

# 一分钟一千万 天猫双十一背后的互动游戏引擎

范淑宾（范导） 淘天集团 高级技术专家

## 讲师简介

“



范淑宾（花名：范导）  
高级技术专家

阿里17年专注各个核心系统技术研发：

- 参与Alibaba国际站会员系统、小二工作台的设计与维护；
- 参与Aliexpress速卖通联盟平台建设，覆盖超40%的平台订单；
- 淘天集团互动基础平台负责人、天猫双十一互动游戏架构师，完成整个互动基础平台从0到1的建设，稳定支撑集团多个BU的上百个互动类业务场景。

”

# 目录

- 互动游戏的背景与挑战
- 核心实践： 从不确定中找确定性
- 核心实践： 规则调控->观测预判->快速扩容
- 核心实践： 降本提效
- 天猫双11幻想岛总动员成果展示
- 总结与启示

## 亮点介绍

- 领域驱动设计(DDD)，在超高QPS、超大数据量业务场景下的实践。
- 游戏模式、玩法模型的抽象实践。
- “反微服务化”，收敛资源、减少依赖，从IO密集转向计算密集。
- 充分利用存储特性实现高并发、强一致兼得，可灵活扩展和无感替换的设计。

# 互动游戏的背景与挑战

# 互动？游戏？互动游戏？

“互联网经验告诉我们，高频场景的增长潜力始终高于低频场景，如何从低频转向高频，是所有经营者竭尽全力地追求。”

—— 雷军《小米创业思考》



## 天猫双11大促互动游戏



2019

游戏行为

做任务、升级、拉人助力



2021

虚拟资产

积分、等级



2023

平台权益

红包、卡卷

# 大促互动游戏的挑战

## 大促互动游戏本质：多人在线实时游戏和大型营销活动的结合体

### 流量不确定

每次都是全新玩法，用户行为很难预测。不同玩法导致不同的流量形态，没有一个稳定可预估的流量模型。

### 研发周期短

往往活动前一段时间才确定具体玩法，留给产品研发的时间比较短，导致有的事情无法做得很精细化。

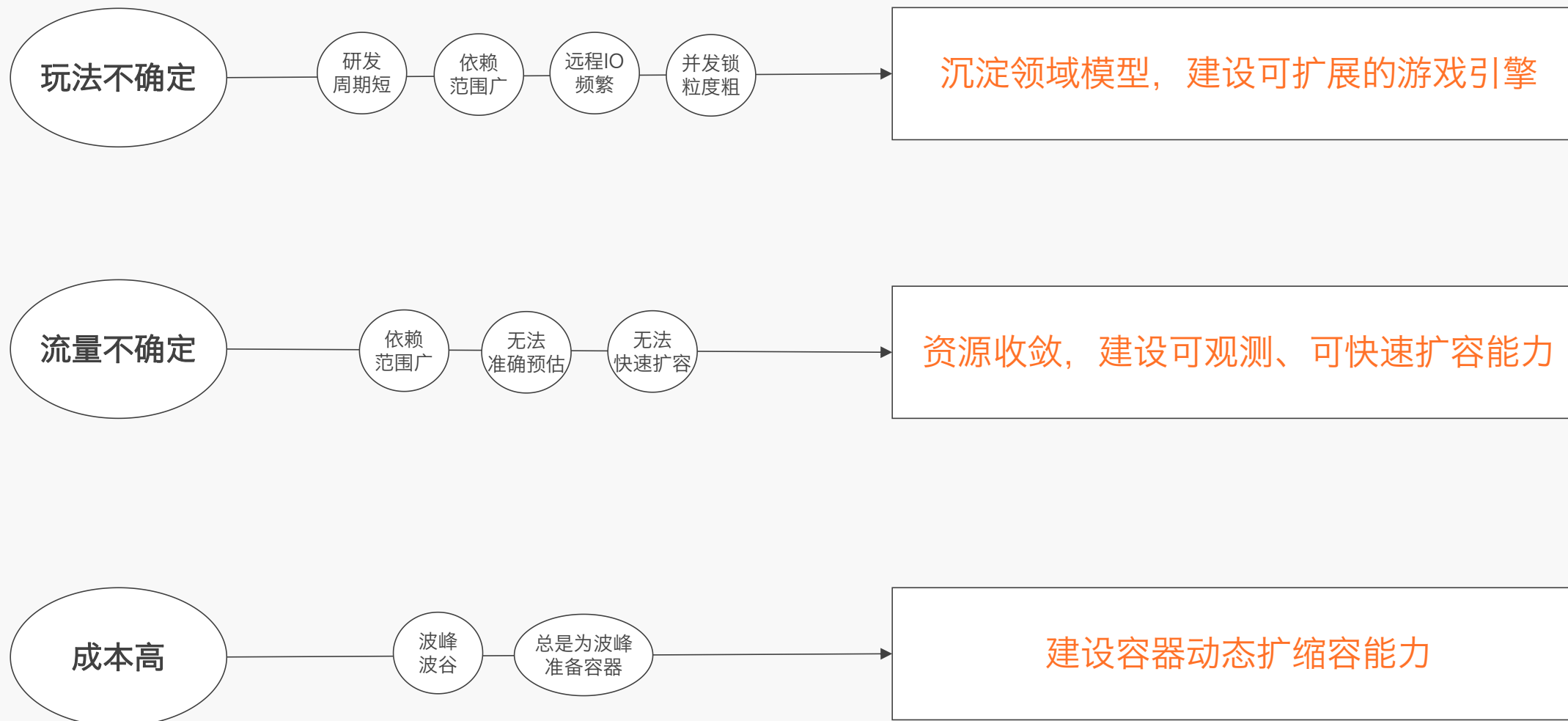
### 用户体验敏感

全民大促前的暖场，内部和外部的关注度很高，游戏玩法、响应的及时性都与用户利益相关，传统游戏里的问题都会在这里放大。

### 高QPS和高数据一致性

游戏和营销的结合体，用户游戏数据往往与权益相关，游戏数据不可丢失，这一点与传统游戏区别很大，但QPS极高。

# 破题思路





# 从不确定中找确定性

——玩法不确定

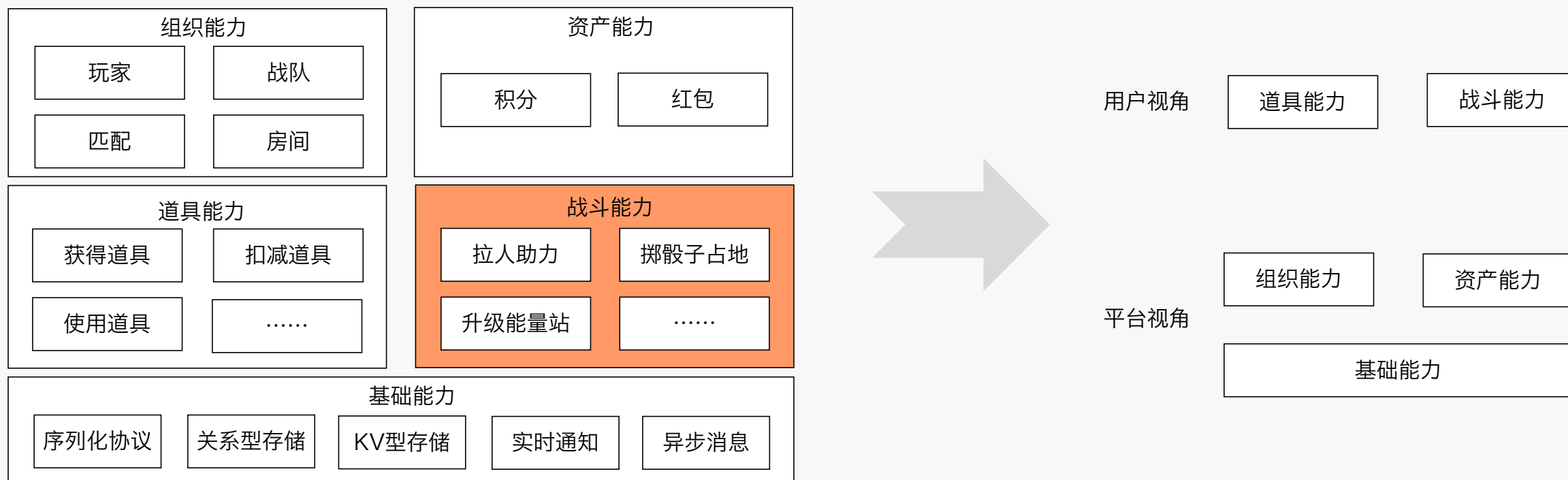
# 核心实践 —— 确定性的抽象

从不确定的玩法中抽象确定的东西，沉淀成可复用的模型与框架(IGF)，可极大提升研发效率

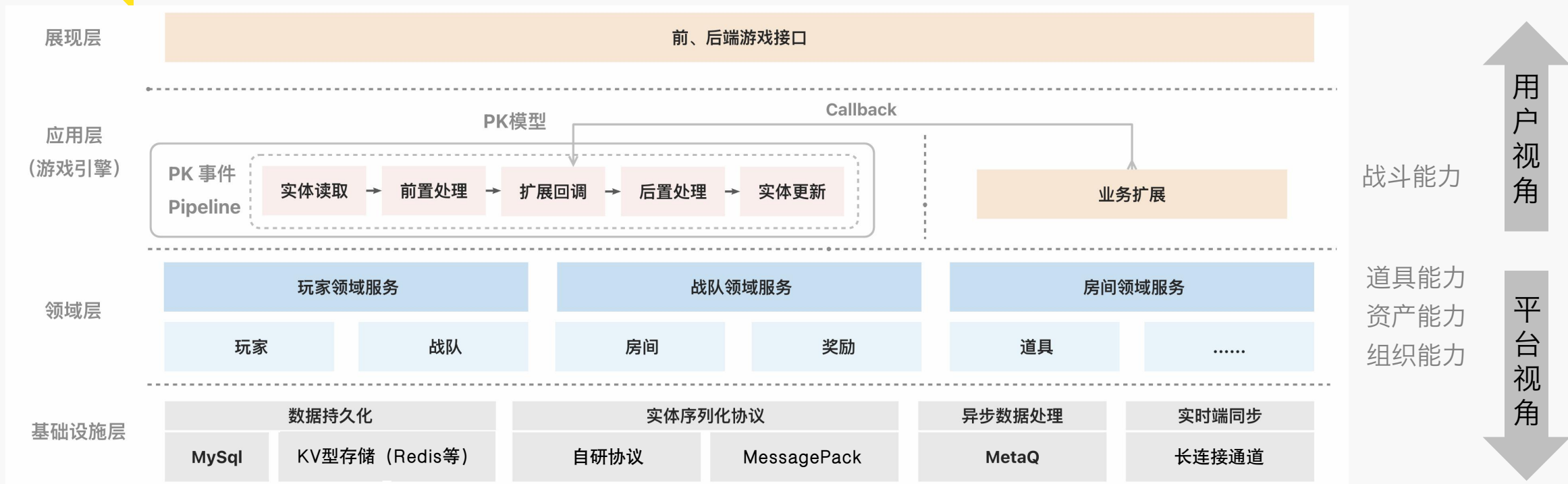
**流程的确定性**——虽然具体玩法不同，但大促游戏的流程是有章可循的



**功能的确定性**——虽然具体玩法不同，但仍有共同的功能模块



# 核心实践 —— 面向领域设计



## 实体、领域服务:

面向实体及领域服务研发，屏蔽底层数据存取、异常流处理等细节，专注业务逻辑实现。

## 聚合、聚合根:

核心游戏数据聚合，高QPS下数据强一制；系统链路收敛，系统扩缩容更灵活；大幅度减少IO，提升性能与稳定性。

## 仓储:

多模式存储业务无感按需切换；多分区存储集群，存储上限成倍预留；精控制数据并发锁范围，提升用户体验。

# 核心实践 —— 领域实体设计

## 贫血模型

VS

## 充血模型

```
try {
    Long playerId = 12345L;
    Long upgradeCost = 10000L;
    Long upgradeLevel = 1L;

    // 先扣减升级消耗的资产
    Property playerCoin = PropertyService.findProperty(playerId, "COIN");
    playerCoin.setAmount(playerCoin.getAmount - upgradeCost);
    PropertyService.updateProperty(playerCoin);

    // 再给用户增加等级
    Player player = PlayerService.findPlayer(playerId);
    player.setLevel(player.getLevel() + upgradeLevel);
    PlayerService.updatePlayer(player);
} catch (Throwable e) {
    // 扣资产或加等级失败了怎么办???
    // 高流量下的IO放大怎么办???
}
```

```
// 查询要操作的玩家
IgfPlayer player = IgfPlayer.getById(12345L);

// 确定升级要消耗的资产
Property cost = new Property();
cost.setType("COIN");
cost.setAmount(10000L);

// 玩家升级 (upgradeHandler为业务扩展点)
player.useTool(cost, upgradeHandler);
```

Handler 怎么设计???

2021双十一喵糖总动能量站升级场景示例伪代码

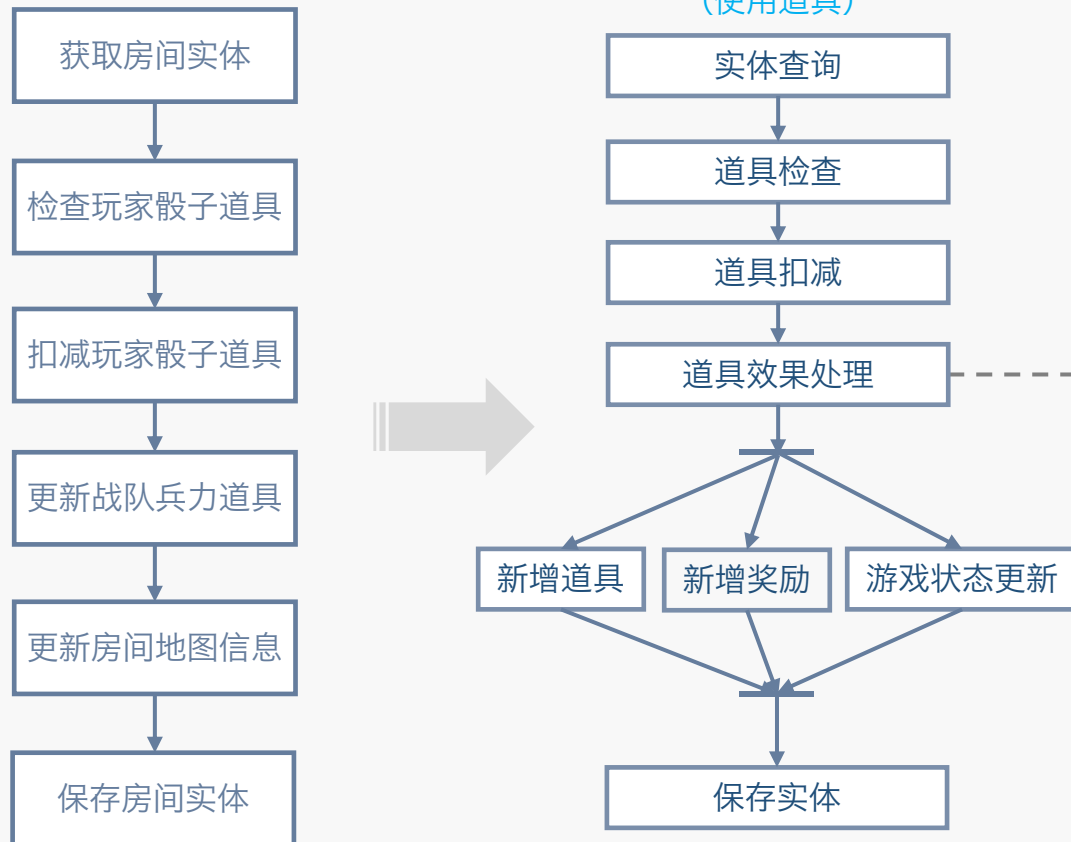
# 核心实践 —— 领域服务扩展设计

游戏战斗逻辑中

不变的部分：每个游戏事件都有相同的执行流程和模式。

变化的部分：每个游戏事件都有独立的执行逻辑和结果。

面向领域的喵糖总动员PK处理逻辑



变化的战斗逻辑

```
public class DiceHandler implements GameLogicHandler {
    @Override
    public void useTool(Player player, Tool usedTool, Room battleRoom) {
        // 掷骰子抢地块逻辑处理
        if (usedTool.getType() == "DICE") {
            useDice(player, battleRoom);
        }

        // 其他类型道具处理
    }

    private void useDice(Player player, Room battleRoom) {
        int randomForce = Random.nextInt(1, 6);

        // 获取指定的地块
        BattleField field = battleRoom.findField(player.getSelectedField());

        // 玩家已经占领地块直接累加兵力
        if (field.getWinnerId() == player.getId()) {
            field.setForce(field.getForce() + randomForce);
            return;
        }

        // 玩家未占领地块，且兵力足够可占领
        if (randomForce > field.getForce()) {
            field.setForce(randomForce - field.getForce());
            field.setWinnerId(player.getId());
            return;
        }

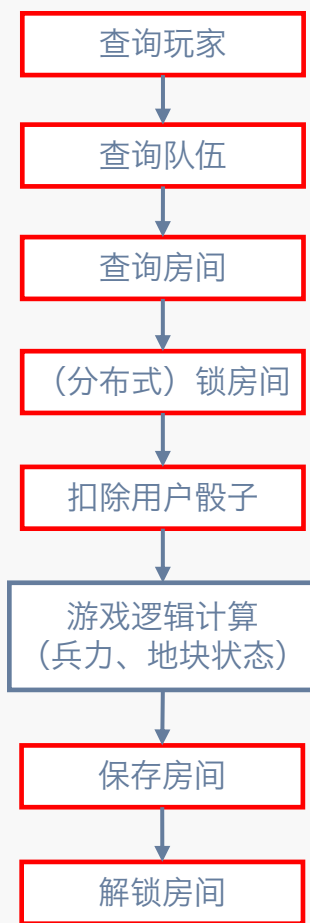
        // 玩家未占领地块，且兵力不足
        field.setForce(field.getForce() - randomForce);
    }
}
```

2021双十一喵糖总动员PK示例伪代码

# 核心实践 —— 鱼（性能）和熊掌（一致性）如何兼得

2021双十一喵糖总动员——投骰子处理流程

业务研发花费大量精力做服务、存储的编排

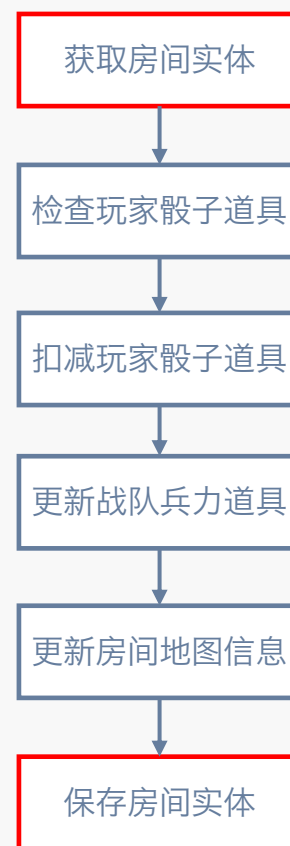


最少8次远程IO  
(红框部分)

核心策略：面向领域、减少IO、数据聚合

红框为产生IO的操作

面向领域的喵糖投骰子处理流程



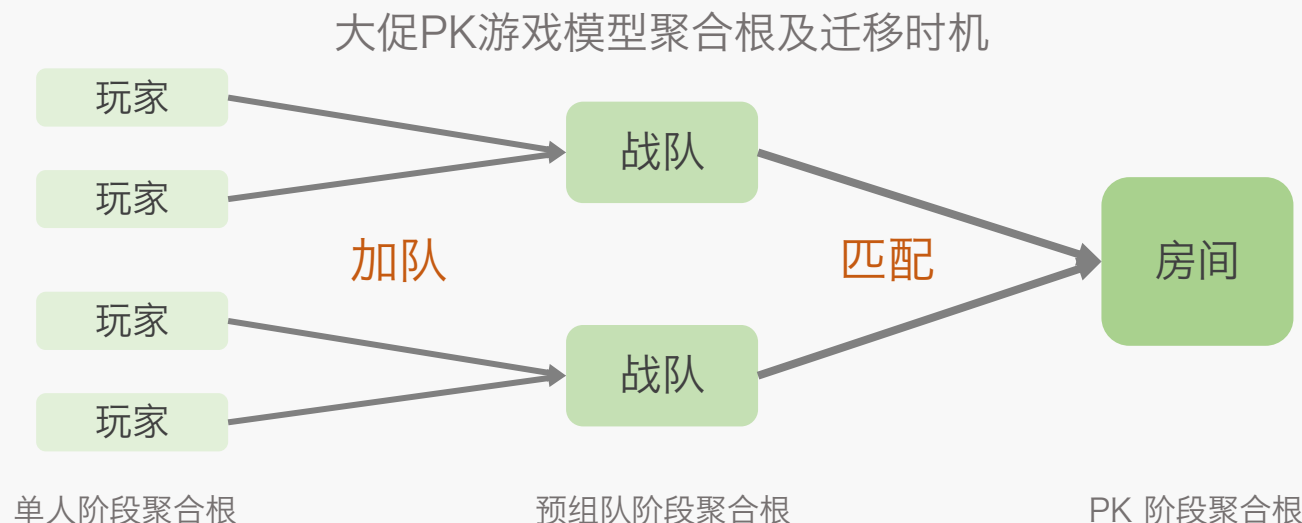
只需2次存储IO  
(红框部分)

纯内存操作  
专注游戏逻辑

# 核心实践 —— 聚合的力量

## 如何识别聚合根？

如果一个聚合只有一个实体，那么这个实体就是聚合根；如果有多个实体，那么我们可以思考聚合内哪个对象有独立存在的意义并且可以和外部直接进行交互。



## 极简IO:

聚合根纬度的数据，理论上只需要一次IO即可取得整个PK房间中所有战队及所有玩家的数据；同时，也只需要一次IO便可将所有玩家、战队的的数据变更写回到存储中去。

## 高性能:

所有对聚合根实体的业务逻辑操作，全部在内存中完成，实现战斗逻辑无需关注存储细节，高效、高性能。

## 原子性:

聚合根数据写入单次可完成，天然具备原子性，多实体数据变化不存在数据不一致问题。



# 核心实践 —— 聚合的问题1

## 聚合根的问题——数据大小

互动大促游戏以多人PK模式为主，一个房间的玩家数大多在2~20个之间，中间有着10倍的差距，聚合后的数据大小对存储运行时性能有很大的影响。

**性能劣化：**无论是关系型存储还是KV型存储，性能最优的单行或单KEY数据大小都会有建议大小。

**带宽瓶颈：**带宽是存储中比较昂贵和有限的资源，从单例看带宽资源非常有限，过大的数据体积会高QPS下会导致带宽打满。

```
public class Entity {  
    /**  
     * 实体ID  
     */  
    protected Long id;  
  
    /**  
     * 创建时间  
     */  
    protected Date gmtCreate;  
  
    /**  
     * 业务域  
     */  
    protected String bizCode;  
  
    /**  
     * 业务子场景  
     */  
    protected String subBizCode;  
  
    /**  
     * 是否有效  
     */  
    protected Boolean valid;  
}
```

JSON 序列化结果(107B)

```
{  
  "id":123456789,  
  "gmtCreate":1669691203590,  
  "bizCode":"SOME_ACTIVITY",  
  "subBizCode":"SOME_SCENE",  
  "valid":true  
}
```

定制序列化协议优化

```
{  
  "entityClass": "xxx.Entity",  
  "entityDef": {  
    "id": {  
      "idx": 0,  
      "type": "long"  
    },  
    "gmtCreate": {  
      "idx": 1,  
      "type": "date"  
    },  
    "bizCode": {  
      "idx": 2,  
      "type": "string"  
    },  
    "subBizCode": {  
      "idx": 3,  
      "type": "string"  
    },  
    "valid": {  
      "idx": 4,  
      "type": "boolean"  
    }  
  }  
}
```

Entity实体结构声明

先声明

再序列化

每个实体结构声明  
全局唯一

定制序列化结果(52B)

```
[  
  123456789,  
  1669691203590,  
  SOME_ACTIVITY,  
  SOME_SCENE,  
  1  
]
```

数据  
大小  
减小  
51%



## 核心实践 —— 聚合的问题2

### 聚合根的问题——并发锁粒度大

玩家数据聚合在房间中后，为了保证个人数据的修改不会产生相互覆盖的问题，那么势必要对整体房间加锁。

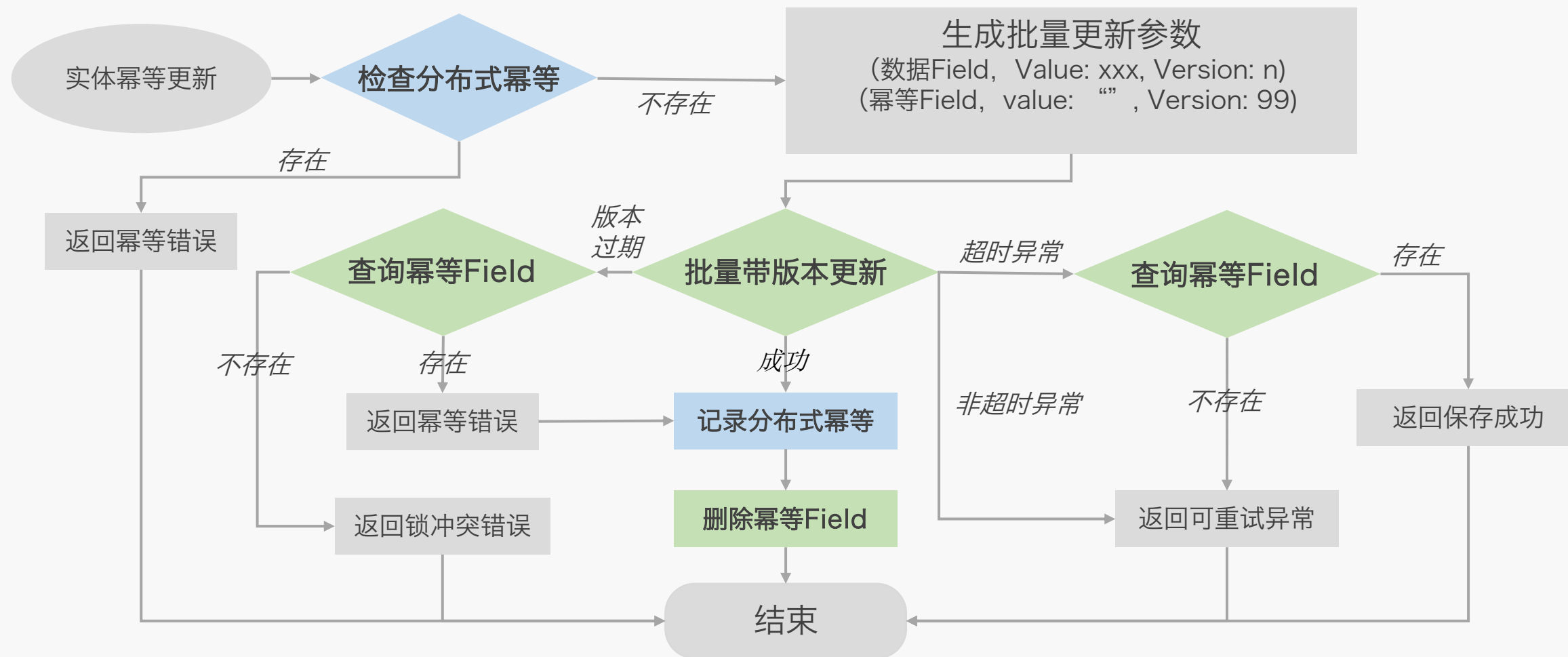


通过这种存储设计，平均的局部写入数据大小比整体写入数据大小减小超过70%。

# 核心实践 —— 聚合的问题3

## 聚合根的问题——迁移后的幂等保障

玩家和战队数据随着匹配结算不断迁移存储实例，原有的幂等数据无法随迁，幂等数据确实会导致核心游戏资产超发问题，严重可产生资损。



跨存储的通用幂等保障方案（以Redis存储为例）

# 规则调控->观测预判->快速扩容

——流量不确定

## 核心实践 —— 通过游戏规则削峰

运营&产品

按照游戏的方式运作

流量错峰

从最开始的运营策略要考虑流量应对，完全没必要全网所有用户都集中同一个场次同一个时间段，这样只会造成人为的高峰。

### 2023天猫双11 幻想岛大作战限时翻倍规则

用户心智：两小时内派出的仔仔数量翻倍。

- 时间约束：9:00 ~ 19:00
- 触发时机：队员上线随机触发
- 触发次数：每天每队只有一次机会
- 持续时间：2小时

公平

刺激

趣味



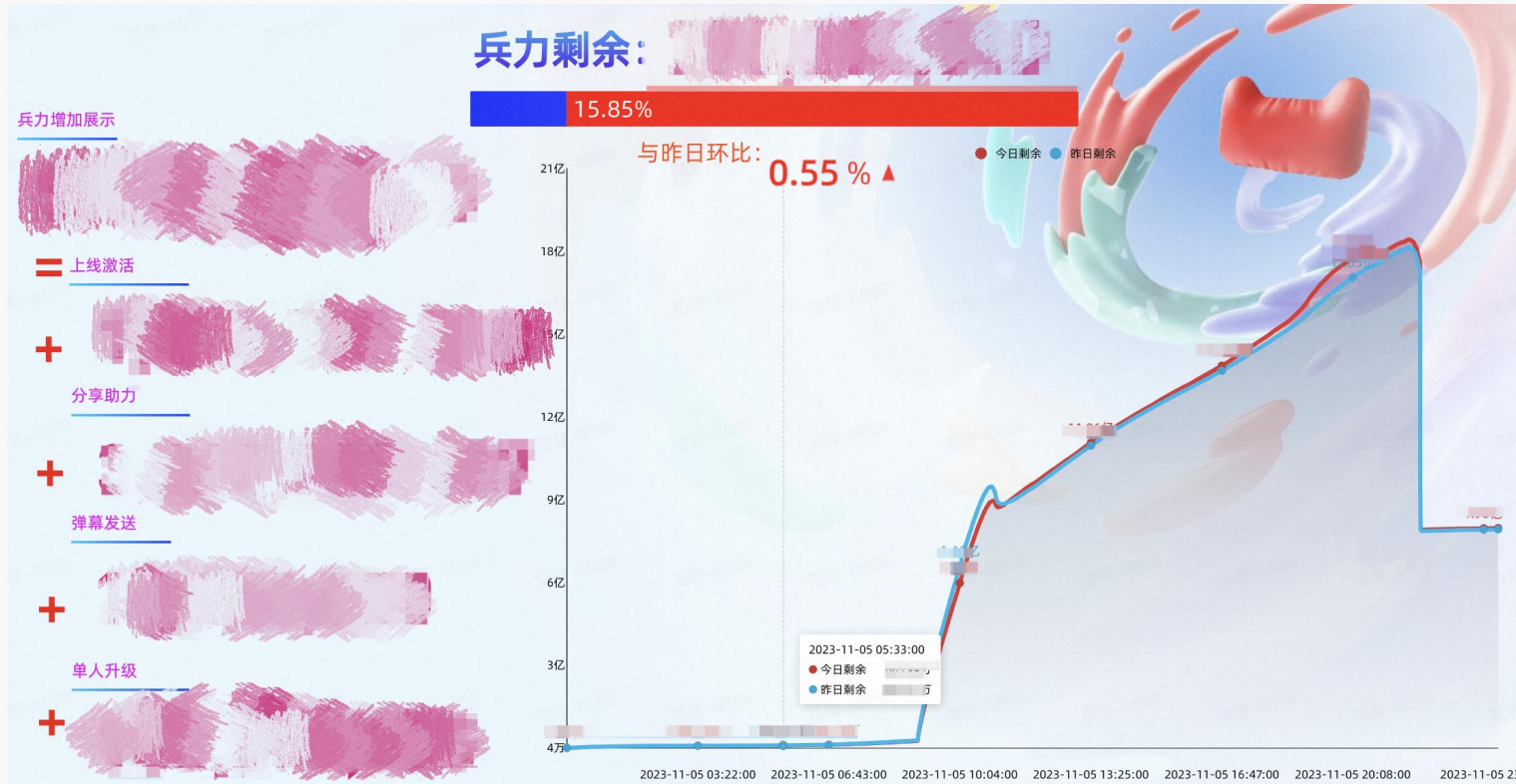
# 核心实践 —— 观测预判系统水位

研发

建立流量预估模型

不能只关注功能逻辑

建立装备数与QPS之间的关系计算模型，通过实时计算技术结合历史数据的对比，预判系统水位。



2023天猫双11 幻想岛大作战兵力大盘

$$QPS = \frac{\text{剩余总兵力} \times \text{消耗比例}}{\text{平均出兵个数} \times \text{持续时间}}$$



## 核心实践 —— 消除快速扩容的瓶颈

简化依赖、调用链路收敛后，容器的快速扩容已经不是问题。但还需要保证强依赖的存储不会成为瓶颈。



2023天猫双11 幻想岛大作战仓储设计

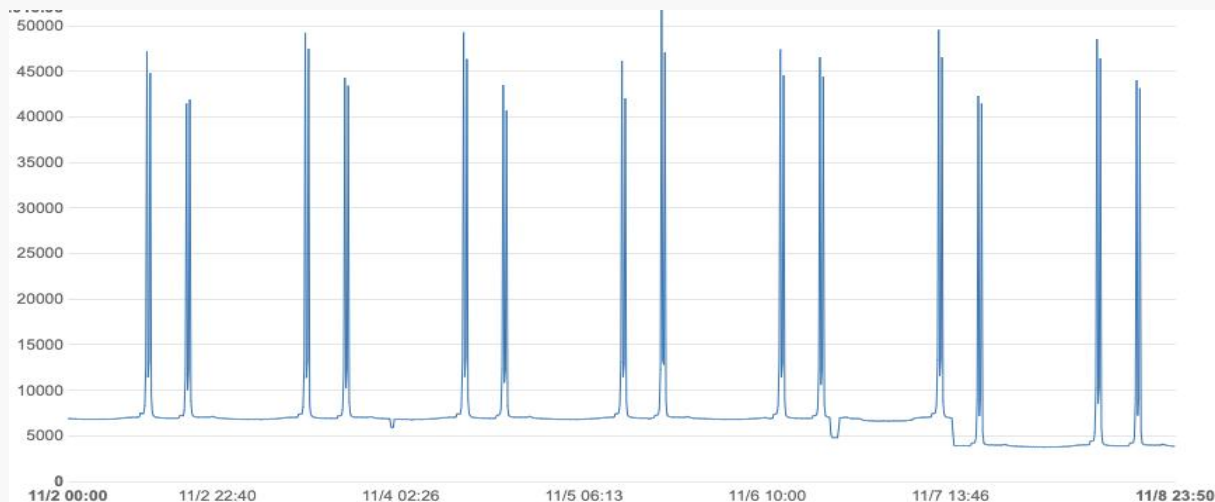
可灵活配置存储集群数量，为双11预留**千万级**QPS和空间扩容空间

# 弹性扩缩容

——运维成本高

# 核心实践 —— 弹性扩缩容

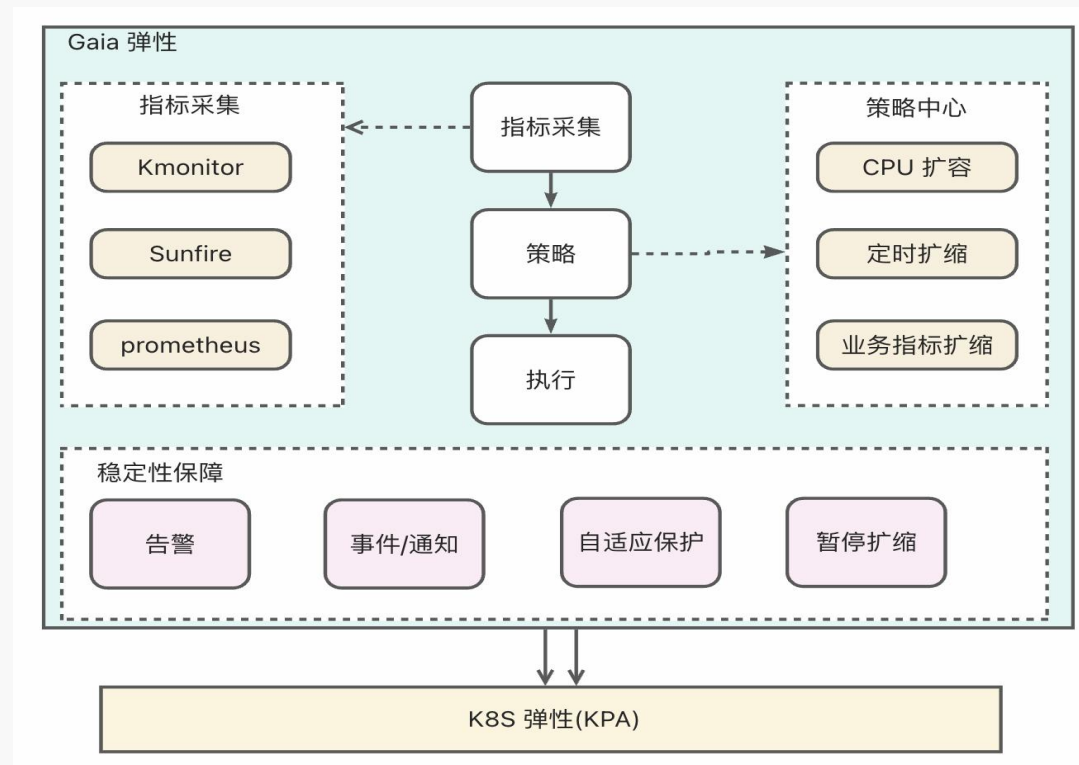
大促玩法周期性明显、峰谷差异巨大



通过弹性扩缩容，大促互动游戏在闲时可节省50%以上的容器资源。

## 过载保护

- 集群层面：对于部分需要配置集群维度不能随着机器数变化而影响到总限流值的场景，GAIA 提供了机器数改变时自动调整单机限流值，从而让总限流值保持不变的能力。
- 单机层面：容器层面提供自适应限流，在 CPU 瞬时超过固定值后会自动触发限流(400-600毫秒作出响应)。



## 弹性策略

- 按照不同业务场景支持多种扩缩策略的组合配置。
- 支持 QPS, RT 等业务指标的组合弹性。
- 支持扩缩容步长来控制单次扩缩容对业务的抖动。

TIPS: GAIA 是淘天自研的一站式函数研发平台(FAAS)。



## 成果展示

## 用户


体验提升

180万+在线

200万+QPS

平均6ms服务端RT

## 技术

幸福感提升

5000万+战队

1分钟匹配1000万战队

30分钟结束值班

## 业务

成本下降

单机吞吐量提升300%

资源利用率提升100%

运维成本下降50%

# 总结与启示

## 做好取舍

- 本地化 VS 服务化
- IO密集 VS 计算密集
- 绝对一致 VS 最终一致
- 聚合存储 VS 分布存储

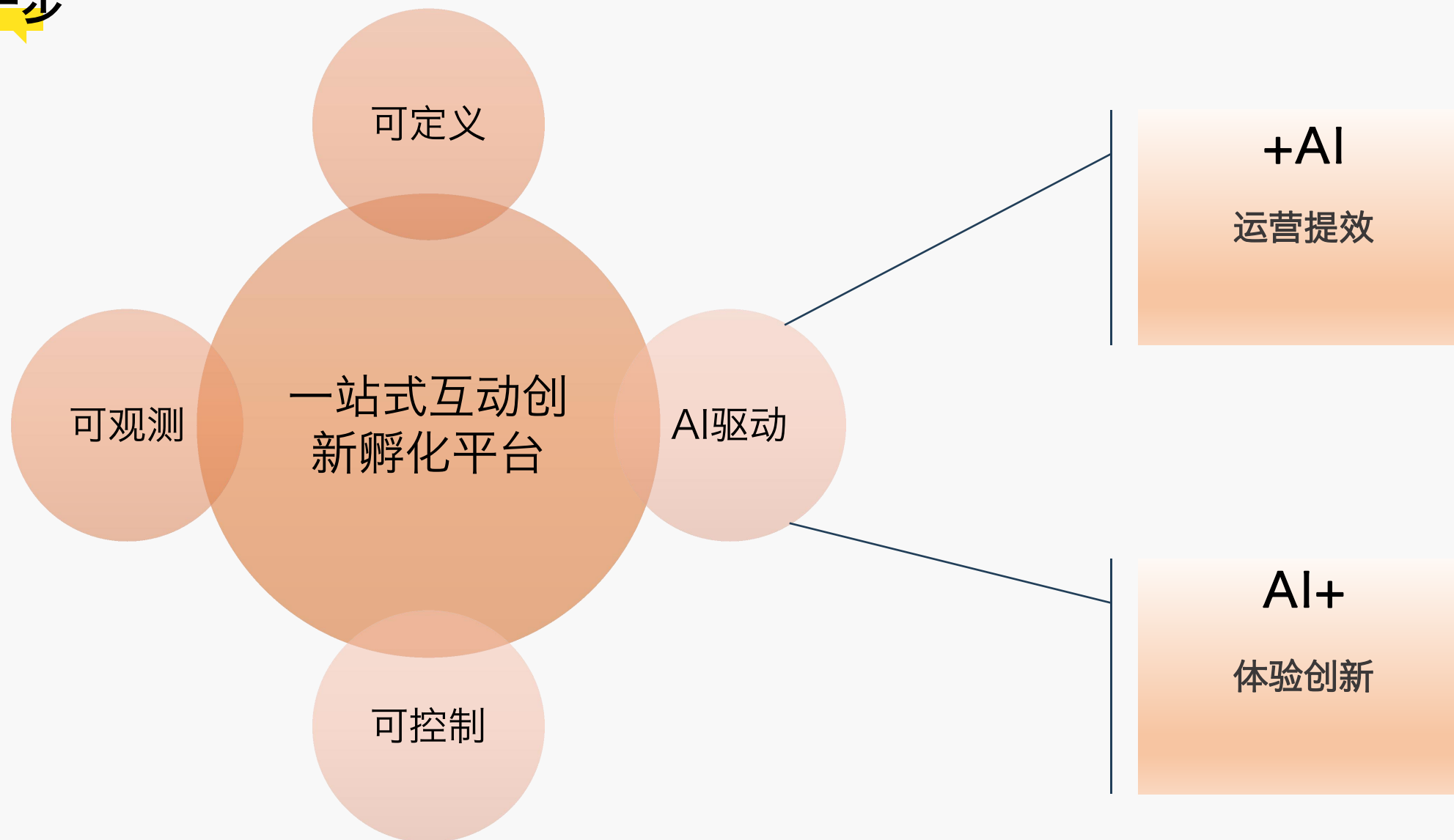
## 打破常规

- 自上而下的领域设计
- 通用的存储结构设计
- 最常用的并不一定适合你
- 性能问题不只有技术方案

## 理解下游

- 依赖服务/中间件的SLA
- 存储单行/单KEY的大小
- 存储事务的范围
- 存储乐观锁的影响范围
- 单个实例的带宽瓶颈

下一步





微信官方公众号：壹佰案例  
关注查看更多年度实践案例