Fallback & Join Handler

Introduction

The `IJoinHandler` and `IReconnectHandler` interfaces are providing plugins with an easy API to change some of the most important parts of your network's behaviour.

Setting the handlers

The ProxyServer object holds one instance of each interface, accessible using `ServerInfo#setReconnectHandler(IReconnectHandler)` and `ServerInfo#setJoinHandler(IJoinHandler)`. **Setting them to null will cause massive issues**. Rather implement NO-OP handlers.

IJoinHandler

The `IJoinHandler` interface only requires one method to be implemented, namely the `determineServer(ProxiedPlayer)` method. This method is called whenever a player connects **to the proxy**.
This method can only return an instance of `ServerInfo` or `null`. If the method returns a ServerInfo object, the player's initial connection while be established to that ServerInfo.
If null is returned, the player will be disconnected from the proxy as there is no server available for it to use.

Use case

This method can be used perfectly if you are having multiple lobby-instances in your network (for example). You can then implement f.e. a Round-Robin determination model to evenly distribute players over your lobby-instances. You could also send player to servers depending on where they were last, or depending on any other set of conditions that you would like to enforce.

IReconnectHandler

The `IReconnectHandler` is called whenever a player is disconnected from a downstream server. This can be caused by a kick, a server shutdown or even a Proxy <-> Downstream timeout. This method will then be called in order to determine the future of the player.
The interface only requires the implementation of the `getFallbackServer(ProxiedPlayer, ServerInfo, String)` method, where the ServerInfo is the information holder of the downstream server the player was disconnected from, and the String is the reason the player was disconnected with (if given).

Use case

In some network concepts, you could want to prevent players from being kicked from the network. This could be the case f.e. for minigames-servers where servers might crash / close down, but you'd still want the player to stay on the proxy but instead be sent to your lobby. In that case you'd just return the ServerInfo of the lobby to transfer the player to.

Important is that you can filter this input by using the `kickMessage` method parameter. With that you could catch players which are being kicked for "Internal Server Error" or "Server closed", but still completly disconnect players that are kicked for "You are banned" or "You have been kicked". If you are returning a ServerInfo object, you can also send the player titles, text messages or other types of output to notify him of the disconnect.

Important note

This note is regarding the performance of this system. You **should not** execute any time-expensive code in either of these methods, as that causes some players to lag while the code is running. Instead, try to run f.e. SQL queries periodically in the background, store the results easily usable in-memory and access those results in the method.

Examples

Setting the handlers

```
ProxyServer server = ProxyServer.getInstance();
```

```
IReconnectHandler reconnectHandler = new MyCustomReconnectHandler();

IJoinHandler joinHandler = new MyCustomJoinHandler();

server.setJoinHandler(joinHandler);

server.setReconnectHandler(reconnectHandler);
```

Custom IJoinHandler

[A simple example plugin](#)

```java
public class VanillaJoinHandler implements IJoinHandler {

    private final ProxyServer server;

    public VanillaJoinHandler(ProxyServer server) {
        this.server = server;
    }


    @Override
    public ServerInfo determineServer(ProxiedPlayer player) {
        return
this.server.getServer(this.server.getConfiguration().getPriorities().get(0));
    }
}
```

Excerpt of the WaterdogPE code.
Returns the first server from the server priority list.

Custom IReconnectHandler

```java
public class TestFallbackHandler implements IReconnectHandler {
    @Override
    public ServerInfo getFallbackServer(ProxiedPlayer proxiedPlayer, ServerInfo serverInfo,
String s) {
        for(ServerInfo i : proxiedPlayer.getProxy().getServers()){
            if(!i.getServerName().equals(serverInfo.getServerName())){
                return i;
            }
```

```
        }
        return null;
    }
}
```

Tries to find a server which is not the server the player was kicked from. If none was found, return null.