TiD 2017
质量竞争力大会

遗留系统的设计演进实践

关于我

- 王辉，软件工程师@ZTE
- 非典型通信男,专业码砖十余年，
- 从事过嵌入式系统软件、Java企业应用软件的开发，目前从事智能家居系统的设计和开发。
- 希望与你聊聊代码那点事儿。

目录

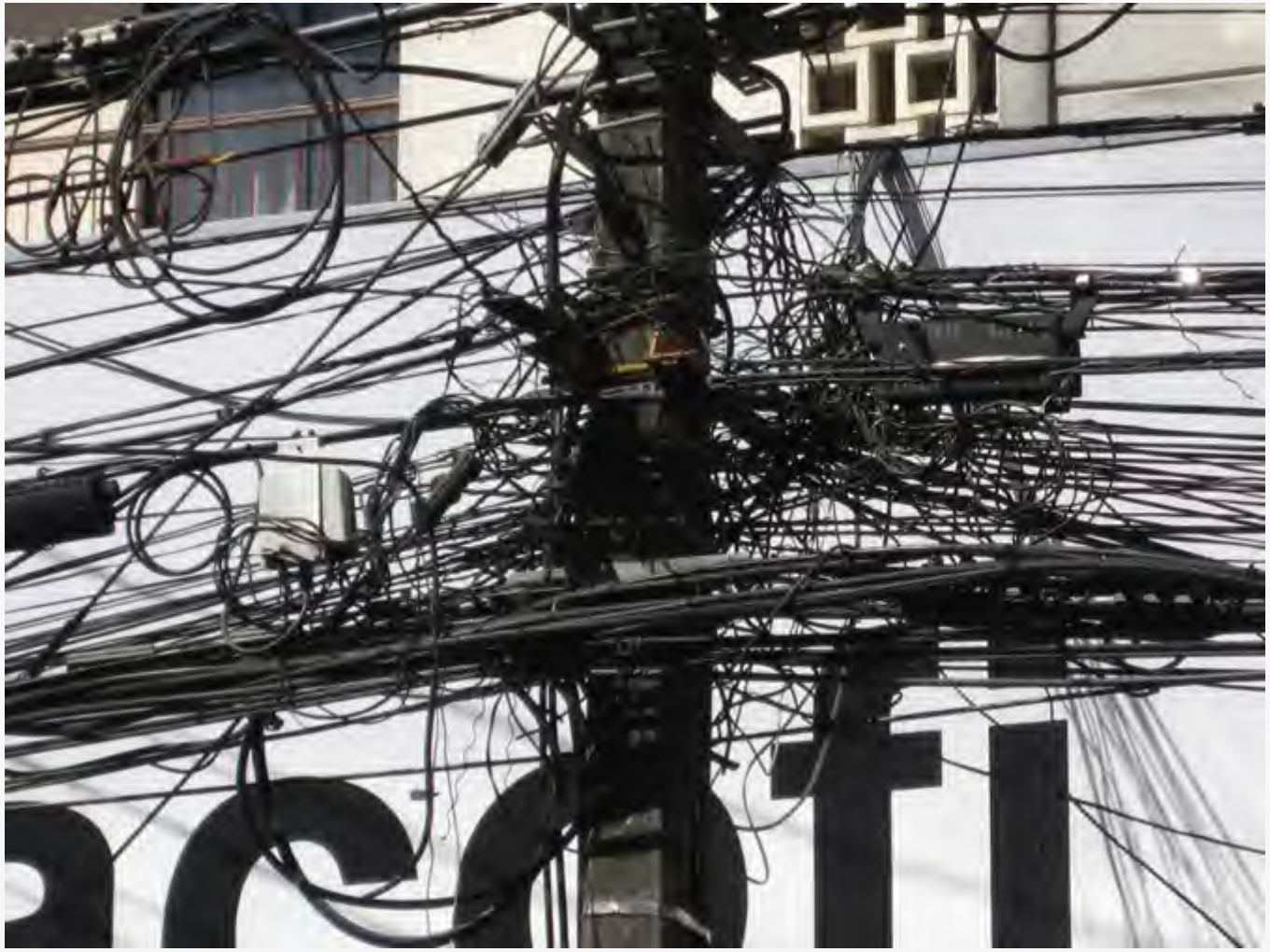
01

背景与问题

软件开发的困境

- “业务域”与“实现域”相割裂
- 产品开发成本逐步上升
- 遗留系统重用困难

常见的遗留系统



遗留系统的困境

- 系统设计没有“与时俱进”
- 缺少安全防护网，无人敢动
- 技术债务累积，**重用**越来越难
- 代码的“坏味道”逐渐累积

代码的坏味道

- 业务知识点分散
- 呈现了过多的“细节”
- 简单重复
- 缺乏层次感
- 没有体现问题领域
- 没有体现业务的核心关注点
- ……

```
T_PortState_Struct AdaptNIPIPortSt(WORD32 dwPortState)
{
  /* [illegible] */
  T_PortState_Struct boardPortState  = {0,0,0,0,0,0};
  getSinglePortState(dwPortState, NIPI_MFE1_PORT_POSITON , &boardPortState.MFE1PortState);
  getSinglePortState(dwPortState, NIPI_MFE2_PORT_POSITON , &boardPortState.MFE2PortState);
  getSinglePortState(dwPortState, NIPI_MFE3_PORT_POSITON , &boardPortState.MFE3PortState);
  getSinglePortState(dwPortState, NIPI_MFE4_PORT_POSITON , &boardPortState.MFE4PortState);
    return boardPortState;
}
```

```
#define NIPI_MFE1_PORT_POSITON     0x00000001
#define NIPI_MFE2_PORT_POSITON     0x00000002
#define NIPI_MFE3_PORT_POSITON     0x00000004
#define NIPI_MFE4_PORT_POSITON     0x00000008
```

```
#define MEUIM_MFE1_PORT_POSITON       0x00000001
#define MEUIM_MFE2_PORT_POSITON       0x00000002
#define MEUIM_MFE3_PORT_POSITON       0x00000004
#define MEUIM_MFE4_PORT_POSITON       0x00000008
#define MEUIM_SFE_PORT_POSITON        0x00000010
#define MEUIM_MGE_PORT_POSITON        0x00000020
```

```
#define MT64NIC_SFE_PORT_POSITON      0x00000001
#define MT64NIC_MGE_PORT_POSITON      0x00000002
#define MT64NIC_MFE1_PORT_POSITON     0x00000004
#define MT64NIC_MFE2_PORT_POSITON     0x00000008
```

```
WORD32  R04_IsMasterPriorSlave(WORD32 dwMasterPortSt,
                               WORD32 dwSlavePortSt)
{
  WORD32  dwMstPortUpWt = 0;
  WORD32  dwSlvPortUpWt = 0;
  T_PortState_Struct masterPortSt = {0 ,0,0,0,0,0,};
  T_PortState_Struct slavePortSt = {0 ,0,0,0,0,0,};

  masterPortSt = AdaptPortState(dwMasterPortSt);
  slavePortSt = AdaptPortState(dwSlavePortSt);
  ProcSlvPortStByMoniPara(&slavePortSt);
  dwMstPortUpWt = CalPortStUpWeight(masterPortSt);
  dwSlvPortUpWt = CalPortStUpWeight(slavePortSt);

  if(dwMstPortUpWt >=dwSlvPortUpWt)
    return TRUE;
  else
    return FALSE;
}
```

```
T_PortState_Struct AdaptPortState(WORD32 dwPortSt)
{
  WORD32 portType = PORT_TYPE_UNKNOW;
  T_PortState_Struct boardPortState = {0 ,0,0,0,0,0,};
    portType = R04getPortType();
    switch (portType)
    {
    case PORT_TYPE_GEB:
      boardPortState = AdaptGEBPortState(dwPortSt);
      break;
    case PORT_TYPE_NIPI:
      boardPortState = AdaptNIPIPortSt(dwPortSt);
      break;
    case PORT_TYPE_SIPI:
      boardPortState = AdaptSIPIPortSt(dwPortSt);
      break;
    case PORT_TYPE_MEUIM:
      boardPortState = AdaptMEUIMPortSt(dwPortSt);
      break;
    case PORT_TYPE_MT64_MNIC:
      boardPortState = AdaptMT64NicPortSt(dwPortSt);
      break;
    default:
      printf("Invalid portType  %d \n",portType);
      break;
    }
    return boardPortState;
}
```

坏味道的“直觉”

- 辗转反侧，却不得要领

债台高筑

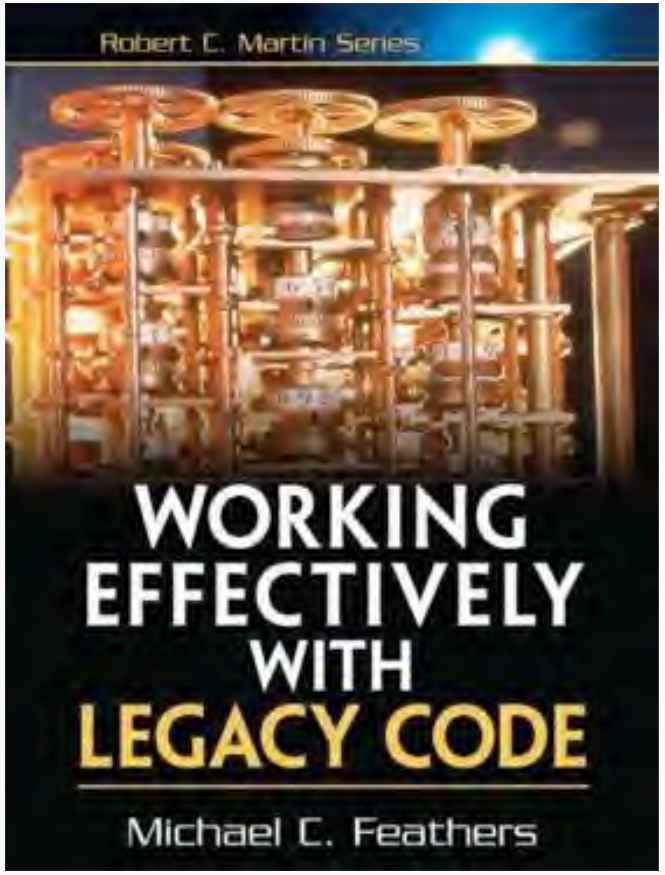
“走自己的路，让别人无路可走”

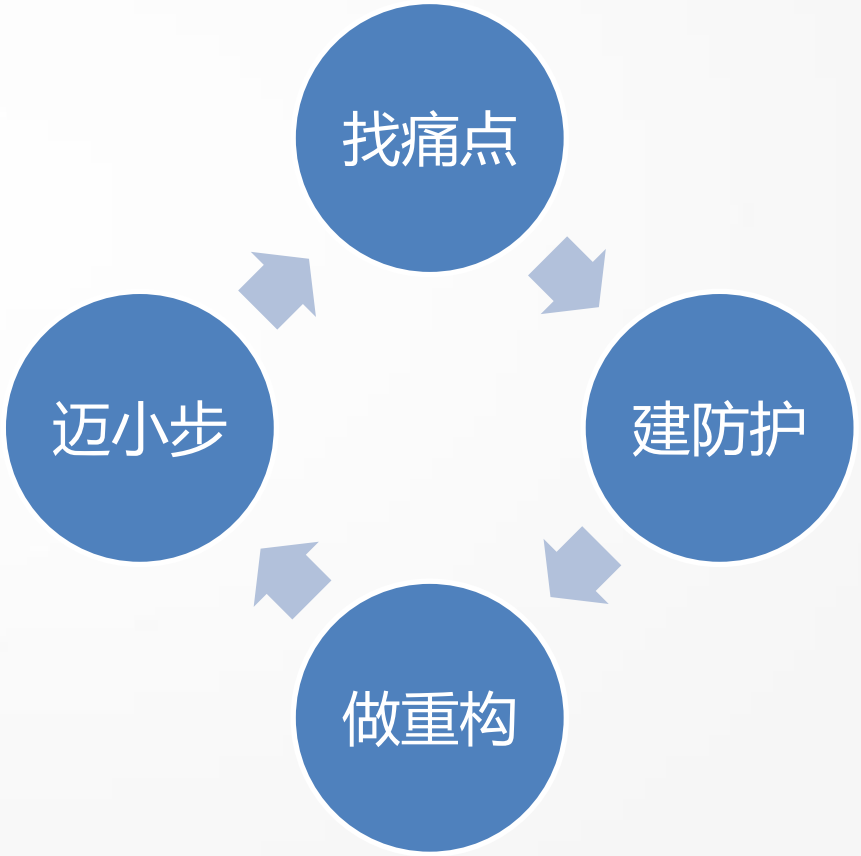
——佚名

02

走出泥潭

遗留代码改进的一般策略





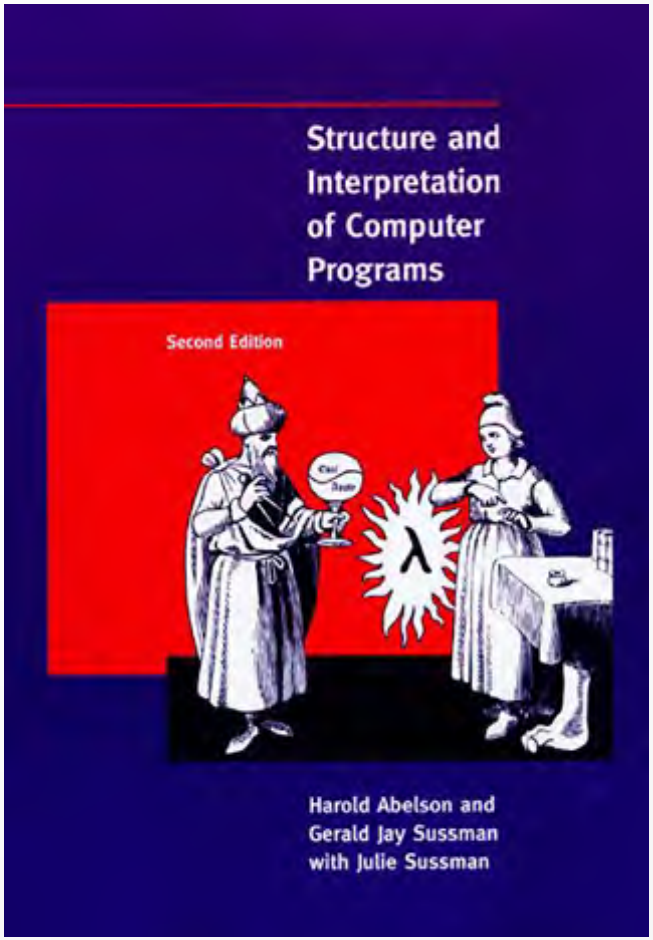
改善设计的更高目标

- 寻找业务模型，构建合适的抽象

一本好书

计算机程序的构造与解释

SICP



内容简介

1 构造过程抽象

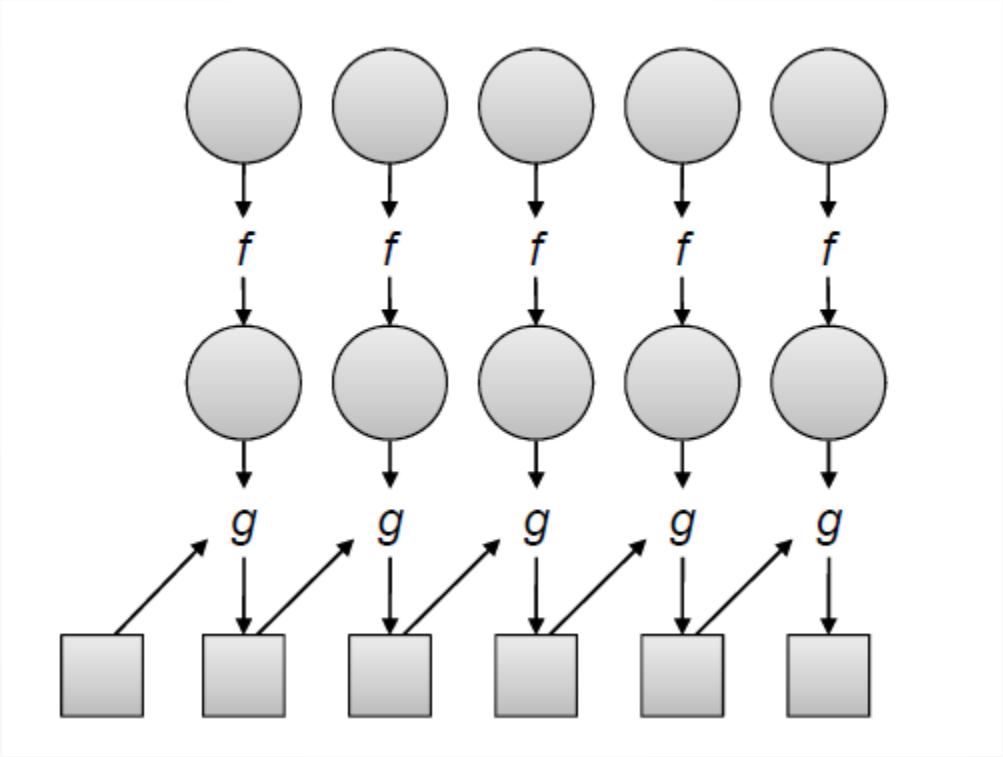
2 构造数据抽象

3 模块化、对象和状态

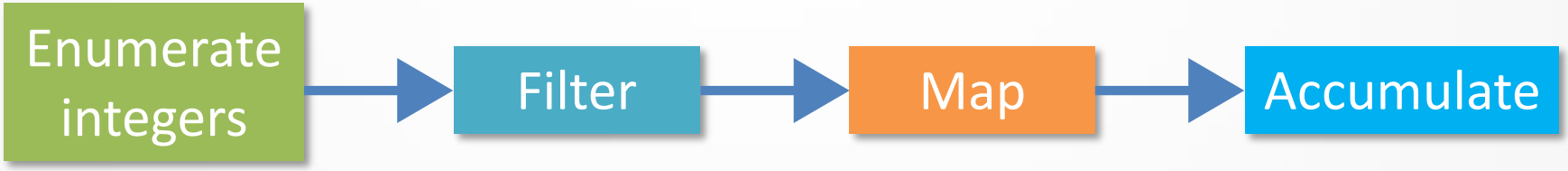
构造过程抽象

- 将待解决的问题分解为一个个小的子问题，每一个子问题分别由一个独立的模块、函数、类等来完成。
- 分解出来的模块、函数或者类能够完成一个相对独立的功能，并且能够在其它的系统或模块中被重复引用

示例: map&reduce模型



示例:“信号流”模式



过程抽象的收益

- 不同层次的过程在实现上相互独立
- 系统在其中某一个部分的实现被替换的情况下，不需要修改设计仍然能正常工作
- 通过**组合**的方式，构建出功能更加强大和复杂的系统

示例:

```
WORD32  R04_IsMasterPriorSlave_ref(WORD32 dwMasterPortSt,WORD32 dwSlavePortSt)
{
    unsigned int weight_master_ports =
            weight_of_board_ports(
                    make_board_port_state( board_port_def_map[ R04getPortType() ].def(),
                                           dwMasterPortSt));


    unsigned int weight_of_slave_ports =
            weight_of_board_ports(
                    adjust_port_state_by_monitor_data(
                            make_board_port_state( board_port_def_map[ R04getPortType() ].def(),
                                                   dwSlavePortSt)));

    return (weight_master_ports >= weight_of_slave_ports) ? TRUE : FALSE ;
}
```

构造数据抽象

- 将复杂数据结构的**使用**和它的**构造**分离开来，通过使用“抽象数据”的方式，具体数据结构的用户可以通过明确定义的一系列接口对其进行访问和操作。
- 数据的构造——*constructor*
- 数据的使用——*selector*

数据抽象的收益

- 分离数据对象的表示以及数据对象的使用，隐藏数据对象的内部特征，对于外部环境而言是透明的。
- 在定义具体的数据表示时，不用关心该数据被使用的方式，两者相互独立。
- 基于*constructors*和*selectors* ，构造出一套新的适用于此领域的新“语言” ，提升业务代码的概念层次。

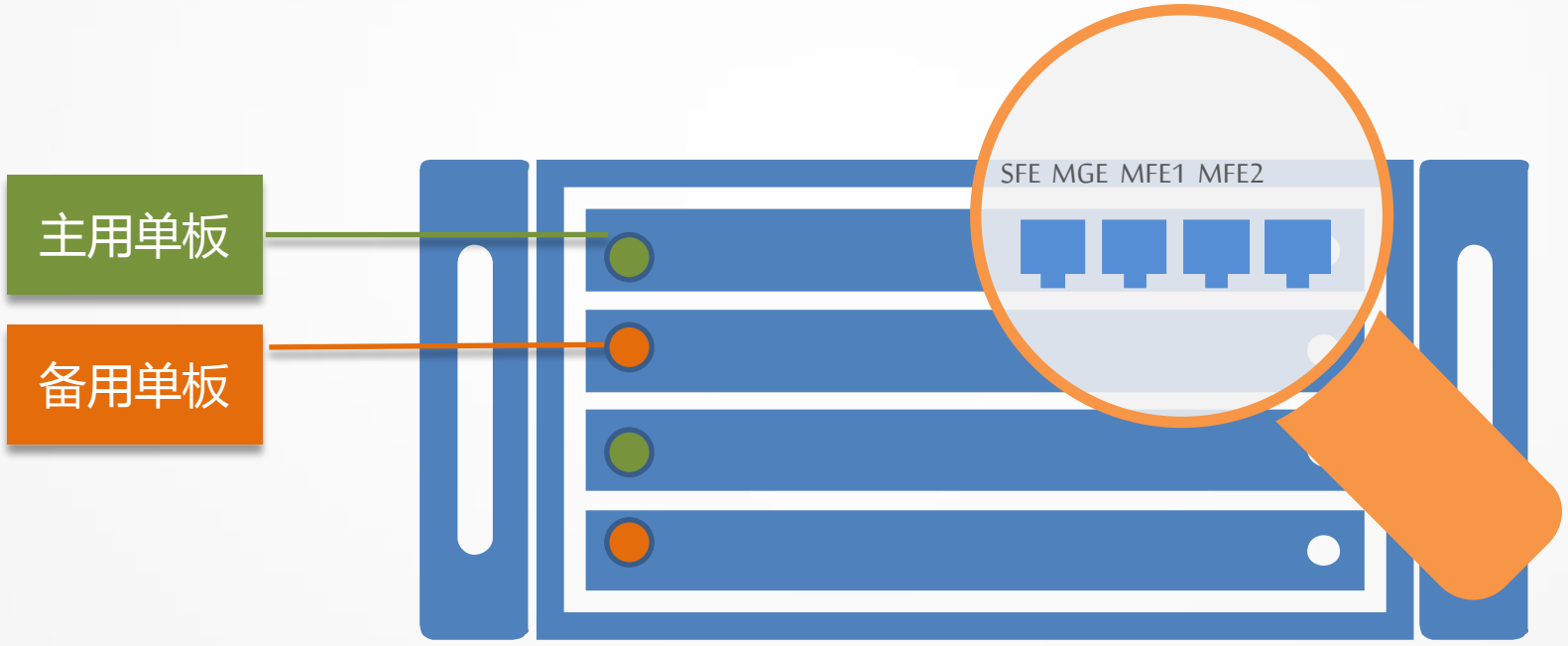
03

案例分析

需求分析

- 系统控制功能是电信设备中保证系统可用性的重要功能，对系统中各种类型的单板提供主备保护。
- 系统控制功能根据端口的实时状态，判断是否需要在主备单板之间进行切换。
- 不同类型的端口的重要性不同（信令口高于媒体口）

通过主备实现高可用



现有业务流程

1. 根据单板的端口定义，将系统上报的实时状态转换为通用的端口状态对象。
2. 根据通用的端口状态对象，计算主用和备用两个单板的端口加权值。
3. 根据主备单板的端口加权值，判断是否需要在主备之间进行切换。

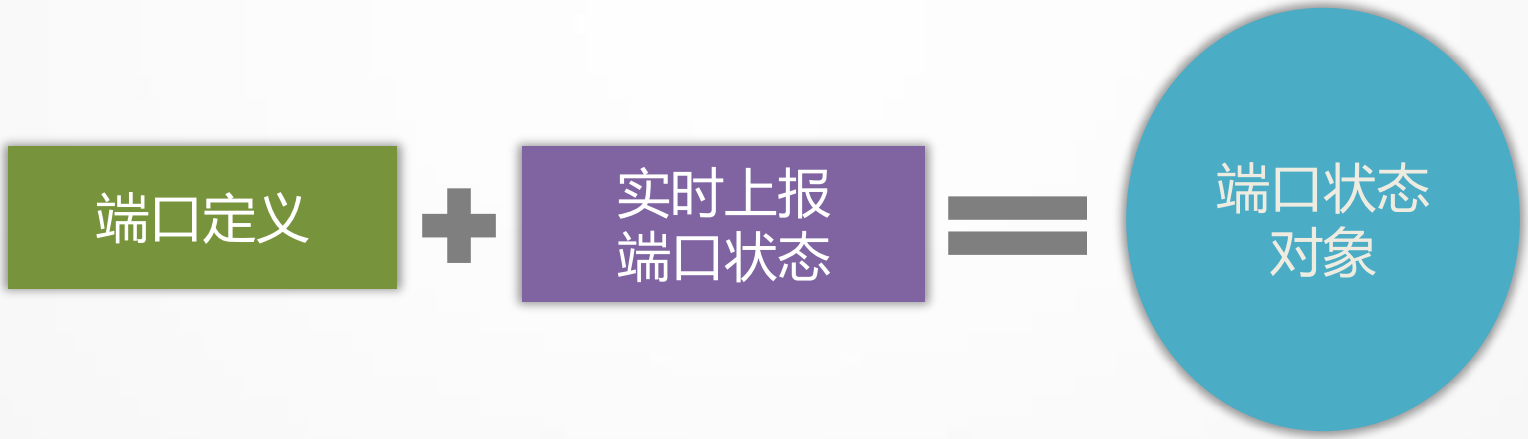
多问自己几遍

这个需求究竟要干什么？

挖掘业务本质

- 在这个问题中，最核心的元素是**端口的定义**和实际上报的**端口状态值**。
- 其它所有的计算和判断都是围绕此数据对象进行。

构造数据抽象



单板端口配置定义

```
struct board_port_entity
{
    unsigned int  pos;
    unsigned char name[8];
};
struct board_port_definition
{
    struct board_port_entity ports[8];
};
```

一个实例

```
static struct board_port_definition
nipi_board_ports_def =
{
    {
        /*  pos     name    */
            {1,     "MFE1"},
            {2,     "MFE2"},
            {3,     "MFE3"},
            {4,     "MFE4"},
    }
};
```

生成更多实例

```
static struct board_port_definition
geb_board_ports_def =
{
    {
        /*  pos     name    */
            {1,     "MGE"},
    }
};
```

```
    {
        /*  pos     name    */
            {1,     "SFE"},
    }
};
```

```
                pos     name    */
                {1,     "MFE1"},
                {2,     "MFE2"},
                {3,     "MFE3"},
                {4,     "MFE4"},
                {5,     "SFE"},
                {6,     "MGE"},
    }
};
```

单板端口状态对象

```
typedef unsigned int board_origin_status;
struct board_port_status
{
    struct board_port_definition def;
    board_origin_status status;
};
```

定义constructor

```
struct board_port_status
make_board_port_state(struct board_port_definition def,
                      board_origin_status orgin)
{
    struct board_port_status st;
    st.def = def;
    st.status = orgin;

    return st;
}
```

定义selectors

- *get_board_port_status*，获取某一个具体端口的状态。
  *get_board_port_status(st, “FE1”)*获取FE1端口的状态；

- *set_board_port_status*，设置某一个具体端口的状态。
  set_board_port_status(st, “FE1”，PORT_STATE_INSERVICE) 设置FE1端口的状态为“激活”；

- *weight_of_spec_port*，获取某一个端口的权重值。
  weight_of_spec_port(“SFE”) 获取此单板信令口的权重。

基于selectors构建高层接口

```
unsigned int weight_of_board_ports(struct board_port_status st)
{
    unsigned int i;
    unsigned int weight = 0;
    for(i = 0; i< sizeof(st.def.ports)/sizeof(struct board_port_entity); i++)
    {
        if(get_board_port_status(st, (const char*)st.def.ports[i].name)
            == PORT_STATE_OUTSERVICE)
            continue;

        weight += weight_of_spec_port((const char*)st.def.ports[i].name);
    }
    return weight;
}
```

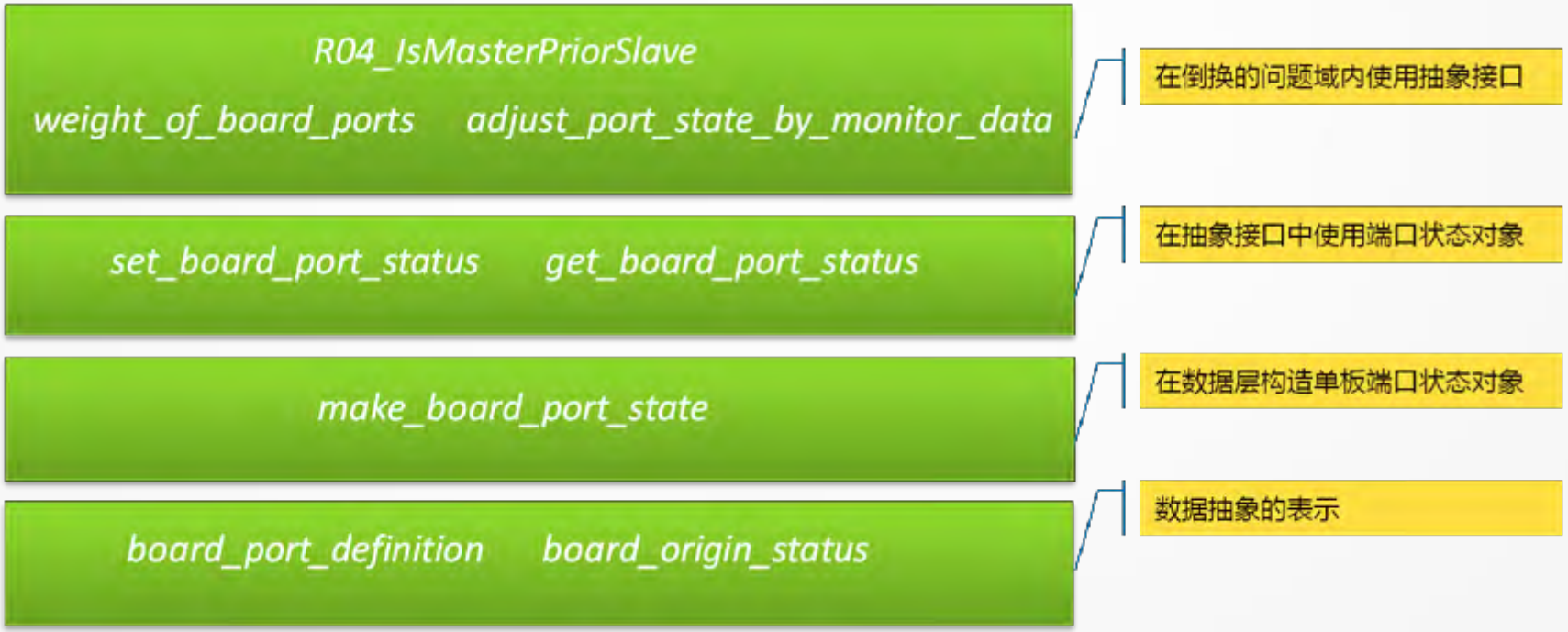
组合

```
WORD32  R04_IsMasterPriorSlave_ref(WORD32 dwMasterPortSt,WORD32 dwSlavePortSt)
{
    unsigned int weight_master_ports =
            weight_of_board_ports(
                    make_board_port_state( board_port_def_map[ R04getPortType() ].def(),
                                           dwMasterPortSt));


    unsigned int weight_of_slave_ports =
            weight_of_board_ports(
                    adjust_port_state_by_monitor_data(
                            make_board_port_state( board_port_def_map[ R04getPortType() ].def(),
                                                   dwSlavePortSt)));

    return (weight_master_ports >= weight_of_slave_ports) ? TRUE : FALSE ;
}
```

业务层次


R04_IsMasterPriorSlave
weight_of_board_ports
adjust_port_state_by_monitor_data
在倒换的问题域内使用抽象接口
set_board_port_status
get_board_port_status
在抽象接口中使用端口状态对象
make_board_port_state
在数据层构造单板端口状态对象
board_port_definition
board_origin_status
数据抽象的表示

设计评估

- 知识点集中
- 语义更清晰，体现业务规则
- 易于扩展

04

回顾与总结

发现领域模型

- 在这个问题中，最核心的应该是处理板的端口状态对象。
- 基于端口状态对象进行设计和实现。

业务需求

端口配置规范

端口对象的实例

如何评价一个设计的“简单性”？

追求简单性

- 不只是各种技巧、原则和模式的堆积。
- 从业务问题出发，寻找出适合于业务领域的模型，让模型与问题相匹配。
- 增加程序的模块化特性，使得程序具有很好的可增长型（$additivity$）

简单性的“直觉”

契合问题领域的设计，会是一个好的设计！

逐步偿还技术债务

走自己的路，让后来的人有路可走！

05

简书：zhizhuwang