

[TECH01]

TVP Tencent Cloud
Valuable Professional

intel

/serverless/DAYS

Techo x TVP 开发者峰会

/serverless/DAYS China
2021

无服务器, 大有未来
Serverless, Empower More

Serverless 落地应用的趋势

王晓波

</>

β

/...

#

☰

∞

{ }



王晓波

同程旅行



Serverless

我们为什么要用Serverless? 只是为了架构设计得好看一点吗?

用Serverless解决什么问题? 只是为了弹性付费吗?

利用Serverless怎么助力业务? 只是为了更快的需求变现吗?

架构设计的过程：基本是一个填坑与挖坑的过程

填了哪些坑？又挖了哪些坑？

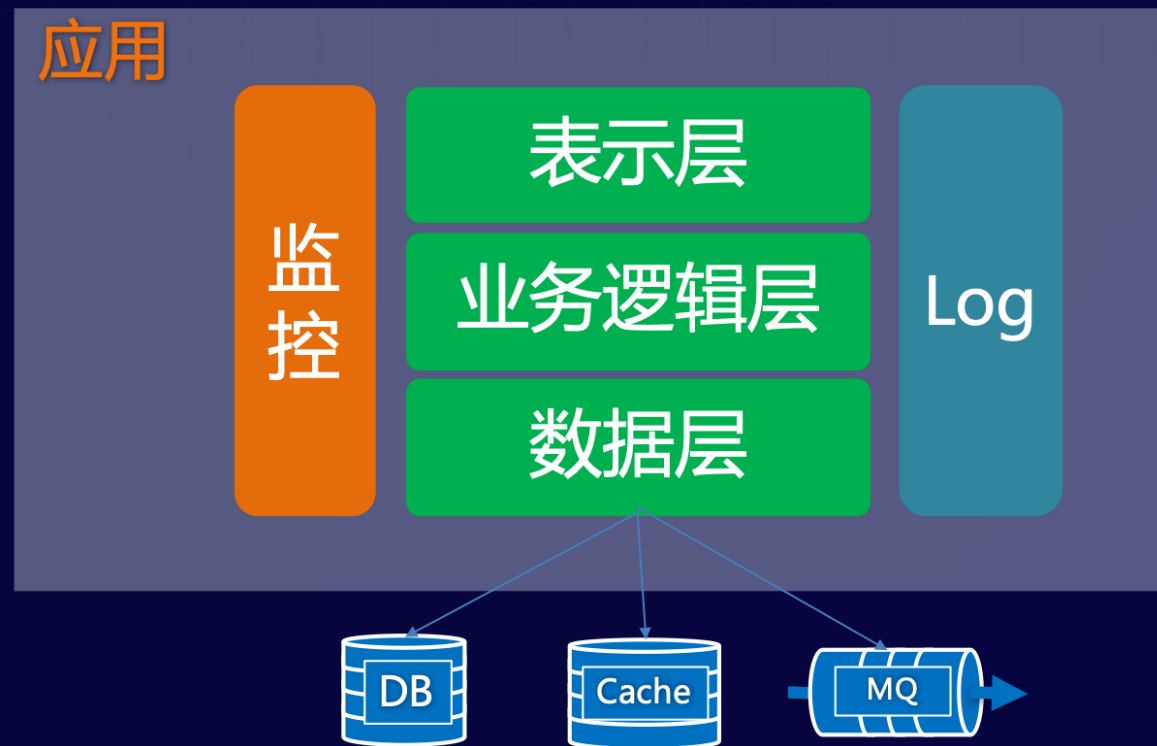


让我们看看最早的系统架构

[TECH00]

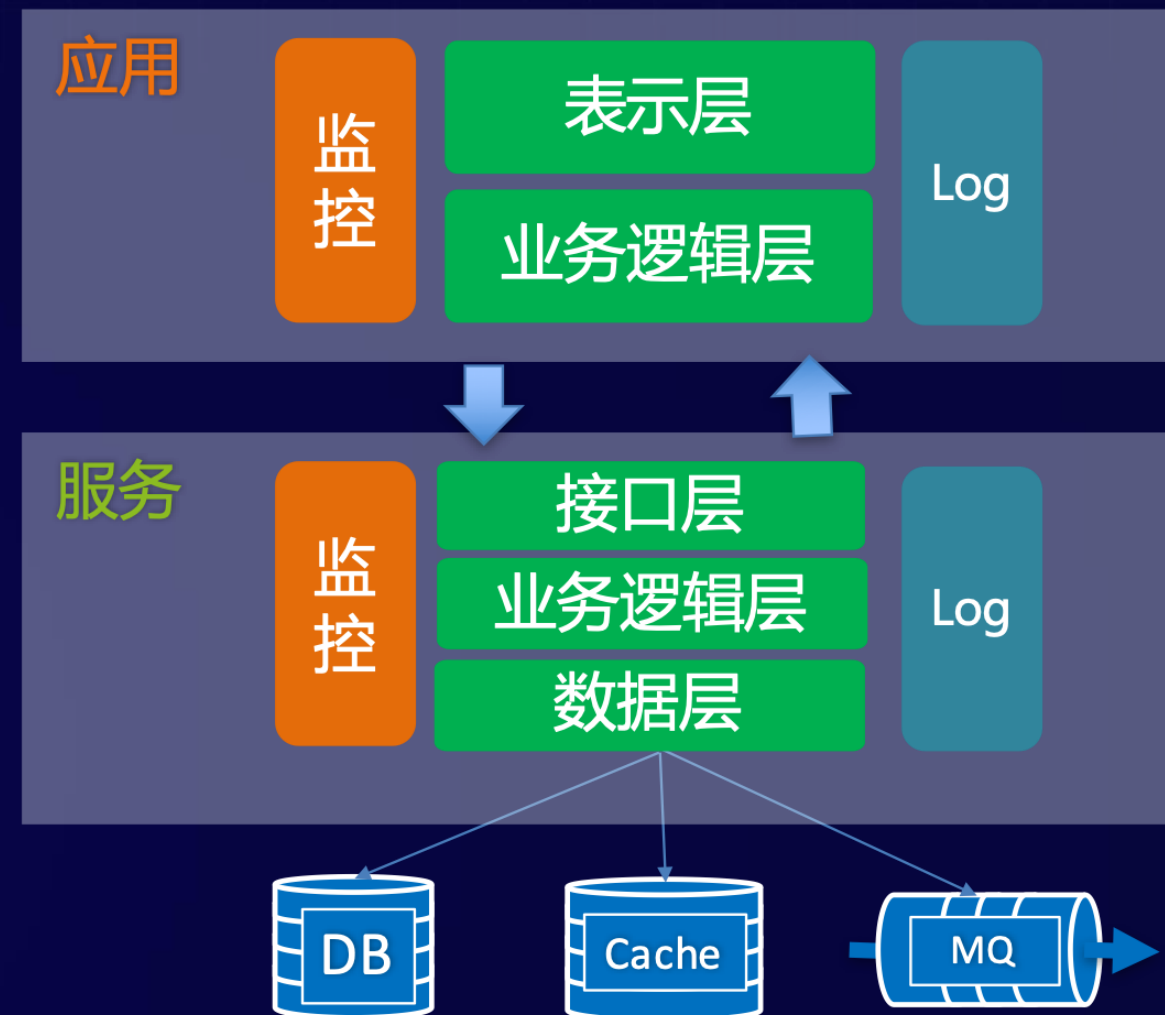
TVP Tencent Cloud
Valuable Professional

/serverless/DAYS



优点:

- 开发方便 [一两个人也能做]
- 部署简单 [开发自己运维]
- 开发快 [无基础也能做]
- 管理简单 [一人一个系统从头到脚]



特点:

- 使用多年大家都理解
- 也有粗粒度的服务展现
- 拥有一定的系统伸缩性能力
- 重了框架轻了架构



一个字“乱”，两个字“太乱”

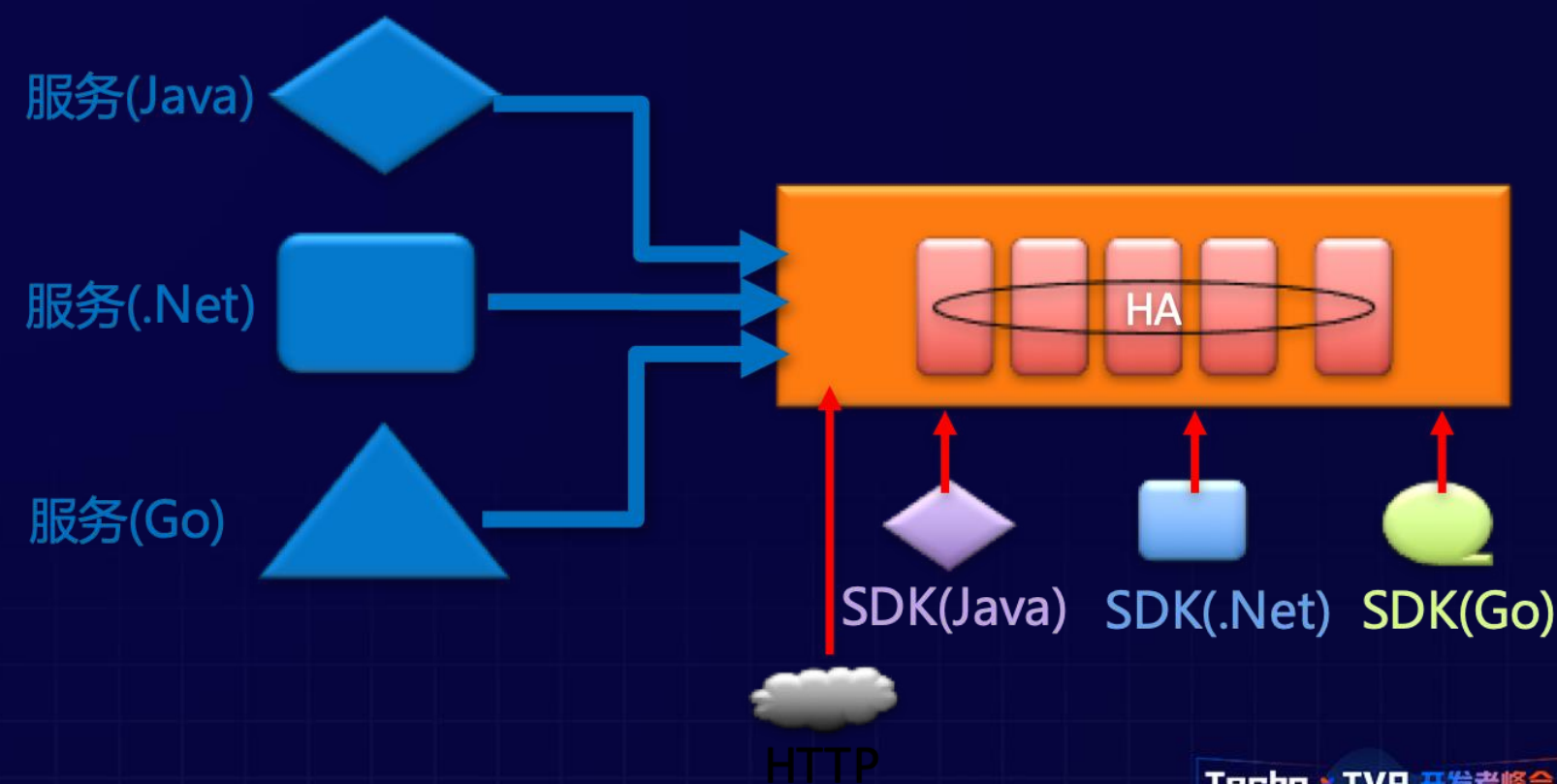
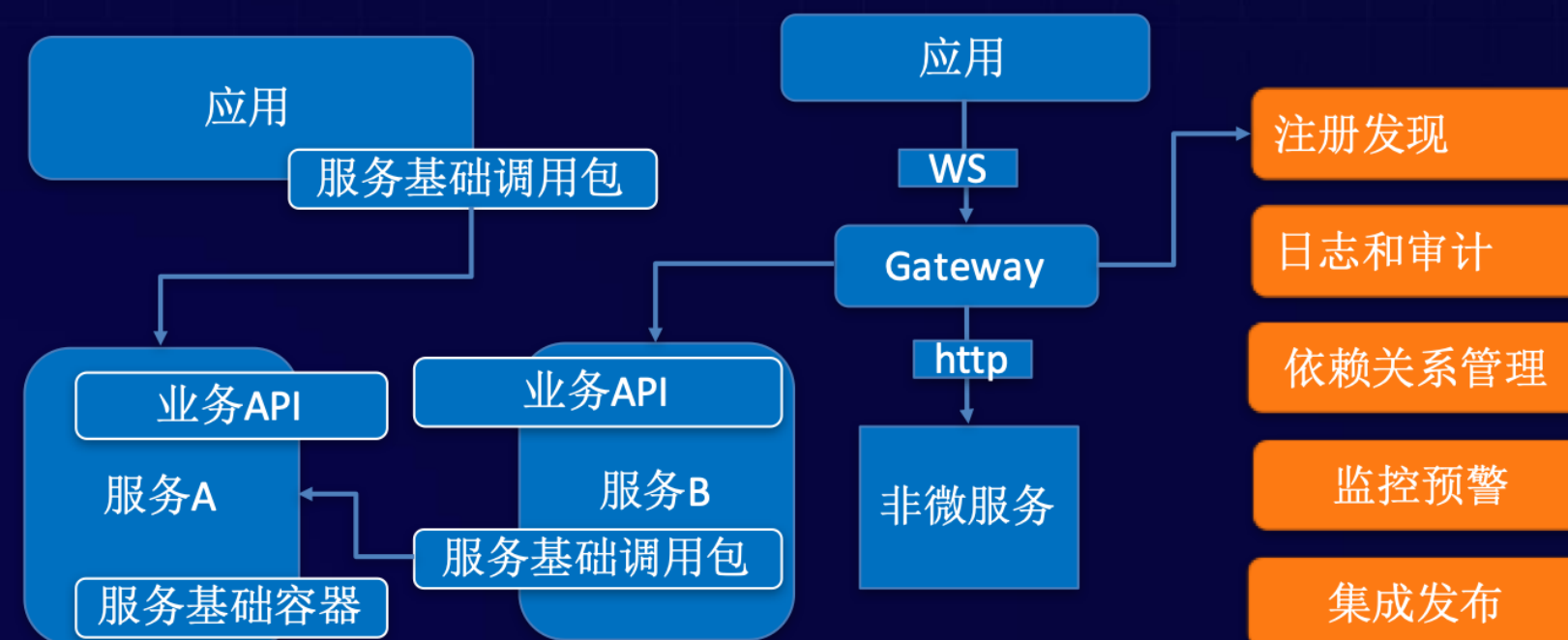
问题：

- 应用增加到几千多个，想一想玩不动了
- 流量是海量，每天数亿级请求量
- 原来很简单的一句SQL，现在没法用了
- 原来的小方法，现在是万行的代码
- 原来实例化调用三层框架，现在是麻花
- 服务器加到不知道放到哪里好
- 各种背不完的锅



那为什么现在还有大量的单体应用在新增呢？

- 大量的业务逻辑代码开始解耦
- 原来的大象变成了一个个可原子化的服务
- 每个服务都可以单独部署和运维
- 调用关系也清晰明了
- 可以从单一语言向java、nodeJS、GO多语言开发了

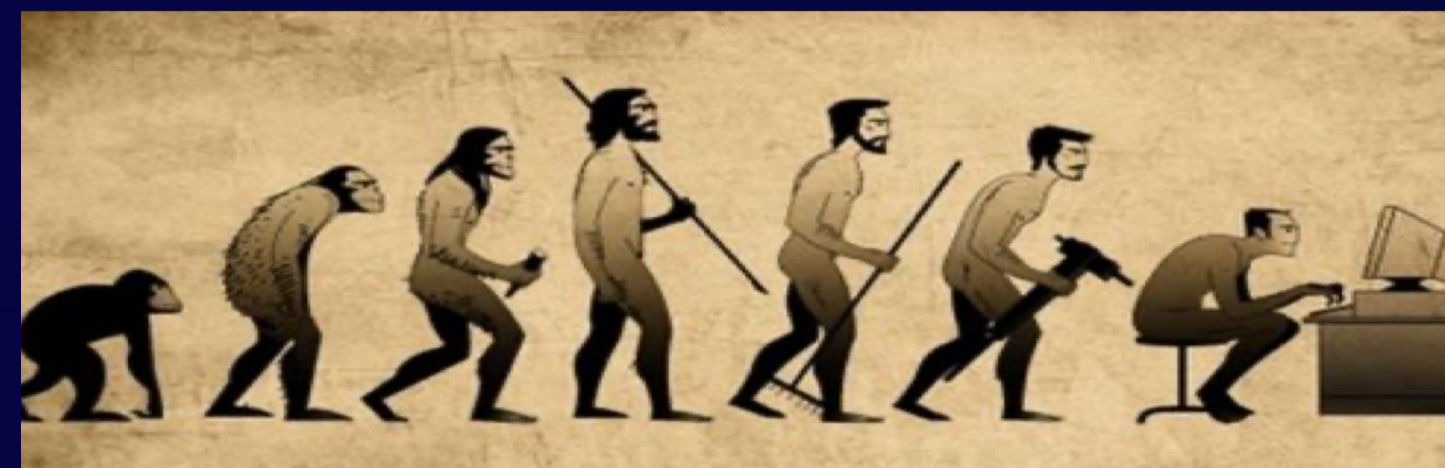
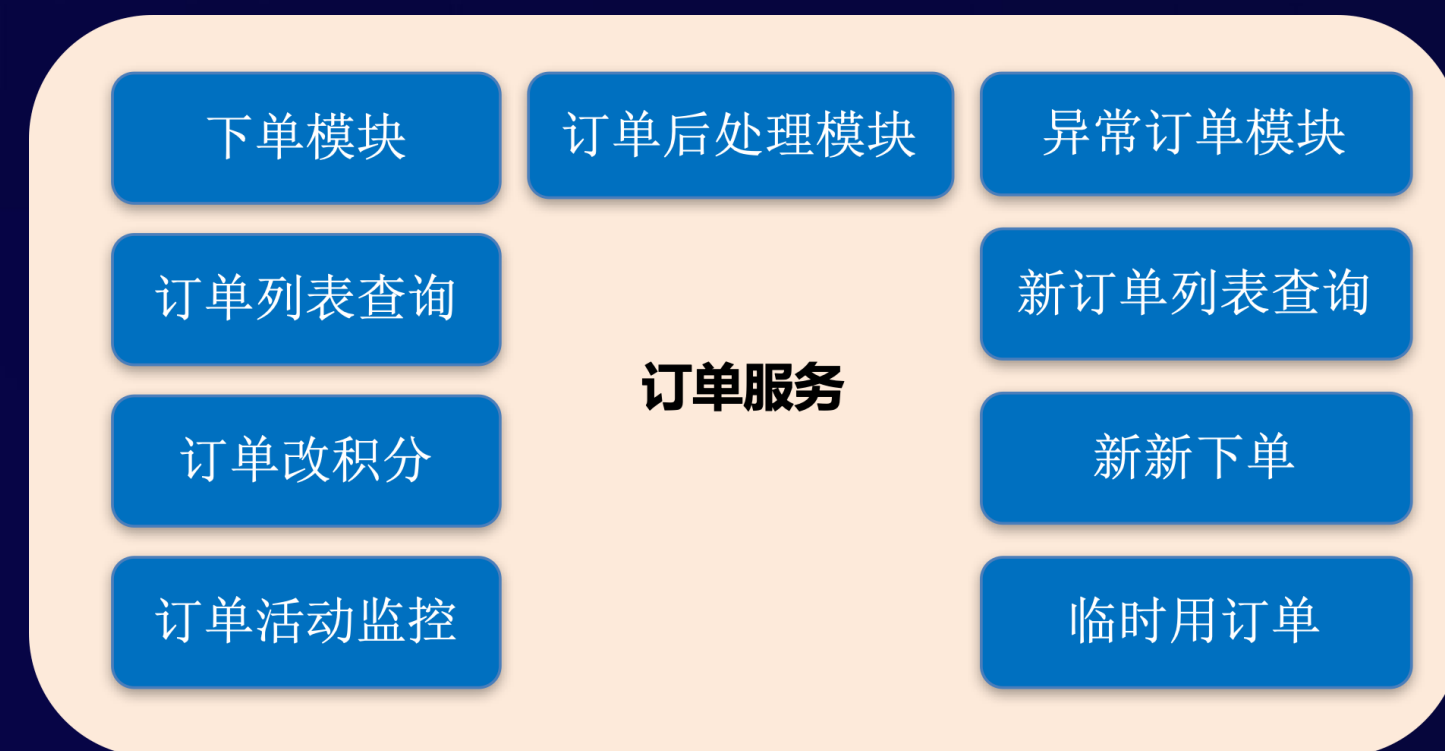




大量服务以业务原子为单位划分，系统变化快，
服务变成大胖子

问题：

- 非核心服务与核心服务来回变化
- 原服务的代码不敢改〔程序员的特性，产品的锅〕
- 服务的数量成几何倍增涨
- 伪微服务的出现〔微服务也变肥了，当你发现总是在它出现后〕
- 新应用比较多〔新的小单体应用，要速度长大〕
- 什么样的服务算微服务



- 微服务整体推进要时间，不可能一次改完，中间过程需要过渡
- 还有很多为快速打样做的单体应用，或一些简单应用
- 一个微服务经过长时间迭代更新后也需要新重构

问题:

- 整个体系中并不是只有微服务
- 管理，调用都会因为这个不得不变得复杂起来了
- 写一个微服务的成本也并不小
- 开发过程的调试比较麻烦
- 服务的数量越来越多，容器也是资源
- 伪DEVOPS
- 微服务基础设施自己是否微服务化了





总结起来这一切就是自己挖个坑和自己再填上坑:

1. 用一两个小时写个服务并上线
2. 不用关心服务器在哪部多少
3. 只需要配管人员配置, 接下来就完全是代码的事
4. 开发人员可以像操作IDE那样完成他不熟悉的技能
5. 代码没跑最好不要计算资源成本
6. 不需要人人都是全栈工程师, 他就想好好写代码
7. 运维就是智能的, 让系统与算法去解决问题





再看看天天在用的编程框架方便吗？ 服务化了吗？

[TECH

TVP

Tencent Cloud
Valuable Professional

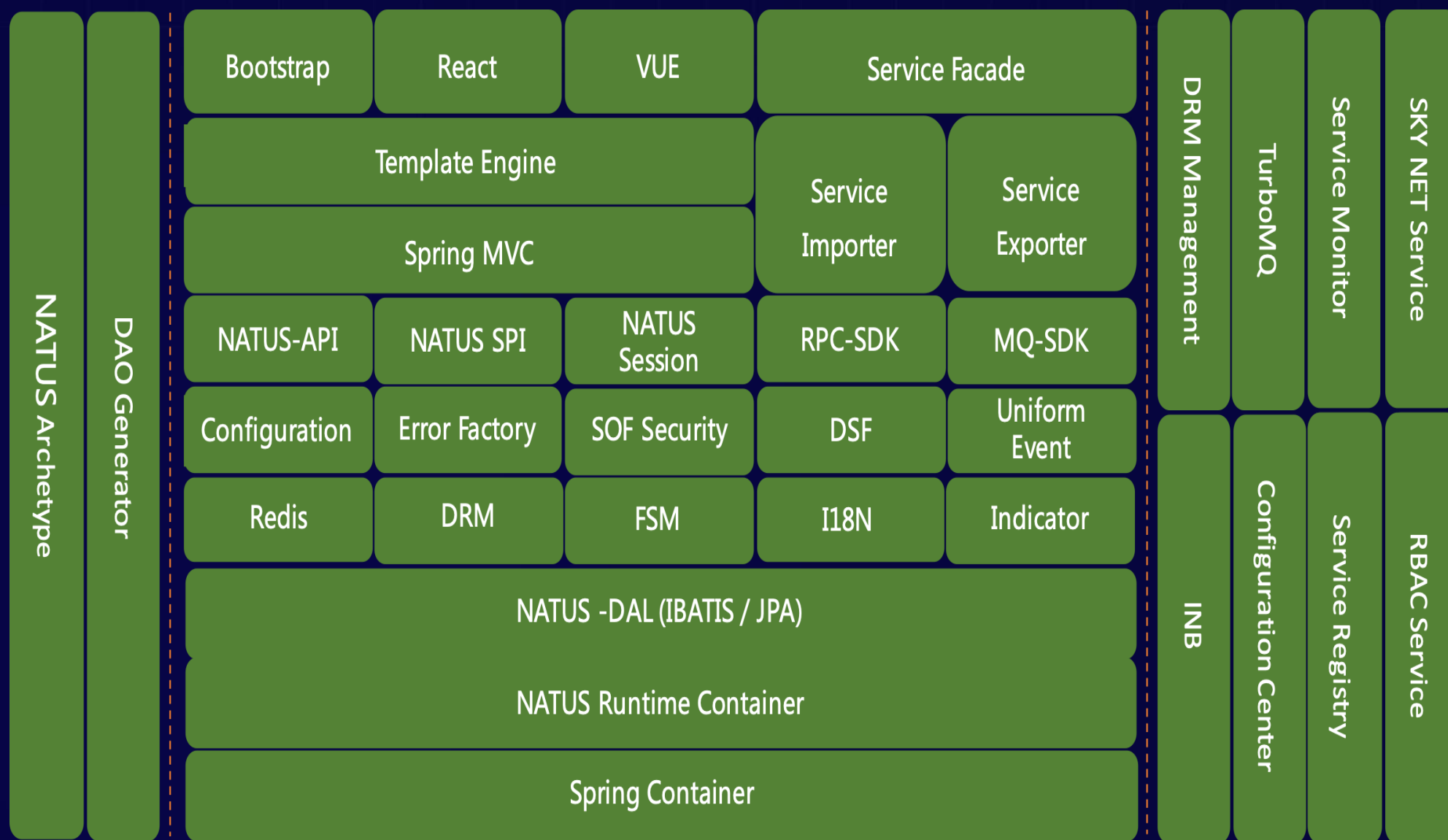
/serverless/DAYS

- 各种组件
- 各种规则
- 各种配置
- 各种工程结构

```
<table sqlname="flight_order">
  <sql id="order.columns">
    id, trade_order_serial_no, order_serial_no, order_state, order_source, contact_name, contact_phone,
    contact_email, airline_type, gmt_take_off,
    gmt_arrive, gmt_return_take_off, gmt_return_arrive, departure_city_code, arrival_city_code, gmt_issue, booker,
    gmt_book, total_money, reimbursement_airway,
    reimbursement_type, member_id, member_name, merchant_id, is_delete, trace_id, operator, CreateTime, UpdateTime,
    env, auto_issue_retry_count, gmt_first_issue
  </sql>

  <operation name="updateOrderState" paramtype="parameterObject">
    <sql>
      UPDATE flight_order
      SET
        order_state = #orderState#,
        UpdateTime = CURRENT_TIMESTAMP
      WHERE
        order_serial_no = #orderSerialNo# AND env = #env#
    </sql>
  </operation>

  <operation name="queryBySerialNo" multiplicity="one">
    <sql>
      SELECT
        <include refid="order.columns"/>
      FROM flight_order
      WHERE
        order_serial_no = ? AND is_delete = 0 AND env = #env#
    </sql>
  </operation>
</table>
```



如何安安静静地写代码？？？



程序员幸福吗??? 我姓: 增, 名: 加, 字: 重构

- 真的纯业务逻辑吗
- 真的能独立部署吗

```
@Override
public ResponseContext<BookResponseV0> invoke(BookRequestV0 request) throws BaseOrderCoreInterfaceException {
    try {
        // 获取缓存快照 (航班信息 + 价格信息)
        SearchDetailResV0 snap = flightDetailsSnapshotService.retrieveFlightDetails(request.getTraceId());

        // 基础校验
        BookContext context = new BookContext(request, snap);
        doBaseValidates(context);

        // 调用预订校验接口(强校验)
        BookValidateResponseDTO bvResponse = doBookValidate(context);

        // 校验结果并获取最新价格
        SnapClientPriceInfo snapClientPriceInfo = processResponse(context, bvResponse);

        // 处理业务 创单
        OrderInfoV0 order = doBusiness(context, snapClientPriceInfo);

        // 创建返回结果
        return createResult(snapClientPriceInfo, order, request);
    } catch (OrderBizException | GatewayException | AssemblyException | ValidatorException e) {
        throw new BaseOrderCoreInterfaceException(request.getTraceId(), e.getError(), e);
    }
}
```

```
/**
 * @see com.ly.sof.spi.context.StartupCallback#startup(com.ly.sof.spi.context.SOFContext)
 */
@Override
public void startup(SOFContext context) {
    LogContextUtils.setModule(MODULE);
    LoggerUtils.info(logger, "Register resource engine service proxy...", new Object[] {"Registry"});
    Map<String, T> engineServicesMap = Maps.newConcurrentMap();
    try {
        try {
            List<EngineServiceMetadata> engineServiceMetadatas = loadEngineServiceMetas();
            for (EngineServiceMetadata serviceMeta : engineServiceMetadatas) {
                // 创建资源引擎服务代理存根, 获取引擎服务动态代理对象
                Reference<T> reference = createEngineServiceRef(serviceMeta);
                LoggerUtils.info(logger, "reference:{}", FastJsonUtils.toJSONString(reference));
                T proxy = reference.getObject();
                engineServicesMap.put(serviceMeta.getEngineCode(), proxy);
            }
        } catch (Exception e) {
            throw new IllegalStateException(e.getMessage(), e);
        }

        if (MapUtils.isEmpty(engineServicesMap)) {
            throw new IllegalStateException("Cannot register resource engine service proxy");
        }

        engineServices.clear();
        engineServices.putAll(engineServicesMap);
        LoggerUtils.info(logger, "Register resource engine service proxy, online engines: {}", new Object[] {"Registry", engineServices.keySet()});
    } finally {
        LogContextUtils.removeAll();
    }
}

@Override
public void shutdown() {
    LogContextUtils.removeAll();
    try {
        LoggerUtils.info(logger, "Shutdown resource engine service proxy, online engines: {}", new Object[] {"Registry", engineServices.keySet()});
        engineServices.clear();
    } catch (Exception e) {
        throw new IllegalStateException(e.getMessage(), e);
    }
}
```



环境, 框架, 依赖, 太难搞
调试, 性能, 安全, 太复杂



- 各种环境的统一性
- 各种开发框架的烦心事
- 各种调用的乱依赖
- 代码以外花了多少时间
- 开发调试最烦对外依赖
- 性能往往做好后才细想
- 安全出了问题才懂了它
- 专业领域还有多少



部署, 运维, 扩容, 太麻烦

[TECH00]

TVP Tencent Cloud
Valuable Professional

/serverless/DAYS



- 部署在哪, 多少实例
- 各种配置, 上线, 下线
- 扩个容各种关注点
- 真的可以人人是全栈吗



Serverless 在不同研发团队眼中就像硬币的两面

业务系:

- 我的业务模型很重要
- 我的业务逻辑应该与代码一一对应
- 工程进度要越快越好

基础系:

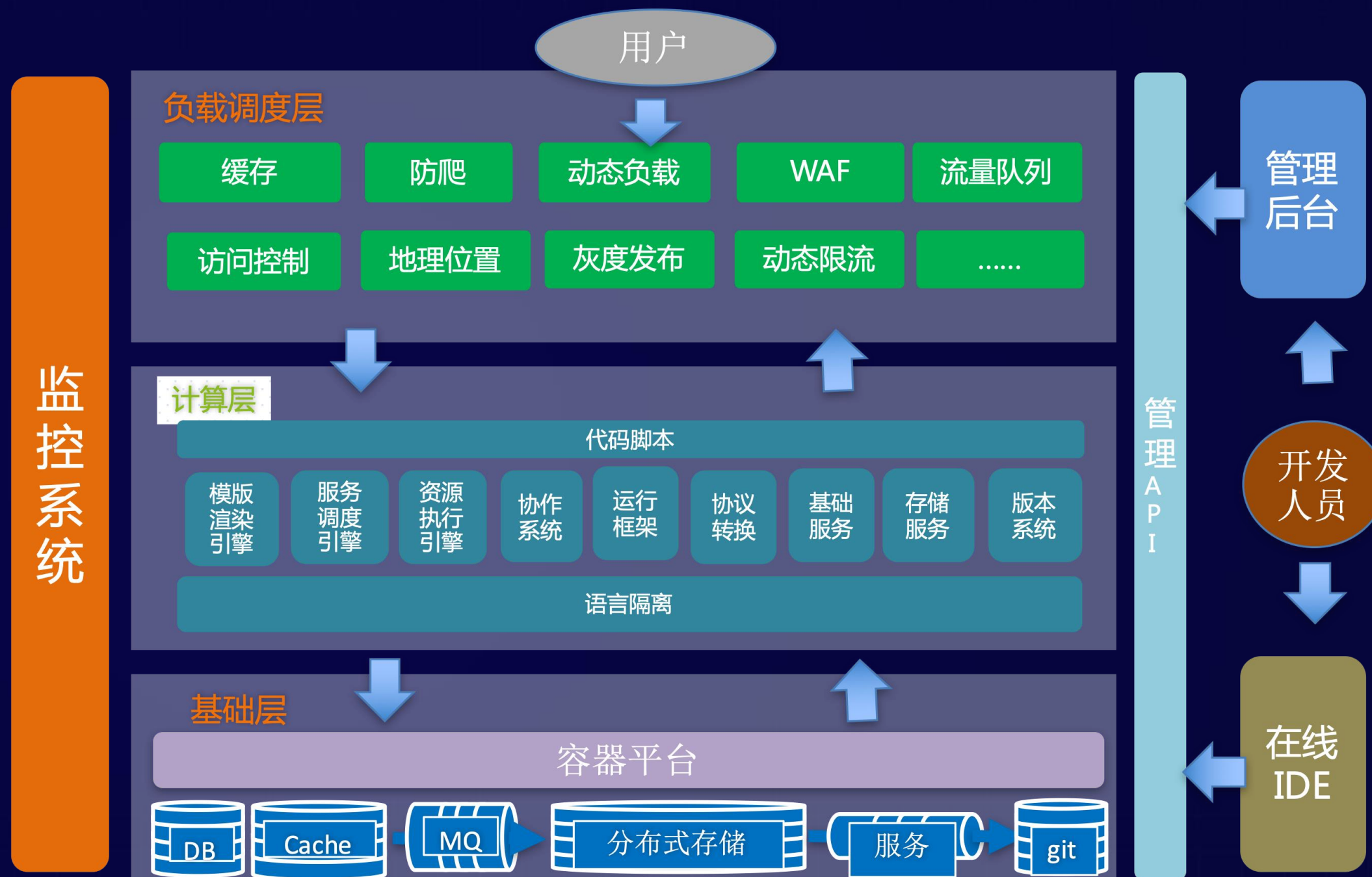
- 开发时我是银弹, 故障时我又不是银弹
- 业务不就是CRUD吗
- 把你程序拉起来我的弹性工作就完了

一条 SQL 到一个服务的距离到底有多远？

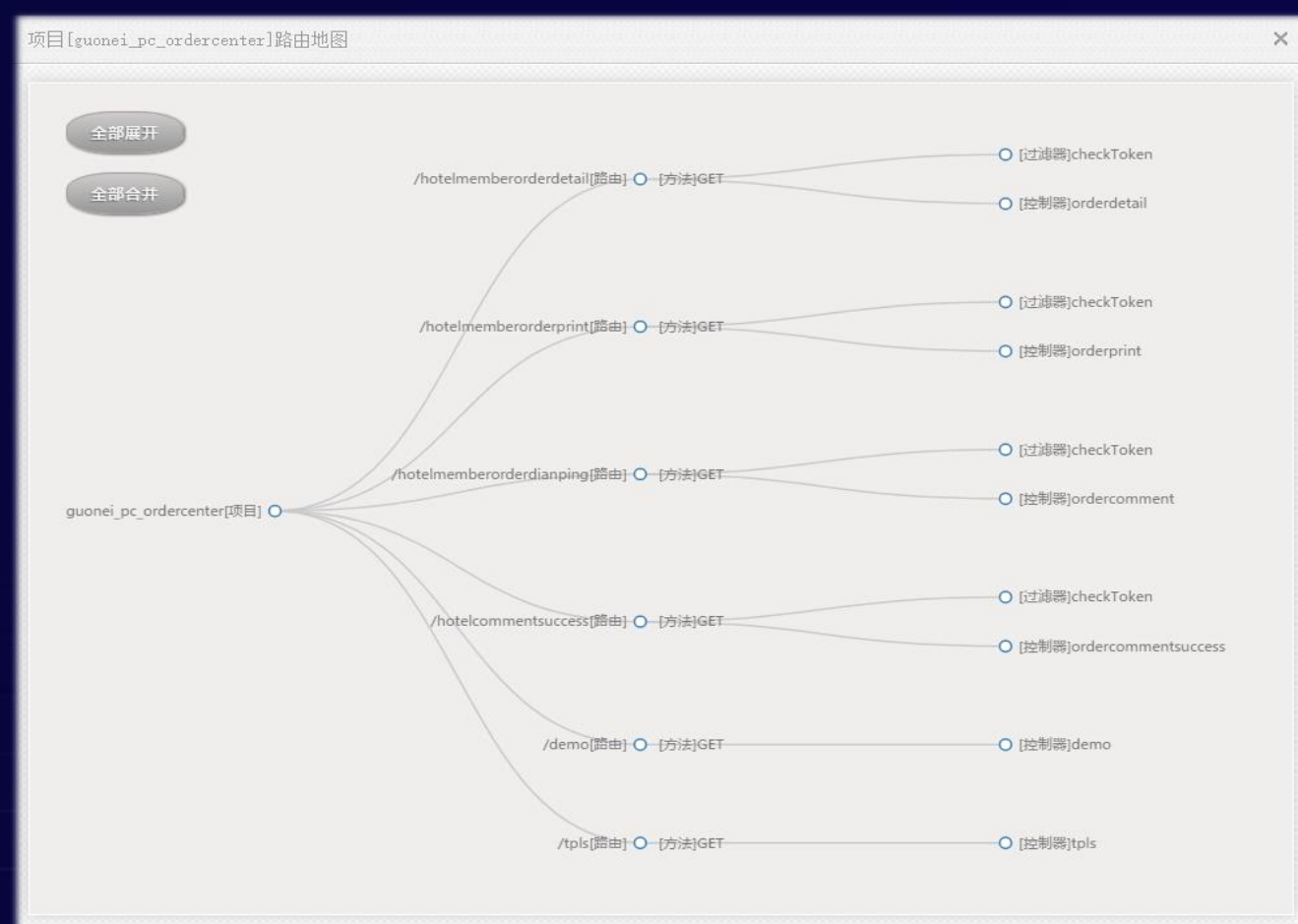
一个代码脚本能不能就是一个微服务？

如果重构是天性，能不能让重构变得廉价？

1. 居于原有基础服务
2. 利用原有容器平台
3. 集成能集成的一切



- 底层容器级隔离
- 语言级隔离（语言VM）
- 通过Gateway访问到应用
- 根据流量自动拉起应用和自动扩容部署实例





- 编程SDK升级无感
- 所有使用均于代码方式使用

```
tpls.js • orderprint.js orderdetail.js ordercommentsuccess.js ordercomment.js demo.js
1 let data = await utils.pGetTplAsync(["ordercommentsuccess.js", "ordercomment.js", "demo.js"], req);
2 res.send(data);
3 await appendOrderAsync("ordercomment.js", data);
   append
   clearCookie
   cookie
   get
   header
   json
   jsonp
   redirect
   renderAsync
   send
   status
   type
```

```
reqOption = {
  url: `${util.getUrlPrefix()}${orderId}`,
  headers: {
    "Referer": "http://www.tencentcloud.com",
    "Host": "${util.getHost()}",
    "Cookie": "${util.getCookie()}"
  },
  timeout: 5000
};

orderInfo = {};
s1 = Date.now();
[err, resp, rspInfo] = await utils.pGetTplAsync(["ordercommentsuccess.js", "ordercomment.js", "demo.js"], req);
s2 = Date.now() - s1;
console.log(`Request time: ${s2}ms`);
var rspJson = JSON.parse(rspInfo);
if (rspJson && rspJson.data) {
  orderInfo = rspJson.data;
} else {
  res.redirect("/");
  return;
}

Object.keys(orderInfo).length === 0 ? {
  console.error(resp);
  res.redirect("/");
  return;
}
```

创建一个新数据源

MySQL 数据源

只读

0

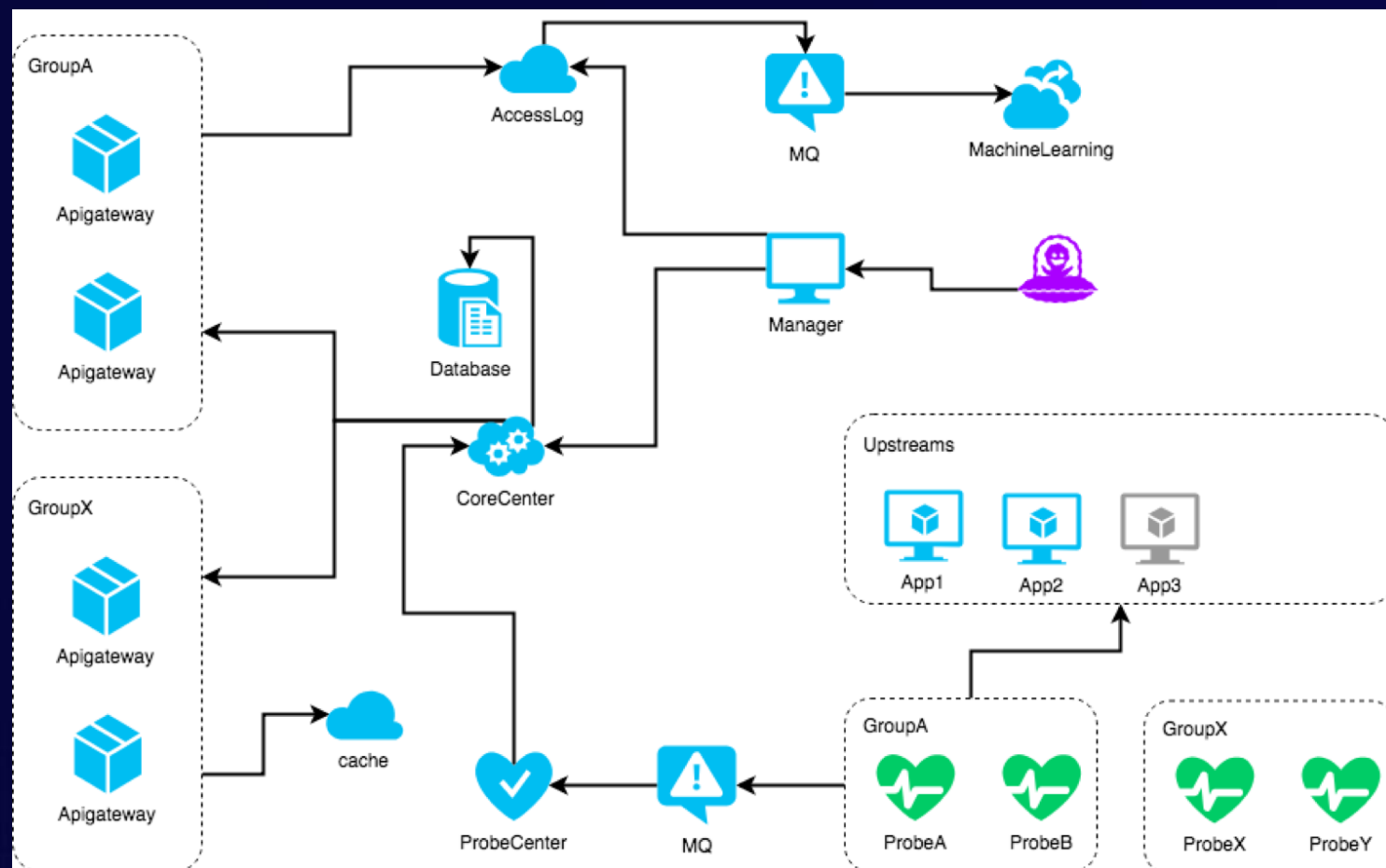
备注

确定 取消

例：从订单数据库中拉出数据

`Server.DB.Order.Get("SELECT * FROM ORDER")`

• Serverless服务毫秒级别的弹性扩容和缩容的能力



多种负载模式

分组模式：分组负载 轮询模式：轮询集群中所有机器
弹性模式：根据访问情况自动增减负载

根据访问量，自动扩容和缩容集群

将应用的分级为6级，根据请求量和后端应用的响应时间，动态调整负载机器数量



热加载

免去每次对负载均衡的Reload操作，减少系统吞吐量的影响

适用任何应用

动态负载均衡器可以适用任何应用，包括：JAVA, IIS, Go 等等





- 无服务器架构不能只解决线上的问题
- 开发过程的痛苦更多
- 将所有的功能可视化，可配置化，不再需要本地环境，有多少环境不再被关注

```
1 var orderId = req.query.id || 0;
2 var isUploadPic = parseInt(req.query.isuploadpic) || 0; //0: 没有上传图片 1: 上传图片
3 var reqOption = {
4   url: `${utils.config.reqUrlPrefix}${orderId}`,
5   headers: {
6     "Referer": "http://leonidmanager.17uoft.com/ide/",
7     "Host": `${utils.config.reqUrlHost}`,
8     "Cookie": `${utils.config.reqUrlCookie}`
9   },
10  timeout: 5000
11 };
12 var orderInfo = {};
13 var s1 = Date.now();
14 var [err, resp, rspInfo] = await requestAsync(reqOption);
15 var s2 = Date.now() - s1;
16 if (rspInfo && rspInfo !== null) {
17   var rspJson = JSON.parse(rspInfo);
18   if (rspJson && rspJson.rspStatus) {
19     orderInfo = rspJson.rspInfo;
20   } else {
21     res.redirect(`${utils.config.reqUrlPrefix}${orderId}`);
22     return;
23   }
24 }
25 if (Object.keys(orderInfo).length === 0) {
26   console.error(resp);
27   res.redirect(`${utils.config.reqUrlPrefix}${orderId}`);
28   return;
29 }
30 if (orderInfo.OrderFlag !== "已点译") {
31   res.redirect(`${utils.config.reqUrlPrefix}${orderId}`);
32   return;
33 }
34 let pageFootSign = `${utils.config.reqUrlPrefix}${orderId}` + orderInfo.LongOrderSerialId;
35 pageFootSign = "cnwidescreenfoot" + encodeURIComponent(pageFootSign);
36 let data = await utils.pGetTplAsync(["cnwidescreenjscss", "cnwidescreenhead", "cnmember_leftmenu", pageFootSign], req);
37 data.orderInfo = orderInfo;
38 data.isUploadPic = isUploadPic;
39 data.s2 = s2;
```

90%

初始化项目脚手架

申请git
安装框架
安装模块
调试脚手架

系统框架自带

80%

数据源、路由和日志

定义编写路由
数据源连接
拦截器、控制器
日志记录

MVC+系统进行配置

40%

编码开发

工具函数，常用类库
逻辑编写
程序调试
代码版本控制

云端编辑器编写代码

90%

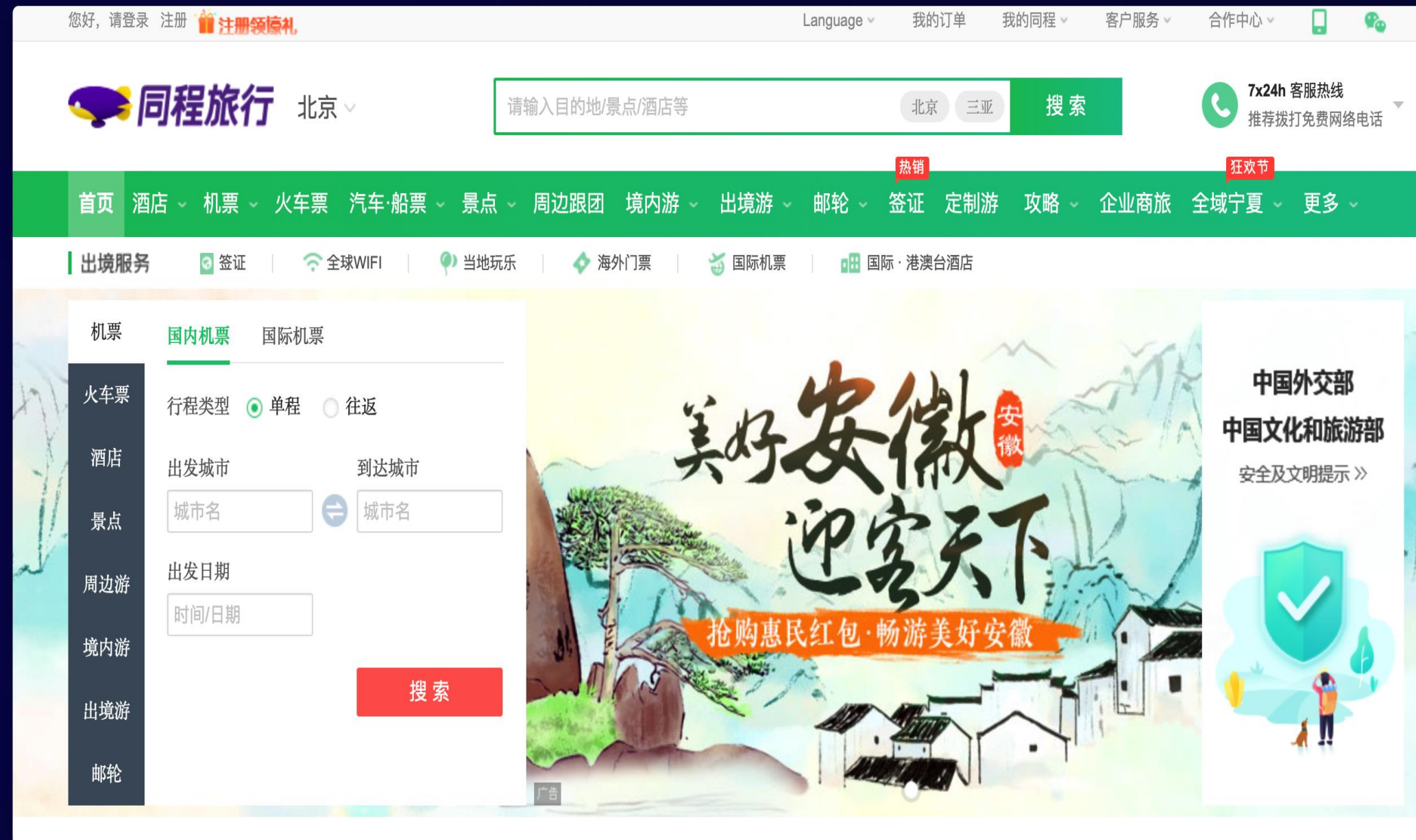
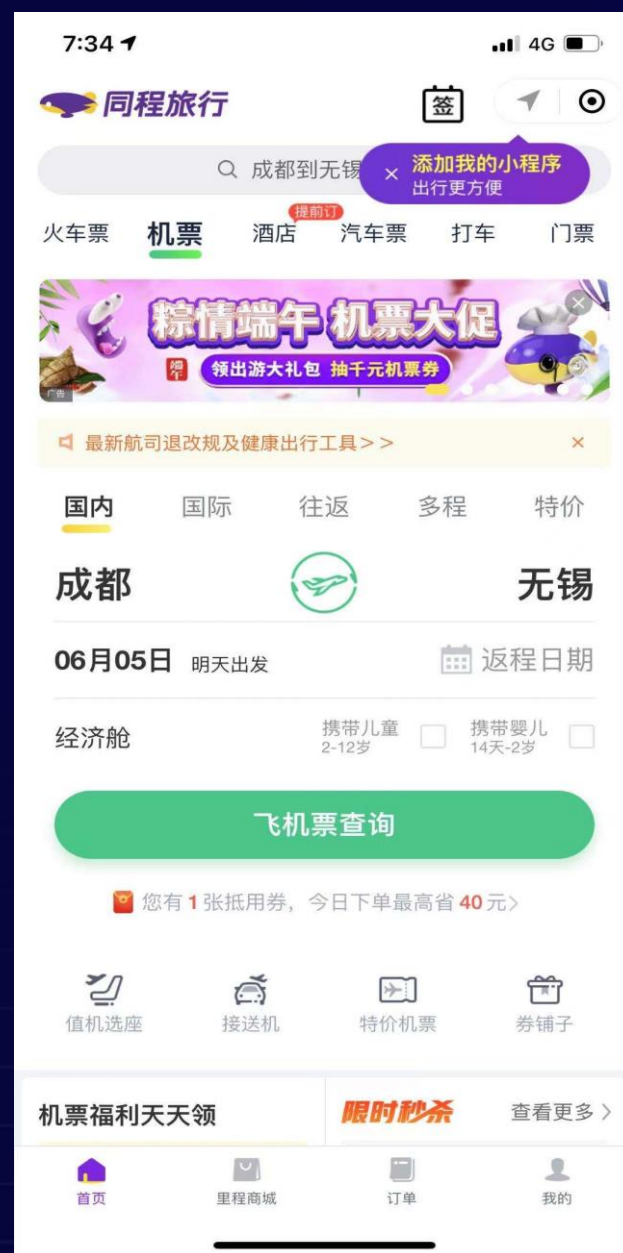
发布和运维

申请应用上线
配置运行环境
申请访问路径
应用运行情况和日志

应用系统一键发布

我们把什么放在Serverless平台上跑了？

- 所有的网页
- 所有的活动推广
- 部分变化量大的后台





- 一些逻辑简单的业务服务
- 一些逻辑变化快的业务服务
- 大量的临时的小服务

```
ipDefender.lua f1.lua main.lua x index.lua utils.lua
29 else
30     console.error('设置黑名单失败 ip:' .. v.ip .. 'err:' .. err)
31 end
32 end
33 end
34 end
35 else
36     console.error('未获取到黑名单列表')
37 end
38 console.error('设置黑名单列表完成')
39
40 console.error('设置配置列表完成')
41 [], 10)
42 if err then
43     console.error('计时器启动失败' .. err)
44 end
45
46 local requestText = req.raw_body
47 if requestText and requestText ~= '' then
48     local json, err = global.json_parse(requestText)
49     if err then
50         console.error('代理转发请求失败 json反序列化失败:' .. err)
51         utils.response(res, utils.RspType.Exception, utils.RspCode.RspCode_7000)
52     else
53         local resp, body, err = req.proxy()
54         if resp and resp.status >= 400 then
55             console.error('代理转发请求失败 response status:' .. resp.status)
56             utils.response(res, utils.RspType.Exception, utils.RspCode.RspCode_7000)
57         elseif err then
58             console.error('代理转发请求失败 error:' .. err)
59             utils.response(res, utils.RspType.Exception, utils.RspCode.RspCode_7000)
60         else
61             res.send(body)
62         end
63     end
64 else
65     console.error('代理转发请求失败 未传入请求体')
66     utils.response(res, utils.RspType.Exception, utils.RspCode.RspCode_7000)
67 end
```



Serverless的使用情况：配套功能集成

[TECHNO]

TVP Tencent Cloud Valuable Professional

/serverless/DAYS

弹性计算服务



房型酒店信息交通点评(39)

入住 2017-06-28 周三 离店 2017-06-29 周四 修改日期 共1晚

含早餐 有窗 床型 立即确认 免费取消 可订 支付类型

特价单人(无窗)
大床(1.5米) | 免费有线 | 最多可住2人 | 面积22㎡ | 楼层1-2层

¥222起
共7个产品

名称	床型	早餐	窗户	取消规则	日均价	
[同程自营]	大床	双早	无窗	不可取消	¥222	预订 在线付
(双人入住) 代理	大床	双早	无窗	不可取消	¥229	预订 在线付
[标准价-双早] 退	单人床	双早	无窗	不可取消	¥240	预订 在线付
特价单人(无窗) 退	大床	无早	无窗	不可取消	¥273	预订 在线付
特价单人(无窗) 退	大床	无早	无窗	免费取消	¥288	预订 到店付

查看更多产品

附近酒店

周边同级酒店

北京君鑫酒店
距离该酒店57米
¥199起

飘HOME连锁酒店(北京五棵松店)
距离该酒店124米
¥160起

北京毛林惠丰大酒店
距离该酒店454米
¥376起

海友酒店(北京定慧寺店)
距离该酒店549米
¥139起

北京长荣酒店
距离该酒店616米
¥209起

如家快捷酒店(北京定慧桥五路居地铁站店)
距离该酒店701米
¥226起

*Serverless*不是银弹

微服务体系

稳定的核心服务

中台的基础

事件的触发源

系统的核心流程

Serverless平台

即用即走的服务

不停变化的服务

耗资源的服务

事件型服务

低代码

业务组件

基础组件

可配置性

Serverless工具体系

开发测试环境服务

DEVOPS 服务

生产基础设施服务

*Serverless*的核心价值不在于省下哪几台服务器的钱

THANK YOU!

感谢聆听!