|  |  |
| --- | --- |
| Оригинал (EN) | Перевод (RU) |
| **Using the Transport Layer API**  In addition to the High Level networking API that we provide - which provides easy-to-use systems for managing your players, networked GameObjects, and other common requirements - we also give access to a lower level API called the “Transport Layer”. This provides you with the ability to build your own networking systems at a lower level, which can be useful if you have more specific or advanced requirements for your game’s networking.  The Transport Layer is a thin layer working on top of the operating system’s sockets-based networking. It’s capable of sending and receiving messages represented as arrays of bytes, and offers a number of different “quality of service” options to suit different scenarios. It is focused on flexibility and performance, and exposes an API within the UnityEngine.Networking.NetworkTransport class.  The Transport Layer supports base services for network communication. These base services include:   * Establishing Connections * Communicating using a variety of “quality of services” * Flow control * Base statistics * Additional services like communication via relay server or local discovery   The Transport Layer can use two protocols: UDP for generic communications, and WebSockets for WebGL. To use the Transport Layer directly, the typical workflow would be as follows:   1. Initialize the Network Transport Layer 2. Configure network topology 3. Create Host 4. Start communication (handling connections and sending/receiving messages) 5. Shut down library after use.   **Initializing the Network Transport Layer**  When initializing the Network Transport Layer, you can choose between the default initialization, with no arguments, or you can provide parameters which control the overall behaviour of the network layer, such as the maximum packet size and the thread timout limit.  // Initializing the Transport Layer with no arguments (default settings)  NetworkTransport.Init();  // An example of initializing the Transport Layer with custom settings  GlobalConfig gConfig = new GlobalConfig();  gConfig.MaxPacketSize = 500;  NetworkTransport.Init(gConfig);  In the 2nd example above, the Transport Layer is initialized with a custom “MaxPacketSize” value specified of 500. Custom Init values should only be used if you have an unusual networking environment and are familiar with the specific settings you need. As a rule of thumb, if you are developing a typical multiplayer game designed to be played across the internet, the default Init() settings with no arguments should be appropriate.  **Configuration**  The next step is configuration of connection between peers. You may want to define several communication channels, each with a different quality of service level specified to suit the specific types of messages that you want to send, and their relative importance within your game.  ConnectionConfig config = new ConnectionConfig();  int myReiliableChannelId = config.AddChannel(QosType.Reliable);  int myUnreliableChannelId = config.AddChannel(QosType.Unreliable);  In the example above, we define two communication channels with different quality of service values. “QosType.Reliable” will deliver message and assure that the message is delivered, while “QosType.Unreliable” will send message without any assurance, but will do this faster.  It’s also possible to specify configuration settings specifically for each connection, by adjusting properties on the ConnectionConfig object. However, when making a connection from one client to another, the settings should be the same for both connected peers or the connection will fail with a CRCMismatch error.  **Topology**  The final step of network configuration is topology definition. Network topology defines how many connections allowed and what connection configuration will used:  HostTopology topology = new HostTopology(config, 10);  Here we created topology which allow up to 10 connections, each of them will configured by parameters defines in previous step. | **Использование Транспортного уровня интерфейса прикладного программирования (API)**  В дополнение к предоставляемому нами высокоуровневому сетевому Интерфейсу прикладного программирования (API), позволяющему использовать простые системы управления игроками, сетевые игровые объекты GameObjects и отвечающему прочим обычным техническим требованиям, мы также обеспечиваем доступ к низкоуровневому API, называемому "Транспортный уровень". Это дает Вам возможность построения собственных сетевых систем на более низком уровне, что может быть удобно при наличии у вас специфических или повышенных требований к сетевой игре.  Транспортный уровень - это тонкая прослойка, работающая поверх основанной на сокетах сетевой операционной системы. Он может отправлять и получать сообщения, представленные в виде массивов байтов и предлагает ряд различных "по качеству обслуживания" опций, подходящих к различным сценариям. Он нацелен на гибкость и производительность и предоставляет Интерфейс прикладного программирования (API) внутри класса UnityEngine.Networking.NetworkTransport.  Транспортный уровень поддерживает базовые сервисы для сетевых коммуникаций. Эти базовые сервисы включают в себя:   * Установку соединений * Обмен данными с использованием различного "качества обслуживания" * Управление потоком * Базовую статистику * Дополнительные сервисы типа обмена данными через сервер ретрансляции или локальное обнаружение   Транспортный уровень может использовать два протокола: Протокол пользовательских дейтаграмм (UDP) для универсальной связи и соединения WebSocket для WebGL. Для непосредственного использования Транспортного уровня необходимо выполнить стандартную последовательность действий:   1. Инициализируйте Сетевой транспортный уровень. 2. Сконфигурируйте топологию сети 3. Создайте хост 4. Начните коммуникацию (обработку соединений и отправку/получение сообщений) 5. Отключите библиотеку после использования.   **Инициализация Сетевого транспортного уровня.**  При инициализации Сетевого транспортного уровня Вы можете выбрать инициализацию по умолчанию, без задания параметров, или Вы можете указать параметры, которые управляют общим поведением сетевого уровня, таким как максимальный размер пакета и лимит ожидания потока.  //Инициализация Транспортного уровня без задания аргументов (настройки по умолчанию)  NetworkTransport.Init();  // Пример инициализации Транспортного уровня с пользовательскими настройками  GlobalConfig gConfig = new GlobalConfig();  gConfig.MaxPacketSize = 500;  NetworkTransport.Init(gConfig);  Во втором примере Транспортный уровень инициализируется с заданным пользователем значением "MaxPacketSize" равным 500 Пользовательскими значениями следует пользоваться, только если у вас нестандартная сетевая среда и вы знакомы со специфическими настройками, которые вам нужны. На практике, при разработке типичной многопользовательской онлайновой игры следует использовать настройки Init() по умолчанию без задания аргументов.  **Конфигурация**  Следующий шаг - конфигурация одноранговой связи. Возможно, вы захотите определить несколько каналов связи, каждый с своим уровнем "качества обслуживания", подходящим определенному типу отправляемых вами сообщений, а также их относительную важность в вашей игре.  ConnectionConfig config = new ConnectionConfig();  int myReiliableChannelId = config.AddChannel(QosType.Reliable);  int myUnreliableChannelId = config.AddChannel(QosType.Unreliable);  В приведенном выше примере мы определили два канала связи с различными значениями качества обслуживания. “QosType.Reliable” отправляет сообщение и гарантирует его доставку, в то время как “QosType.Unreliable” отправит сообщение без какой-либо гарантии, но сделает это быстрее.  Также возможно задать настройки конфигурации для каждого соединения, настроив свойства объекта ConnectionConfig. Однако, при соединении двух клиентов, оба подключаемых пира должны обладать одинаковыми настройками, в противном случае соединения не произойдет, и система выдаст ошибку CRCMismatch.  **Топология**  Последний шаг конфигурации сети – определение топологии. Топология сети определяет, сколько может быть соединений, и какая их конфигурация будет использоваться:  HostTopology topology = new HostTopology(config, 10);  В данном примере мы создали топологию, позволяющую использовать до 10 соединений, каждое из которых конфигурировано с помощью параметров, определенных на предыдущем этапе. |