**COMP3029J Software Systems Architecture**

SHEN Yixi & SUN Yue & XU Junmeng & YANG Shaonan & ZOU Keming & ZHANG Mingkun

**Design Style**

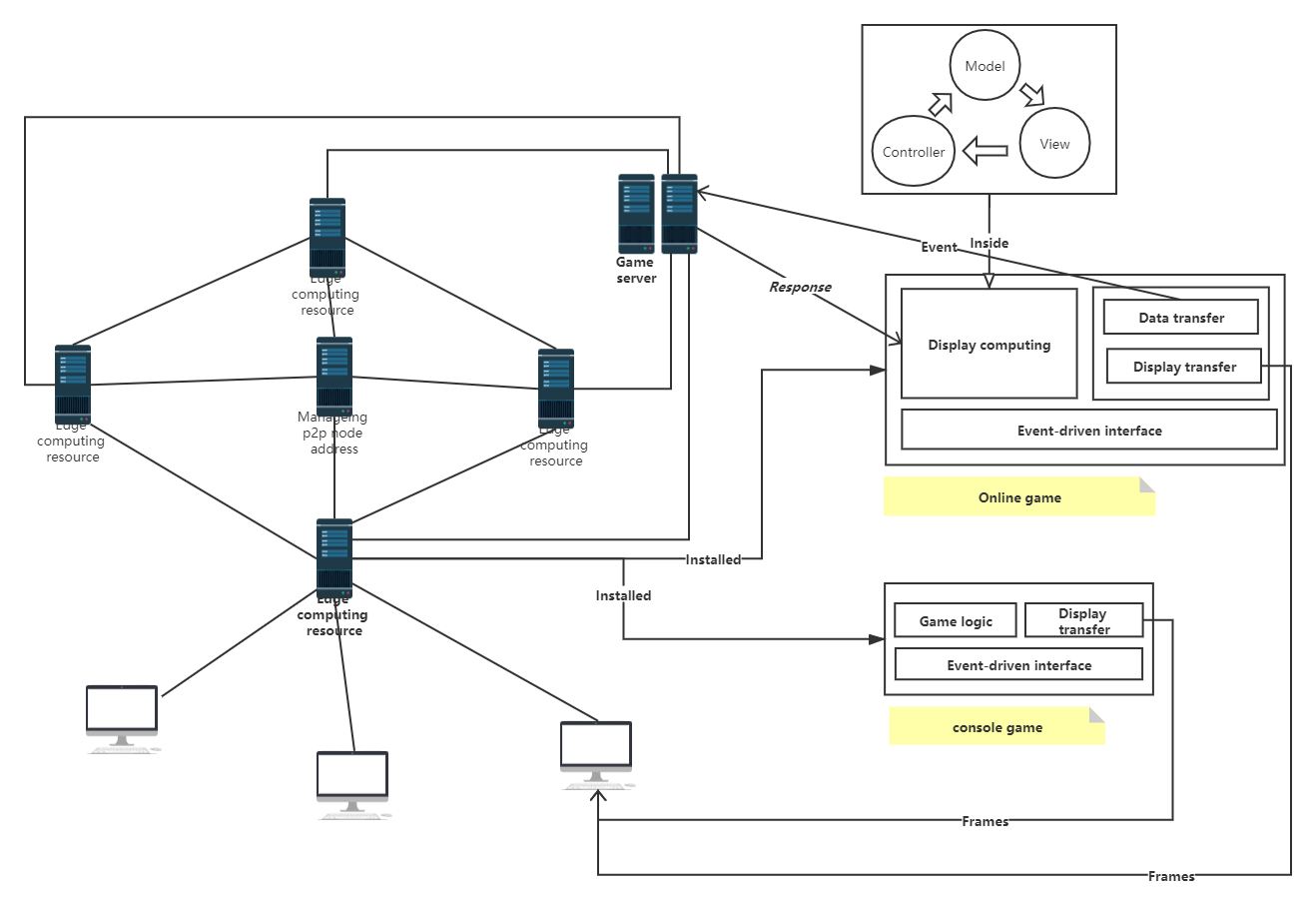
Design style involved：

Overall style：Client-server Style

Game or video server architecture：Layered Architecture

Edge server architecture：Peer-to-Peer Style

Client game architecture：Event Driven Style

Handling other issues：Model-View-Controller (MVC)

1. **Overall style**

Reference Three-tier Client-server Style

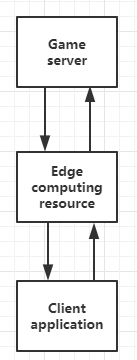
**Premise assumption：**

Because the main point of discussion in the title is the processing of client-side transactions, so for the setting of the game or video main server, we only consider the case of one main server.

**Element：**

Game Video Server, Edge Server, Client

**Schematic diagram：**



**Interactive process：**

During the client / server model, the proactive request method is adopted：    
  
Game video server-side:

First, the server must be started and provide corresponding services according to the request:

1. Open a communication channel and inform the localhost that it is willing to receive client requests at a well-known address (Zhouzhikou, such as FTP 21);

2. Wait for the client's request to reach the port; (Here, because the edge server is the middle layer of the client-server, the edge server can also be regarded as the client's proxy, that is, the client here refers to the edge server)

3. Receive the repeated service request, process the request, and send a response signal. After receiving a concurrent service request, a new process must be activated to handle this client request (for example, fork, exec in UNIX systems). The new process handles this client request and does not need to respond to other requests. After the service is completed, the communication link between this new process and the client is closed and terminated.

4. Return to the second step and wait for another customer request.

5. Shut down the server   
  
Client:

1. Open a communication channel and connect to a specific port of the host where the server is located; (same as above, where the edge server acts as a proxy for the game server, and the client considers the edge server to be the target server of the link)

2. Send a service request message to the server, wait for, and receive a response; continue to make requests ...

3. After the request ends, close the communication channel and terminate.

Edge server:

As the middle layer, it refers to the client and server that the above two parties refer to, respectively.

When processing requests, data processing operations are performed. For example, after receiving the data from the client, the huge data will be calculated or filtered and processed into a result that can be more conveniently processed by the game server, and then sent to the game server. After receiving the data from the game server, it is also processed to send only the necessary data to the client. Among them, the edge server also plays a role in controlling the game. In other words, it is like a small game server with some logic processing functions, which we will introduce in detail later.

**Design reasons**：

Due to the bottleneck of computing resources, we do not put a lot of calculation or data processing on the client, and because of the problem of delay, that is, the time required for data interaction between the server and the client may be too long, or delay due to other problems Customers create a bad gaming experience. We designed an edge server between the client and the server as the middle layer, which is also responsible for processing part of the logic so that the client's game is reflected more quickly, while also reducing the burden on the server. The overall network The load is also greatly reduced. At the same time, because of the reduction in the amount of data transmitted, the security of the data can be more assured. In addition, when the game server is down, the edge server can temporarily store part of the data, which improves the overall robustness.

1. **Game or video server architecture:**

Reference Layered Architecture

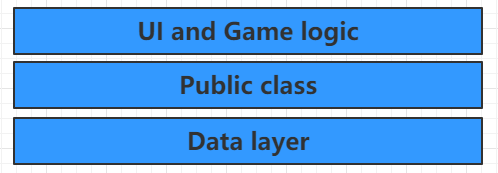
**Premise assumption：**

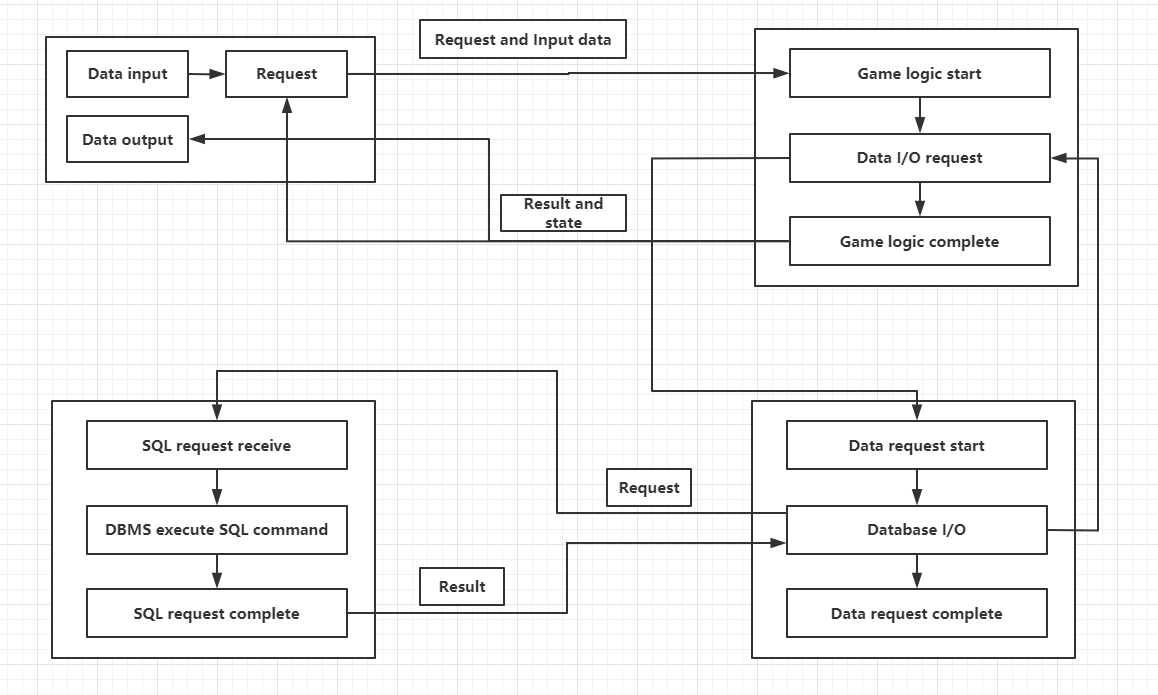
First of all, the video server is relatively simple and can be included in the game server, so we will only describe the game server in detail. Here we do general treatment of game types (online games, stand-alone games), all regarded as, the game server only stores or processes key data, such as player's basic information, victory, and defeat, past battle information, friends, permanent equipment And so on, that is, non-critical data in-game operations, and the main game entities do data storage or processing, such as the determination of attacks between players. For edge servers, they are equivalent to common computer hosts and can run multiple applications simultaneously. In addition to computing functions, they also have storage backup functions, such as game games like Jedi Survival or Overwatch, we put some data in a game on the edge server; or games such as open maps, where the player is located The data of a certain area is placed on the edge server, which we will explain in detail later.

**Element：**

Common class library layer, Data layer, UI and business logic layer

**Schematic diagram：**



The presentation layer is responsible for processing user input and output to the customer (for efficiency reasons, it may perform legality verification before transmitting user input upwards). The business logic is responsible for establishing the database connection, generating SQL statements to access the database according to the user's request, and returning the results to the client. The data layer is responsible for the actual database storage and retrieval responds to the data processing requests of the functional layer and returns the results to the functional layer.

**Interactive process：**

The game server focuses on the public library layer and data layer, while the edge server focuses on the UI and business logic layers. For computing, the edge server is equivalent to running the game application instead of the client and converts the calculated picture into a video stream, which is sent to the client and presented. Regarding data, the main role of the data on the edge server is to support players to complete the current game, rather than in permanent data storage, or support future game continuation, so the data storage here is only temporary, and is completed after around After the game, or a game after a period of time, or after entering another different game area, certain necessary data of the player will be transmitted to the game server, such as how many enemies are defeated, what equipment the player has retained and so on. In addition, it is also responsible for the logical processing of the client's screen display, that is, the UI. In contrast, the game server can intermittently update the edge server with some player-requested or necessary data without paying too much attention to the UI.

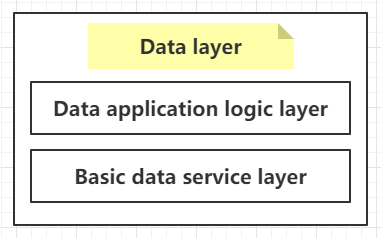
1. Public library layer

The public class library includes some imported components related to the existing system through the class library (not dependent on the current system)

The public class library is used as the bottom layer to implement other high-level layers. Since the public class library layer does not depend on the existing system, it can be referenced by all modules.

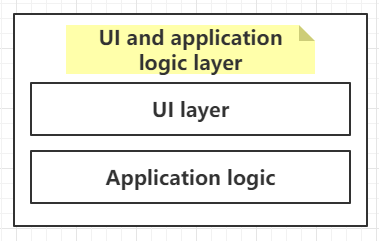
2. Data layer

Data layer processing is the data processing center of the system. The data layer can be subdivided into two small layers: basic data service layer and data business logic layer



Basic data services provide public basic services similar to data storage (database or file storage) (such as the three-layer Model layer and DAL layer, which provide data access interfaces and data corresponding entities), while the data business logic layer Provide data service interfaces for higher layers (UI and business logic layers) (similar to BLL layer, organize data forms for higher layers).

1. UI and business logic layer



The UI business logic layer can be subdivided into a business logic layer and a UI layer. The business logic layer mainly processes the interaction logic of the UI layer and the corresponding business logic. The UI layer and the business logic layer can be regarded as being on the same layer.

**Design reasons**：

In this form, the three layers are placed on different hardware systems, so the flexibility is very high, and it can adapt to the increase in the number of clients and the change in processing load. For example, when adding a new business process, a server with a loading function layer can be added accordingly. Therefore, the larger the system scale, the more obvious the advantages of this form. This is very suitable for games with a large number of players or games with high concurrency.

1. **Edge server architecture:**

Reference Peer-to-Peer Style，Centralized Topology

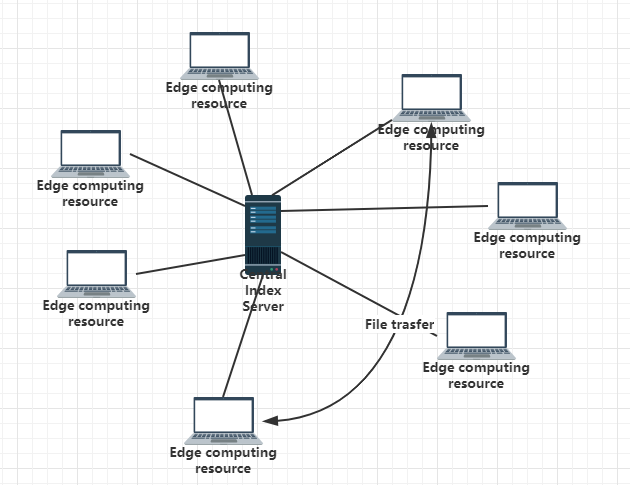
**Premise assumption：**

Multiple edge servers form a p2p network, and within a certain geographic area, it is controlled by a central server. The central server has a directory with connected edge servers, and regularly checks the status of the updated edge server. In special cases, the edge server can also be scheduled. On the edge server, it is divided into 90% of computing storage resources and 10% of storage resources.

**Element：**

Edge server, Central index server

**Schematic diagram：**



**Interactive process：**

*Edge server:*

*The computing storage resource is mainly responsible for processing the data interacting with the UI of the client, which is to convert the game screen into a video stream and send it to the client. It is responsible for ensuring the smoothness of the user screen and correct logical judgment. Both computing storage resources and storage resources store the data of the authorities' games or games within the current scope. To ensure that when this server is down, the player's current status data will not be lost. Even in the worst case, even if the current state of a resource is lost, there will be a backup of another resource and not all will be lost.*

*There is a delay detection between the edge server and the client. When the client encounters a delay, the edge server will quickly report to the central directory server. If the client delay does not recover within a very small period, the central server Find the neighboring server and transfer the information to the current edge server. This edge server can transfer the client data to the target server from the storage resource part to complete the server-client data migration. In addition to delay detection, there is also signal strength detection. This detection can solve the problem of customer movement. When the customer signal strength is weak and reaches a certain value, consistent with the above method, it will be quickly reported to the central directory server, and then the data will be migrated and transferred Client to the new edge server.*

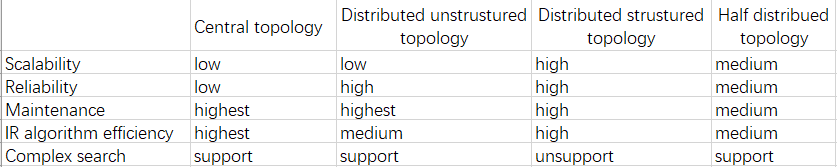
*When the load status of the edge server in the customer's area is relatively full if a new user wants to join the server, first determine whether the user wants to watch VR video or play VR game. If they watch the video, they can notify the central directory server and directly Connect to the nearest available edge server. If you are playing a VR game, it is determined whether there are users who watch VR videos. Because the real-time requirement for viewing VR videos is not high, we can make some compromises to users who watch videos. For example, in the calculation process before the new user joins, pre-process the video resource and place it on the storage resource, and notify the central directory server to find a recently available edge server. In this way, newly added users can replace the resources occupied by video computing, and the pre-stored videos are transferred to users by retrieving storage resources. When the user watching the video has a request to watch the new video, it will directly operate on the new edge server, which not only meets the user's need to watch the video but also ensures that the player who plays the VR game is linked to the fastest server.*

*Central index server：*

The central catalog server provides catalog query service for each program in the network, and the transmission content does not need to go through the central server. This network has a relatively simple structure. The central server is only responsible for scheduling or searching for edge servers, so the burden is greatly reduced. Due to the centralized management of the directory, it is an optional solution for network management and control. For example, when dealing with the downtime of an edge server, the central directory server can quickly find another edge server closest to the player, notify the player and the new edge server, directly link the player to it, and connect it to the previous edge server Data is migrated to the new edge server. The data migration here is the process mentioned above.

**Design reasons**：

As far as the p2p structure is concerned, robustness has the advantages of resistance to attack and high fault tolerance. Because the service is distributed among various nodes, the destruction of some nodes or the network has little effect on other parts. P2P networks can automatically adjust the overall topology when some nodes fail and maintain the connectivity of other nodes. Privacy protection, because the transmission of information is scattered between nodes without going through a centralized link, the possibility of users ’private information being eavesdropped and leaked is greatly reduced. Load balancing reduces the requirements on the computing power and storage capacity of traditional C / S structure servers. At the same time, because the resources are distributed in multiple nodes, the load balance of the entire network is better achieved. As far as the centralized topology is concerned, maintenance is simple and resource discovery efficiency is high. Because the discovery of resources depends on a centralized directory system, the discovery algorithm is flexible and efficient and can implement complex queries. Although the centralized topology has lower scalability and lower reliability than other structures, neither of these two is its main business. Its main business is to maintain good maintenance and can play a long-term role. The data storage and the discovery algorithm are highly efficient, and can quickly find the next available node to use the edge server when the edge server fails.



1. **Client game architecture:**

Reference Event Driven Style

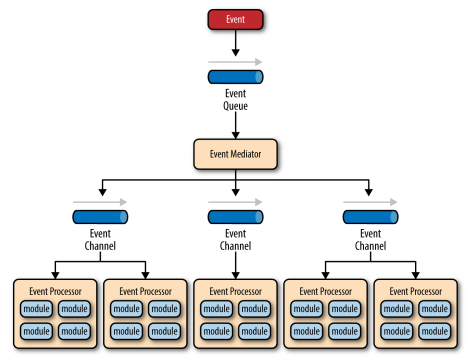
**Premise assumption：**

Similarly, the video aspect is considered to be included in the game, and only the game will be described here. The client game is mainly responsible for UI display, processing the received data, and docking the corresponding screen elements, thereby generating a response, and also changing the data processing screen according to the user's operation and transmitting necessary data to the edge server. In layman's terms, VR glasses are equivalent to the usual display, and the operators are the mouse and keyboard, and it is the signals generated by these operators that need to be transmitted.

**Element：**

Operator, Client, VR glasses

**Schematic diagram：**



**Interactive process：**

*Input:*

*The subversive concept of VR input devices is spatial meaning. It is different from traditional input devices in that the most direct meaning is to spatialize input parameters, which allows developers to track the user's handle spatial position and trajectory to create more possibilities.*

Menu Button：

The menu button is a unified functional button. According to the naming of HTC VIVE, there should be no more definitions that deviate from the concept of the menu in determining the function of the button. What is interesting is that in VR there are two controllers, divided into left and right hands. This allows us to differentiate the left and right-hand menu keys for calling out different interfaces, which can simplify the interface hierarchy depth better.

Trackpad：

Trackpad, as the name suggests, it is a touchpad. It provides key operations in four directions (up, down, left, and right), as well as more delicate sliding coordinate trajectory operations on the panel. In this way, we can use it as a positional movement tool modeled up, down, left and right, or as a fine adjustment tool for fine coordinates.

Trigger：

The trigger key provides a dynamic value between 0-1, resulting from the whole process of the user pulling the trigger. It is also the most frequently used button in the game, and its main function is to confirm the operation.

Grip Button：

The side buttons should be used as an auxiliary button function that is not commonly used in terms of location. Used for some necessary but not commonly used game behavior control.

The controller's space six degrees of freedom:

The six degrees of freedom of the controller is the only real VR concept input among the above input modes because it is three-dimensional. In simple terms, six degrees of freedom provides developers with a sense of the controller's rotation angle and the coordinates of the virtual space, which provides an input mode for the game's more three-dimensional concept. The use of six degrees of freedom will be combined with the above-mentioned operations to a greater extent.

## *Output：*

Force feedback output:

In HTC VIVE, the controller provides a vibration function, which can provide force feedback that we use for player game interaction. The vibration of the handle is not the first time it appears in the output mode of the game. Both the console game console and the mobile phone have a vibration function, and the application of the vibration function in the game is not the first day.

Image display output:

HMD (Head Mounted Display Device), which provides the concept of panoramic viewing virtual reality, that is, the screen display is three-dimensional. It includes headspace tracking and positioning, as well as a closed, larger field of view experience. This is different from the traditional game design (mirror design) because there will be countless mirrors in the degree of freedom relationship scenes, which naturally requires no visual dead ends in the design.

Audio output method:

Audio output is an extremely important issue in VR, and it is also a requirement for a three-dimensional state. The position of the audio playback point, the player's head model, and the environment state determine how the audio is transmitted to the user's ears.

**Design reasons**：

Support software reuse, easy to achieve concurrent processing. With good scalability, new components can be introduced through registration without affecting existing components. Can simplify the customer code, simplify the customer's equipment needs, a low configuration can still experience the effects that require the high calculation to achieve. Generally speaking, it is suitable for the operation of VR games, and also for viewing VR videos.

1. **Handling other issues:**

Reference Model-View-Controller (MVC)

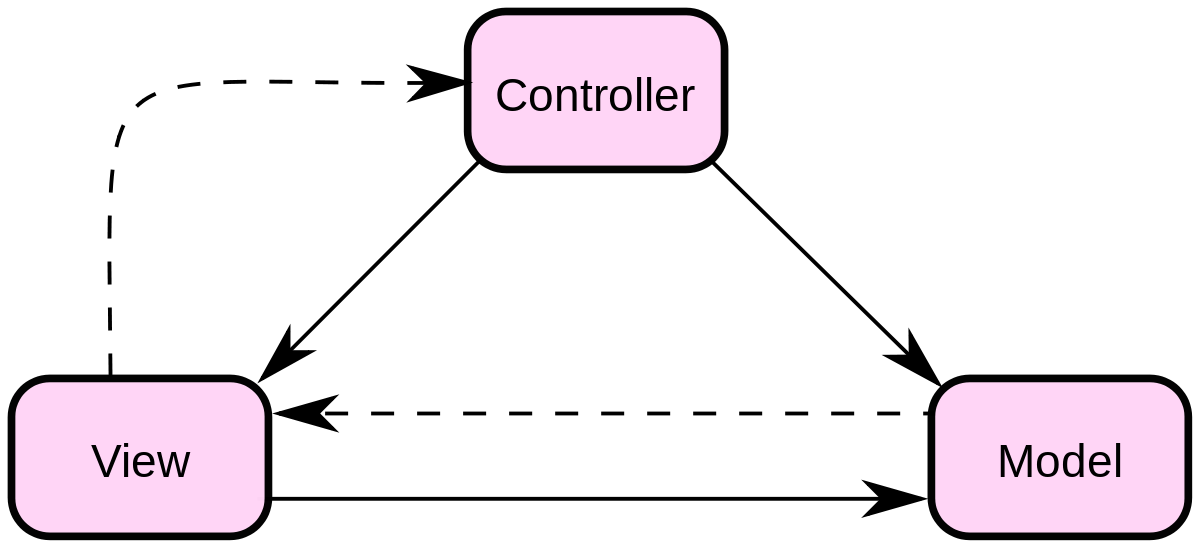
**Premise assumption：**

The client serves as the view layer, the edge server serves as the controller layer, and the game video server serves as the model layer.

**Element：**

Edge Server, Client, Game Video Server

**Schematic diagram：**



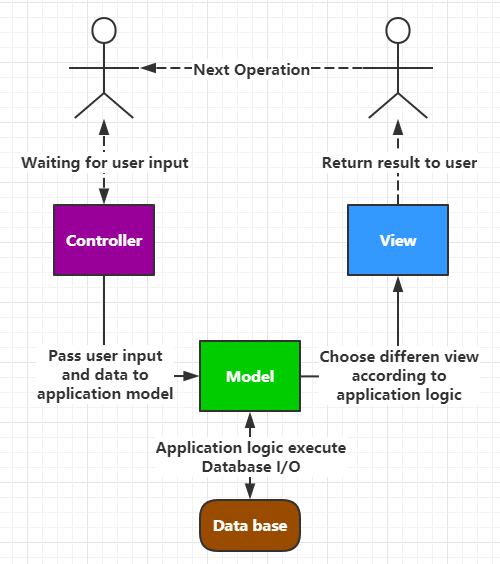
Model - The model represents an object or JAVA POJO that accesses data. It can also have logic to update the controller when data changes.

View - The view represents the visualization of the data contained in the model.

Controller - The controller acts on the model and the view. It controls the flow of data to model objects and updates the view as data changes. It separates the view from the model.

**Interactive process：**

The client serves as the view layer, the edge server serves as the controller layer, and the game video server serves as the model layer.



**Design reasons**：

In the course of normal games, the client's screen will not be obviously stuck, to ensure the smooth operation of the screen. When the game server fails, the edge server can continue to maintain the operation of the picture, that is, the transmission of the video stream. To ensure that the client screen will not stop, or interfere with the normal game, this mode can be used when the model layer has problems, the view and controller layers still retain some functions, allowing players to get a good experience. The coupling is low, and the view layer and the business layer are separated, which allows the view layer code to be changed without recompiling the model and controller code. Similarly, the change of an application's business process or business rules only needs to change the MVC model layer. High reusability: The MVC pattern allows various views to be used to access the same server-side code because multiple views can share a model. Fast deployment and low life cycle costs, MVC reduces the technical content of developing and maintaining user interfaces. Using the MVC model has greatly reduced the development time.

**Design Principles**

Since we have adopted a variety of architectural styles, it will be a clear way to analyze the design principles of our project according to different styles.

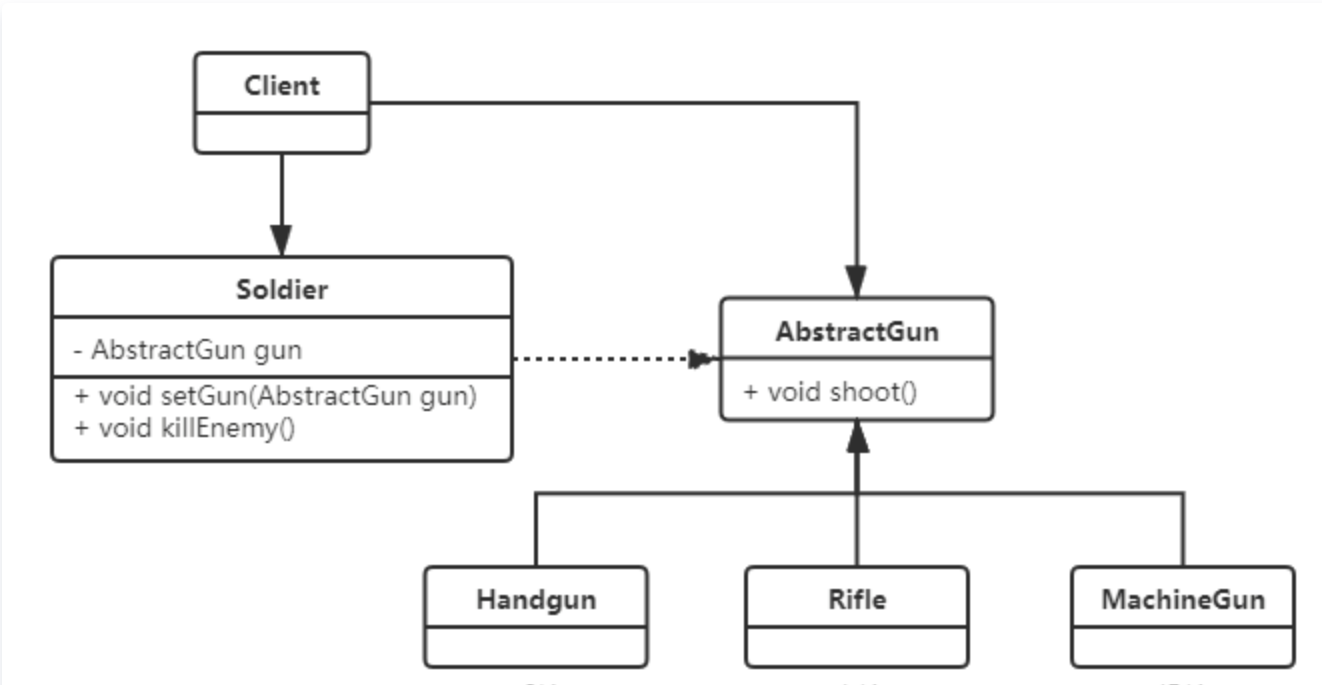
**Client-server Style**

Client-server architecture is a kind of two-layer architecture normally. In this structure, the application is divided into two parts: the server part and the client part. The server part shares the information and functions to multiple users and performs background services, such as controlling the operation of the shared database, and part of the data is exclusively for the user, responsible for performing the front desk functions. It also has powerful functions in error prompts, online help, etc. , What is more, it can switch freely between subprograms.

In our project, we divide it into three-layer which is client, edge server, and main server. We designed the edge server as a middle layer to connect the client and main server and it is not only a data transfer station but also a deal with some logical calculation. That means the edge server will calculate or filter the huge data firstly in order to change the data to be the handy result for the main server when it receives the data from the client and then sends the result. When the data is returned, the edge server will filter the necessary data and send it to the client. In this way, it can reduce the delay caused by the interaction between client and server or internet congestion so that the game can react faster. At the same time, it can reduce the risk of main server overload and keep the security of data because of the less data transmission. There is another advantage is that the edge server can store and resend the data when the main server down which prevents the data from losing.

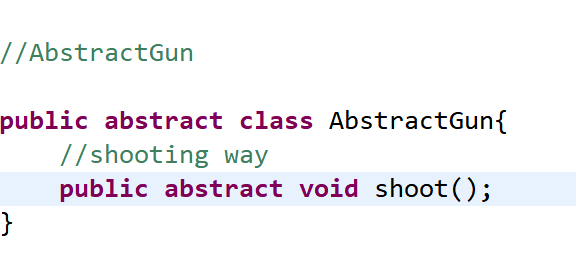
The main features using client-server architecture is that this structure has strong interaction and fast response. The Liskov Substitution Principle means as long as the parent class can appear, the child class can appear, and the replacement of the child class will not produce any errors or exceptions. This guarantee strong interaction and fast response. The following explains the Liskov Substitution Principle with a shooting game in a different way.

A subclass must fully implement the method of the parent class. When we design a system, we often define an interface or abstract class, and then code to achieve, the calling class is directly passed into the interface or abstract class. Such as when we describe the gun in the CS

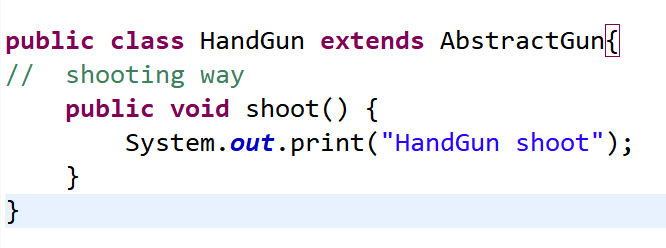


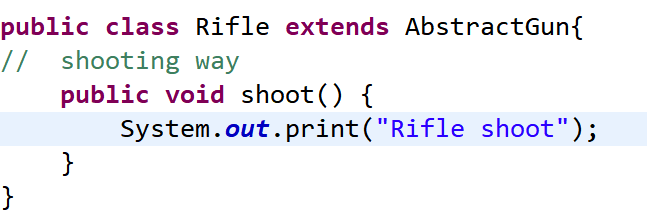
The main responsibility of the gun is shooting. How to shoot is implemented in each specific subclass. The soldier class Soldier defines a method killEnemy, which uses the gun to kill the enemy. The specific gun is only known when it is called.

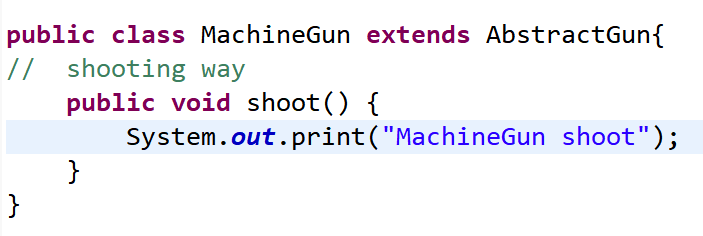
AbstractGun source code:



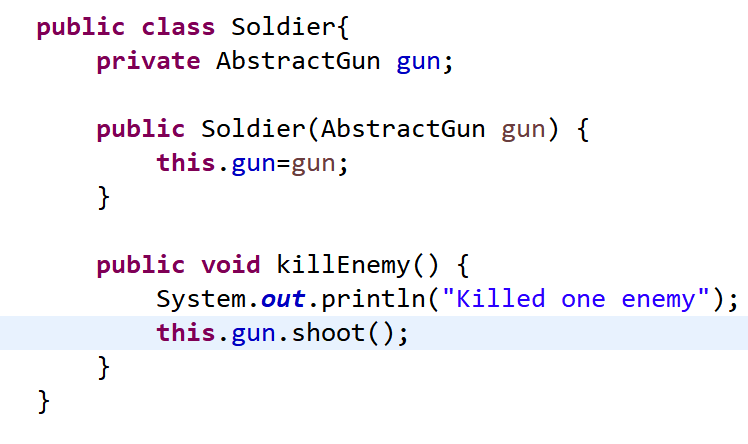
The code of Handgun, Rifle, Machinegun implement:



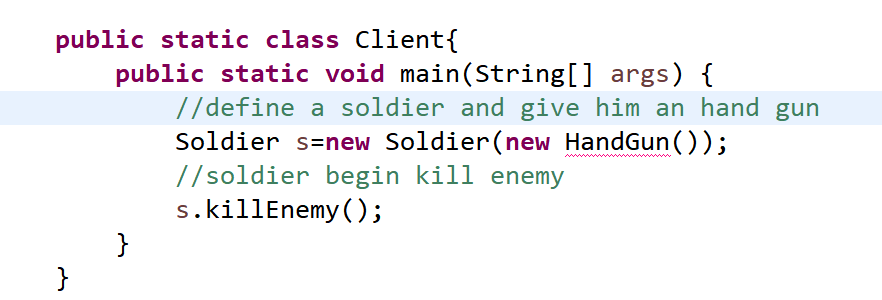




The source code of solider: The gun used in the killEnemy method of the soldier class is abstract, and the specific time of the gun needs to be determined by the client (Client) calling the constructor of the Soldier.



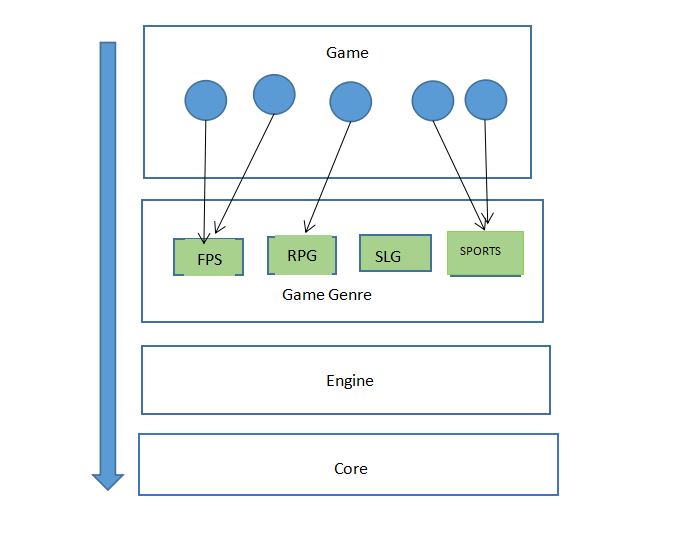
The code of client:



When calling other classes in a class, you must use the parent class or interface. If you cannot use the parent class or interface, it means that the design of the class has violated the LSP principle. This principle maintains the code sharing, reducing the workload of creating classes, each subclass has the attributes and methods of the parent class and also improve code reuse and scalability.

**Layered Architecture**

Layered architecture is a structure that divides a complex system into layers. Each layer does its own thing, and each layer is one-way dependent. The game is also designed according to the layered design ideas. Games can be divided into core layer, engine layer, game type layer, game layer. This layered architecture design can help us deconstruct complex systems, to achieve the singularity of each subsystem or module. Each layer in the layered architecture is highly cohesive, and each layer should be loosely coupled with the layers below it.



The layering in the game can be divided into five layers

1. Collect information around

The first layer is equivalent to a global perceptron that can store information about players around. Just like in shooting games, we need to sense the players around us and update the real-time status of players

2. Update decision layer-what to do

The decision-making layer is responsible for deciding what the agent should do at this time, for example, if I want to go to a certain position, I want to attack, release skills, etc. Based on all the current information, make a "what" decision The content will be encapsulated in a "request" structure and continue to pass down

3. Update behavior layer

This layer mainly adjusts multiple behaviors based on the "request" information of the previous layer to complete the target request

Open-close Principle

Introduce the concept of open and closed layers, which can help us define the relationship between different layers and the transfer of requests. Pass the necessary information between each layer in the layered architecture. If a layer is marked as open, it means that the request can go directly through this layer to its next layer

**Peer-to-Peer Style**

Peer-to-Peer Style

Peer-to-peer style means that every node can be a client or a server, they receive data from others and they also can send data to someone else.

In our project, we use a peer-to-peer style on our server design. We use several edge servers to build the peer-to-peer network, and there is a central index server that is used to help control those edge servers. This central index server stores all the index of every edge server, and the central index server will check edge server status to decide whether the edge server needs dispatching.

The edge servers are used to dealing with the data that interact with clients that are controlled by users and convert the game screen into a video stream and send it to the clients.

In our design, 90% of the edge server resources will be used on calculating the data, and others will be used on storing data. These stored data ensure that users’ status will not be lost if the edge server that the user connection is shut down. If the edge server is shut down, the central index server will notice that and search the index to find out the nearest edge server to the user so that the shutdown edge server can send the file that storing user data to a new server and user can continue the game. In this process, the edge server that shuts down can be taken as a client that sends data to the other edge server. This is a p2p style design.

The central index server is only responsible for scheduling or searching edge servers, so the burden is greatly reduced. If one of the edge servers is down, the central index server is able to find the nearest edge server and connect the user with it.

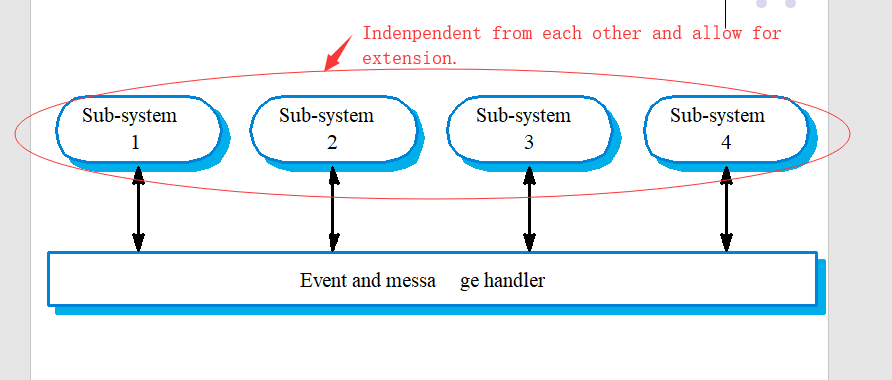
This kind of p2p network is strong and high fault tolerance. Load balancing reduces the requirements of computing power and storage capacity of traditional C / S structure servers. At the same time, because resources are distributed in multiple nodes, the load balancing of the whole network is better realized.

Law of Demeter

Our design on edge servers, central index server, and clients follow the law of Demeter. The clients that are controlled by users only communicate with certain edge servers. Edge server sends data to the central game server and receives data from it. There is no interaction between clients and the central game server.

**Event Driven Style**

The event-driven architecture has a central event handler for receiving event when an event occurs, the handler should take the event message and broadcast it to all its subsystem, it is up to the sub-system to decide whether it can handle this event. One event can be handled by multiple sub-systems concurrently. This kind of architecture should have the following characteristic, functional decoupling, maintainable, extendable, robust, and reusable.



Following [the Single Responsibility Principle](http://blog.csdn.net/vking_wang/article/details/8450886" \t "_blank) to achieve functional decoupling, each subsystem should only be designed for handling one type of event. So that changes to one type of event handling do not affect others. This kind of design also has a benefit when one subsystem is down, the remaining system can still be running.

Every sub-system should regard the same from the view of an event handler so that it would inform every sub-system when an event occurs, [Liskov Substitution Principle](http://blog.csdn.net/vking_wang/article/details/8455621" \t "_blank) can be applied to achieve this purpose.

Applying [Interface Segregation Principle](http://blog.csdn.net/vking_wang/article/details/8455631" \t "_blank) to the interface between event handler and sub-system to remove the dependency between event handler class and sub-system implementation.

**Model-View-Controller (MVC)**

We use the “Model View Controller” to divide business logic and interface. We use the game server as a model layer, the client as a view layer, edge server as a controller layer. The client device is responsible for displaying the interface and uploading user operations to the edge server. The program runs on edge servers and processes users’ operation data. Edge server as controller upload game data to the game server and return video data to the user. Game server process data from edge servers and return to edge servers.

Based on the Law of Demeter, there is no interaction between the game server and client interface. Edge servers as a medium of communication between them. It makes low coupling between layers and each layer has high independence. This allows one of the layers to be edited without changing other layers and a different view can reuse the same server codes. It increases code Reusability and system scalability. Different people focus on different layers in development. It reduces development time, facilitates management, and subsequent maintenance.

**Design Pattern**