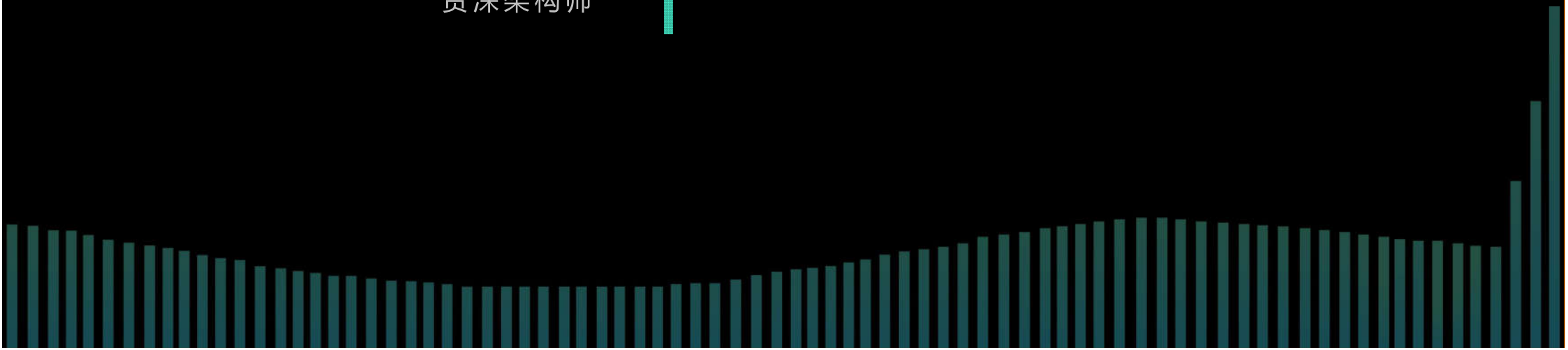
C++ Summit 2020

连少华
资深架构师

Modern C++ 整洁代码最佳实践



内容概览

架构设计基础

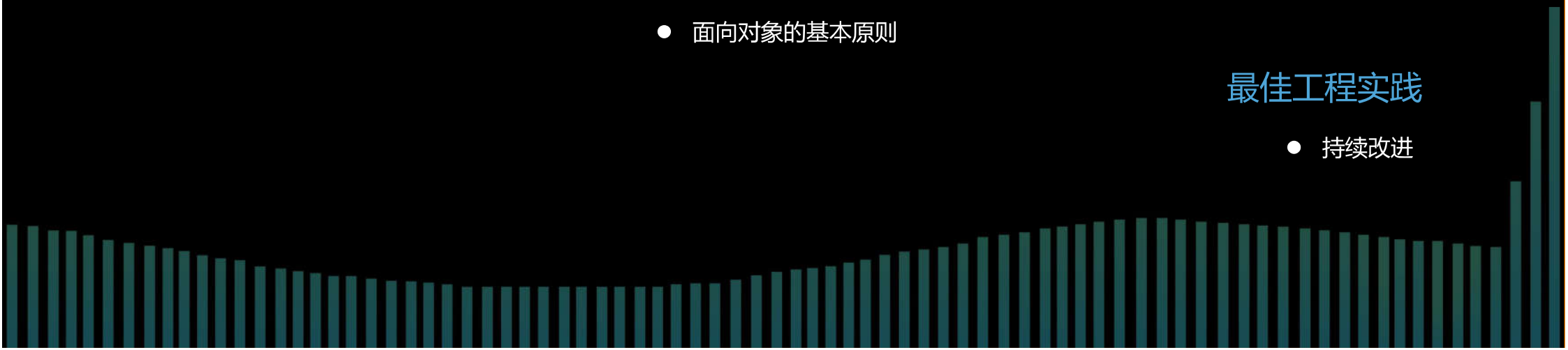
- 架构的基本概念
- 常见架构分类
- 架构方法论

Modern C++整洁之道

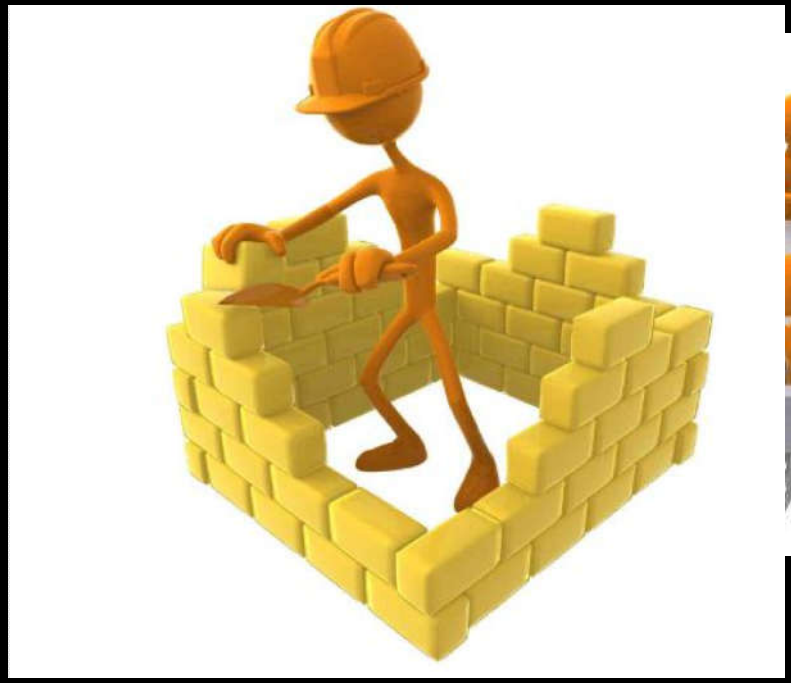
- 整洁的基本原则
- 整洁的基本规范
- Modern C++
- 面向对象的基本原则

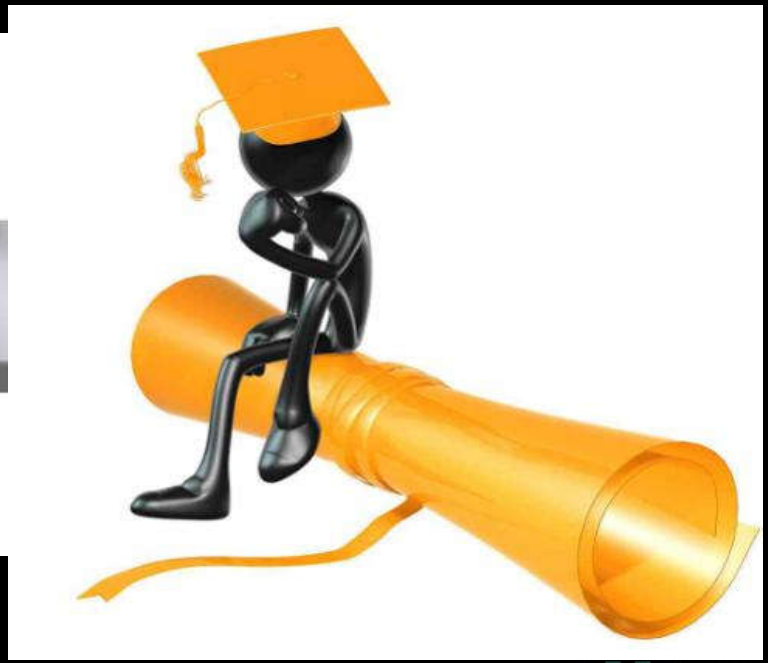
最佳工程实践

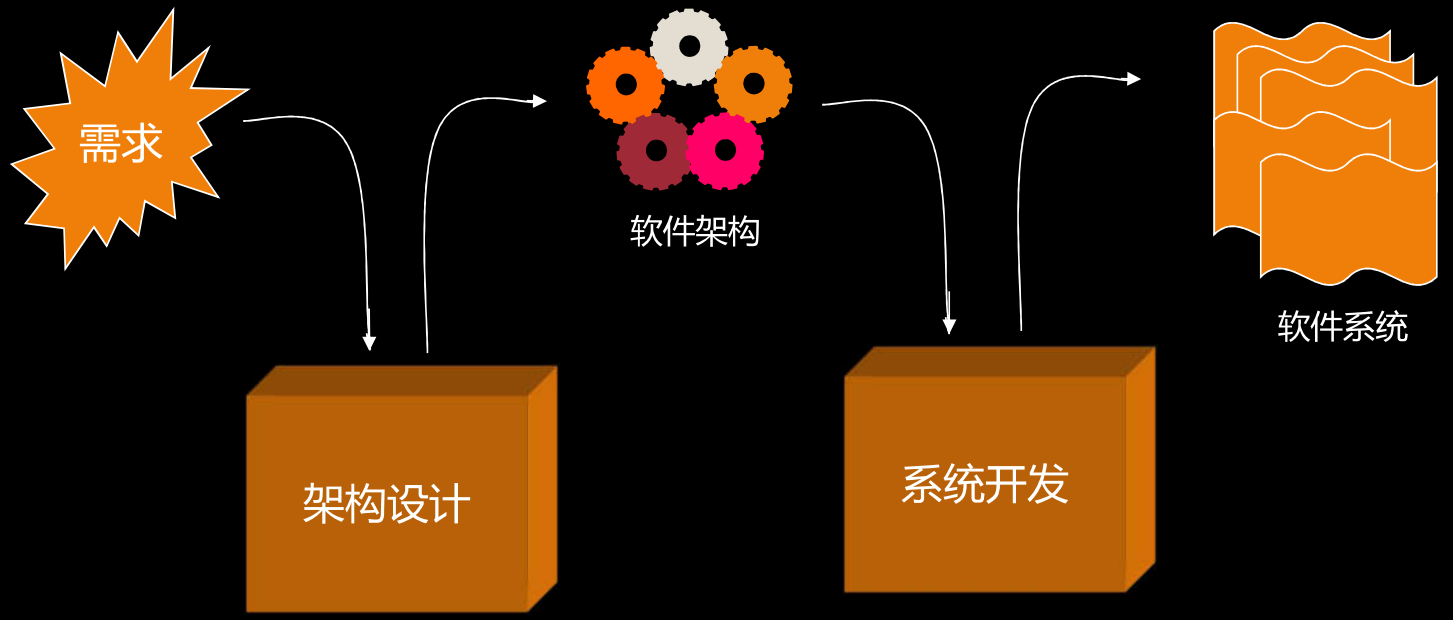
- 持续改进



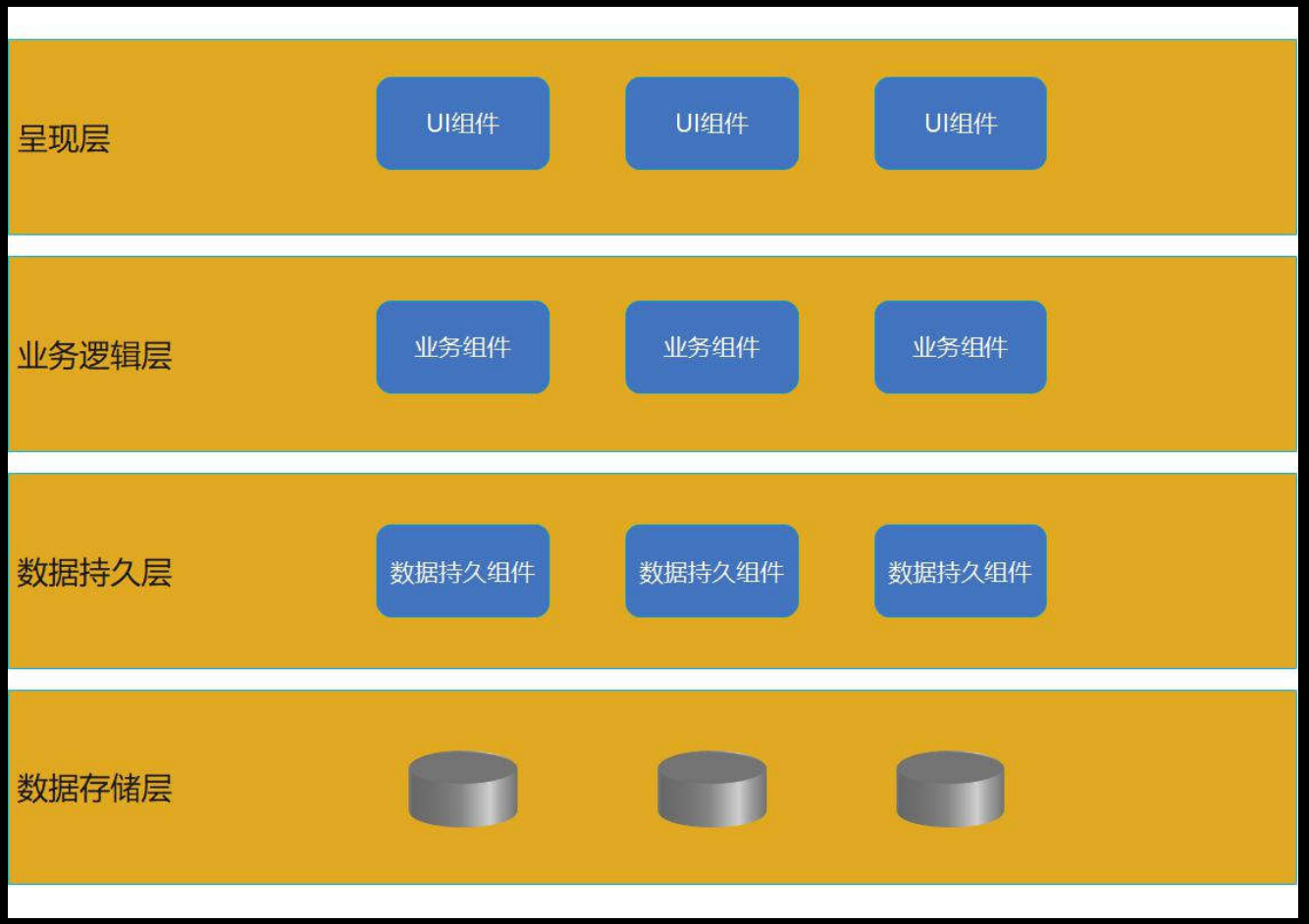
- 在大街上，问一百个人可能有一百零一种说法
- 目前对软件架构的理解大致可分为剑宗和气宗两大门派





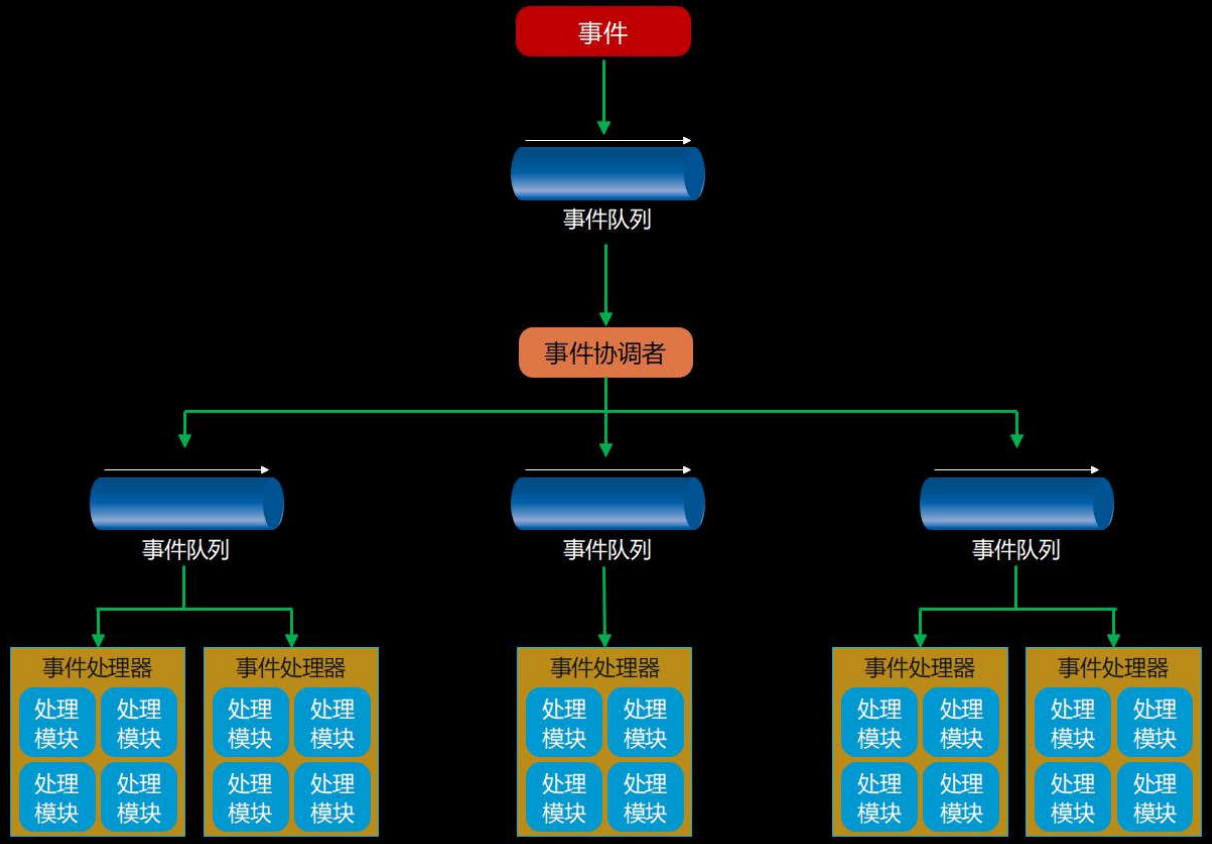

需求
软件架构
软件系统
架构设计
系统开发

I、分层架构



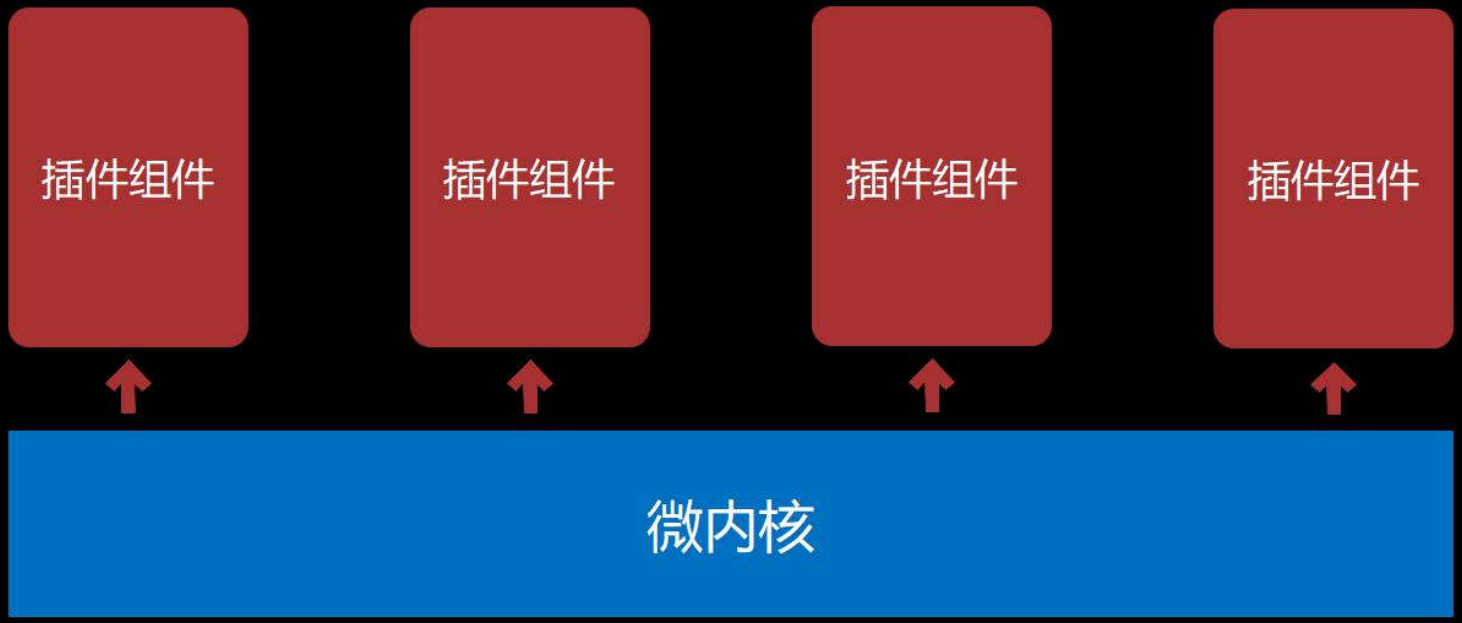
MVC
MVVM
MVP
SSH
Blockchain

II、事件驱动架构



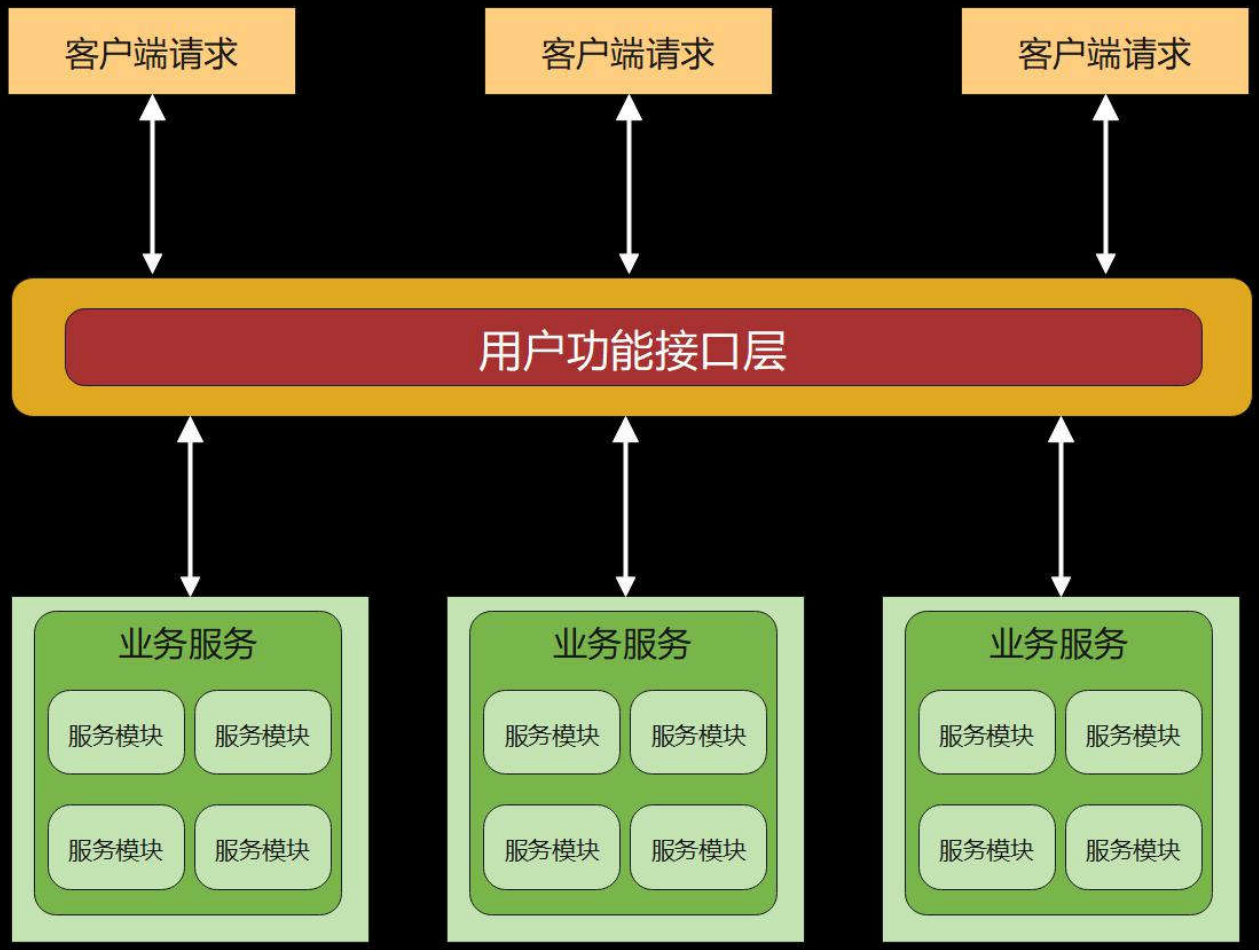
Node.js
Flink

III、微内核架构

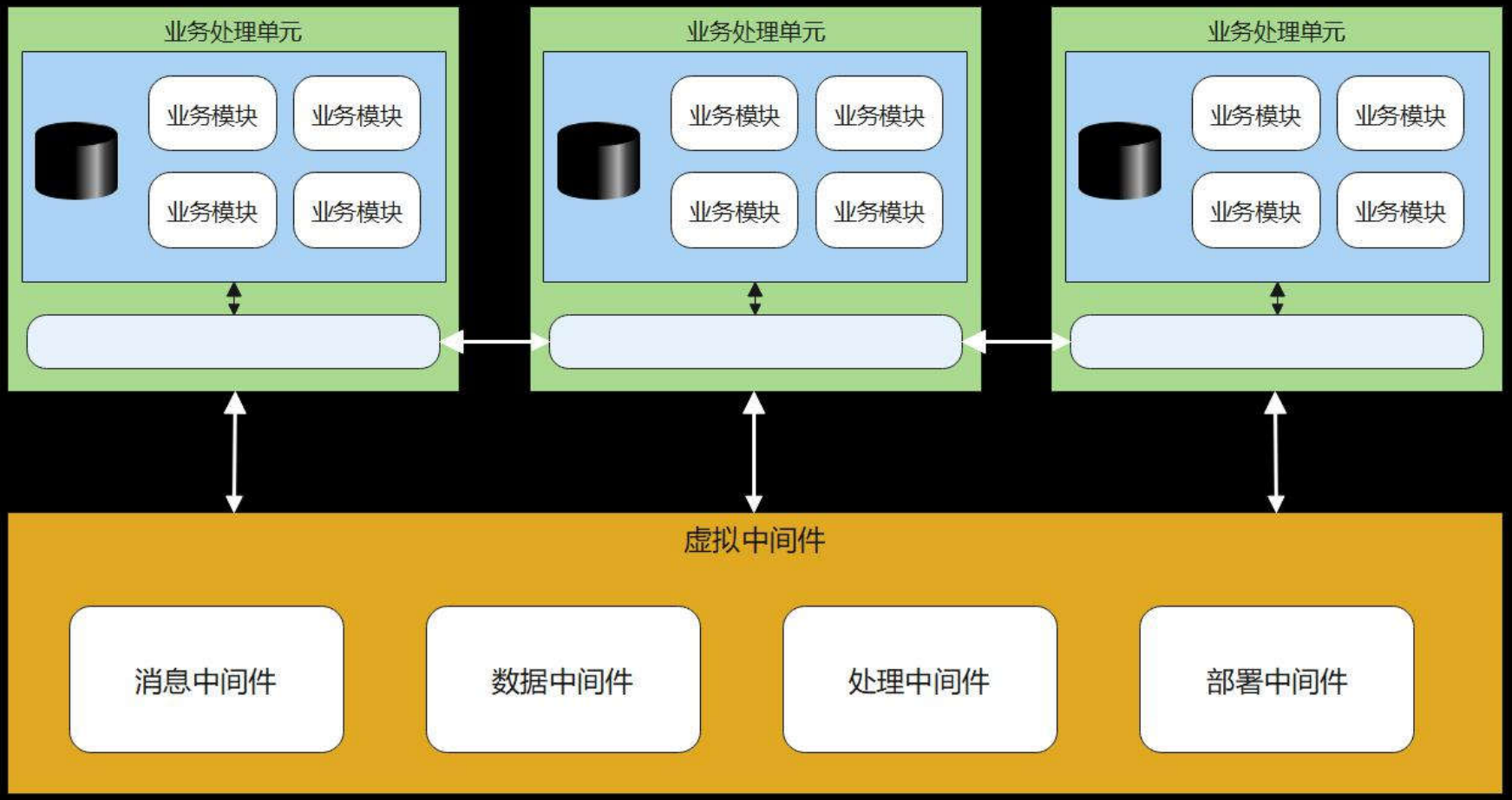


Eclipse
Vim

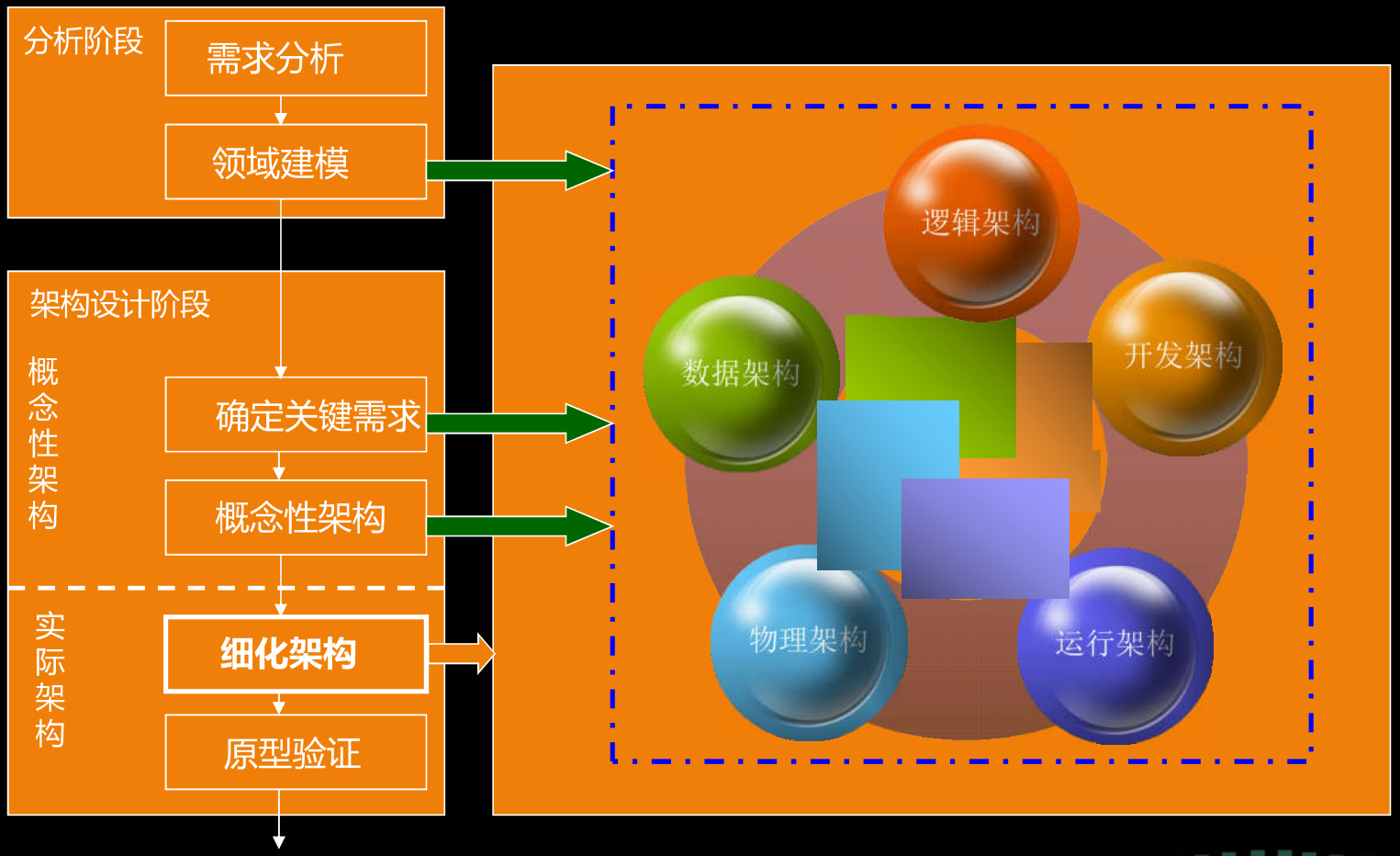
IV、微服务架构


客户端请求
客户端请求
客户端请求
用户功能接口层
业务服务
服务模块
服务模块
服务模块
服务模块
业务服务
服务模块
服务模块
服务模块
服务模块
业务服务
服务模块
服务模块
服务模块
服务模块

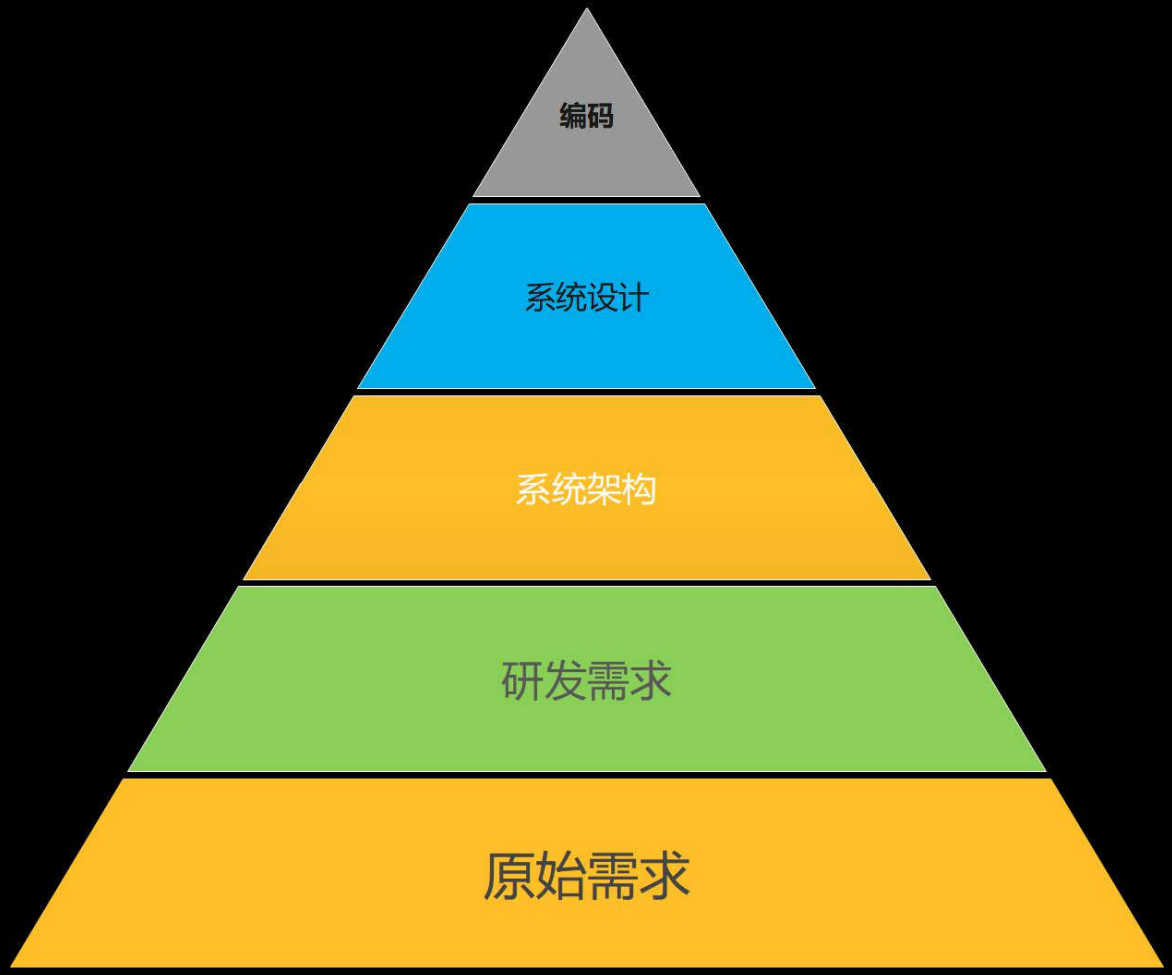
V、云架构


业务处理单元
业务模块
业务模块
业务模块
业务模块
业务处理单元
业务模块
业务模块
业务模块
业务模块
业务处理单元
业务模块
业务模块
业务模块
业务模块
虚拟中间件
消息中间件
数据中间件
处理中间件
部署中间件

架构设计基础——架构方法论：软件架构的基本过程




全面认识需求
识别关键需求
1
2
3
4
多视图探寻
尽早验证


编码
系统设计
系统架构
研发需求
原始需求

✓ 保持简单和直接原则（KISS)

KISS (Keep It Simple，Stupid)原则是指代码的设计和实现越简单越好，在满足系统需求的前提下，任何没有必要的复杂都是需要避免的。因为人们喜欢简单、容易学习和易于使用的事物，同时，简单也可以缩短交付时间，降低公司成本。

大道至简。把简单的事情复杂化是没事找事情，把复杂的事情简单化才是一种能力。

对于程序员来讲，关注简单和保持简单可能是最困难的事情之一，因为程序员是最聪明的一群人，他们每天都在做着“改变世界”的事情。

✓ 不需要原则(YAGNI)

不需要（You Are NOT Gonna Need It)指出千万不要进行过度设计,也就是说，不要写目前用不上，将来也许用得上的代码，否则，就破坏了KISS原则了。在日常讨论中，我们常常会这样“以后也许会用到这个功能...”，对于“将来也许”的功能，建议在真正有必要的时候再写代码，因为将来的事情...

✓ 避免复制原则（DRY)

避免复制原则（Do not Repeat Yourself)指出尽量在项目中减少重复的代码行、重复的方法和重复的模块，这就意味着项目中的每个事物都应该是唯一的。其实很多的原则、思想、模式和框架最本质的思想都是在消除重复。

任何时候都尽量避免Ctrl+C、Ctrl+V，因为重复意味着臃肿。可以想象一下，当修改一段代码时，也必须相应的修改另一段重复的代码，事实是，往往会漏改或忘改那段重复的代码。

✓ 信息隐藏原则

信息隐藏原则（Information Hiding）指出，当一个模块（一段代码）调用另一个模块（另一段代码）时，调用者不应该知道被调用者的内部实现。这就要求被调用者把自己的实现“隐藏”起来，只提供必要的接口，所以信息隐藏是系统模块化的基本原则。

信息隐藏有很多的优点，如：限制了模块变化的范围；显著提高了模块的复用性；模块具备更好的可测试性；当修复缺陷时，对依赖模块基本没有影响。

✓ 高内聚原则

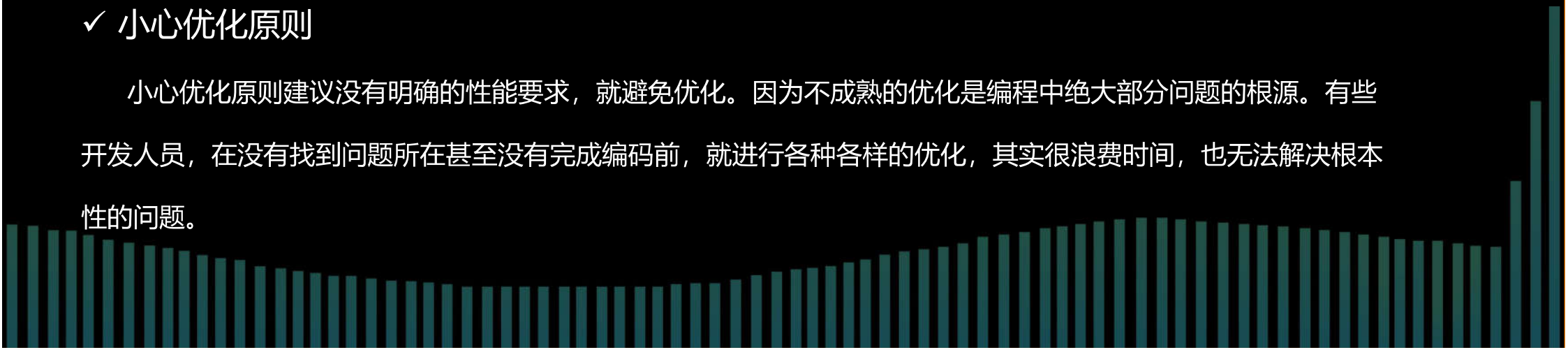
内聚是衡量内部间聚集和关联的程度，高的意思是内部间的关系要简单明了，不要牵强附会。高内聚是信息隐藏原则的扩展，是从功能角度进行度量的。内聚可分为偶然内聚、逻辑内聚、时间内聚、过程内聚、通信内聚、顺序内聚、功能内聚，功能内聚是最强的内聚。

✓ 松耦合原则

松耦合是衡量模块间相互联系的紧密程度。耦合的强弱与模块间接口的复杂性、调用方式和传输数据有直接关系。耦合可分为非直接耦合、数据耦合、标记耦合、控制耦合、外部耦合、公共耦合、内容耦合。

✓ 小心优化原则

小心优化原则建议没有明确的性能要求，就避免优化。因为不成熟的优化是编程中绝大部分问题的根源。有些开发人员，在没有找到问题所在甚至没有完成编码前，就进行各种各样的优化，其实很浪费时间，也无法解决根本性的问题。



✓ 名称应该自解释

应望文知意，长度适中，使用简单，能自我解释和描述。如：flag、list、data、vec、num、Info等都是不太好的名字，像isRunning、orderInfo、orderList、productNumber等是比较好的命名。

示例：

```
unsigned int num;
bool flag;
std::vector<Book> bookVector;
std::string info;
unsigned int totalPriceOfCustomerBooksToday;
```

VS

++11（或更高

```
unsigned int numberOfBook;
bool isRunning;
std::vector<Book> books;
std::string orderInfo;
unsigned int priceOfBooks;
```

✓ 使用域中的名称

如果有领域驱动设计（Domain-Driven Design，DDD)环节，那么在写代码之前，建议先仔细阅读DDD中的命名，DDD中的术语、名称比较专业，同时命名也保证了统一。

✓ 避免冗余的名称

类（或模块）中的数据成员或函数成员名称尽量不要带有类（或模块）的名称，也不要包含类型的名称。

示例：

```
enum class PersonSex:int
{
    Male=0,
    Female=1
};

class Person:public std::enable_shared_from_this<Person>
{
public:
    string getPersonName() const;
    int setPersonName(string name);

    int getPersonAge() const;
    int setPersonAge(int age) const;

    PersonSex getPersonSex() const;
    int setPersonSex(PersonSex newSex) const;

private:
    string personName;
    int personAge;
    PersonSex personSex;
};
```

VS

```
enum class Sex:int
{
    Male=0,
    Female=1
};

class Person:public std::enable_shared_from_this<Person>
{
public:
    string getName() const;
    int setName(string newName);

    int getAge() const;
    int setAge(int newAge);

    Sex getSex() const;
    int setSex(Sex newSex);

private:
    string name;
    int age;
    Sex sex;
};
```

✓ 避免晦涩难懂的缩写

在单词不是很长的时候尽量不要使用自定义缩写，一是有歧义，二是记忆和学习的成本太高了，如：idx、bmw、rfg等；但是可以使用一些与行业相关的“知名缩写”，如：MD、TRD、TGW、UDP、TCP等

示例：

```
std::size_t idx;
Car ctw;
unsigned int nBottles;
bool rfg;
```

VS

```
std::size_t index;
Car carToWash;
unsigned int bottleAmount;
bool runFlag;

unsigned int trdAmount;
unsigned int tgwStatus;
```

✓ 避免相同的名称用于不同的目的

就像一个公司中有两个同名的人一样，看到名字时不知道这个名字到底指的是谁。

✓ 避免匈牙利命名和命名前缀

尽量不要把类型信息加入到变量的名称中，因为经常会出现变量类型与名称表示的类型不致的情况——类型前缀不可信，如：pszName、strInfo、iSize、fFlag等。现在很多高级的IDE都支持了知能提示，所以尽早的抛弃匈牙利命名法吧。

示例：

```
bool fEnable;
unsigned long long ullContainerSize;
int iIndex;
char * pszTitle;
vector<Book> vecBook;
double dLastPrice;
float fPrice;
char * gsc_pszPrefixString;
```

VS

```
bool isEnable;
unsigned long long containerSize;
int index;
char * title;
vector<Book> books;
double lastPrice;
float price;
char * prefixString;
```

✓ 代码应该自注释

请理解一句话——真相只能在代码中找到。所以，能不写注释就不要写注释，因为事实已经证明了注释往往与代码不一致，这个时候，我们应该相信代码还是应该相信注释呢？

✓ 不要为易懂的代码写注释

```
/*
 * @brief proxy_manager
 *     It manages xpub_xsub_proxy and router_proxy instances
 *     It can not be inherited
 */
class proxy_manager final
{
    private:
        //Define proxy_vector type
        using proxy_vector=std::vector<proxyIPtr>;
    public:
        //...
        //Initialize proxy_manager instance
        error_code Initialize() noexcept;
        //Start to run proxy_manager instance
        error_code Run() noexcept;
        //Release all resources allocated in Initialize function
        error_code Uninitialize() noexcept;

    private:
        //Initialize flag
        std::atomic_bool isInitialized=false;
        //Running flag
        std::atomic_bool isRunning=false;
        //The Vector with proxy instance
        proxy_vector proxies;
};
```

VS

```
class proxy_manager final
{
    private:
        using proxy_vector=std::vector<proxyIPtr>;

    public:
        //...

        error_code Initialize() noexcept;
        error_code Run() noexcept;
        error_code Uninitialize() noexcept;

    private:
        std::atomic_bool isInitialized=false;
        std::atomic_bool isRunning=false;
        proxy_vector proxies;
};
```

✓ 不要通过注释禁用代码

被禁用的代码增加了代码的混乱程度，却没有带来任何的好处，提交到版本管理系统的代码“永久”不会丢失。

✓ 不要写块注释

文件头的创建、更改记录、版权声明、版本控制、大段大段的//或/*...*/的注释。

```
/* Copyright (C)
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 *
 */
//############################################################################
//Change log:
//2006-06-14 (dc cheng) fix bug #11086
//2006-06-14 (nn xu) fix bug #11072
//2006-05-01 (y yu) create file
//############################################################################

/**
 * @file memory_pool.cpp
 * @brief memory pool
 * @author shlian
 * @version 1.0
 * @date 2020-09-28
 */
```

```
/**
 * @brief MQ proxy
 */
class mq_proxy final
{
public:
    //--------------------
    //public interface
    //--------------------

    //.....

protected:
    //------------------------------
    //override or protected methods
    //------------------------------

    //.....

private:
    //-------------------------
    //private member functions
    //-------------------------

    //......

    //-------------------------
    //private data members
    //-------------------------

    //.....
}
```

Modern C++整洁之道——整洁的基本规范：合理的注释

✓ 有些注释是必要的

从源代码生成帮助文档时，可以适当的写一些简要的注释，如经常使用的Doxygen工具提供的几种注释风格。

```
/**
 * @brief proxy interface
 */
class mq_proxy
{
public:
   //...
   /**
    * @brief Initialize the proxy,allocate resources needed
    *
    * @param thread_number the number of thread to create
    *
    * @return ok:successful,other:error
    */
   error_code Initialize(int thread_number) noexcept;

   /**
    * @brief Start the proxy
    *
    * @return ok:successfull,other:error
    */
   error_code Start() noexcept;

   /**
    * @brief Stop the proxy
    *
    * @return  ok:successfull,other:error
    */
   error_code Stop() noexcept;

   /**
    * @brief Wait the proxy to complete all jobs
    *
    * @return ok:successfull,other:error
    */
   error_code Wait() noexcept;

   /**
    * @brief Uninitialize the proxy to releaee resources
    *
    * @return ok:successfull,other:error
    */
   error_code Uninitialize() noexcept;
};
```

✓ 只做一件事情

一个函数应该做一件事情，且应该仅仅做好这一件事情。如果函数体量比较大、没有合适的名字、人为了划分了几个段落、圈复杂度比较度、入参比较多等，都是函数做了太多事情的标志。

✓ 让函数尽可能的小

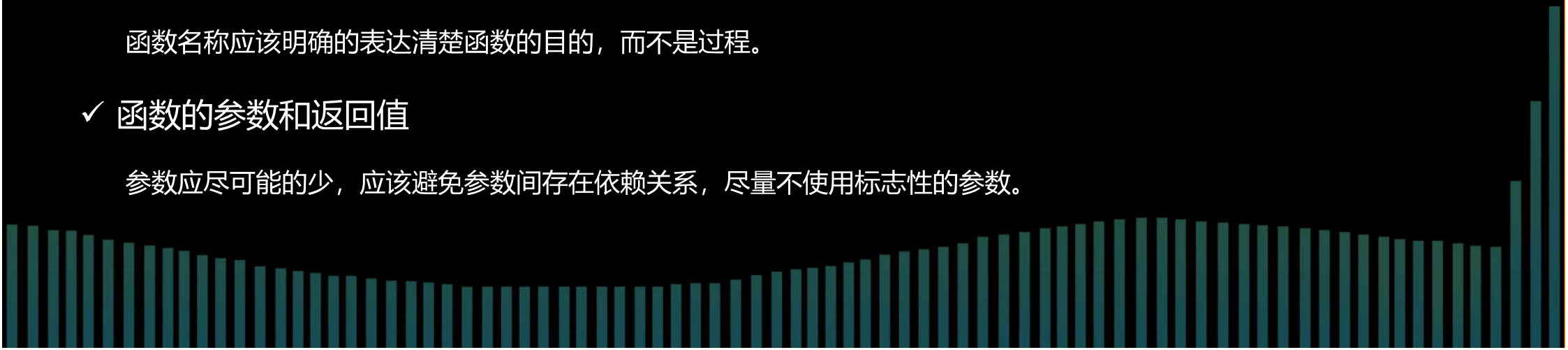
函数体应该尽可能的小（参见第一条）。函数调用开销并不是系统的瓶颈，并且现在的编译器会优化掉函数的调用开销，编译器比你聪明得多。

✓ 使用容易理解的名称

函数名称应该明确的表达清楚函数的目的，而不是过程。

✓ 函数的参数和返回值

参数应尽可能的少，应该避免参数间存在依赖关系，尽量不使用标志性的参数。



- ✓ 尽量使用C++的std::string、std::stream替代C风格的char*

  不安全，且性能上的“提升”不足以掩盖其带来的问题。

- ✓ 避免使用printf、str、mem系列的函数，如：snprintf、strncpy、memcpy等

  不安全，且性能上的“提升”不足以掩盖其带来的问题。

```
const char *prefix="CPP-Summit 2020";
const unsigned int title_length=50+1;
char *title=new char[title_length];

strcpy(title,prefix);
//strncpy(title,prefix,title_length);
//memcpy(title,prefix,title_length);
//memcpy(title,prefix,strlen(prefix)+1);

snprintf(title,title_length,"%s%s",prefix,"shenzhen");
printf("%d\n",title);
```

VS

```
const char *prefix="CPP-Summit 2020";
std::ostringstream oss;
oss<<prefix<<"shenzhen";
string title(oss.str());
cout<<title<<endl;
```

Modern C++整洁之道——整洁的基本规范：避免C风格的代码

- ✓ 使用标准库的容器替代C风格的数组

  std::array比较安全，不会越界，且兼容STL接口，并且还与C风格的数组兼容。

- ✓ 使用C++类型转换代替C风格的强制转换

  尽量避免类型转换！！C++类型转换会在编译期进行检查，而C风格的则不会。

- ✓ 尽量避免使用宏

  尽量避免使用宏，宏定义的函数并不能带来效率上的提升，并且会导致文件体量庞大。

```
std::array<int,20> topNumbers;
topNumbers[0]=100;
topNumbers[19]=-100;
topNumbers[200]=30;

for_each(topNumbers.begin(),topNumbers.end(),[](int element){
        cout<<element<<endl;
        });

int *number=topNumbers.data();
for(size_t i=0;i<topNumbers.max_size();++i)
{
    cout<<number[i]<<endl;
}
```

```
const char *title="CPP-Summit 2020";

int *value1=(int*)title;
char *value2=(char *)title;
float *value3=(float*)title;
void *value4=(void*)title;

int *val1=static_cast<int*>(title);
char *val2=static_cast<char*>(title);
char *val3=const_cast<char*>(title);
void *val4=reinterpret_cast<void*>(title);
```

Modern C++整洁之道——整洁的基本原则：Modern C++

- 获得，析构时释放”
- std::shared_pt
- 栈内存、make_xx
- ”→“零原则”

VS

```
class summit final
{
public:
    summit()
    {
    }

    ~summit();
    summit(const summit &other);
    summit & operator=(const summit &other);

    summit(summit && other);
    summit & operator=(summit &&other);

private:
    string title="CPP-Summit 2020";
    unsigned int beginDate=20201204;
    unsigned int endDate=20201205;
    string address="shenzhen";
    vector<string> titles;
    unsigned int numberOfConferee=2000;
};

void build_summit()
{
    auto summitInstance=std::make_shared<summit>();

    auto  summitInstance2=std::make_unique<summit>();
}
```

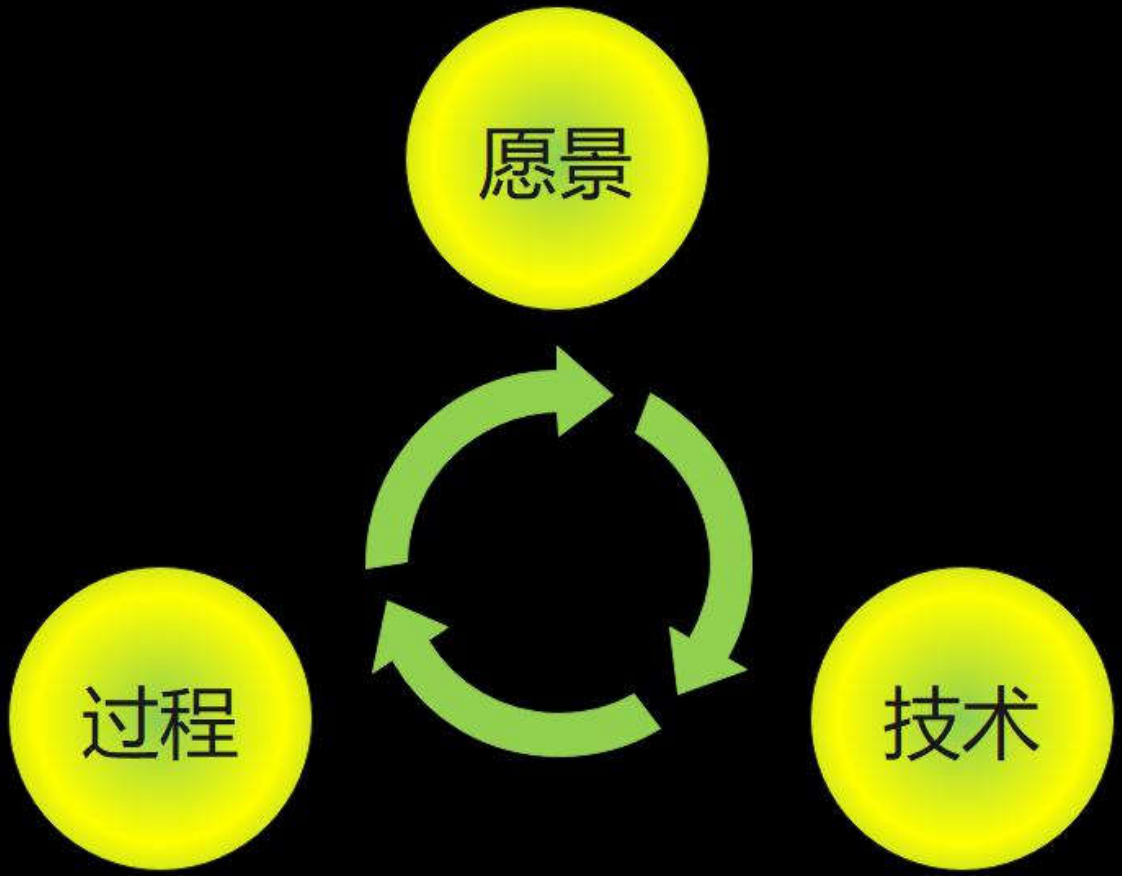
```
class summit final
{
public:
    summit()
    {
    }

private:
    string title="CPP-Summit 2020";
    unsigned int beginDate=20201204;
    unsigned int endDate=20201205;
    string address="shenzhen";
    vector<string> titles;
    unsigned int numberOfConferee=2000;
};

void build_summit()
{
    auto summitInstance=std::make_shared<summit>();

    auto summitInstance2=std::make_unique<summit>();
}
```

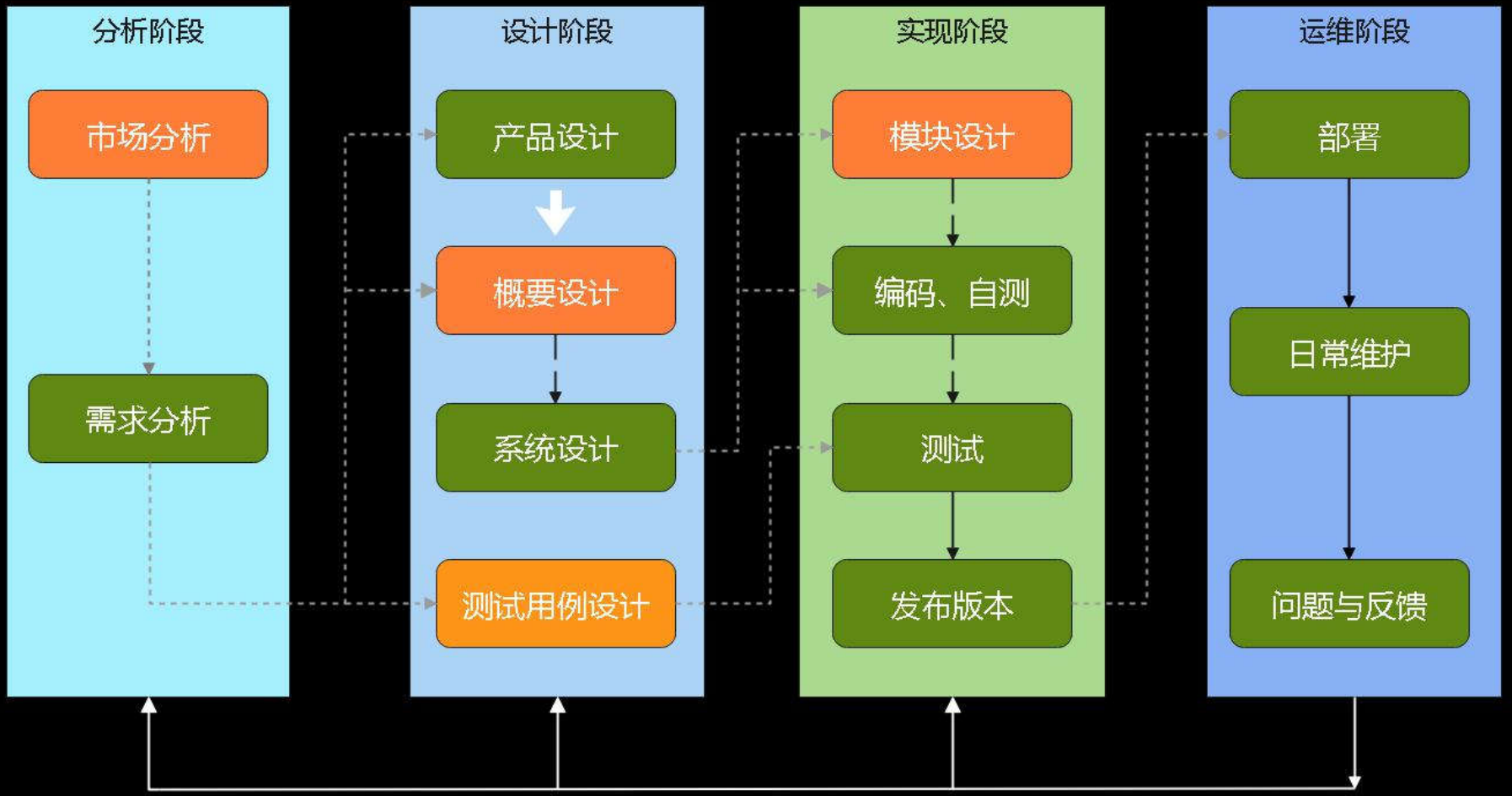
Modern C++整洁之道——整洁的基本原则：Modern C++

- 必要的时候，使用自动类型推导
- 尽可能的
- 熟悉stl
- 尽量使用
- 使用Typ
- 学习、使

```
struct Money{
    long double value;
};

constexpr Money operator"" _CNY(long double value)
{
    return Money{value=value};
}

int main(int argc, char *argv[])
{
    auto moneys={1.0_CNY,10.0_CNY,125.0_CNY,300.0_CNY};
    long double sum=0;

    for_each(moneys.begin(),moneys.end(),[&sum](const auto & money){
            sum+=money.value;
            cout<<money.value<<endl;
            });
    cout<<"sum="<<sum<<endl;
}
```

- 让类尽可能的小，尽量避免出现“工具类”、“万能类”
- 单一职责（Single Responsibility Principle，SRP）
- 开闭原则（Open-Close Principle，OCP）
- 里氏替换原则（Liskov Substitution Principle，LSP）
- 接口隔离原则（Interface Segregation Principle，ISP）
- 无环依赖原则（Acyclic Dependencies Principle，ADP）
- 依赖倒置原则（Dependence Inversion Principle，DIP）
- 迪米特法则，最少知识原则（Least Knowledge Principle，LKP）
- 避免贫血类
- 优先使用组合而不是继承
- 尽量避免使用静态成员，单例可能并不是一个好的设计模式。

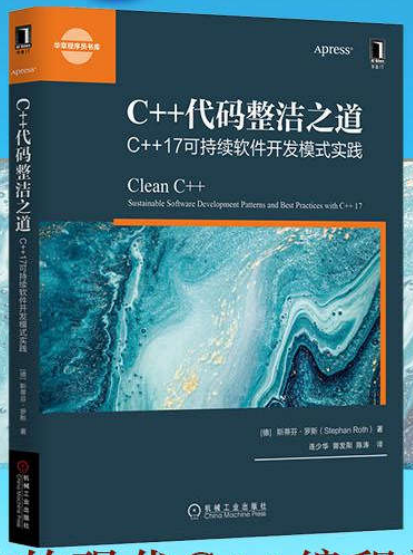

愿景
过程
技术


分析阶段
市场分析
需求分析
设计阶段
产品设计
概要设计
系统设计
测试用例设计
实现阶段
模块设计
编码、自测
测试
发布版本
运维阶段
部署
日常维护
问题与反馈

持续改进

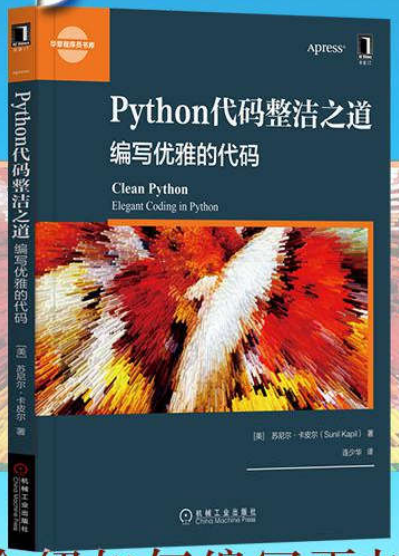
代码整洁之道

C++



掌握高效的现代C++编程法则；
学会应用C++设计模式和习惯用法；
创建可维护、可扩展的软件

Python



通过示例介绍如何编写更加整洁、优雅的Python代码，并介绍一些非常有用的工具

Thank You!