```kotlin
package app.control



import SPUtils

import android.content.*

import android.os.IBinder

import app.control.common.util.Utils

import app.control.dmanager.download.DownloadManager

import app.control.gesture.accessibility.BaseAccessibilityService

import app.control.gesture.background.GestureReceiverManager

import app.control.model.MemberInfo

import app.control.receiver.KeepLiveWidgetProvider

import app.control.receiver.PushTaskReceiver

import app.control.service.SocketClient

import app.control.service.TaskJobService

import com.common.um.CommonApplication

import com.google.gson.Gson

import com.tencent.bugly.crashreport.CrashReport



/**

 * Created by danbo on 2019-11-07.

 */

class ControlApplication : CommonApplication() {
```

```
//  创建单例
private object SingletonHolder {
    var socketClient: SocketClient? = null
}


companion object {
    fun getSocketClient(): SocketClient? {
        return SingletonHolder.socketClient
    }
}

private val socketConnection = object : ServiceConnection {
    override fun onServiceDisconnected(name: ComponentName?) {
    }

    override fun onServiceConnected(name: ComponentName?, binder: IBinder?) {
        if (binder is (SocketClient.SocketBinder)) {
            SingletonHolder.socketClient = binder.service
        }
    }
```

```
}

override fun onCreate() {
    super.onCreate()
    //初始化下载管理器
    DownloadManager.getInstance().init(this, 3)

    Utils.init(this)

    //在应用层就获取 token 同步用户信息
    token = SPUtils.getData(AppConstant.TOKEN, "") as String
    if (token.isNotEmpty()) {
        memberInfo = Gson().fromJson(
            SPUtils.getData(AppConstant.MEMBER_INFO, "") as String,
            MemberInfo::class.java
        )
    }

    //SocketClient 服务
    val bindIntent = Intent(this, SocketClient::class.java)
    bindService(bindIntent, socketConnection, Context.BIND_AUTO_CREATE)
```

```
        //初始化辅助功能基类
        BaseAccessibilityService.getInstance().init(applicationContext)
        //接收广播永久获取 action
        GestureReceiverManager.getInstance().bindRecordReceiver(this)

        //推送广播接收器
        val pushReceiver = PushTaskReceiver()
        val pushFilter = IntentFilter()
        pushFilter.addAction(PushTaskReceiver.ACTION_PUSH_MESSAGE)
        registerReceiver(pushReceiver, pushFilter)

        //启动发送桌面小部件广播
        val intent = Intent(this, KeepLiveWidgetProvider::class.java)
            .setAction("android.appwidget.action.APPWIDGET_UPDATE")
        sendBroadcast(intent)

        val jobServiceIntent = Intent(this, TaskJobService::class.java)
        startService(jobServiceIntent)//执行任务

        CrashReport.initCrashReport(this, "2236918d66", false)
    }
```

```
}

package app.control.ui


import SPUtils

import android.Manifest

import android.app.Dialog

import android.content.Intent

import android.graphics.Color

import android.net.Uri

import android.os.Build

import android.os.Bundle

import android.os.Handler

import android.provider.Settings

import android.view.*

import androidx.annotation.IdRes

import androidx.annotation.MenuRes

import androidx.core.view.ViewCompat

import androidx.databinding.DataBindingUtil

import androidx.databinding.ViewDataBinding

import androidx.fragment.app.Fragment

import androidx.lifecycle.ViewModelProviders
```

```
import app.control.ActivityStackManager

import app.control.AppConstant

import app.control.AppConstant.REQUEST_CODE

import app.control.BR

import app.control.R

import app.control.common.BaseView

import app.control.common.util.AndroidUtil

import app.control.common.util.PermissionUtils

import app.control.common.util.Utils

import app.control.common.util.VMUtil

import app.control.gesture.accessibility.BaseAccessibilityService

import app.control.viewmodel.BaseViewModel

import app.control.viewmodel.MainViewModel

import app.control.widget.dialog.DoubleButtonDialog

import app.control.widget.dialog.SingleButtonDialog

import app.control.widget.toast.CommonToast

import com.trello.rxlifecycle2.components.support.RxAppCompatActivity



/**

 * Created by danbo on 2019/11/07.

 */

abstract class BaseActivity<VM : BaseViewModel> : RxAppCompatActivity(), BaseView {
```

```
companion object {

    const val GESTURE_CODE = 202

    val PERMISSIONS = when {

        Build.VERSION.SDK_INT >= Build.VERSION_CODES.P -> listOf(

            Manifest.permission.READ_EXTERNAL_STORAGE,

            Manifest.permission.READ_PHONE_STATE,

            Manifest.permission.WRITE_EXTERNAL_STORAGE,

            Manifest.permission.WAKE_LOCK,

            Manifest.permission.DISABLE_KEYGUARD,

            Manifest.permission.ACCESS_NETWORK_STATE,

            Manifest.permission.FOREGROUND_SERVICE

        )

        Build.VERSION.SDK_INT >= Build.VERSION_CODES.M -> listOf(

            Manifest.permission.READ_EXTERNAL_STORAGE,

            Manifest.permission.READ_PHONE_STATE,

            Manifest.permission.WRITE_EXTERNAL_STORAGE,

            Manifest.permission.WAKE_LOCK,

            Manifest.permission.DISABLE_KEYGUARD,

            Manifest.permission.ACCESS_NETWORK_STATE

        )

        else -> listOf(

            Manifest.permission.READ_PHONE_STATE,
```

```
                Manifest.permission.WRITE_EXTERNAL_STORAGE,
                Manifest.permission.ACCESS_NETWORK_STATE,
                Manifest.permission.DISABLE_KEYGUARD
            )
        }
    }

    fun setStatusBarColor(statusColor: Int) {
        //取消状态栏透明
        window.clearFlags(WindowManager.LayoutParams.FLAG_TRANSLUCENT_STATUS)
        //添加 Flag 把状态栏设为可绘制模式

window.addFlags(WindowManager.LayoutParams.FLAG_DRAWS_SYSTEM_BAR_BACKGROU
NDS)
        //设置状态栏颜色
        window.statusBarColor = statusColor
        //设置系统状态栏处于可见状态
        window.decorView.systemUiVisibility = View.SYSTEM_UI_FLAG_VISIBLE
        //让 view 不根据系统窗口来调整自己的布局
        val mContentView = window.findViewById(Window.ID_ANDROID_CONTENT) as
ViewGroup
        val mChildView: View? = mContentView.getChildAt(0)
```

```
        if (mChildView != null) {
            ViewCompat.setFitsSystemWindows(mChildView, false)
            ViewCompat.requestApplyInsets(mChildView)
        }
    }

    open lateinit var mBinding: ViewDataBinding

    open var mViewModel: VM? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        window.decorView.systemUiVisibility = (
                View.SYSTEM_UI_FLAG_LAYOUT_STABLE
                        or View.SYSTEM_UI_FLAG_LIGHT_STATUS_BAR)

        //状态栏透明
        window.statusBarColor = Color.TRANSPARENT
        window.navigationBarColor = resources.getColor(R.color.navigationBarColor)

        Utils.init(this)
```

```
ActivityStackManager.getInstance().addActivity(this)


val layoutId = getLayoutId()
if (layoutId == 0) {
    return
}


// Obtain ViewModel from ViewModelProviders
val vm = VMUtil.getVM<VM>(this)
vm?.let {
    mViewModel = ViewModelProviders.of(this).get(vm::class.java)
}
//bind context
mViewModel?.context = this


// Obtain binding
mBinding =
    DataBindingUtil.setContentView(this, layoutId)


//bind ViewModel
mBinding.setVariable(getBR(), mViewModel)
```

```
// LiveData needs the lifecycle owner
mBinding.lifecycleOwner = this


val agreed = SPUtils.getData(AppConstant.LICENCE_AGREED, false) as Boolean
if (agreed) {
    //优先检查授权
    checkPermissions()
} else {//未同意协议弹窗
    DoubleButtonDialog(
        this,
        null, "服务协议和隐私政策",
        null, "退出", "同意", object : DoubleButtonDialog.DialogClickListener {
            override fun onClick(dialog: Dialog) {
                super.onClick(dialog)
                SPUtils.saveData(AppConstant.LICENCE_AGREED, true)
                //检查授权
                checkPermissions()
            }

            override fun onCancel(dialog: Dialog) {
                super.onCancel(dialog)
```

```
                            finish()
                        }
                    }
                ).show()
            }
        }


    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (requestCode == GESTURE_CODE) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M
&& !Settings.canDrawOverlays(this)) {
                CommonToast.middleShow("取消授权")
                //回调用户服务不可用
                if (mViewModel is MainViewModel) {
                    (mViewModel as MainViewModel).onServiceUnavailable("用户取消授权")
                }
            } else {//检查辅助功能
```

```
                    CommonToast.middleShow("授权成功")
                    //继续请求辅助服务
                    checkAccessibility()
                }
                return
            }
            if (PermissionUtils.checkMorePermissions(this, PERMISSIONS.toTypedArray()).size >
0) {

                CommonToast.middleShow("为保证程序功能正常使用,请同意授权请求")
                Handler().postDelayed({
                    //继续请求
                    checkPermissions()
                }, 2000)
            } else {
                //授予成功
                initView()
            }
        }

    fun checkBattery() {
        //电池优化提醒
```

```
        AndroidUtil.ignoreBatteryOptimization(this)
    }


    //1
    fun checkPermissions() {
        PermissionUtils.checkAndRequestMorePermissions(
            this,
            PERMISSIONS.toTypedArray(),
            REQUEST_CODE
        ) { initView() }
    }


    open var checkAccessibility = false//是否检查辅助服务
    fun checkControlPermission(): Boolean {


        if (Build.VERSION.SDK_INT < Build.VERSION_CODES.N) {
            checkAccessibility = false


            SingleButtonDialog(
                this,
                R.drawable.ic_warning,
                null,
```

```
                "手势操作暂不支持您的系统", "确定",
                null
            ).show()

            //android 7.0 以下不可控制
            if (mViewModel is MainViewModel) {
                (mViewModel as MainViewModel).onServiceUnavailable("用户系统不支持控制
")
            }
            return false
        }

        checkAccessibility = true

        if (!Settings.canDrawOverlays(this)) {//没有全屏权限
            DoubleButtonDialog(
                this,
                null,
                "授权",
                "控制操作需要开启全屏监控", "取消", "授权",
                object : DoubleButtonDialog.DialogClickListener {
                    override fun onClick(dialog: Dialog) {
```

```
                        openDrawOverlays()
                    }


                    override fun onCancel(dialog: Dialog) {
                        super.onCancel(dialog)
                        checkAccessibility = false
                        if (mViewModel is MainViewModel) {
                            (mViewModel as MainViewModel).onServiceUnavailable("用户取消控制")
                        }
                    }


                }
            ).show()
            return false
        } else {//检查辅助权限
            return checkAccessibility()
        }
    }


    fun openDrawOverlays() {
        startActivityForResult(
```

```
            Intent(
                Settings.ACTION_MANAGE_OVERLAY_PERMISSION,
                Uri.parse("package:$packageName")
            ), GESTURE_CODE
        )
    }


    //检查辅助功能权限
    private fun checkAccessibility(): Boolean {
        return if (!BaseAccessibilityService.getInstance().checkAccessibilityEnabled(false)) {
            //服务没有在运行,打开辅助页面
            DoubleButtonDialog(
                this,
                null,
                "授权",
                "控制操作需要在 无障碍/辅助功能 中\n 开启远程耦控制", "取消", "启用",
                object : DoubleButtonDialog.DialogClickListener {
                    override fun onClick(dialog: Dialog) {
                        if (mViewModel is MainViewModel) {
                            BaseAccessibilityService.getInstance()
                                .goAccess(mViewModel as MainViewModel)
                        } else {
```

```
                            BaseAccessibilityService.getInstance().goAccess(null)
                        }
                    }

                    override fun onCancel(dialog: Dialog) {
                        super.onCancel(dialog)
                        checkAccessibility = false
                        if (mViewModel is MainViewModel) {
                            (mViewModel as MainViewModel).onServiceUnavailable("用户取消控制!")
                        }
                    }
                }
            ).show()
            false
        } else {//
            //服务已启用
            if (mViewModel is MainViewModel) {
                (mViewModel as MainViewModel).onServiceConnected()
            }
            checkAccessibility = false
```

```
            true
        }
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        if (menuResId() != 0) {
            menuInflater.inflate(menuResId(), menu)
        }
        return super.onCreateOptionsMenu(menu)
    }

    override fun getBR(): Int {
        return BR.viewModel
    }

    @MenuRes
    protected open fun menuResId(): Int {
        return 0
    }

    /**
     * 创建 Fragment
```

```
     */
    fun startFragment(fragment: Fragment?, @IdRes layout: Int) {
        if (fragment == null) {
            return
        }

        supportFragmentManager.beginTransaction()
            .replace(layout, fragment).commit()
    }

    override fun onDestroy() {
        super.onDestroy()
        ActivityStackManager.getInstance().removeActivity(this)
    }
}

package app.control.ui

import SPUtils
import android.app.Dialog
import android.content.Intent
import android.os.Handler
import app.control.*
```

```
import app.control.dmanager.download.DownloadManager

import app.control.model.MemberInfo

import app.control.viewmodel.SplashViewModel

import app.control.widget.dialog.DoubleButtonDialog

import com.google.gson.Gson



/**

 * Created by danbo on 2019/11/07.

 */

class SplashActivity : BaseActivity<SplashViewModel>(), SyncListener,

    DoubleButtonDialog.DialogClickListener {



    override fun getLayoutId(): Int {

        return R.layout.activity_spalash

    }



    override fun initView() {

        //在应用层就获取 token 同步用户信息

        token = SPUtils.getData(AppConstant.TOKEN, "") as String

        if (token.isNotEmpty()) {

            memberInfo = Gson().fromJson(

                SPUtils.getData(AppConstant.MEMBER_INFO, "") as String,
```

```
                MemberInfo::class.java
            )
        }

        //清理库
        DownloadManager.getInstance().clear()

        init()
    }

    private fun init() {
        Handler().postDelayed({
            if (token.isEmpty()) {//未登录
                jumpToLogin()
            } else {//已登录
                jumpToHome()
            }
        }, 3000)
    }

    private fun jumpToHome() {
        //token 刷新成功,同步用户信息
```

```
        refreshToken(this, false, this)
    }

    private fun jumpToLogin() {
        startActivity(Intent(this, LoginActivity::class.java))
        finish()
    }

    override fun onSyncSuccess() {
        //同步完成跳转至首页
        startActivity(Intent(this, MainActivity::class.java))
        finish()
    }

    override fun onSyncFail(reason: String?) {
        if (this.isDestroyed) {
            return
        }
        DoubleButtonDialog(
            this,
            R.drawable.ic_warning,
            null,
```

```
            "同步失败,请检查网络!",
            "退出",
            "重试",
            this
        ).show()
    }

    override fun onClick(dialog: Dialog) {
        //
        refreshToken(this, true, this)
    }

    override fun onCancel(dialog: Dialog) {
        finish()
    }

    override fun onBackPressed() {
        //super.onBackPressed()
    }

}
```

```
package app.control.ui


import app.control.BR

import app.control.R

import app.control.checkUpdate

import app.control.viewmodel.LoginViewModel


class LoginActivity : BaseActivity<LoginViewModel>() {

    override fun getLayoutId(): Int {

        return R.layout.activity_login

    }


    override fun initView() {

        mBinding.setVariable(BR.data, mViewModel?.titleViewModel)

        mViewModel?.titleViewModel?.context = mBinding.root.context


        checkUpdate(this, true)

    }
package app.control.viewmodel


import SPUtils

import android.content.Intent
```

```
import android.text.Editable

import android.text.TextWatcher

import android.view.View

import androidx.databinding.ObservableField

import androidx.lifecycle.MutableLiveData

import app.control.AppConfig

import app.control.AppConstant.ACCOUNT

import app.control.AppConstant.MEMBER_INFO

import app.control.AppConstant.MENU_TITLE

import app.control.AppConstant.TOKEN

import app.control.AppConstant.URL

import app.control.R

import app.control.common.CountDownManager

import app.control.common.util.Base64

import app.control.common.util.PhoneUtil

import app.control.common.util.ScreenInfo

import app.control.memberInfo

import app.control.model.LoginModel

import app.control.net.RetrofitProvider

import app.control.net.response.BasicResponse

import app.control.net.subscribers.DefaultObserver

import app.control.token
```

```
import app.control.ui.*

import app.control.widget.SimpleTextWatcher

import app.control.widget.dialog.SingleButtonDialog

import app.control.widget.toast.CommonToast

import com.google.gson.Gson

import io.reactivex.android.schedulers.AndroidSchedulers

import io.reactivex.schedulers.Schedulers

import org.reactivestreams.Subscription

import java.nio.charset.Charset



/**

 * Created by danbo on 2019-11-07.

 */

class LoginViewModel : BaseViewModel(), OnTitleClickListener {

    /**

     * 倒计时

     */

    var mSubscription: Subscription? = null



    val isRegister = ObservableField<Boolean>().apply {

        set(false)

    }
```

```
val verifyLogin = MutableLiveData<Boolean>().apply {
    value = false
}

val titleViewModel = TopBarViewModel("登录", null, "注册", this)

//密码输入区是否可见
var passwordEnable = ObservableField<Boolean>().apply {
    set(true)
}
var commitEnable = ObservableField<Boolean>().apply {
    set(false)
}
var account = MutableLiveData<String>().apply {
    value = ""
}
var verifyAuthCode = MutableLiveData<String>().apply {
    value = ""
}
var verifyEnable = ObservableField<Boolean>().apply {
    set(false)
}
```

```
var password = MutableLiveData<String>().apply {
        value = ""
}

init {
        account.value = SPUtils.getData(ACCOUNT, "") as String
}

fun cleanAccount(view: View) {
        //清空账号取消保存
        cleanSavedAccount()
}

private fun cleanSavedAccount() {
        account.value = ""
        password.value = ""

        SPUtils.saveData(ACCOUNT, "")
}

var editAccountWatcher: TextWatcher = object : SimpleTextWatcher() {
```

```
        override fun afterTextChanged(s: Editable?) {
            passwordVisible.value = passwordVisible.value

            if (account.value.isNullOrEmpty() || account.value?.length != 11) {
                verifyEnable.set(false)
                commitEnable.set(false)
            } else {
                verifyEnable.set(true)
                if (!password.value.isNullOrEmpty()) {
                    commitEnable.set(true)
                }
            }

        }
    }
    var countdown = MutableLiveData<String>().apply {
        value = "获取验证码"
    }
    var verifyCodeWatcher: TextWatcher = object : SimpleTextWatcher() {
        override fun afterTextChanged(s: Editable?) {
            if (verifyAuthCode.value.isNullOrEmpty()) {
                if (isRegister.get()!!) {
```

```
                    passwordEnable.set(false)
                } else {
                    //验证码登录
                    commitEnable.set(false)
                }

            } else {
                if (verifyEnable.get()!! && verifyAuthCode.value!!.length == 6) {
                    if (isRegister.get()!!) {
                        passwordEnable.set(true)
                        commitEnable.set(true)
                    } else {
                        //验证码登录
                        commitEnable.set(true)
                    }
                }
            }

        }
    }
    var editPasswordWatcher: TextWatcher = object : SimpleTextWatcher() {
        override fun afterTextChanged(s: Editable?) {
```

```
            if (password.value.isNullOrEmpty()) {
                commitEnable.set(false)
            } else {
                if(verifyAuthCode.value?.length == 6){
                    verifyEnable.set(true)
                }
                if (verifyEnable.get()!!) {
                    if (isRegister.get()!!) {
                        commitEnable.set(!(password.value.isNullOrEmpty()))
                    } else {
                        commitEnable.set(true)
                    }
                }
            }

        }
    }

    //密码是否可见
    var passwordVisible = MutableLiveData<Boolean>().apply {
        value = false
    }
```

```
fun swapVisible(view: View) {
        passwordVisible.value?.let {
            passwordVisible.value = !it
        }
}

fun swapLogin(view: View) {
        verifyLogin.value?.let {
            verifyLogin.value = !it
        }

        if (verifyLogin.value!!) {//验证码登录
            //密码区不可见
            passwordEnable.set(false)
            password.value = ""
        } else {//密码登录
            passwordEnable.set(true)
            verifyAuthCode.value = ""
        }
}
```

```
fun updatePassword(view: View) {
        startActivity(UpdatePasswordActivity::class.java)
}


override fun onMenuClick() {
        super.onMenuClick()
        if (isRegister.get()!!) {
            //切换至登录
            titleViewModel.title.value = "登录"
            titleViewModel.menu.value = "注册"

            isRegister.set(false)
            passwordEnable.set(true)

            //恢复密码登录
            verifyLogin.value = false
            //预处理登录

        } else {
            //切换至注册
            titleViewModel.title.value = "注册"
            titleViewModel.menu.value = "登录"
```

```
            isRegister.set(true)
            passwordEnable.set(false)
            //预处理注册
        }
    }

    fun getVerifyCode(view: View) {
        if (!verifyEnable.get()!!) {
            return
        }

        if (PhoneUtil.isMobileNO(account.value) == false) {
            CommonToast.middleShow("请输入正确的手机号码!")
            return
        }
        if (!isRegister.get()!!) {//是登录
            getVerifyCode()
            return
        }

        //检查用户是否已注册
```

```
        (context as BaseActivity<*>).let {
            RetrofitProvider.getApiService()
                .checkMemberExist(account.value)
                .compose(it.bindToLifecycle())
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(object : DefaultObserver<BasicResponse<Any>>(it, true) {
                    override fun onSuccess(response: BasicResponse<Any>) {
                        //成功及手机号不存在
                        getVerifyCode()
                    }
                })
        }

    }

    private fun getVerifyCode() {
        (context as BaseActivity<*>).let {
            RetrofitProvider.getApiService()
                .getVerifyCode(account.value)
                .compose(it.bindToLifecycle())
                .subscribeOn(Schedulers.io())
```

```
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(object : DefaultObserver<BasicResponse<Any>>(it, true) {
                override fun onSuccess(response: BasicResponse<Any>) {
                    CommonToast.middleShow("验证码已发送,请注意查收!")
                    CountDownManager.timeCountDown(
                        this@LoginViewModel, 59,
                        mSubscription
                    )
                }

                override fun onFail(response: BasicResponse<Any>) {
                    super.onFail(response)
                    countdown.value = "获取验证码"
                    verifyEnable.set(true)

                    SingleButtonDialog(
                        context!!,
                        R.drawable.ic_warning,
                        null,
                        response.message,
                        "我知道了",
                        null
```

```
                    ).show()
                }

                override fun onException(reason: ExceptionReason) {
                    super.onException(reason)
                    countdown.value = "获取验证码"
                    verifyEnable.set(true)
                }
            })
    }
}


fun commit(view: View) {
    if (!commitEnable.get()!!) {
        return
    }

    if (PhoneUtil.isMobileNO(account.value) == false) {
        CommonToast.middleShow("请输入正确的手机号码!")
        return
    }
```

```
        if (isRegister.get()!!) {
            register()
        } else {
            login()
        }
    }

    private fun login() {
        if (verifyLogin.value!!) {
            //验证码登录
            (context as BaseActivity<*>).let {
                val screenInfo =
                    Base64.encode(
                        Gson().toJson(ScreenInfo.getInstance())
                            .toByteArray(Charset.defaultCharset())
                    )
                RetrofitProvider.getApiService()
                    .verifyLogin(account.value, verifyAuthCode.value, screenInfo)
                    .compose(it.bindToLifecycle())
                    .subscribeOn(Schedulers.io())
                    .observeOn(AndroidSchedulers.mainThread())
```

```
                    .subscribe(object : DefaultObserver<BasicResponse<LoginModel>>(it,
true) {

                        override fun onSuccess(response: BasicResponse<LoginModel>) {



                            response.data?.let {

                                token = it.token//登录 token 不为 null

                                memberInfo = it.member



                                SPUtils.saveData(ACCOUNT, account.value)

                                SPUtils.saveData(MEMBER_INFO, Gson().toJson(it.member))

                                SPUtils.saveData(TOKEN, token)



                                startActivity(MainActivity::class.java)

                                finish()

                            }

                        }

                    })

            }

            return

        }

        (context as BaseActivity<*>).let {

            val screenInfo =
```

```
                    Base64.encode(


Gson().toJson(ScreenInfo.getInstance()).toByteArray(Charset.defaultCharset())

                    )
                RetrofitProvider.getApiService()
                    .login(account.value, password.value, screenInfo)
                    .compose(it.bindToLifecycle())
                    .subscribeOn(Schedulers.io())
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribe(object : DefaultObserver<BasicResponse<LoginModel>>(it, true) {
                        override fun onSuccess(response: BasicResponse<LoginModel>) {


                            response.data?.let {
                                token = it.token//登录 token 不为 null
                                memberInfo = it.member


                                SPUtils.saveData(ACCOUNT, account.value)
                                SPUtils.saveData(MEMBER_INFO, Gson().toJson(it.member))
                                SPUtils.saveData(TOKEN, token)


                                startActivity(MainActivity::class.java)
                                finish()
```

```
                        }

                    }

                })

        }

    }


    fun deviceLogin(view: View) {

        (context as BaseActivity<*>).let {

            val screenInfo =

                Base64.encode(


Gson().toJson(ScreenInfo.getInstance()).toByteArray(Charset.defaultCharset())

                )

            RetrofitProvider.getApiService()

                .deviceLogin(screenInfo)

                .compose(it.bindToLifecycle())

                .subscribeOn(Schedulers.io())

                .observeOn(AndroidSchedulers.mainThread())

                .subscribe(object : DefaultObserver<BasicResponse<LoginModel>>(it, true) {

                    override fun onSuccess(response: BasicResponse<LoginModel>) {


                        response.data?.let {
```

```
                        token = it.token//登录 token 不为 null
                        memberInfo = it.member

                        SPUtils.saveData(MEMBER_INFO, Gson().toJson(it.member))
                        SPUtils.saveData(TOKEN, token)

                        startActivity(MainActivity::class.java)
                        finish()
                    }
                }
            })
    }
}

private fun register() {
    if (PhoneUtil.isMobileNO(account.value) == false) {
        CommonToast.middleShow("请输入正确的手机号码!")
        return
    }

    if (verifyAuthCode.value?.length != 6) {
        CommonToast.middleShow("请输入正确的验证码!")
```

```
            return
        }

        (context as BaseActivity<*>).let {
            val screenInfo =
                Base64.encode(

Gson().toJson(ScreenInfo.getInstance()).toByteArray(Charset.defaultCharset())
                )
            RetrofitProvider.getApiService()
                .register(account.value, password.value, verifyAuthCode.value, screenInfo)
                .compose(it.bindToLifecycle())
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(object : DefaultObserver<BasicResponse<LoginModel>>(it, true) {
                    override fun onSuccess(response: BasicResponse<LoginModel>) {

                        response.data?.let {
                            token = it.token//登录 token 不为 null
                            memberInfo = it.member

                            SPUtils.saveData(ACCOUNT, account.value)
```

```
                            SPUtils.saveData(MEMBER_INFO, Gson().toJson(it.member))
                            SPUtils.saveData(TOKEN, token)

                            startActivity(MainActivity::class.java)
                            finish()
                        }
                    }
                })
        }
    }

    fun agreement(view: View) {
        val intent = Intent(context, WebViewActivity::class.java)
        intent.putExtra(URL, AppConfig.POLICY_URL)
        intent.putExtra(MENU_TITLE, "介绍")

        context?.startActivity(intent)
    }
}

}
```

```
package app.control.viewmodel



import android.app.Dialog

import android.content.Intent

import android.os.Build

import android.view.View

import app.control.ControlApplication

import app.control.adapter.HomeNavigatorAdapter

import app.control.adapter.ViewPagerAdapter

import app.control.common.util.AndroidUtil

import app.control.common.util.AsyncManager

import app.control.common.util.NetworkUtils.Companion.NETWORK_NONE

import app.control.common.util.NotifyUtils

import app.control.common.util.VibratorUtil

import app.control.gesture.ConnectManager

import app.control.gesture.accessibility.AccessibilityListener

import app.control.gesture.background.GestureReceiver.Companion.CONTROL_ACTION

import

app.control.gesture.background.GestureReceiver.Companion.CONTROL_ACTION_START

import app.control.model.BindMember

import app.control.net.RetrofitProvider

import app.control.net.domain.ConnectRequest
```

```
import app.control.net.domain.ConnectResponse

import app.control.net.domain.ControlResponse

import app.control.net.domain.SocketMessage

import app.control.net.response.BasicResponse

import app.control.net.subscribers.DefaultObserver

import app.control.receiver.NetworkEvent

import app.control.receiver.SocketBroadcastReceiver

import app.control.receiver.SocketEvent

import app.control.ui.BaseActivity

import app.control.ui.MainActivity

import app.control.ui.SearchActivity

import app.control.ui.fragment.BaseFragment

import app.control.ui.fragment.HostFragment

import app.control.ui.fragment.TargetFragment

import app.control.widget.dialog.DoubleButtonDialog

import app.control.widget.tab.ViewPagerHelper

import app.control.widget.tab.buildins.commonnavigator.CommonNavigator

import app.control.widget.toast.CommonToast

import com.google.gson.Gson

import io.reactivex.android.schedulers.AndroidSchedulers

import io.reactivex.schedulers.Schedulers

import kotlinx.android.synthetic.main.activity_main.view.*
```

```
import java.util.*


/**
 * Created by danbo on 2019-11-07.
 */
class MainViewModel : BaseViewModel(), NetworkEvent,
    SocketEvent, AccessibilityListener {


    val connectedState = ControlApplication.getSocketClient()?.connectingState


    private val titles: ArrayList<String> = arrayListOf()
    private val fragments = arrayListOf<BaseFragment<*>>()
    fun initPager() {
        //添加推荐
        titles.add("控制列表")
        titles.add("申请列表")
        fragments.add(TargetFragment())//我控谁列表
        fragments.add(HostFragment())//谁控我列表



        //adapter
        (context as MainActivity).let {
```

```
            it.mBinding.root.view_pager.apply {

                adapter = ViewPagerAdapter(it.supportFragmentManager, fragments)



                //navigator

                val navigator = CommonNavigator(context)

                navigator.isAdjustMode = true

                navigator.adapter = HomeNavigatorAdapter(this, titles)



                it.mBinding.root.indicator.navigator = navigator



                //bind

                ViewPagerHelper.bind(it.mBinding.root.indicator, this)

            }

        }

    }



    override fun onNetworkChange(netWorkState: Int) {

        //网络切换就断开连接

        ConnectManager.getInstance().stopConnect(context!!)



        if (netWorkState == NETWORK_NONE) {

            CommonToast.middleShow("无网络连接！")
```

```
            return
        }

        reConnect(null)
    }

    fun reConnect(view: View?) {
        ControlApplication.getSocketClient()?.startRequest()
    }

    fun fabClick(view: View) {
        startActivity(SearchActivity::class.java)
    }

    override fun onReceiveMessage(socketMessage: SocketMessage) {
        when (SocketMessage.Type.valueOf(socketMessage.type)) {
            SocketMessage.Type.BIND_REQUEST -> {
                //切换到被控列表并刷新
                (context as MainActivity).let {
                    it.mBinding.root.view_pager.setCurrentItem(1, true)
                    fragments[it.mBinding.root.view_pager.currentItem].refreshData("绑定请
求")
```

```
                }


            }
            SocketMessage.Type.BIND_RESPONSE -> {
                //切换到控制列表并刷新
                (context as MainActivity).let {
                    it.mBinding.root.view_pager.setCurrentItem(0, true)
                    fragments[it.mBinding.root.view_pager.currentItem].refreshData("绑定响
应")
                }
            }
            SocketMessage.Type.CONNECT_REQUEST -> {
                //解锁屏幕
                AndroidUtil.wakeUpAndUnlock(context!!)


                //用户请求连接
                val connectRequest = Gson().fromJson(socketMessage.data,
ConnectRequest::class.java)
                handleConnectRequest(connectRequest)
            }
            SocketMessage.Type.CONTROL_REQUEST -> {
                //用户请求控制
```

```
                handleControlRequest()
            }
            SocketMessage.Type.DISCONNECT_REQUEST -> {//断开连接请求
                //停止处理
                ConnectManager.getInstance().connectedBindMember?.let {
//                    startActivity(MainActivity::class.java)
//                    Handler().postDelayed({
//                        SingleButtonDialog(
//                            context!!,
//                            R.drawable.ic_remote_off, null,
//                            "用户断开了您的连接！", "确定", null
//                        ).show()
//                    }, 500)
                    CommonToast.middleShow("用户断开了您的连接！")
                }

                ConnectManager.getInstance().stopConnect(context!!)
            }
        }
    }

    private fun handleConnectRequest(connectRequest: ConnectRequest) {
```

```
        context.let {
            RetrofitProvider.getApiService()
                .getHostBindMember(connectRequest.memberId)
                .compose((it as BaseActivity<*>).bindToLifecycle())
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(object : DefaultObserver<BasicResponse<BindMember>>(it, true)
{

                    override fun onSuccess(response: BasicResponse<BindMember>) {
                        if (response.data == null) {
                            doConnectResponse(false)
                            return
                        }
                        response.data?.let {
                            ConnectManager.getInstance().connectedBindMember = it

                            if (it.connect == 1) {//默认直连,直接请求截屏
                                requestScreeShot()
                                return
                            }
                            //如果 MainActivity 不在前台,不弹框
                            if ((context as MainActivity).isPaused) {
```

```
            return
        }
        //弹框提示同意/拒绝连接
        val reason = if (connectRequest.reason.isNullOrEmpty()) {
            ""
        } else {
            "(${connectRequest.reason})"
        }

        DoubleButtonDialog(
            context!!,
            null,
            "连接提示",
            "${it.member?.username} 请求连接\n$reason",
            "拒绝", "同意",
            object : DoubleButtonDialog.DialogClickListener {
                override fun onClick(dialog: Dialog) {
                    //同意后请求截屏
                    requestScreeShot()
                }

                override fun onCancel(dialog: Dialog) {
```

```
                                    //拒绝连接响应
                                    doConnectResponse(false)
                                }


                            }
                        ).show()
                    }
                }

                override fun onFail(response: BasicResponse<BindMember>) {
                    super.onFail(response)
                    doConnectResponse(false)
                }

                override fun onException(reason: ExceptionReason) {
                    super.onException(reason)
                    doConnectResponse(false)
                }
            })
        }
    }
```

```
//同意后请求截屏
private fun requestScreeShot() {
    ConnectManager.getInstance().connectedBindMember?.let {
        (context as MainActivity).requestScreenShot()
    }
}

fun doConnectResponse(accept: Boolean) {
    ConnectManager.getInstance().connectedBindMember?.let {
        NotifyUtils.cancel(SocketBroadcastReceiver.SOCKET_CONNECT_NOTIFY_ID)

        val gson = Gson()

        val connectResponse = ConnectResponse()
        connectResponse.memberId = it.member!!.id
        connectResponse.isAccept = accept
        connectResponse.isOnline = true

        val socketMessage = SocketMessage()
        socketMessage.type = SocketMessage.Type.CONNECT_RESPONSE.value()
        socketMessage.data = gson.toJson(connectResponse)
```

```
ControlApplication.getSocketClient()?.sendMessage(gson.toJson(socketMessage))


            if (!accept) {   //拒绝连接
                //停止掉所有
                ConnectManager.getInstance().stopConnect(context!!)
            }
        }

    }

    private fun handleControlRequest() {
        ConnectManager.getInstance().connectedBindMember?.let {
            if (Build.VERSION.SDK_INT < Build.VERSION_CODES.N) {
                //android 7.0 以下不可控制
                doControlResponse(false, "用户系统不支持控制!")
                return
            }
            //如果默认允许控制,则直接连
            if (it.control == 1) {
                //检查辅助授权
                (context as MainActivity).checkControlPermission()
```

```
            return
        }


        //如果 MainActivity 不在前台,不弹框
        if ((context as MainActivity).isPaused) {
            return
        }
        //弹框提示同意/拒绝连接
        DoubleButtonDialog(
            context!!,
            null,
            "控制提示",
            "${it.member?.username} 请求控制您的手机!",
            "拒绝", "同意",
            object : DoubleButtonDialog.DialogClickListener {
                override fun onClick(dialog: Dialog) {
                    //同意后请求辅助功能
                    (context as MainActivity).checkControlPermission()
                }


                override fun onCancel(dialog: Dialog) {
                    //拒绝连接响应
```

```
                        doControlResponse(false, "用户拒绝控制!")
                    }


                }
            ).show()
        }
    }

    override fun onServiceConnected() {
        ConnectManager.getInstance().connectedBindMember?.let {
            //服务已准备好,组织响应
            doControlResponse(true, "")

            //允许了控制后开启手势服务
            startRecord()
        }
    }

    override fun onServiceUnavailable(reason: String?) {
        ConnectManager.getInstance().connectedBindMember?.let {
            //服务已准备好,组织响应
            doControlResponse(false, reason)
```

```
        }


    }


    private fun doControlResponse(accept: Boolean, reason: String?) {
        ConnectManager.getInstance().connectedBindMember?.let {
            AsyncManager.me().execute({
                NotifyUtils.cancel(SocketBroadcastReceiver.SOCKET_CONTROL_NOTIFY_ID)
            }, 1000)

            val gson = Gson()

            val controlResponse = ControlResponse()
            controlResponse.memberId = it.member!!.id
            controlResponse.isOnline = true
            controlResponse.isAccept = accept
            controlResponse.reason = reason

            val socketMessage = SocketMessage()
            socketMessage.type = SocketMessage.Type.CONTROL_RESPONSE.value()
            socketMessage.data = gson.toJson(controlResponse)
```

```
ControlApplication.getSocketClient()?.sendMessage(gson.toJson(socketMessage))
        }


    }


    //开始执行接收执行手势
    private fun startRecord() {
        context?.apply {
            VibratorUtil.vibrate(this)
            val intent = Intent()
            intent.action = CONTROL_ACTION
            //开始录制
            intent.putExtra(CONTROL_ACTION, CONTROL_ACTION_START)


            sendBroadcast(intent)
        }


    }
}
```