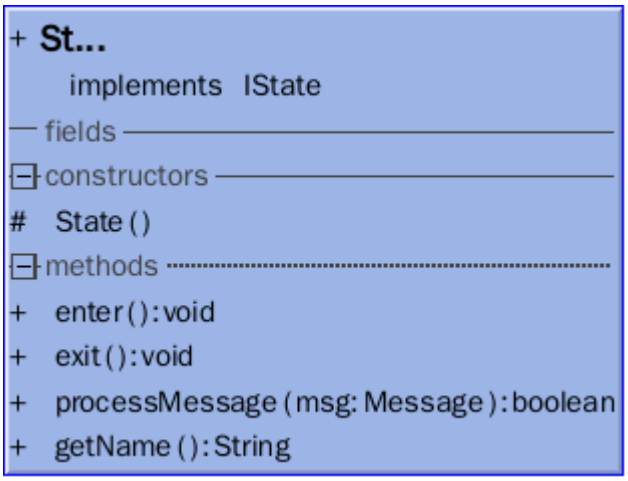
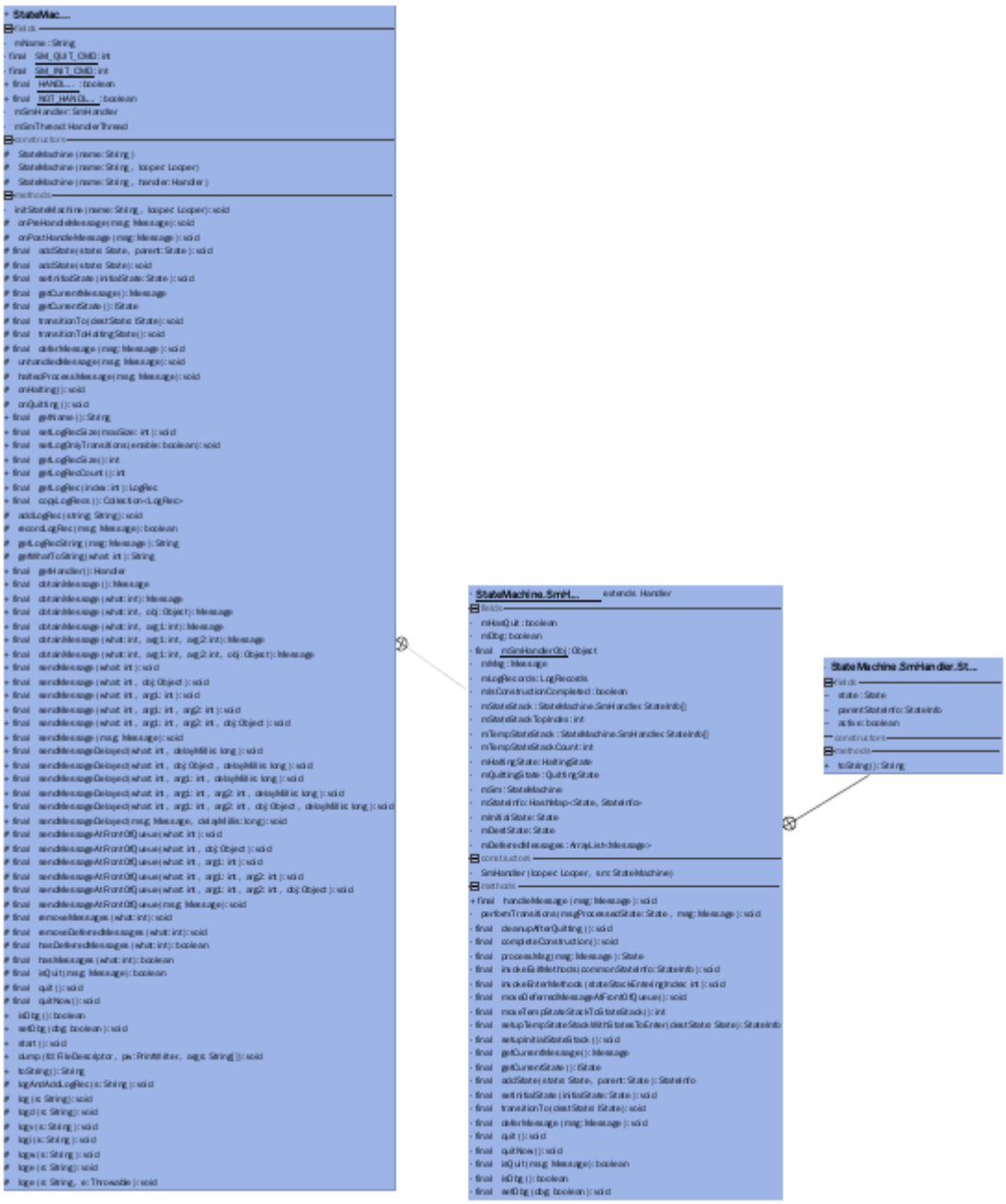
前言

本文基于Android N进行的源码分析，Android在Wifi和蓝牙大量的使用了状态机去进行管理不同状态下接收的消息，本文分析的源码在GitHub上可以找到，因为StateMachine依赖比较少，所以可以直接抽出来进行编译调试，Android系统中也提供了对应的Demo，有兴趣的欢迎到GitHub上下载

类的结构图

本文会涉及到一些StateMachine的内部类以及结构，





状态机的初始化

```
    //StateMachine.java
    protected StateMachine(String name) {
        //这里创建了一个HandlerThread
        mSmThread = new HandlerThread(name);
        mSmThread.start();
        Looper looper = mSmThread.getLooper();

        initStateMachine(name, looper);
    }

    private void initStateMachine(String name, Looper looper) {
        mName = name;
        //创建SmHandler的实例
        mSmHandler = new SmHandler(looper, this);
    }
```

添加状态

这里是以一个官方提供的例子做了修改之后写的，HelloState继承子StateMachine，具体可以看代码

```
    //HelloState.java
    protected HelloState(String name) {
        //这个就会调用StateMachine的初始化
        super(name);
        log("Actor E");

        // Add states, use indentation to show hierarchy
        //调用基类的方法
        addState(mP1);
        addState(mS1, mP1);
        addState(mS2, mP1);
        addState(mP2);

        // Set the initial state
        setInitialState(mS1);
        log("Actor X");
    }
```

```
    //StateMachine.java
    protected final void addState(State state) {
        mSmHandler.addState(state, null);
    }
    protected final void addState(State state, State parent) {
      mSmHandler.addState(state, parent);
    }

    //StateMachine.SmHandler
    private final StateInfo addState(State state, State parent) {
      if (mDbg) {
        mSm.log("addStateInternal: E state=" + state.getName() + ",parent="
                + ((parent == null) ? "" : parent.getName()));
      }
      StateInfo parentStateInfo = null;
      //判断类是否为空
      if (parent != null) {
        //private HashMap<State, StateInfo> mStateInfo = new HashMap<State, StateInfo>
();
        //从Map中取出父的State是否已经添加
        parentStateInfo = mStateInfo.get(parent);
        if (parentStateInfo == null) {
          // Recursively add our parent as it's not been added yet.
          //添加父State
          parentStateInfo = addState(parent, null);
        }
      }
      StateInfo stateInfo = mStateInfo.get(state);
      //如果当前的State不在Map中，那就直接添加
      if (stateInfo == null) {
        stateInfo = new StateInfo();
        mStateInfo.put(state, stateInfo);
      }

      // Validate that we aren't adding the same state in two different hierarchies.
      //检查我没是否有添加两个相同的State
      if ((stateInfo.parentStateInfo != null)
          && (stateInfo.parentStateInfo != parentStateInfo)) {
        throw new RuntimeException("state already added");
      }
      //初始化State
      stateInfo.state = state;
      stateInfo.parentStateInfo = parentStateInfo;
      stateInfo.active = false;
      if (mDbg) mSm.log("addStateInternal: X stateInfo: " + stateInfo);
      return stateInfo;
    }

    //addState(mP1);
    //addState(mS1, mP1);
    //addState(mS2, mP1);
    //addState(mP2);

    //setInitialState(mS1);
```

```
    //根据这个的添加顺序可以得出以下的图
              P1         P2
            /   \
(Initial)S1      S2
```

状态机的启动

```
//StateMachine.java
public void start() {
    // mSmHandler can be null if the state machine has quit.
    SmHandler smh = mSmHandler;
    if (smh == null) return;

    /** Send the complete construction message */
    smh.completeConstruction();
}
//StateMachine.SmHandler
private final void completeConstruction() {
    if (mDbg) mSm.log("completeConstruction: E");

      /**
        * Determine the maximum depth of the state hierarchy
        * so we can allocate the state stacks.
        */
    int maxDepth = 0;
    //遍历Map,寻找树的最大深度，用于确定数组的大小
    for (StateInfo si : mStateInfo.values()) {
      int depth = 0;
      for (StateInfo i = si; i != null; depth++) {
        i = i.parentStateInfo;
      }
      if (maxDepth < depth) {
        maxDepth = depth;
      }
    }
    if (mDbg) mSm.log("completeConstruction: maxDepth=" + maxDepth);
    //创建两个数组，这两个数组存储的是当前StateInfo和父StateInfo
    //并且这两个数组是相反的，例如：mStateStack是P1,S1，那么mTempStateStack是S1,P1
    mStateStack = new StateInfo[maxDepth];
    mTempStateStack = new StateInfo[maxDepth];
    setupInitialStateStack();

    /** Sending SM_INIT_CMD message to invoke enter methods asynchronously */
    //见发送消息
    sendMessageAtFrontOfQueue(obtainMessage(SM_INIT_CMD, mSmHandlerObj));

    if (mDbg) mSm.log("completeConstruction: X");
}

private final void setupInitialStateStack() {
    if (mDbg) {
      mSm.log("setupInitialStateStack: E mInitialState=" + mInitialState.getName());
    }
    //根据例子，获取是S1的State
    StateInfo curStateInfo = mStateInfo.get(mInitialState);
    for (mTempStateStackCount = 0; curStateInfo != null; mTempStateStackCount++) {
      mTempStateStack[mTempStateStackCount] = curStateInfo;
      if (mDbg) {

        mSm.log("setupInitialStateStack: " + " mTempStateStack[" +
```

```
mTempStateStackCount + "] : " +
                    mTempStateStack[mTempStateStackCount].state.getName());
          }
          curStateInfo = curStateInfo.parentStateInfo;
        }
        //此时mTempStateStack[0] = S1.StateInfo, mTempStateStack[1] = P1.StateInfo
        //mTempStateStackCount = 2

        // Empty the StateStack
        mStateStackTopIndex = -1;
        //mStateStackTopIndex = 1
        moveTempStateStackToStateStack();
    }

    private final int moveTempStateStackToStateStack() {
        int startingIndex = mStateStackTopIndex + 1;
        int i = mTempStateStackCount - 1;
        int j = startingIndex;
        //把两个状态的位置倒过来
        while (i >= 0) {
          if (mDbg) mSm.log("moveTempStackToStateStack: i=" + i + ",j=" + j);
          mStateStack[j] = mTempStateStack[i];
          j += 1;
          i -= 1;
        }
        mStateStackTopIndex = j - 1;

        if (mDbg) {
          mSm.log("moveTempStackToStateStack: X mStateStackTop=" + mStateStackTopIndex
                  + ",startingIndex=" + startingIndex + ",Top="
                  + mStateStack[mStateStackTopIndex].state.getName());
        }
        //至于为什么要返回一个startingIndex，见切换状态
        return startingIndex;
    }
```

状态机发送消息

这个地方有两种情况，第一种状态机自己发消息，第二种其他地方发送消息

处理状态机自己发送的消息

```
//sendMessageAtFrontOfQueue(obtainMessage(SM_INIT_CMD, mSmHandlerObj));
//以初始化消息为例,sendMessageAtFrontOfQueue这个其实是Handler的方法，
//StateMachine.SmHandler
@Override
public final void handleMessage(Message msg) {
  if (!mHasQuit) {
    if (mSm != null && msg.what != SM_INIT_CMD && msg.what != SM_QUIT_CMD) {
      //啥都没有
      mSm.onPreHandleMessage(msg);
    }

    if (mDbg) mSm.log("handleMessage: E msg.what=" + msg.what);

    /** Save the current message */
    mMsg = msg;

    /** State that processed the message */
    State msgProcessedState = null;
    if (mIsConstructionCompleted) {
      /** Normal path */
      msgProcessedState = processMsg(msg);
    } else if (!mIsConstructionCompleted && (mMsg.what == SM_INIT_CMD)
              && (mMsg.obj == mSmHandlerObj)) {
      /** Initial one time path. */
      mIsConstructionCompleted = true;
      invokeEnterMethods(0);
    } else {
      throw new RuntimeException("StateMachine.handleMessage: "
                                 + "The start method not called, received msg: " + msg);
    }
    //这个在Normal path时才会有用,真正切换状态的地方，见切换状态
    performTransitions(msgProcessedState, msg);

    // We need to check if mSm == null here as we could be quitting.
    if (mDbg && mSm != null) mSm.log("handleMessage: X");

    if (mSm != null && msg.what != SM_INIT_CMD && msg.what != SM_QUIT_CMD) {
      mSm.onPostHandleMessage(msg);
    }
  }
}
//mStateStackTopIndex = 1
private final void invokeEnterMethods(int stateStackEnteringIndex) {
  for (int i = stateStackEnteringIndex; i <= mStateStackTopIndex; i++) {
    if (mDbg) mSm.log("invokeEnterMethods: " + mStateStack[i].state.getName());
    mStateStack[i].state.enter();
    mStateStack[i].active = true;
  }
}
//顺序调用P1.enter, S1.enter方法，并且两个都是已经启动了，至此，状态机启动真正完毕，等待发送消息
```

其他地方发送消息

```
//StateMachine.java
public final void sendMessage(Message msg) {
  // mSmHandler can be null if the state machine has quit.
  SmHandler smh = mSmHandler;
  if (smh == null) return;
  //最终还是调用SmHandler的sendMessage
  smh.sendMessage(msg);
}

//StateMachine.SmHandler
private final State processMsg(Message msg) {
  //取出的是S1
  StateInfo curStateInfo = mStateStack[mStateStackTopIndex];
  if (mDbg) {
    mSm.log("processMsg: " + curStateInfo.state.getName());
  }

  if (isQuit(msg)) {
    transitionTo(mQuittingState);
  } else {
    //交给对应的State去处理消息，对应方法见例子处理消息
    while (!curStateInfo.state.processMessage(msg)) {
      /**
        * Not processed
        */
      //获取父的State继续处理
      curStateInfo = curStateInfo.parentStateInfo;
      //如果所有的没处理，那么就打个错误信息出来
      if (curStateInfo == null) {
        /**
          * No parents left so it's not handled
          */
        mSm.unhandledMessage(msg);
        break;
      }
      if (mDbg) {
        mSm.log("processMsg: " + curStateInfo.state.getName());
      }
    }
  }
  //返回当前的State
  return (curStateInfo != null) ? curStateInfo.state : null;
}
```

处理消息

```java
//HelloState.java
class S1 extends State {
  @Override
  public void enter() {
    log("mS1.enter");
  }
  @Override
  public boolean processMessage(Message message) {
    log("S1.processMessage what=" + message.what);
    if (message.what == CMD_1) {
      // Transition to ourself to show that enter/exit is called
      //其实这个方法并不会开始切换，只是说下一次的切换到此为止
      transitionTo(mS1);
      //返回true之后才会开始进行切换
      return IState.HANDLED;
    } else {
      // Let parent process all other messages
      return IState.NOT_HANDLED;
    }
  }
  @Override
  public void exit() {
    log("mS1.exit");
  }
}
```

切换状态

```
//StateMachine.java
protected final void transitionTo(IState destState) {
  mSmHandler.transitionTo(destState);
}
//StateMachine.SmHandler
private final void transitionTo(IState destState) {
  //只是设置目标State
  mDestState = (State) destState;
  if (mDbg) mSm.log("transitionTo: destState=" + mDestState.getName());
}

//真正的切换状态是发生在返回之后，难点到了
//StateMaChine.SmHandler 假设: S1-->S2的切换
private void performTransitions(State msgProcessedState, Message msg) {
  //orgState = S1
  State orgState = mStateStack[mStateStackTopIndex].state;
  State destState = mDestState;
  if (destState != null) {
    /**
      * Process the transitions including transitions in the enter/exit methods
      */
    while (true) {
      if (mDbg) mSm.log("handleMessage: new destination call exit/enter");
      //找到公共的父State， mTempStateStackCount = 1
      StateInfo commonStateInfo = setupTempStateStackWithStatesToEnter(destState);
      //从父State往下开始依次退出
      invokeExitMethods(commonStateInfo);
      //这个就不说了，见状态机的启动
      //stateStackEnteringIndex = 0；mStateStack[0] = S2, mStateStack[1] = P1
      int stateStackEnteringIndex = moveTempStateStackToStateStack();
      //只会调用S2.enter()
      invokeEnterMethods(stateStackEnteringIndex);
      //这个是处理前一个状态的deferMessage()
      moveDeferredMessageAtFrontOfQueue();

      if (destState != mDestState) {
        // A new mDestState so continue looping
        destState = mDestState;
      } else {
        // No change in mDestState so we're done
        break;
      }
    }
    mDestState = null;
  }
  if (destState != null) {
    if (destState == mQuittingState) {
      mSm.onQuitting();
      cleanupAfterQuitting();
    } else if (destState == mHaltingState) {
      mSm.onHalting();
    }

  }
```

```
}

private final StateInfo setupTempStateStackWithStatesToEnter(State destState) {
  //mTempStateStackCount重新被赋值为0
  mTempStateStackCount = 0;
  //获取当前StateInfo, S2.StateInfo
  StateInfo curStateInfo = mStateInfo.get(destState);
  //使用do while的原因是destState.active为false并且还要注意，此时mTempStateStack已经被改变了，变
成了0-->S2, 1-->P1， 且mTempStateStackCount = 1
  do {
    mTempStateStack[mTempStateStackCount++] = curStateInfo;
    //curStateInfo是P1
    curStateInfo = curStateInfo.parentStateInfo;
  } while ((curStateInfo != null) && !curStateInfo.active);
    //只有当状态为null,或者是StateInfo.active为true时，才跳出循环
  if (mDbg) {
    mSm.log("setupTempStateStackWithStatesToEnter: X mTempStateStackCount="
            + mTempStateStackCount + ",curStateInfo: " + curStateInfo);
  }
  return curStateInfo;
}

private final void invokeExitMethods(StateInfo commonStateInfo) {
  if (mDbg) mSm.log("invokeExitMethods: mStateStackTopIndex: ");
  //mStateStatackTopIndex = 1
  //退出的仅S1，不会退出P1,因为P1是S1的父State

  while ((mStateStackTopIndex >= 0)
         && (mStateStack[mStateStackTopIndex] != commonStateInfo)) {
    State curState = mStateStack[mStateStackTopIndex].state;
    if (mDbg) mSm.log("invokeExitMethods: " + curState.getName());
    curState.exit();
    mStateStack[mStateStackTopIndex].active = false;
    mStateStackTopIndex -= 1;
  }
  //到最后mStateStackTopIndex = 0,相当于初始化的值
}
```

总结

至此状态机的一些套路我们已经理的差不多了，果然是走的最长的路就是你的套路，至于更多的细节大家可以把源码下载下来仔细研究。原本我以为我之前已经对Android状态机已经了解的差不多了，但是写了这篇文章之后才会了解到状态机的复杂性还是挺高的，尤其是它的状态栈的变化，一不小心就容易弄错，建议大家看的时候把Dbg打开，这样调试对于理解状态机会更深一些。不过其实这个还没有讲完，比如：deferMessage()方法，S1切换到S1的状态会有什么改变，S1到P2的状态切换又会发生什么，欢迎大家与我一起讨论。。