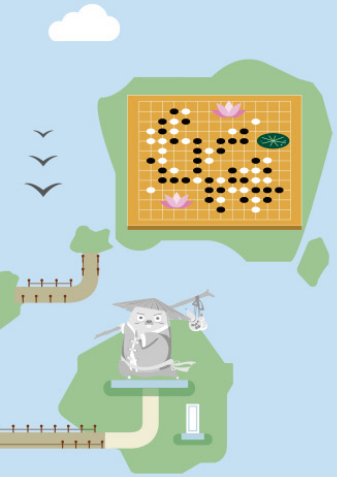
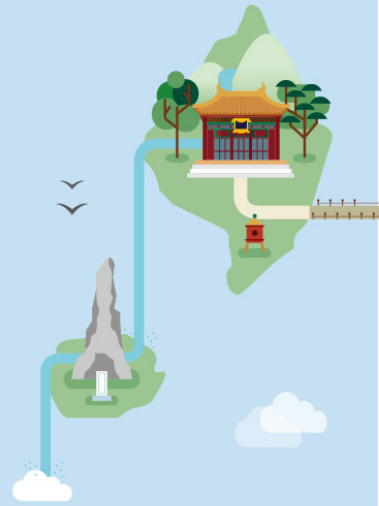


Go 语言在讯联扫码
支付系统中的成功实践

2017-04-02

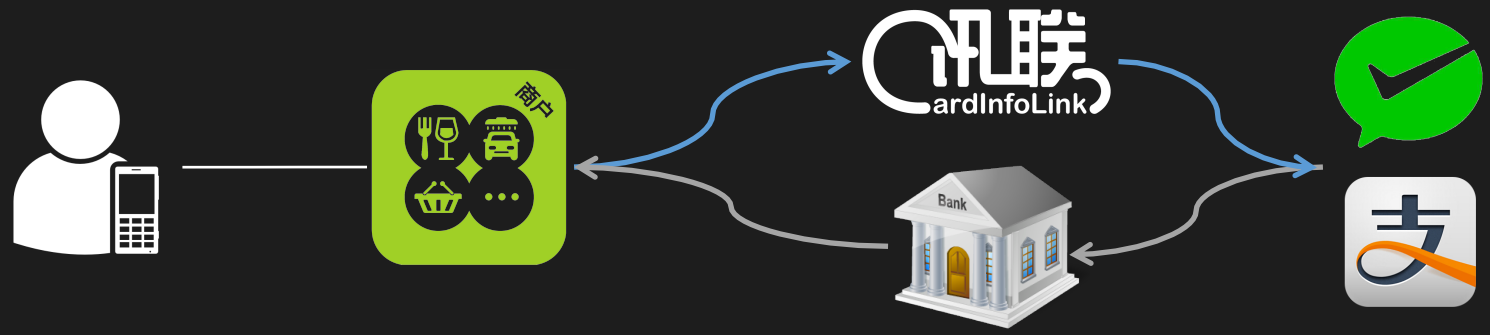
Jacky



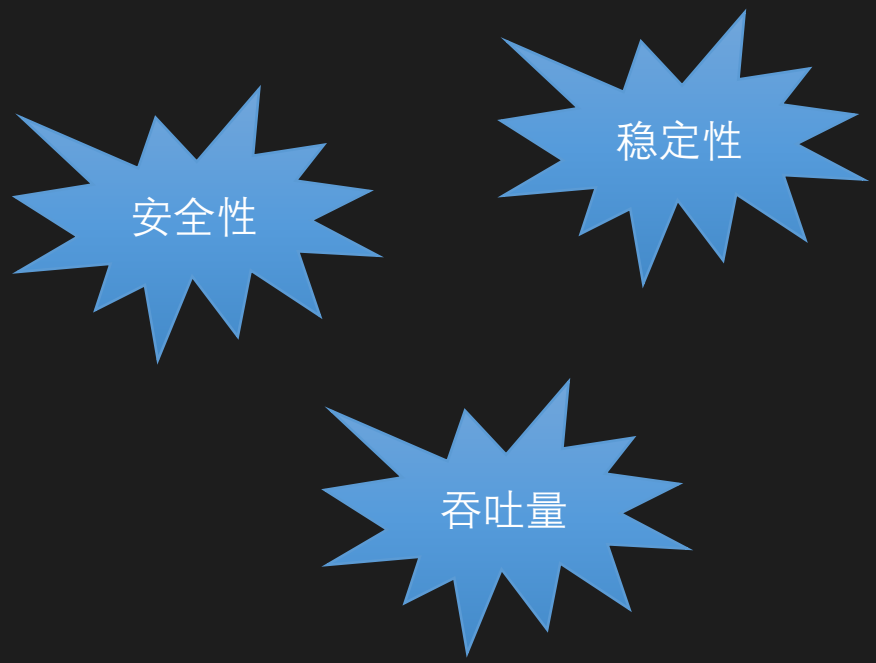
主要内容

- 金融支付系统的一些特点
- 我们的扫码支付系统技术选型
- 系统迭代过程中的架构演进
- 与Go相关的一些坑

业务流程



- 信息流
  - 实时交易服务：API Gateway
  - 商户对账服务：批处理
  - 商户服务：web portal



技术选型

- 业务需求
- 技术需求
- 团队需求

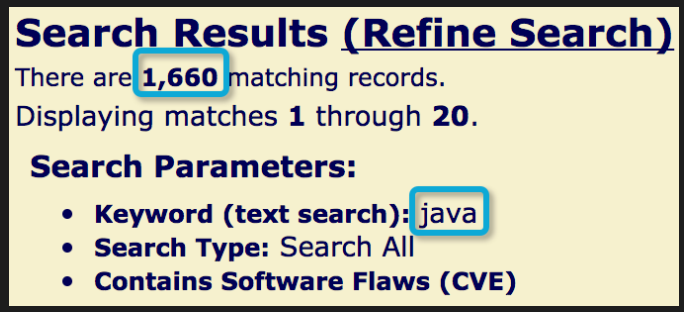
技术选型

- 团队背景
  - C，Java，Golang
- Golang技术特点
  - 快速上手，学习曲线平滑，开发效率高
  - 天生支持并发编程
  - 简洁的错误处理：defer、panic、recover

技术选型

- 安全性

https://web.nvd.nist.gov/view/vuln/search



技术选型

- 稳定性
  - 高可用架构
    - 应用无状态，支持横向扩展

技术选型

- 吞吐量
  - 并发处理能力
    - http接口
    - RSA加解密


```
~/workspace_go/src/github.com/jackyvictory java -version
java version "1.7.0_75"
Java(TM) SE Runtime Environment (build 1.7.0_75-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.75-b04, mixed mode)
~/workspace_go/src/github.com/jackyvictory go version
go version go1.7.5 darwin/amd64
~/workspace_go/src/github.com/jackyvictory
```

```
~/workspace_go/src/github.com/jackyvictory/http ab -n 10000 -c 10 http://localhost:5678/
Concurrency Level:      10
Time taken for tests:   0.815 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      1270000 bytes
HTML transferred:       100000 bytes
Requests per second:    12275.32 [#/sec] (mean)
Time per request:       0.815 [ms] (mean)
Time per request:       0.081 [ms] (mean, across all concurrent requests)
Transfer rate:          1522.43 [Kbytes/sec] received
```

```go
func main() {
  runtime.GOMAXPROCS(runtime.NumCPU())
  http.Handle("/", httpRouter())
  http.ListenAndServe(":5678", nil)
}

func httpRouter() (mux *http.ServeMux) {
  mux = http.NewServeMux()
  mux.HandleFunc("/", myHandler)
  return
}

func myHandler(w http.ResponseWriter, r *http.Request) {
  retStr := "hello http"
  retBytes := []byte(retStr)
  w.Write(retBytes)
}
```

```
~/workspace_go/src/github.com/jackyvictory/http ab -n 10000 -c 10 http://localhost:8765/
Concurrency Level:      10
Time taken for tests:   0.891 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      1050000 bytes
HTML transferred:       100000 bytes
Requests per second:    11229.20 [#/sec] (mean)
Time per request:       0.891 [ms] (mean)
Time per request:       0.089 [ms] (mean, across all concu
Transfer rate:          1151.43 [Kbytes/sec] received
```

```java
public static void main(String[] args) {
    try {
        HttpServer hs = HttpServer.create(new InetSocketAddress(8765), 0);
        hs.createContext("/", new MyHandler());
        int threadMinCount = 100;
        int threadMaxCount = 200;
        int checkPeriod = 2;
        ThreadPoolExecutor threadPool = new ThreadPoolExecutor(
                        threadMinCount, threadMaxCount, checkPeriod, TimeUnit.SECONDS,
                        new ArrayBlockingQueue(100),
                        new ThreadPoolExecutor.CallerRunsPolicy());
        hs.setExecutor(threadPool);
        hs.start();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Goph
ww

```
~/workspace_go/src/github.com/jackyvictory/rsa ▶ /usr/bin/time -lp ./rsa
real         2.78
user         8.56
sys          0.11
  20070400  maximum resident set size
         0  average shared memory size
         0  average unshared data size
         0  average unshared stack size
      4940  page reclaims
         1  page faults
         0  swaps
         0  block input operations
         0  block output operations
         0  messages sent
         0  messages received
         0  signals received
      5922  voluntary context switches
     10119  involuntary context switches
```

```
~/workspace_go/src/github.com/jackyvictory/rsa ▶ /usr/bin/time -lp java Rsa
real         7.74
user        29.29
sys          0.83
 179183616  maximum resident set size
         0  average shared memory size
         0  average unshared data size
         0  average unshared stack size
     49611  page reclaims
         0  page faults
         0  swaps
         0  block input operations
         7  block output operations
        13  messages sent
        12  messages received
       190  signals received
         7  voluntary context switches
     37059  involuntary context switches
```

```go
func main() {
  runtime.GOMAXPROCS(runtime.NumCPU())
  var count = 1000
  var wg sync.WaitGroup
  for i := 0; i < count; i++ {
    wg.Add(1)
    go func() {
      defer wg.Done()
      cipherBytes, err := rsa.EncryptPKCS1v15(rand.Reader, pub, []byte(clearMsg))
      if err != nil {
        panic(err)
      }
      _, err = rsa.DecryptPKCS1v15(rand.Reader, priv, cipherBytes)
      if err != nil {
        panic(err)
      }
    }()
  }
  wg.Wait()
}
```

```java
public static byte[] RSAEncrypt(PublicKey pub, byte[] clearText) {
  try {
    Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);
    cipher.init(Cipher.ENCRYPT_MODE, pub);
    return cipher.doFinal(clearText);
  } catch(Exception e) {
    e.printStackTrace();
  }
  return null;
}

public static byte[] RSADecrypt(PrivateKey priv, byte[] cipherText) {
  try {
    Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);
    cipher.init(Cipher.DECRYPT_MODE, priv);
    return cipher.doFinal(cipherText);
  } catch(Exception e) {
    e.printStackTrace();
  }
  return null;
}
```

技术选型

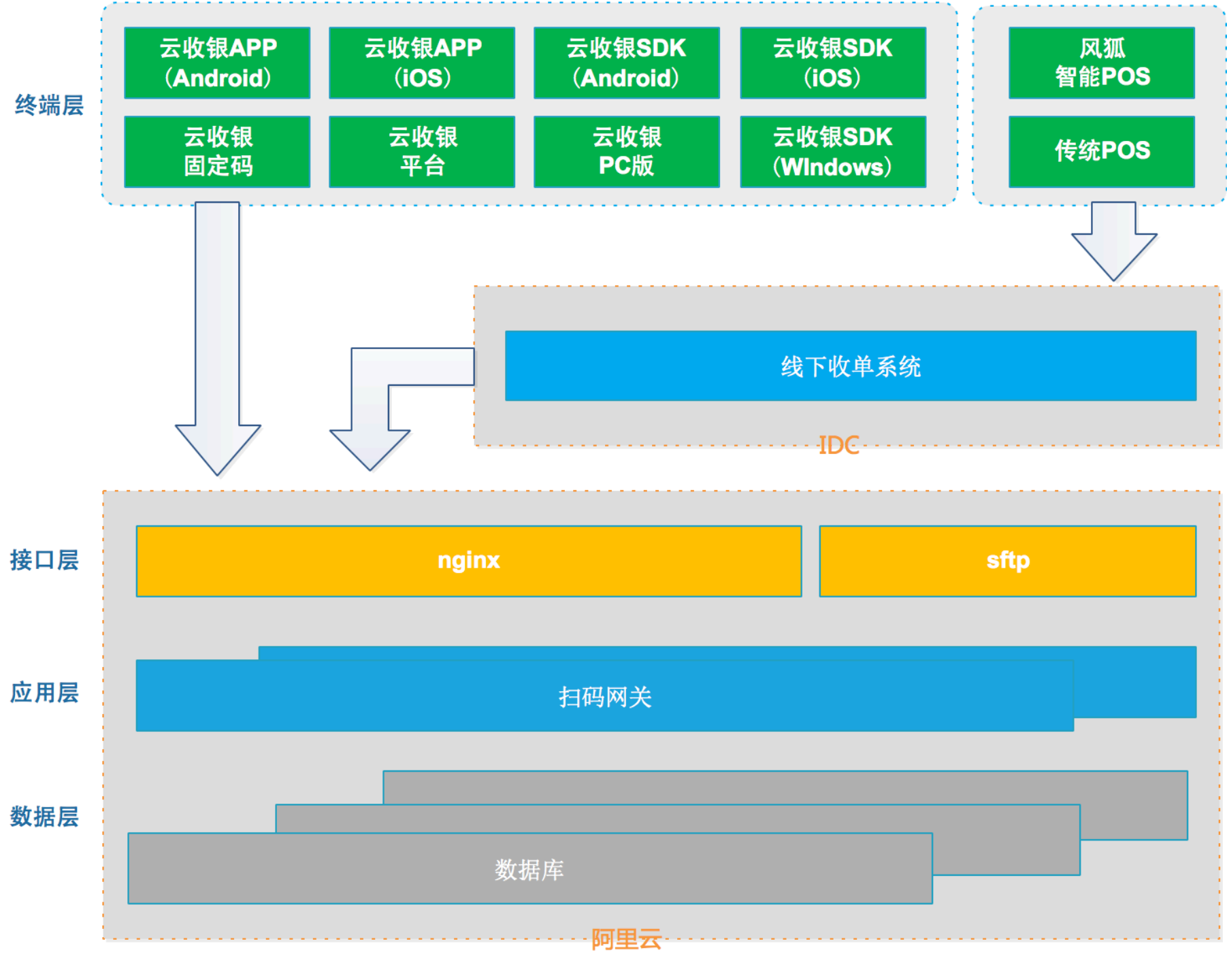
- 总结

  作为需要快速原型、快速迭代的项目，

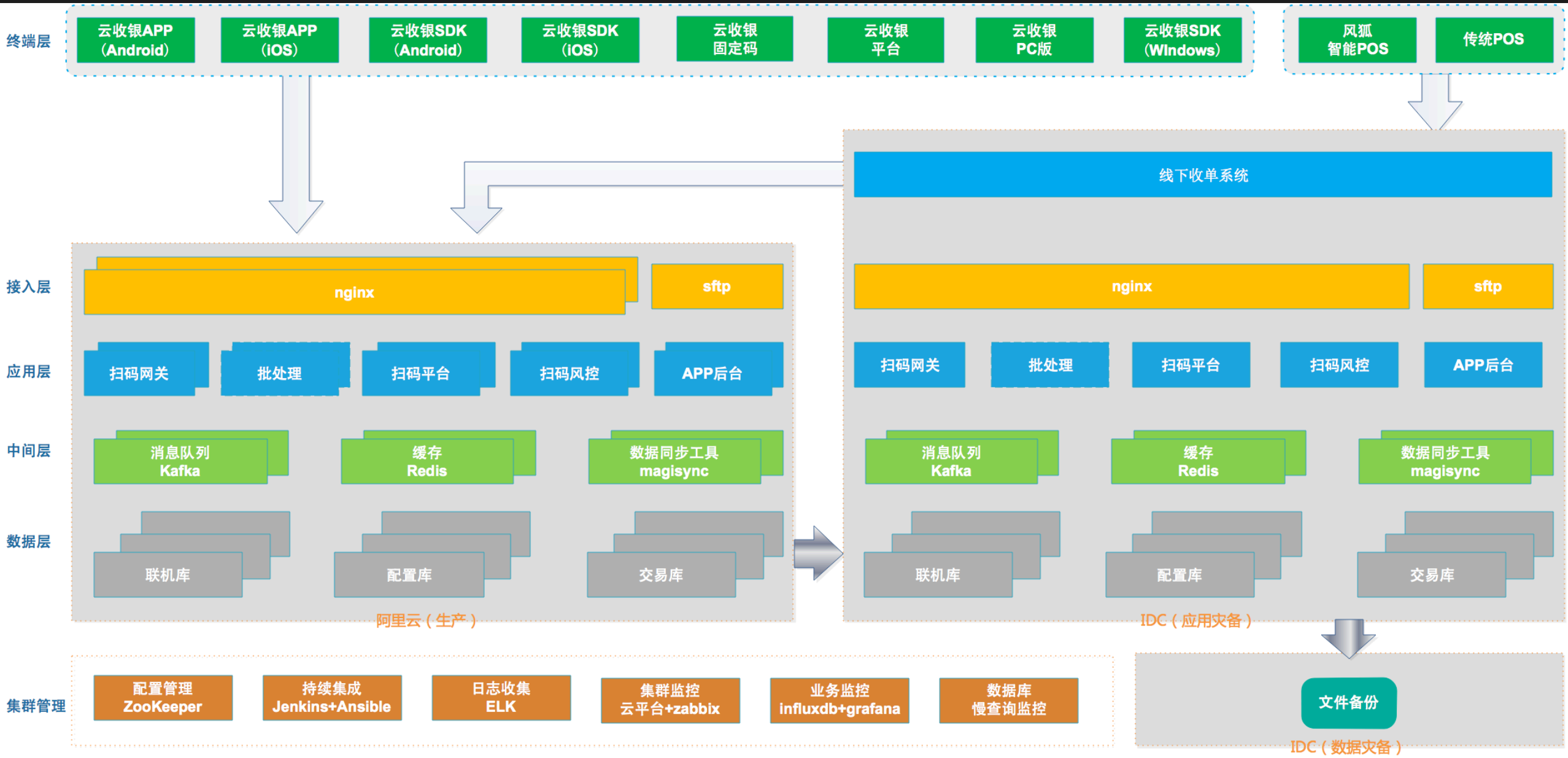
  在满足当前和未来可预期的高可用、吞吐量等业务需求的前提下，

  Golang的高效开发效率、简单的部署和运维，是我们拥抱Golang的主要原因。

初始版本


终端层
云收银APP（Android）
云收银APP（iOS）
云收银SDK（Android）
云收银SDK（iOS）
云收银固定码
云收银平台
云收银PC版
云收银SDK（Windows）
风狐智能POS
传统POS
线下收单系统
IDC
接口层
nginx
sftp
应用层
扫码网关
数据层
数据库
阿里云

架构调整


终端层
云收银APP（Android）
云收银APP（iOS）
云收银SDK（Android）
云收银SDK（iOS）
云收银固定码
云收银平台
云收银PC版
云收银SDK（Windows）
风狐智能POS
传统POS
线下收单系统
接入层
nginx
sftp
应用层
扫码网关
批处理
扫码平台
扫码风控
APP后台
中间层
消息队列 Kafka
缓存 Redis
数据同步工具 magisync
数据层
联机库
配置库
交易库
阿里云（生产）
IDC（应用灾备）
集群管理
配置管理 ZooKeeper
持续集成 Jenkins+Ansible
日志收集 ELK
集群监控 云平台+zabbix
业务监控 influxdb+grafana
数据库 慢查询监控
文件备份
IDC（数据灾备）

一些坑

- 变量作用域
- chan操作

```go
func ListenScanPay(addr string) {
  ln, err := net.Listen("tcp", addr)
  if err != nil {
    log.Errorf("fail to listen %s port: %s", addr, err)
    return
  }
  log.Infof("ScanPay TCP is listening, addr=%s", addr)

  go func() {
    for {
      conn, err := ln.Accept()
      if err != nil {
        log.Errorf("listener fail to accept: %s ", err)
        return
      }

      go catch.TcpRecoverWrap(handleConnection)(conn)
    }
  }()
}
```

```go
func TcpRecoverWrap(handle func(ln net.Conn)) func(ln net.Conn) {
  return func(ln net.Conn) {
    defer func() {
      if r := recover(); r != nil {
        log.Errorf("TCP REQ PANIC: %s", toErr(r))
        ln.Write([]byte("server error, please retry."))
      }
      ln.Close()
    }()
    handle(ln)
  }
}
```

```go
func handleConnection(conn net.Conn) {
  var writeMsgQueue = make(chan string, queueSize)
  defer close(writeMsgQueue)

  go func() {
    for {
      select {
      case writeMsg, ok := <-writeMsgQueue:
        if !ok {
          log.Debug("writeMsgArray closed")
          return
        }

        err := write(conn, writeMsg)
        if err != nil {
          log.Errorf("write failed: %v", err)
          continue
        }
      }
    }
  }()
  var reqBytes []byte
  var err error
  for {
    reqBytes, err = read(conn)
    if err != nil {
      if err == errPing {
        continue
      }
      log.Error(err)
      return
    }

    go func() {
      msg := ScanPayHandle(reqBytes)
      writeMsgQueue <- msg
      return
    }()
  }
}
```

```go
func handleConnection(conn net.Conn) {
  var writeMsgQueue = make(chan string, queueSize)
  var tcpConnClosed = make(chan int, 1)
  defer func() {
    tcpConnClosed <- 1
  }()

  go func() {
    for {
      select {
      case <-tcpConnClosed:
        log.Debug("receive tcp close signal")
        return

      case writeMsg, ok := <-writeMsgQueue:
        if !ok {
          log.Debug("writeMsgArray closed")
          return
        }
        err := write(conn, writeMsg)
        if err != nil {
          log.Errorf("write failed: %v", err)
          continue
        }
      }
    }
  }()

  for {
    reqBytes, err := read(conn)
    if err != nil {
      if err == errPing {
        continue
      }
      log.Error(err)
      return
    }

    go func() {
      msg := ScanPayHandle(reqBytes)
      writeMsgQueue <- msg
      return
    }()
  }
}
```

Thanks & QA