

**杨海**

**15680571195**

**2023年5月4日**

2023年Android面试题

目录

[1. java面试题目 6](#_Toc1992987845)

[1.1 面向对象的基本概念 6](#_Toc1844268656)

[1.2 什么是匿名内部类？匿名内部类有什么特点？ 6](#_Toc1991824241)

[1.3 java类加载过程？为什么采用双亲委托加载加载？该加载机制有什么好处？如何打破双亲委托加载机制？ 6](#_Toc1614929051)

[1.4 java中如何实现多线程？如何保证线程安全？ 6](#_Toc66745724)

[1.5什么是反射？如何使用反射？如何优化反射？ 6](#_Toc808919534)

[1.6 什么是注解处理器？如何使用？ 6](#_Toc1939122428)

[2. kotlin面试题目 6](#_Toc618820524)

[2.1 协程的使用及注意事项 6](#_Toc253244447)

[2.2 kotlin中提供的高阶函数有哪些，分别是什么样的使用场景？ 6](#_Toc2114316022)

[2.3 data class ,inner class ,scaled class ,emum class 等使用场景的优势？ 6](#_Toc897474845)

[2.4 什么是扩展函数？ 6](#_Toc2082067034)

[2.5 什么是DSL? 6](#_Toc54612573)

[2.6 kotlin中的反射如何使用？ 7](#_Toc897997142)

[2.7 kotlin中其他重要的关键字，如lazy，init{} 7](#_Toc122894478)

[2.8 泛型如何支持的？ 7](#_Toc1755706979)

[3. jetpack相关面试题目 7](#_Toc1741886273)

[3.1 什么是ViewBinding? 7](#_Toc1385514407)

[3.2 什么是DataBinding? 7](#_Toc1175454028)

[3.3 什么是LiveData以及类似的功能的框架? 7](#_Toc1153779843)

[3.4 什么是ViewModel？为什么ViewModel可以在手机横竖屏时不会进行重建？ 7](#_Toc1947972538)

[3.5 什么是Room?Room如何实现升级？ 7](#_Toc1186247651)

[3.6 什么是Navigation？有什么好处？那些使用场景？ 7](#_Toc26091609)

[3.7 什么是Lifecycle？有哪些应用场景？ 7](#_Toc435008475)

[3.8 什么是Hilt？有哪些特殊的使用场景需要注意？ 7](#_Toc1153104937)

[3.9 什么是Pagging3？和RecyclerView有什么优缺点？ 7](#_Toc1342245631)

[4. Android架构相关面试题目 7](#_Toc1954092129)

[4.1 什么是MVC？ 8](#_Toc958998532)

[4.2 什么是MVP?MVP如何实现？有什么优缺点？ 8](#_Toc1023556589)

[4.3 什么是MVVM?如何实现？什么是单向数据流？有什么好处？什么是双向数据流？如何改为单向数据流？ 8](#_Toc1571578853)

[4.4 什么是组件化？ 8](#_Toc1624407918)

[4.5什么是插件化？ 8](#_Toc464273515)

[4.6 serializable和parcelable的使用场景？ 8](#_Toc1236877054)

[5. 第三方库使用及实现原理 8](#_Toc550943618)

[5.1 Okhttp3的使用及原理 8](#_Toc1907385509)

[5.2 Retrofit的使用及原理 8](#_Toc1939850994)

[5.3 Glide的使用及原理 8](#_Toc2126411051)

[5.4 Rxjava的使用，原理及注意点 8](#_Toc167680783)

[5.5 Hlit使用，原理及特殊场景 8](#_Toc712375017)

[5.6 LeakCannery的实现原理 8](#_Toc665578694)

[5.7 Handler+Message实现原理 8](#_Toc138792835)

[5.8 RecyclerView实现原理及特殊场景问题解决 8](#_Toc523937203)

[6. 性能优化相关题目 9](#_Toc1129618121)

[6.1 启动优化注意点及实现方案 9](#_Toc1736320167)

[6.1.1. Android的启动流程是什么？ 9](#_Toc177767686)

[6.2内存抖动发现及优化手段 9](#_Toc591745625)

[6.3内存泄露发现及解决办法 9](#_Toc471950118)

[6.4包体积优化 9](#_Toc1408524855)

[6.4.1图片大小计算方法 9](#_Toc1364997104)

[6.4.2 如何检查无效资源删除 9](#_Toc2086009674)

[6.4.3如何有效找到未使用的文件 9](#_Toc1894053643)

[6.4.4 如何有效解决包体积问题 9](#_Toc1209478420)

[6.5电量优化 9](#_Toc1771086085)

[6.6 如何加固？ 9](#_Toc372999528)

[7. 数据结构和算法 10](#_Toc498301503)

[7.1 如何判定一个程序的时间复杂度？ 10](#_Toc1914621268)

[7.2 常用的数据结构如HashMap采用的是什么数据结构？在高并发场景有什么问题？如何解决？CurrentHashMap。 10](#_Toc1144684628)

[7.3 SpareArray数据结构是如何实现的？有什么优缺点？ 10](#_Toc1556032970)

[8. jetpackCompose面试相关题目 10](#_Toc190273624)

[8.1 什么是响应式编程？ 10](#_Toc325648185)

[8.2 什么是重组？ 10](#_Toc1380712739)

[9. flutter面试题目 10](#_Toc2078198538)

[10. JNI面试相关题目 10](#_Toc1608793358)

[10.1 C和C++常用的知识点，如extern的使用场景？ 10](#_Toc23368529)

[10.2 如何实现交叉编译？ 10](#_Toc1912843149)

[11. 音视频开发相关题目 10](#_Toc1324609653)

[11.1 Ffmpeg的使用及基础算法 11](#_Toc1898953169)

[11.2 OpenCV的使用及注意事项 11](#_Toc1951433316)

[12. 计算机网络 11](#_Toc1369485028)

[12.1 什么是TCP/IP，TCP是如何保证通信的可靠性的？ 11](#_Toc205137050)

[12.2 什么是https？ 14](#_Toc1027145915)

[13. 业务场景 14](#_Toc1767838819)

[13.1 输入支付密码逻辑设计 14](#_Toc1630774688)

[13.2 登录状态管理 14](#_Toc96394555)

[13.3 二维码登录逻辑设计 14](#_Toc900616047)

# java面试题目

## 1.1 面向对象的基本概念

## 1.2 什么是匿名内部类？匿名内部类有什么特点？

## 1.3 java类加载过程？为什么采用双亲委托加载加载？该加载机制有什么好处？如何打破双亲委托加载机制？

## 1.4 java中如何实现多线程？如何保证线程安全？

## 1.5什么是反射？如何使用反射？如何优化反射？

## 1.6 什么是注解处理器？如何使用？

# kotlin面试题目

## 2.1 协程的使用及注意事项

## 2.2 kotlin中提供的高阶函数有哪些，分别是什么样的使用场景？

## 2.3 data class ,inner class ,scaled class ,emum class 等使用场景的优势？

## 2.4 什么是扩展函数？

## 2.5 什么是DSL?

## 2.6 kotlin中的反射如何使用？

## 2.7 kotlin中其他重要的关键字，如lazy，init{}

## 2.8 泛型如何支持的？

# jetpack相关面试题目

## 3.1 什么是ViewBinding?

## 3.2 什么是DataBinding?

## 3.3 什么是LiveData以及类似的功能的框架?

## 3.4 什么是ViewModel？为什么ViewModel可以在手机横竖屏时不会进行重建？

## 3.5 什么是Room?Room如何实现升级？

## 3.6 什么是Navigation？有什么好处？那些使用场景？

## 3.7 什么是Lifecycle？有哪些应用场景？

## 3.8 什么是Hilt？有哪些特殊的使用场景需要注意？

## 3.9 什么是Pagging3？和RecyclerView有什么优缺点？

## 3.10 DataStore的使用

### 3.10.1 导入配置信息

导入配置注意事项：放在最前面，不然会出现不能导入成功的问题

|  |
| --- |
| *//datastore 导入配置是放在最前面，不然会出现不能导入成功的问题* implementation "androidx.datastore:datastore-preferences:1.0.0"  implementation "androidx.datastore:datastore-preferences-core:1.0.0" |

### 3.10.2 创建工具类

|  |
| --- |
| import android.content.Context import androidx.datastore.preferences.core.\* import androidx.datastore.preferences.preferencesDataStore  import kotlinx.coroutines.flow.Flow import kotlinx.coroutines.flow.map  *//配置的文件名字* private val Context.*dataStore* by *preferencesDataStore*(name = "user\_preferences")  class DataStoreUtil(context: Context) {  private val dataStore = context.*dataStore* companion object {  *//每个单独的key的名字* private val USER\_NAME\_KEY = *stringPreferencesKey*("user\_name")  private val USER\_AGE\_KEY = *intPreferencesKey*("user\_age")  private val USER\_IS\_STUDENT\_KEY = *booleanPreferencesKey*("user\_is\_student")  private val USER\_GRADE\_KEY = *floatPreferencesKey*("user\_grade")  }   suspend fun setUserName(name: String) {  dataStore.edit **{** preferences **->** preferences[USER\_NAME\_KEY] = name  **}** }  suspend fun setUserAge(age: Int) {  dataStore.edit **{** preferences **->** preferences[USER\_AGE\_KEY] = age  **}** }  suspend fun setUserIsStudent(isStudent: Boolean) {  dataStore.edit **{** preferences **->** preferences[USER\_IS\_STUDENT\_KEY] = isStudent  **}** }  suspend fun setUserGrade(grade: Float) {  dataStore.edit **{** preferences **->** preferences[USER\_GRADE\_KEY] = grade  **}** }  val userNameFlow: Flow<String> = dataStore.data  .*map* **{** preferences **->** preferences[USER\_NAME\_KEY] ?: ""  **}** val userAgeFlow: Flow<Int> = dataStore.data  .*map* **{** preferences **->** preferences[USER\_AGE\_KEY] ?: 0  **}** val userIsStudentFlow: Flow<Boolean> = dataStore.data  .*map* **{** preferences **->** preferences[USER\_IS\_STUDENT\_KEY] ?: false  **}** val userGradeFlow: Flow<Float> = dataStore.data  .*map* **{** preferences **->** preferences[USER\_GRADE\_KEY] ?: 0f  **}** } |

*internal备注：在 Kotlin 中，internal 修饰符用于修饰在模块内部可见的声明。模块可以是编译在一起的一组 Kotlin 文件或一组 jar 包。具体来说，internal 可以在同一模块中的任何代码中访问，但是无法在模块之外的代码中访问。*

*internal 可以用于修饰类、函数、属性和其他声明，它可以提高代码的封装性和安全性。*

### 3.10.3 使用示例

|  |
| --- |
| class MainActivity : AppCompatActivity() {  val userPreferences by *lazy* **{** DataStoreUtil(this@MainActivity) **}** lateinit var databinding: ActivityMainBinding  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  databinding = DataBindingUtil.setContentView(this, R.layout.*activity\_main*)  setContentView(databinding.*root*)  databinding.save.setOnClickListener **{** val recordText = databinding.et.*text*.toString().*trim*()  if (recordText.*isEmpty*()) {  Toast.makeText(this@MainActivity, "is empty", Toast.*LENGTH\_SHORT*).show()  return@setOnClickListener  }  *lifecycleScope*.launchWhenCreated **{** *//保存数据* userPreferences.setUserName(recordText)  **}  }** databinding.showRecord.setOnClickListener **{** *lifecycleScope*.launchWhenCreated **{** *//获取数据* userPreferences.userNameFlow.collect **{** databinding.record.*text* = **it  }  }  }** } } |

### 3.10.4 DataStore原理

DataStore 是一种 Android Jetpack 组件，用于存储键值对数据，它可以用来代替 SharedPreferences。相比 SharedPreferences，DataStore 的优点在于它更加安全、可靠、支持异步和协程操作等。

DataStore 内部实现了不同的存储方案，比如 Preferences DataStore 和 Proto DataStore，用于存储基本类型和序列化数据。其中 Preferences DataStore 是基于 SharedPreferences 实现的，但它的数据存储在内存中，而不是存储在磁盘中。

在 Preferences DataStore 中，数据被存储在一个 XML 文件中，并且可以指定多个文件来存储不同的数据。当应用程序第一次使用 DataStore 时，DataStore 会在应用程序的私有目录中创建一个存储 XML 文件的目录。每个文件都由一个 DataStore 对象来管理，可以通过 DataStoreBuilder 来创建 DataStore 对象。

在 Proto DataStore 中，数据被存储在二进制格式的文件中，使用 Protocol Buffers 来序列化和反序列化数据。Proto DataStore 通过定义 Protobuf 的 schema 来指定存储的数据类型，然后通过 Protobuf API 来读写数据。

在使用 DataStore 时，需要首先定义数据类型，然后创建相应的 DataStore 对象。然后就可以使用 DataStore 提供的 API 来读写数据，这些 API 支持异步和协程操作。例如：

|  |
| --- |
| data class User(val id: Int, val name: String)  class UserPreferences(context: Context) {  private val dataStore = context.createDataStore("user\_prefs")   val currentUser: Flow<User?>  get() = dataStore.data.map **{** preferences **->** if (preferences.contains(CURRENT\_USER\_ID) && preferences.contains(CURRENT\_USER\_NAME)) {  User(  preferences[CURRENT\_USER\_ID] ?: 0,  preferences[CURRENT\_USER\_NAME] ?: ""  )  } else {  null  }  **}** suspend fun setCurrentUser(user: User) {  dataStore.edit **{** preferences **->** preferences[CURRENT\_USER\_ID] = user.id  preferences[CURRENT\_USER\_NAME] = user.name  **}** }  companion object {  private val CURRENT\_USER\_ID = preferencesKey<Int>("current\_user\_id")  private val CURRENT\_USER\_NAME = preferencesKey<String>("current\_user\_name")  } } |

以上代码实现了一个 UserPreferences 类，它包含一个名为 currentUser 的 Flow 对象，用于获取当前用户信息，并且提供了一个名为 setCurrentUser 的 suspend 函数，用于设置当前用户信息。数据被存储在名为 "user\_prefs" 的 DataStore 中，并且使用 Preferences DataStore 实现。在定义数据类型时，使用了 preferencesKey 函数来定义键值对的键名。在读写数据时，使用了 Flow 和 suspend 函数来支持异步和协程操作。

### 3.10.5 Proto DataStore的使用示例

### 3.10.6 DataStore面试回答

## 3.11 Room的使用及注意事项

# Android架构相关面试题目

## 4.1 什么是MVC？

## 4.2 什么是MVP?MVP如何实现？有什么优缺点？

## 4.3 什么是MVVM?如何实现？什么是单向数据流？有什么好处？什么是双向数据流？如何改为单向数据流？

## 4.4 什么是组件化？

## 4.5什么是插件化？

## 4.6 serializable和parcelable的使用场景？

# 第三方库使用及实现原理

## 5.1 Okhttp3的使用及原理

## 5.2 Retrofit的使用及原理

## 5.3 Glide的使用及原理

## 5.4 Rxjava的使用，原理及注意点

## 5.5 Hlit使用，原理及特殊场景

## 5.6 LeakCannery的实现原理

## 5.7 Handler+Message实现原理

## 5.8 RecyclerView实现原理及特殊场景问题解决

# 性能优化相关题目

## 6.1 启动优化注意点及实现方案

### 6.1.1. Android的启动流程是什么？

## 6.2内存抖动发现及优化手段

## 6.3内存泄露发现及解决办法

## 6.4包体积优化

### 6.4.1图片大小计算方法

### 6.4.2 如何检查无效资源删除

### 6.4.3如何有效找到未使用的文件

### 6.4.4 如何有效解决包体积问题

## 6.5 ANR原因，排查手段及修复方式，线上诊断？

### 6.5.1 什么是ANR?

ANR（Application no response）是 Android 系统中的一种机制，用于检测应用程序无响应的情况。当应用程序无法在规定的时间内响应用户操作或者主线程在执行耗时操作而无法及时响应其他事件时，系统会认为该应用程序已经“死锁”，会弹出一个 ANR 对话框提示用户关闭该应用程序或等待程序响应。

ANR 可能会出现在一些耗时的操作上，例如磁盘I/O、网络请求或大量计算操作等。为了避免 ANR 的出现，开发者需要在应用程序中注意避免在主线程中执行耗时操作，以及合理地使用异步任务或线程池等机制来进行耗时操作。

### 6.5.2 如何监控ANR?(线上和线下)

### 6.5.3 如何解决ANR?

### 6.5.4 ANR面试回答

## 6.6电量优化

## 6.7 如何加固？

# 数据结构和算法

## 7.1 如何判定一个程序的时间复杂度？

## 7.2 常用的数据结构如HashMap采用的是什么数据结构？在高并发场景有什么问题？如何解决？CurrentHashMap。

## 7.3 SpareArray数据结构是如何实现的？有什么优缺点？

# jetpackCompose面试相关题目

## 8.1 什么是响应式编程？

## 8.2 什么是重组？

# flutter面试题目

# JNI面试相关题目

## 10.1 C和C++常用的知识点，如extern的使用场景？

## 10.2 如何实现交叉编译？

# 音视频开发相关题目

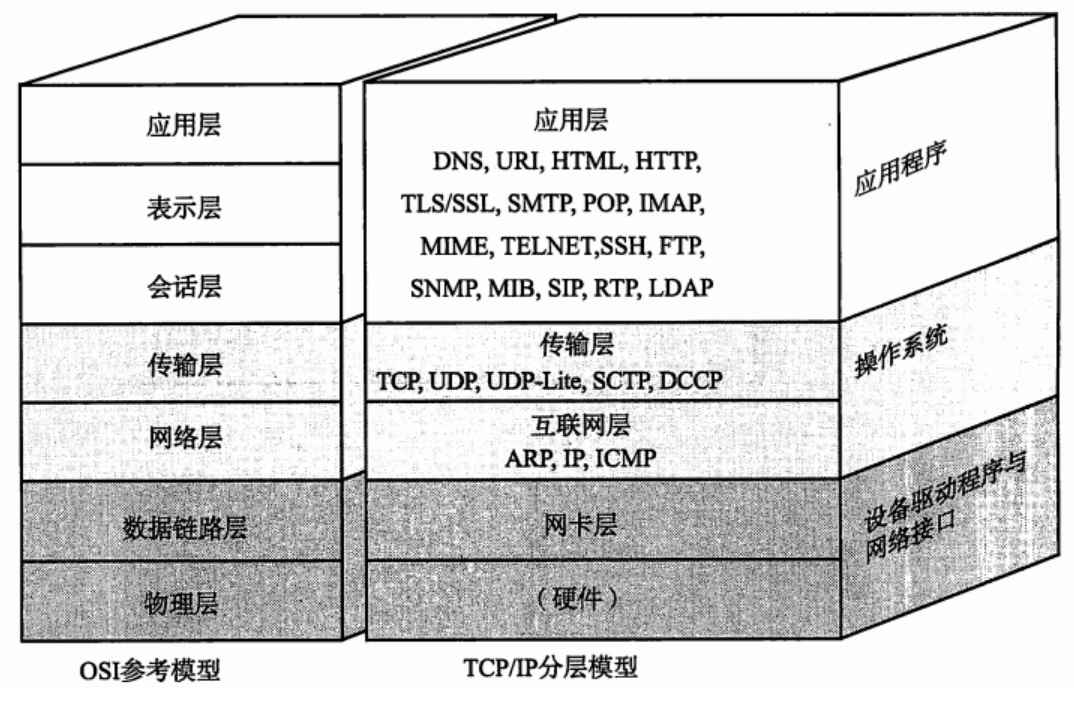
## 11.1 Ffmpeg的使用及基础算法

## 11.2 OpenCV的使用及注意事项

# 12. 计算机网络

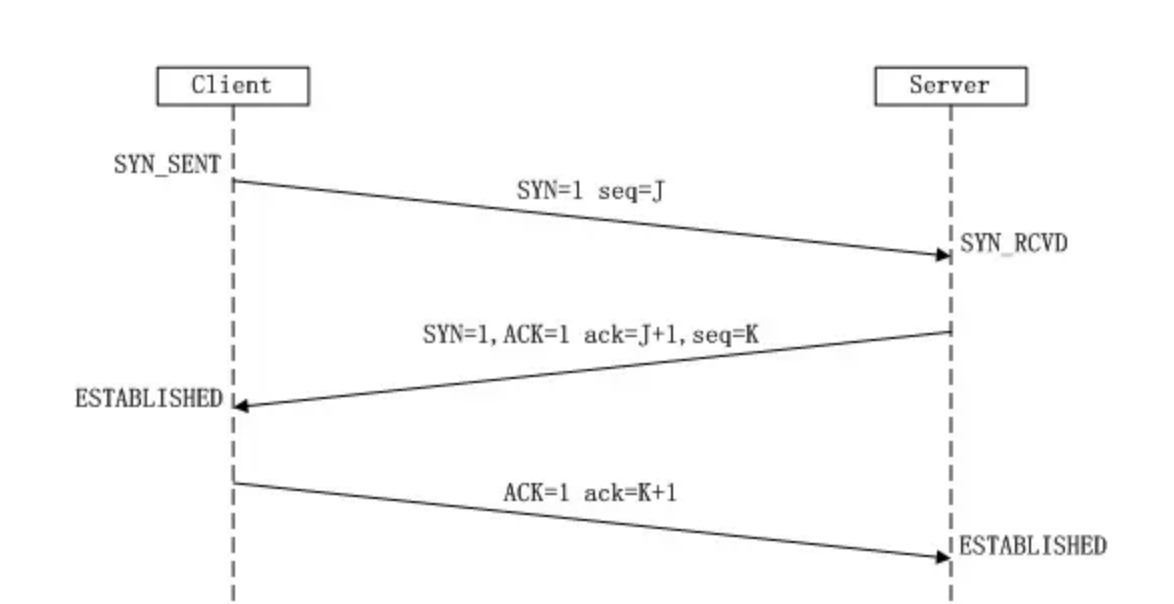
## 12.1 什么是TCP/IP，TCP是如何保证通信的可靠性的？

TCP/IP 指传输控制协议/网际协议(Transmission Control Protocol / Internet Protocol)。实际上是指OSI模型中的传输层和网络层的统称。IP协议是TCP/IP的基础，它负责在网络中传输数据包。TCP协议在IP协议的基础上添加了一些额外的功能，如数据包重传、流量控制和拥塞控制，以确保数据能够以可靠的方式传输。这两个协议结合在一起，使得互联网上的各种计算机和网络设备能够相互通信。

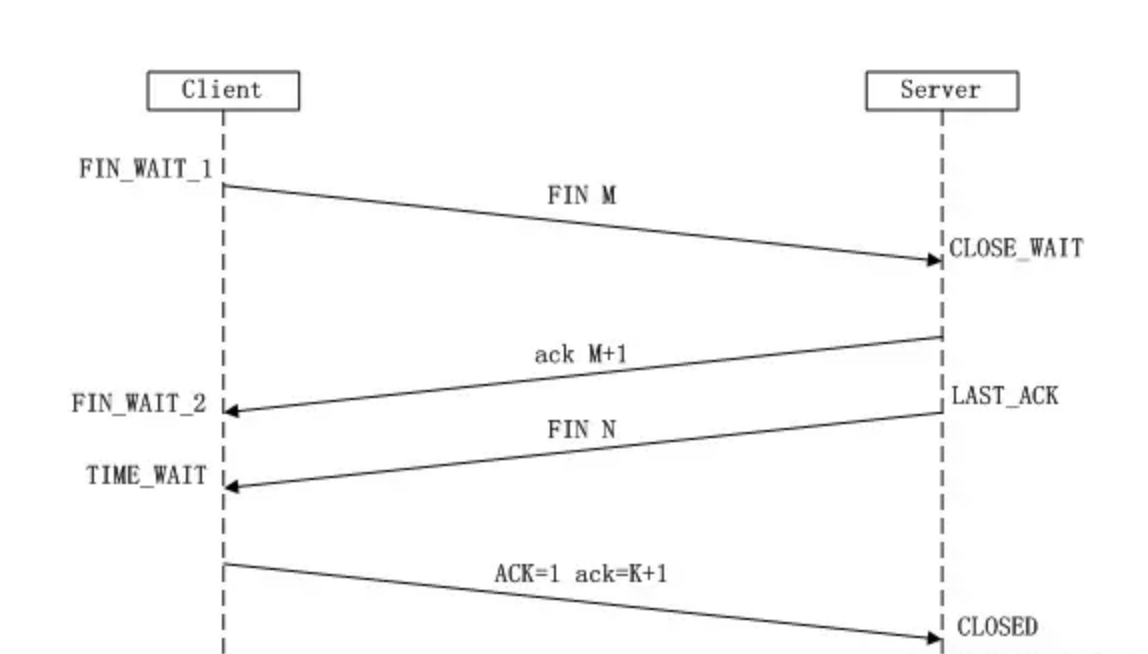


**TCP协议如何保证传输是可靠的？**是通过三次握手，四次分手来保证传输的可靠性。

1. 客户端向服务器发送一个加密协议版本号、加密算法列表、随机数等信息；
2. 服务器回应一个确认消息，包含加密协议版本号、选择的加密算法、服务器证书、随机数等信息；ACK/SYN信号；
3. 客户端再回应一个确认消息，包含加密协议版本号、随机数等信息。



通过以上步骤后，客户端和服务器端就完成了链接。服务器和客户端就会创建一个SOCKET链接。然后开始传输数据，传输完成后，客户端**（客户端根据接收到的数据可以判断是否数据接收完成，而服务器端是不知道客户端是否接收完成数据，所以不能是服务器发起断开链接）**发起断开链接，也就开始了**四次分手过程**。



1. 客户端发起FIN信号给服务器端，服务器收到后给客户端发送ACK信号。
2. 服务器收到客户端发起的断开请求后，也会给服务器发送FIN信号，客户端收到服务器的FIN信号后，客户端给服务器发送ACK信号。

以上过程是为了服务器和客户端释放连接时开辟的资源。

## 12.2 什么是https？及证书校验全过程？

HTTPS（Hypertext Transfer Protocol Secure）是一种安全的网络通信协议，它建立在HTTP协议之上，使用SSL/TLS加密技术对传输的数据进行加密和身份认证。HTTPS协议的目的是为了保护数据的完整性、保密性和身份认证，以防止中间人攻击和窃听等安全问题。

在HTTPS协议中，客户端和服务器之间的通信会通过SSL/TLS协议进行加密，SSL/TLS协议使用数字证书对服务器的身份进行验证，确保通信的安全性和完整性。HTTPS协议广泛应用于在线交易、金融、电子邮件等需要保密和安全的网络通信场景。

### 12.2.1 证书校验全过程

1. 客户端向服务器发送连接请求，包含加密协议版本号、加密算法列表、随机数等信息。

2. 服务器回应一个确认消息，包含加密协议版本号、选择的加密算法、服务器证书、随机数等信息。

3. 客户端从服务器发来的证书中提取公钥，并根据证书颁发机构的公钥，验证证书是否合法。

4. 如果证书合法，客户端生成一个随机数，使用服务器证书中提取的公钥进行加密，并发送给服务器。

5. 服务器使用自己的私钥解密随机数，并使用该随机数生成加密密钥。

6. 客户端和服务器使用该加密密钥进行对称加密和解密，保证通信的安全性。

# 13. 业务场景

## 13.1 输入支付密码逻辑设计

## 13.2 登录状态管理

## 13.3 二维码登录逻辑设计