Zhouxuan-C / Blog

【未完】kRPC -- Java 客户端 #44

Edit New issue

Open Zhouxuan-C opened this issue 6 days ago · 0 comments

Zhouxuan-C commented 6 days ago • edited Owner

原文链接：https://krpc.github.io/krpc/java/client.html

Java 客户端

此客户端提供一套 Java API 用于与 kRPC服务器进行交互。你可以 从 Github下载 jar包。使用 jar包，需要 JDK 1.8。

获取依赖

kRPC 需要 protobuf 和 javatuples 支持，protobuf 可以通过 Maven找到。但是，protobuf 请安装 3.0及以上版本，kRPC与 2.x版本不兼容。

举个栗子，下面的代码展示连接服务器，获取 (查询) 版本并打印出来。

```
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.KRPC;

import java.io.IOException;

public class Basic {
  public static void main(String[] args) throws IOException, RPCException {
    Connection connection = Connection.newInstance();
    KRPC krpc = KRPC.newInstance(connection);
    System.out.println("Connected to kRPC version " + krpc.getStatus().getVersion());
    connection.close();
  }
}
```

连接服务器

想连接服务器，调用 `Connection.newInsrtance()` ，该方法返回一个连接对象。之后所有与服务器的交互都是通过这个对象。创建对象连接服务器时，如果不给任何参数，它将连接本地机器上的默认端口。当然你也指定别的连接选项，也可以给连接起个好♂玩的名字，见代码：

```
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.KRPC;

import java.io.IOException;

public class Connecting {
  public static void main(String[] args) throws IOException, RPCException {
    Connection connection = Connection.newInstance("Remote example", "my.domain.name", 1000, 1001);
    System.out.println(KRPC.newInstance(connection).getStatus().getVersion());
    connection.close();
  }
}
```

调用"远程"程序

kRPC 提供了些程序 (包)，你可以直接调用。这些程序被放在 `krpc.client.service` 中。比如，所有 `SpaceCenter` 提供的方法都被放在 `krpc.client.services.SpaceCenter` 类中。

Assignees
No one—assign yourself

Labels
游戏

Projects
KSP in 游戏

Milestone
No milestone

Notifications

1 participant

想使用这些服务，你需要先实例化它，然后就能调用它的方法和属性啦。下面的代码就展示了，实例化 `SpaceCenter` 并调用 `krpc.client.services.SpaceCenter.SpaceCenter.getActiveVessel()` 来获取一个对象代表活跃的飞船 (to get an object representing the active vessel)，然后设置了飞船的名字，顺便输出了它的高度。

```
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.Vessel;

import java.io.IOException;

public class RemoteProcedures {
  public static void main(String[] args) throws IOException, RPCException {
    Connection connection = Connection.newInstance("Vessel Name");
    SpaceCenter spaceCenter = SpaceCenter.newInstance(connection);
    Vessel vessel = spaceCenter.getActiveVessel();
    System.out.println(vessel.getName());
    connection.close();
  }
}
```

数据流

kRPC 常见的操作就是连续不断地从游戏中获取数据。在不使用数据流的情况下，我们是反复调用"远程"程序 (就是 kRPC提供的方法啦)。比如下面的代码，就是使用循环反复调用方法，然后输出飞船的位置。

```
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.services.KRPC;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.ReferenceFrame;
import krpc.client.services.SpaceCenter.Vessel;

import java.io.IOException;

public class Streaming1 {
  public static void main(String[] args) throws IOException, RPCException {
    Connection connection = Connection.newInstance();
    SpaceCenter spaceCenter = SpaceCenter.newInstance(connection);
    Vessel vessel = spaceCenter.getActiveVessel();
    ReferenceFrame refframe = vessel.getOrbit().getBody().getReferenceFrame();
    while (true) {
      System.out.println(vessel.position(refframe));
    }
  }
}
```

上面这种操作会在服务器和客户端之间，反复的发送、接受信息，这么做的坏处就是需要大量的通信开销。kRPC提供了非常高效的方法，来解决这个问题，叫流 (streams)。

流会服务器上反复执行一个过程 (使用固定的形参) 并将结果返回给客户端。这么做只要发送一个消息给服务器就能建立一个流，然后服务器连续发送数据到客户端，直到流被关闭。下面的例子和上面代码的功能一样：

```
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.Stream;
import krpc.client.StreamException;
import krpc.client.services.KRPC;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.ReferenceFrame;
import krpc.client.services.SpaceCenter.Vessel;

import org.javatuples.Triplet;

import java.io.IOException;

public class Streaming2 {
  public static void main(String[] args) throws IOException, RPCException, StreamException {
    Connection connection = Connection.newInstance();
    SpaceCenter spaceCenter = SpaceCenter.newInstance(connection);
    Vessel vessel = spaceCenter.getActiveVessel();
    ReferenceFrame refframe = vessel.getOrbit().getBody().getReferenceFrame();
    Stream<Triplet<Double,Double,Double>> vesselStream = connection.addStream(vessel, "position", r
```

```
    while (true) {
      System.out.println(vesselStream.get());
    }
  }
}
```

它调用 `Connection.addStream` 使得在程序启动时就创建一个流，接下来反复输出从流得到的位置信息。流会在关闭客户端时自动断开连接。

A stream can be created for any method call by calling `Connection.addStream` and passing it information about which method to stream. The example above passes a remote object, the name of the method to call, followed by the arguments to pass to the method (if any). `Connection.addStream` 返回一个 Stream类型的流对象。流中最近的值可以调用Stream.get获得。调用 `Stream.remove` 可以停止并删除一个流。所有流都会在断开连接时自动停止。

流的同步

kRPC中常见的用法就是等待一个方法返回值，或等待某个属性的值改变，然后采取一些行动。对此，kRPC提供了两种高效机制，监视器(condition variables) 和 回调(callbacks)。

每个流都有一个与之相关的条件变量，只要流的值有变化就会通知它。这么做可以使当前线程阻塞，直到流中的数值发送变化。下面这个例子会一直等待到游戏中的放弃按钮被按下，就是等到 krpc.client.services.SpaceCenter.Control.getAbort()的值为 true，

```
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.Stream;
import krpc.client.StreamException;
import krpc.client.services.KRPC;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.Control;

import java.io.IOException;

public class ConditionVariables {
  public static void main(String[] args) throws IOException, RPCException, StreamException {
    Connection connection = Connection.newInstance();
    SpaceCenter spaceCenter = SpaceCenter.newInstance(connection);
    Control control = spaceCenter.getActiveVessel().getControl();
    Stream<Boolean> abort = connection.addStream(control, "getAbort");
    synchronized (abort.getCondition()) {
      while (!abort.get()) {
        abort.waitForUpdate();
      }
    }
  }
}
```

上面的代码创建一个流，并获取流的条件变量的锁 (同步代码块)，然后反复检查，直到为 true时离开循环。循环体内调用 `waitForUpdate` ，这会使得程序堵塞，直到值发生变化。也可以避免循环 spinning，并且还不消耗任何资源。

The stream does not start receiving updates until the first call to waitForUpdate. This means that the example code will not miss any updates to the streams value, as it will have already locked the condition variable before the first stream update is received.

流允许你创建回调函数，当流的值改变的时候去调用。回调函数应该有个形参存放流的新值，且回调函数不应返回任何东西。举个栗子，下面的代码注册了两个回调函数，当 `krpc.client.services.SpaceCenter.Control.getAbort()` 的返回值改变时调用。

```
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.Stream;
import krpc.client.StreamException;
import krpc.client.services.KRPC;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.Control;

import java.io.IOException;

public class Callbacks {
  public static void main(String[] args) throws IOException, RPCException, StreamException {
    Connection connection = Connection.newInstance();
```

```
    SpaceCenter spaceCenter = SpaceCenter.newInstance(connection);
    Control control = spaceCenter.getActiveVessel().getControl();
    Stream<Boolean> abort = connection.addStream(control, "getAbort");
    abort.addCallback(
      (Boolean x) -> {
        System.out.println("Abort 1 called with a value of " + x);
      });
    abort.addCallback(
      (Boolean x) -> {
        System.out.println("Abort 2 called with a value of " + x);
      });
    abort.start();

    // Keep the program running...
    while (true) {
    }
  }
}
```

Note：When a stream is created it does not start receiving updates until start is called. This is implicitly called when accessing the value of a stream, but as this example does not do this an explicit call to start is required. The callbacks are registered before the call to start so that stream updates are not missed. The callback function may be called from a different thread to that which created the stream. Any changes to shared state must therefore be protected with appropriate synchronization.

自定义事件

一些程序返回事件类型的对象，他们允许你通过调用 `krpc.client.Event.waitFor()` 来等待事件发生。不过实际上，这些都是使用流和条件变量实现的。

自定义事件也可以实现，表达式 API可以让你创建在服务器上运行的代码，且这些代码可以用于构建自定义事件。举个栗子，下面的表达式 `MeanAltitude > 1000` ，然后创建一个事件当返回 true时触发。

```
import krpc.client.Connection;
import krpc.client.RPCException;
import krpc.client.Event;
import krpc.client.StreamException;
import krpc.client.services.KRPC;
import krpc.client.services.KRPC.Expression;
import krpc.client.services.SpaceCenter;
import krpc.client.services.SpaceCenter.Flight;
import krpc.schema.KRPC.ProcedureCall;

import java.io.IOException;

public class CustomEvent {
  public static void main(String[] args) throws IOException, RPCException, StreamException {
    Connection connection = Connection.newInstance();
    KRPC krpc = KRPC.newInstance(connection);
    SpaceCenter spaceCenter = SpaceCenter.newInstance(connection);
    Flight flight = spaceCenter.getActiveVessel().flight(null);

    // Get the remote procedure call as a message object,
    // so it can be passed to the server
    ProcedureCall meanAltitude = connection.getCall(flight, "getMeanAltitude");

    // Create an expression on the server
    Expression expr = Expression.greaterThan(connection,
      Expression.call(connection, meanAltitude),
      Expression.constantDouble(connection, 1000));

    Event event = krpc.addEvent(expr);
    synchronized (event.getCondition()) {
      event.waitFor();
      System.out.println("Altitude reached 1000m");
    }
  }
}
```

客户端 API参考

- *class Connection*
  与 kRPC服务器连接。 所有与 kRPC的交互都通过这个实例。

```
// 使用详细的信息去连接服务器。
/**
 * @param name
 *        描述连接的名字，显示在游戏服务器的窗口。
 *
 * @param address
 *        连接到服务器的地址，也可以是字符串类型的主机名、IP地址
 *        亦或是 java.net.InetAddress类型的对象，默认值是 127.0.0.1
 *
 * @param rpcPort
 *        RPC服务器的端口号，默认值是 50000。
 *        必须与要连接服务器的 RPC端口相匹配
 *
 * @param streamPort
 *        Stream Server 的端口号，默认值是 50001。
 *        必须与要连接服务器的 stream port相匹配，
 *        (Stream Server 和 stream port 不知道)
 */
static ConnectionnewInstance()
static ConnectionnewInstance(String name)
static ConnectionnewInstance(String name, String address)
static ConnectionnewInstance(String name, String address, int rpcPort, int streamPort)
static ConnectionnewInstance(String name, java.net.InetAddress address)
static ConnectionnewInstance(String name, java.net.InetAddress address, int rpcPort, int stre
```

文档生词

单词	释义			单词	释义			单词	释义
provide	提供			interact	交互			containing	包含
requires	需要			available	可找到			via	通过
compatible	兼容			following	下面			queries	查询
compile	编译			done	完成			machine	机器
specify	指定			as follows	如下			remote	远程
functionality	功能			instantiate	实例化			invoke	调用
demonstrates	演示			vessel	船			altitude	高度
common	常见			use case	使用实例			continuously	连续不地
extract	获取			repeatedly	反复			such	这样的
such as	比如			position	位置			approach	方法
requires	需要			significant	大量的			communication	通信
overhead	开销			efficient	非常高效			achieve	实现
executes	执行			procedure	过程			fixed	固定
argument	实参			parameter	形参			requires	需要
single	单一			establish	建立			until	直到
automatically	自动			disconnects	断开连接			recent	最近
obtained	获得			mechanisms	机制			associated	相关
notified	通知			used to	用于			current	当前
block	阻塞			get abort	终止			abort button	放弃按
acquires	获得			leaves	离开			which causes	使得
prevents	预防			consume	消费			processing resources	处理资
whilst	期间			receiving	接受			Custom	自定义
events	事件			implemented	实现			triggered	触发

单词	释义			单词	释义			单词	释义
Reference	参考								
	————				————				———

Zhouxuan-C changed the title from **kRPC -- Java 客户端** to **【未完】kRPC -- Java 客户端** 6 days ago

Zhouxuan-C added this to **KSP** in **游戏** 6 days ago

Zhouxuan-C added the **游戏** label 6 days ago