메모리 점유와 리소스 로딩을 줄일 수 있도록 다음과 같이 구현하였습니다. 여기서 리소스는 텍스쳐, 배경음악 등을 말합니다.

리소스 로딩 방식

1. 오브젝트는 리소스 매니저(싱글턴)를 호출하여 리소스 로딩을 시도
2. 리소스 매니저가 가지고 있는 리소스 리스트에서 이름으로 검색
3. 리소스가 존재한다면 리소스의 레퍼런스 1 증가 후 반환
4. 존재하지 않는다면 리소스 새로 생성 후 리소스 리스트에 추가하고 반환

사용 완료 시 메모리 해제 대신 레퍼런스를 감소시킵니다. 이후 레퍼런스가 0이 되면 메모리를 해제합니다.