Firebase Java Connector

User Manual

Lars J. Nilsson

Firebase Java Connector: User Manual

Lars J. Nilsson

Firebase Java Connector 1.10.5-SNAPSHOT

Published 06/27/2014

Table of Contents

Chapter 1. About Cubeia

Cubeia Ltd

Cubeia Ltd is a distributed systems expert company, registered in England and operating through our office in Stockholm, Sweden. We provide scalable, high availability systems and consultation based on our long experience in the gambling and Internet application industry.

Our main product, Firebase, is a game agnostic, high availability, scalable platform for multiplayer online games. It is developed by Cubeia Ltd and was built from the start with the gaming industry in mind. It provides a simple API for game development using event-driven messaging and libraries for point-to-point client to server communication.

Firebase is a server platform for developing and running online games. It scales from small installations to extremely large, it is built to stand up to hard traffic and can be used for almost any type of game.

Contact

For further information, please contact Cubeia Ltd UK Filial in Stockholm. Bugs should be reported to the Cubeia online support forums. If you do not have access to these forums please contact Cubeia Ltd at the address below.

Cubeia Ltd, UK Fillial
Stora Nygatan 33
11127 Stockholm
Sweden

Email: info (at the cubeia domain)

Corporate Homepage: http://www.cubeia.com

Community Community: http://www.cubeia.org

Chapter 2. Firebase Java Connector

Overview

The Java Connector is a simple library used to connect to, and send, native Firebase packages from a client to a Firebase server. It support native Firebase encryption and connection handshake.

Connecting

The connectors uses constructor injection of the encryption type to use, and optional handshake parameters.

```
[...]

/*
 * Example parameters
 */
int port = 4123;
String host = "localhost";
Encryption enc = Encryption.NONE;

/*
 * Create connector and connect
 */
Connector connector = new SocketConnector(host, port, enc);
connector.connect();

[...]
```

If a handshake is required by the server, this will have to be included in the constructor.

Receiving Packets

To receive packets from the server, a `PacketListener` should be implemented and added to the connector before the connection is established. Like so:

```
[...]

/*
 * Example parameters
 */
int port = 4123;
String host = "localhost";
Encryption enc = Encryption.NONE;

/*
 * Create connector
 */
Connector connector = new SocketConnector(host, port, enc);

/*
 * Add listener
 */
connector.addListener(new PacketListener() {

    @Override
    public void packetRecieved(ProtocolObject packet) {
        // Do something here...
    }
});

/*
 * Connect
 */
connector.connect();

[...]
```

Packets are delivered to the listener using a dedicated thread. In other words, there will never be concurrent delivery of multiple packets. Also, the packet ordering till be preserved.

Sending Packets

Sending packets is done via the `send(ProtocolObject)` method:

```
[...]

ProtocolObject o = //...
connector.send(o);

[...]
```

Encryption

The connector supports Java native SSL and Firebase native packet encryption. The encryption type is is passed as an enumeration to the constructor. The enumeration has the following types:

`NONE`	No encryption. This is the default value.
`NAIVE_SSL`	SSL but where any server certificate is accepted. This is un-secure, but useful for development.
`SSL`	Full SSL, this is configured with Java system properties.
`FIREBASE_NATIVE`	Native Firebase packet encryption.

The naive SSL should only be used for testing and development.

Should you need to configure SSL outside the system properties, you can extend the `SocketConnector` and override its `getSSLSocketFactory(Encryption)` method.

If Firebase native packet encryption is used, the connector will try to wait for the encryption key exchange to finish before returning from the `connect()` method. This interval is defaulted to 5 seconds (5000 millis) but can be configured before connect is called, like so:

```
[...]
```

```
/*
 * Example parameters
 */
int port = 4123;
String host = "localhost";
Encryption enc = Encryption.FIREBAE_NATIVE;
long wait = 10000; // 10 secs

/*
 * Create connector, set key exchange wait
 * period and connect
 */
Connector connector = new SocketConnector(host, port, enc);
((SocketConnector)connector).setKeyExchangeWait(wait);
connector.connect();

[...]
```

Handling IO errors

By default all errors will simply be logged by the connector. To handle IO errors on read, override the `handleReadException(Exception)` method in a subclass.

More Information

Please refer to the community Wiki and message boards for more information about this component: http://www.cubeia.org