> PeerConnection是一个Java层面的API，它的内层包裹着c++的代码，它需要调用c++层面的代码。 它同时也是rtc-Android-sdk中尤为重要的几个类，推荐新入门的同学，可以从这里面入手。

```
//需要在类刚开始初始化的时候，便引入so包，这个so包就是我们用c++缩写
 static {
    System.loadLibrary("jingle_peerconnection_so");
 }
```

```
//Ice: Input Checking Equipment 输入校验设备, 输入校正装置
//检验设备的一些信息的收集状态，开始，收集中，完成
 public enum IceGatheringState { NEW, GATHERING, COMPLETE }
```

```
//链接的状态
public enum IceConnectionState {
    NEW,
    CHECKING,
    CONNECTED,
    COMPLETED,
    FAILED,
    DISCONNECTED,
    CLOSED
  }
```

```
//信号声明
public enum SignalingState {
    STABLE,     //稳定的
    HAVE_LOCAL_OFFER,     //有本地offer
    HAVE_LOCAL_PRANSWER,     //有远程answer
    HAVE_REMOTE_OFFER,     //有远程offer
    HAVE_REMOTE_PRANSWER,     //有远程answer
    CLOSED     //关闭
  }
```

```
  //用来回调的接口，用来监听流发生的改变
  public static interface Observer {

    /** Triggered when the SignalingState changes. */
    public void onSignalingChange(SignalingState newState);

    /** Triggered when the IceConnectionState changes. */
    public void onIceConnectionChange(IceConnectionState newState);

    /** Triggered when the ICE connection receiving status changes. */
    public void onIceConnectionReceivingChange(boolean receiving);

    /** Triggered when the IceGatheringState changes. */
    public void onIceGatheringChange(IceGatheringState newState);

    /** Triggered when a new ICE candidate has been found. */
    public void onIceCandidate(IceCandidate candidate);

    /** Triggered when some ICE candidates have been removed. */
    public void onIceCandidatesRemoved(IceCandidate[] candidates);

    /** Triggered when media is received on a new stream from remote peer. */
    public void onAddStream(MediaStream stream);

    /** Triggered when a remote peer close a stream. */
    public void onRemoveStream(MediaStream stream);

    /** Triggered when a remote peer opens a DataChannel. */
    public void onDataChannel(DataChannel dataChannel);

    /** Triggered when renegotiation is necessary. */
    public void onRenegotiationNeeded();
  }
```

```
//一个对象IceServer里面包含的是三个属性，这个对象通常是在加入房间时被调用的，我们也就是通过这个加入的聊天服务器里面的房间
public static class IceServer {
    public final String uri;
    public final String username;
    public final String password;

    /** Convenience constructor for STUN servers. */
    public IceServer(String uri) {
      this(uri, "", "");
    }

    public IceServer(String uri, String username, String password) {
      this.uri = uri;
      this.username = username;
      this.password = password;
    }

    public String toString() {
      return uri + "[" + username + ":" + password + "]";
    }
  }
```

```
//ice的运送的方式
public enum IceTransportsType { NONE, RELAY, NOHOST, ALL }
```

```
//捆绑的策略，平衡，最大程序捆绑，最大程度兼容
public enum BundlePolicy { BALANCED, MAXBUNDLE, MAXCOMPAT }
```

```
//谈判、无资格的
public enum RtcpMuxPolicy { NEGOTIATE, REQUIRE }
```

```
//tcp候选人的策略,激活，有缺陷的
public enum TcpCandidatePolicy { ENABLED, DISABLED }
```

```
//候选人网络策略，全部，最低的成本
public enum CandidateNetworkPolicy { ALL, LOW_COST }
```

```
秘钥加密的类型
public enum KeyType { RSA, ECDSA }
```

```
不断的收集策略，收集一次，不间断的收集
public enum ContinualGatheringPolicy { GATHER_ONCE, GATHER_CONTINUALLY }
```

```
/**
   *这个一个类，里面有很多属性，这些属性，大多都是上面刚刚定义过的属性，当我们创建PeerConnection的时候，
   *这些都是要被设置的，它的构造方法里面需要接受iceSercers，这个里面就包含了许多和我们配置相关的信息
public static class RTCConfiguration {
    public IceTransportsType iceTransportsType;
    public List<IceServer> iceServers;
    public BundlePolicy bundlePolicy;
    public RtcpMuxPolicy rtcpMuxPolicy;
    public TcpCandidatePolicy tcpCandidatePolicy;
    public CandidateNetworkPolicy candidateNetworkPolicy;
    public int audioJitterBufferMaxPackets;
    public boolean audioJitterBufferFastAccelerate;
    public int iceConnectionReceivingTimeout;
    public int iceBackupCandidatePairPingInterval;
    public KeyType keyType;
    public ContinualGatheringPolicy continualGatheringPolicy;
    public int iceCandidatePoolSize;
    public boolean pruneTurnPorts;
    public boolean presumeWritableWhenFullyRelayed;

    public RTCConfiguration(List<IceServer> iceServers) {
      iceTransportsType = IceTransportsType.ALL;
      bundlePolicy = BundlePolicy.BALANCED;
      rtcpMuxPolicy = RtcpMuxPolicy.NEGOTIATE;
      tcpCandidatePolicy = TcpCandidatePolicy.ENABLED;
      candidateNetworkPolicy = candidateNetworkPolicy.ALL;
      this.iceServers = iceServers;
      audioJitterBufferMaxPackets = 50;
      audioJitterBufferFastAccelerate = false;
      iceConnectionReceivingTimeout = -1;
      iceBackupCandidatePairPingInterval = -1;
      keyType = KeyType.ECDSA;
      continualGatheringPolicy = ContinualGatheringPolicy.GATHER_ONCE;
      iceCandidatePoolSize = 0;
      pruneTurnPorts = false;
      presumeWritableWhenFullyRelayed = false;
    }
  };
```

```
//记录媒体流，发送者，和接受者，还有就是jni层的链接和观察者
  private final List<MediaStream> localStreams;
  private final long nativePeerConnection;
  private final long nativeObserver;
  private List<RtpSender> senders;
  private List<RtpReceiver> receivers;
```

```
//构造方法，需要传入一个Connection，还有就是一个回调的接口
  PeerConnection(long nativePeerConnection, long nativeObserver) {
    this.nativePeerConnection = nativePeerConnection;
    this.nativeObserver = nativeObserver;
    localStreams = new LinkedList<MediaStream>();
    senders = new LinkedList<RtpSender>();
    receivers = new LinkedList<RtpReceiver>();
  }
```

```
//JSEP（JavaScript Session Establishment Protocol，JavaScript 会话建立协议）
//以下便都是和底层c代码的交互部分了，这里面的方法尤为重要
//这里的方法都是，我们调用，然后传入callback等待结果
//createOffer,createAnswer,setLocalDescription,setRemoteDescription都是rtc最基本的函数之一
  public native SessionDescription getLocalDescription();

  public native SessionDescription getRemoteDescription();

  public native DataChannel createDataChannel(String label, DataChannel.Init init);

  public native void createOffer(SdpObserver observer, MediaConstraints constraints);

  public native void createAnswer(SdpObserver observer, MediaConstraints constraints);

  public native void setLocalDescription(SdpObserver observer, SessionDescription sdp);

  public native void setRemoteDescription(SdpObserver observer, SessionDescription sdp);

  public native boolean setConfiguration(RTCConfiguration config);
```

```
//添加和删除对端的描述
public boolean addIceCandidate(IceCandidate candidate) {
    return nativeAddIceCandidate(candidate.sdpMid, candidate.sdpMLineIndex, candidate.sdp);
  }

  public boolean removeIceCandidates(final IceCandidate[] candidates) {
    return nativeRemoveIceCandidates(candidates);
  }
```

```
//添加和删除本地的流
public boolean addStream(MediaStream stream) {
    boolean ret = nativeAddLocalStream(stream.nativeStream);
    if (!ret) {
      return false;
    }
    localStreams.add(stream);
    return true;
  }

  public void removeStream(MediaStream stream) {
    nativeRemoveLocalStream(stream.nativeStream);
    localStreams.remove(stream);
  }
```

```
//添加和获取发送者的信息，例如音量信息，网速的信息都可以从中获得
public RtpSender createSender(String kind, String stream_id) {
    RtpSender new_sender = nativeCreateSender(kind, stream_id);
    if (new_sender != null) {
      senders.add(new_sender);
    }
    return new_sender;
  }

  // Note that calling getSenders will dispose of the senders previously
  // returned (and same goes for getReceivers).
  public List<RtpSender> getSenders() {
    for (RtpSender sender : senders) {
      sender.dispose();
    }
    senders = nativeGetSenders();
    return Collections.unmodifiableList(senders);
  }
```

```
//获取状态，在这里面可以获取到trak的状态
public boolean getStats(StatsObserver observer, MediaStreamTrack track) {
    return nativeGetStats(observer, (track == null) ? 0 : track.nativeTrack);
  }
```

```
 // Starts recording an RTC event log. Ownership of the file is transfered to
  // the native code. If an RTC event log is already being recorded, it will be
  // stopped and a new one will start using the provided file. Logging will
  // continue until the stopRtcEventLog function is called. The max_size_bytes
  // argument is ignored, it is added for future use.
  public boolean startRtcEventLog(int file_descriptor, int max_size_bytes) {
    return nativeStartRtcEventLog(file_descriptor, max_size_bytes);
  }

// Stops recording an RTC event log. If no RTC event log is currently being
  // recorded, this call will have no effect.
  public void stopRtcEventLog() {
    nativeStopRtcEventLog();
  }
```

```
  //这里面都是jni层面的代码，这些方法可以获取一些状态
  public native SignalingState signalingState();

  public native IceConnectionState iceConnectionState();

  public native IceGatheringState iceGatheringState();

  public native void close();
```

```java
//dispose掉当前的这些流，实现清空当前PeerConnection的作用
public void dispose() {
    close();
    for (MediaStream stream : localStreams) {
      nativeRemoveLocalStream(stream.nativeStream);
      stream.dispose();
    }
    localStreams.clear();
    for (RtpSender sender : senders) {
      sender.dispose();
    }
    senders.clear();
    for (RtpReceiver receiver : receivers) {
      receiver.dispose();
    }
    receivers.clear();
    freePeerConnection(nativePeerConnection);
    freeObserver(nativeObserver);
  }
```

```java
//JNI层面的代码，供sdk内部调用的，我们调用的很多sdk层面的代码，然后它们调用这些代码

  private static native void freePeerConnection(long nativePeerConnection);

  private static native void freeObserver(long nativeObserver);

  private native boolean nativeAddIceCandidate(
      String sdpMid, int sdpMLineIndex, String iceCandidateSdp);

  private native boolean nativeRemoveIceCandidates(final IceCandidate[] candidates);

  private native boolean nativeAddLocalStream(long nativeStream);

  private native void nativeRemoveLocalStream(long nativeStream);

  private native boolean nativeGetStats(StatsObserver observer, long nativeTrack);

  private native RtpSender nativeCreateSender(String kind, String stream_id);

  private native List<RtpSender> nativeGetSenders();

  private native List<RtpReceiver> nativeGetReceivers();

  private native boolean nativeStartRtcEventLog(int file_descriptor, int max_size_bytes);

  private native void nativeStopRtcEventLog();
```

以上便是，我对PeerConnection源码的解析，转载请注明出处:[linsir.top]，我的[gitHub地址]，欢迎star，follow~