> PeerConnectionFactory一个用来生成PeerConnection的工厂类，WebRTC中最重要的类之一，用来创建peerConnection的类，同时负责初始化全局和底层交互。

```
//引用so包
static {
    System.loadLibrary("jingle_peerconnection_so");
}
```

```
  public static class Options {
    // Keep in sync with webrtc/base/network.h!
    //未知
    static final int ADAPTER_TYPE_UNKNOWN = 0;
    //以太网
    static final int ADAPTER_TYPE_ETHERNET = 1 << 0;
    //WIFI
    static final int ADAPTER_TYPE_WIFI = 1 << 1;
    //移动网络
    static final int ADAPTER_TYPE_CELLULAR = 1 << 2;
    //vpn
    static final int ADAPTER_TYPE_VPN = 1 << 3;
    //回环
    static final int ADAPTER_TYPE_LOOPBACK = 1 << 4;
    //网络忽略
    public int networkIgnoreMask;
    //解码器
    public boolean disableEncryption;
    //网络监控
    public boolean disableNetworkMonitor;
  }
```

```
  //native层初始化全局，如果初始化成功会返回TRUE,否则返回FALSE
  public static native boolean initializeAndroidGlobals(Object context, boolean initializeAudio,
      boolean initializeVideo, boolean videoHwAcceleration);

  // Field trial initialization. Must be called before PeerConnectionFactory
  // is created.
  //一个初始化工作，应该在创建Factory创建之前
  public static native void initializeFieldTrials(String fieldTrialsInitString);


  // Internal tracing initialization. Must be called before PeerConnectionFactory is created to
  // prevent racing with tracing code.
  //初始化内部描述
  public static native void initializeInternalTracer();
  // Internal tracing shutdown, called to prevent resource leaks. Must be called after
  // PeerConnectionFactory is gone to prevent races with code performing tracing.

  //关闭内部描述
  public static native void shutdownInternalTracer();

  //打开或关闭内部追踪
  // Start/stop internal capturing of internal tracing.
  public static native boolean startInternalTracingCapture(String tracing_filename);
  public static native void stopInternalTracingCapture();
```

```
  //工厂的构造方法，从标签来看，是不建议我们直接使用的
  @Deprecated
  public PeerConnectionFactory() {
    this(null);
  }
```

```
  //工厂类的构造方法，调用native层穿进去一组参数
  public PeerConnectionFactory(Options options) {
    nativeFactory = nativeCreatePeerConnectionFactory(options);
    if (nativeFactory == 0) {
      throw new RuntimeException("Failed to initialize PeerConnectionFactory!");
    }
  }
```

```
//两种创建PeerConnection的方法，要的参数实质上是一样，返回的也一样，目前不明白区别。。。
 public PeerConnection createPeerConnection(PeerConnection.RTCConfiguration rtcConfig,
      MediaConstraints constraints, PeerConnection.Observer observer) {
    long nativeObserver = nativeCreateObserver(observer);
    if (nativeObserver == 0) {
      return null;
    }
    long nativePeerConnection =
        nativeCreatePeerConnection(nativeFactory, rtcConfig, constraints, nativeObserver);
    if (nativePeerConnection == 0) {
      return null;
    }
    return new PeerConnection(nativePeerConnection, nativeObserver);
  }

  public PeerConnection createPeerConnection(List<PeerConnection.IceServer> iceServers,
      MediaConstraints constraints, PeerConnection.Observer observer) {
    PeerConnection.RTCConfiguration rtcConfig = new PeerConnection.RTCConfiguration(iceServers);
    return createPeerConnection(rtcConfig, constraints, observer);
  }
```

```
  //创建本地媒体流
  public MediaStream createLocalMediaStream(String label) {
    return new MediaStream(nativeCreateLocalMediaStream(nativeFactory, label));
  }
```

```
  //创建视频资源，调用的都是native层的代码
 public VideoSource createVideoSource(VideoCapturer capturer) {
    final EglBase.Context eglContext =
        localEglbase == null ? null : localEglbase.getEglBaseContext();
    long nativeAndroidVideoTrackSource =
        nativeCreateVideoSource(nativeFactory, eglContext, capturer.isScreencast());
    VideoCapturer.CapturerObserver capturerObserver =
        new VideoCapturer.AndroidVideoTrackSourceObserver(nativeAndroidVideoTrackSource);
    nativeInitializeVideoCapturer(
        nativeFactory, capturer, nativeAndroidVideoTrackSource, capturerObserver);
    return new VideoSource(nativeAndroidVideoTrackSource);
  }
```

```
//创建音频资源
public AudioSource createAudioSource(MediaConstraints constraints) {
    return new AudioSource(nativeCreateAudioSource(nativeFactory, constraints));
  }
```

```
//创建视频足记与音频足记
public VideoTrack createVideoTrack(String id, VideoSource source) {
    return new VideoTrack(nativeCreateVideoTrack(nativeFactory, id, source.nativeSource));
  }

public AudioTrack createAudioTrack(String id, AudioSource source) {
    return new AudioTrack(nativeCreateAudioTrack(nativeFactory, id, source.nativeSource));
  }
```

```
  // Starts recording an AEC dump. Ownership of the file is transfered to the
  // native code. If an AEC dump is already in progress, it will be stopped and
  // a new one will start using the provided file.
  //使用一定文件空间转码
  public boolean startAecDump(int file_descriptor, int filesize_limit_bytes) {
    return nativeStartAecDump(nativeFactory, file_descriptor, filesize_limit_bytes);
  }

  // Stops recording an AEC dump. If no AEC dump is currently being recorded,
  // this call will have no effect.
  //停止转码
  public void stopAecDump() {
    nativeStopAecDump(nativeFactory);
  }
```

```
//设置状态，对应无参数的构造方法，同样不建议被使用
@Deprecated
  public void setOptions(Options options) {
    nativeSetOptions(nativeFactory, options);
  }
  public void setVideoHwAccelerationOptions(
      EglBase.Context localEglContext, EglBase.Context remoteEglContext) {
    if (localEglbase != null) {
      Logging.w(TAG, "Egl context already set.");
      localEglbase.release();
    }
    if (remoteEglbase != null) {
      Logging.w(TAG, "Egl context already set.");
      remoteEglbase.release();
    }
    localEglbase = EglBase.create(localEglContext);
    remoteEglbase = EglBase.create(remoteEglContext);
    nativeSetVideoHwAccelerationOptions(
        nativeFactory, localEglbase.getEglBaseContext(), remoteEglbase.getEglBaseContext());
  }
```

```
  //处理掉，关闭的
 public void dispose() {
    nativeFreeFactory(nativeFactory);
    networkThread = null;
    workerThread = null;
    signalingThread = null;
    if (localEglbase != null)
      localEglbase.release();
    if (remoteEglbase != null)
      remoteEglbase.release();
  }
```

```
//回调线程
public void threadsCallbacks() {
    nativeThreadsCallbacks(nativeFactory);
  }
```

```
  //输出现在的调用帧
  private static void printStackTrace(Thread thread, String threadName) {
    if (thread != null) {
      StackTraceElement[] stackTraces = thread.getStackTrace();
      if (stackTraces.length > 0) {
        Logging.d(TAG, threadName + " stacks trace:");
        for (StackTraceElement stackTrace : stackTraces) {
          Logging.d(TAG, stackTrace.toString());
        }
      }
    }
  }

  public static void printStackTraces() {
    printStackTrace(networkThread, "Network thread");
    printStackTrace(workerThread, "Worker thread");
    printStackTrace(signalingThread, "Signaling thread");
  }
```

```
  //当线程准备完毕
  private static void onNetworkThreadReady() {
    networkThread = Thread.currentThread();
    Logging.d(TAG, "onNetworkThreadReady");
  }

  private static void onWorkerThreadReady() {
    workerThread = Thread.currentThread();
    Logging.d(TAG, "onWorkerThreadReady");
  }

  private static void onSignalingThreadReady() {
    signalingThread = Thread.currentThread();
    Logging.d(TAG, "onSignalingThreadReady");
  }
```

```
//native层代码

//创建peerConnectionFactory
private static native long nativeCreatePeerConnectionFactory(Options options);

//创建回调的listener
private static native long nativeCreateObserver(PeerConnection.Observer observer);

//创建peerConnection
private static native long nativeCreatePeerConnection(long nativeFactory,
    PeerConnection.RTCConfiguration rtcConfig, MediaConstraints constraints, long nativeObserver);

//创建本地的媒体流
private static native long nativeCreateLocalMediaStream(long nativeFactory, String label);

//创建音频资源
private static native long nativeCreateVideoSource(
    long nativeFactory, EglBase.Context eglContext, boolean is_screencast);

//初始化视频的资源
private static native void nativeInitializeVideoCapturer(long native_factory,
    VideoCapturer j_video_capturer, long native_source,
    VideoCapturer.CapturerObserver j_frame_observer);

//创建视频足记
private static native long nativeCreateVideoTrack(
    long nativeFactory, String id, long nativeVideoSource);

//创建音频资源
private static native long nativeCreateAudioSource(
    long nativeFactory, MediaConstraints constraints);

//创建音频足记
private static native long nativeCreateAudioTrack(
    long nativeFactory, String id, long nativeSource);

//开启ace转码
private static native boolean nativeStartAecDump(
    long nativeFactory, int file_descriptor, int filesize_limit_bytes);

//关闭ace转码
private static native void nativeStopAecDump(long nativeFactory);

//设置配置
@Deprecated public native void nativeSetOptions(long nativeFactory, Options options);

//设置怎么样的转码方式，加速的方式
private static native void nativeSetVideoHwAccelerationOptions(
    long nativeFactory, Object localEGLContext, Object remoteEGLContext);

//不同线程的回调
private static native void nativeThreadsCallbacks(long nativeFactory);

//释放factory的控件
private static native void nativeFreeFactory(long nativeFactory);
```

```
//各种编码格式的相互转换
public static native void nativeARGBToNV21(byte[] src, int width, int height, byte[] dst);

public static native void nativeRGBAToNV21(byte[] src, int width, int height, byte[] dst);

public static native void nativeNV21ToARGB(byte[] src, int width, int height, byte[] dst);

public static native void nativeNV12ToNV21(byte[] src, int width, int height);

public static native void nativeI420ToARGB(byte[] src, int width, int height, byte[] dst);

public static native void nativeI420ToNV21(byte[] src, int width, int height, byte[] dst);
```

以上便是rtc中的peerConnection类~ 以上便是博主的一点点的小见解，欢迎大家和我讨论~~