C++ Summit 2020

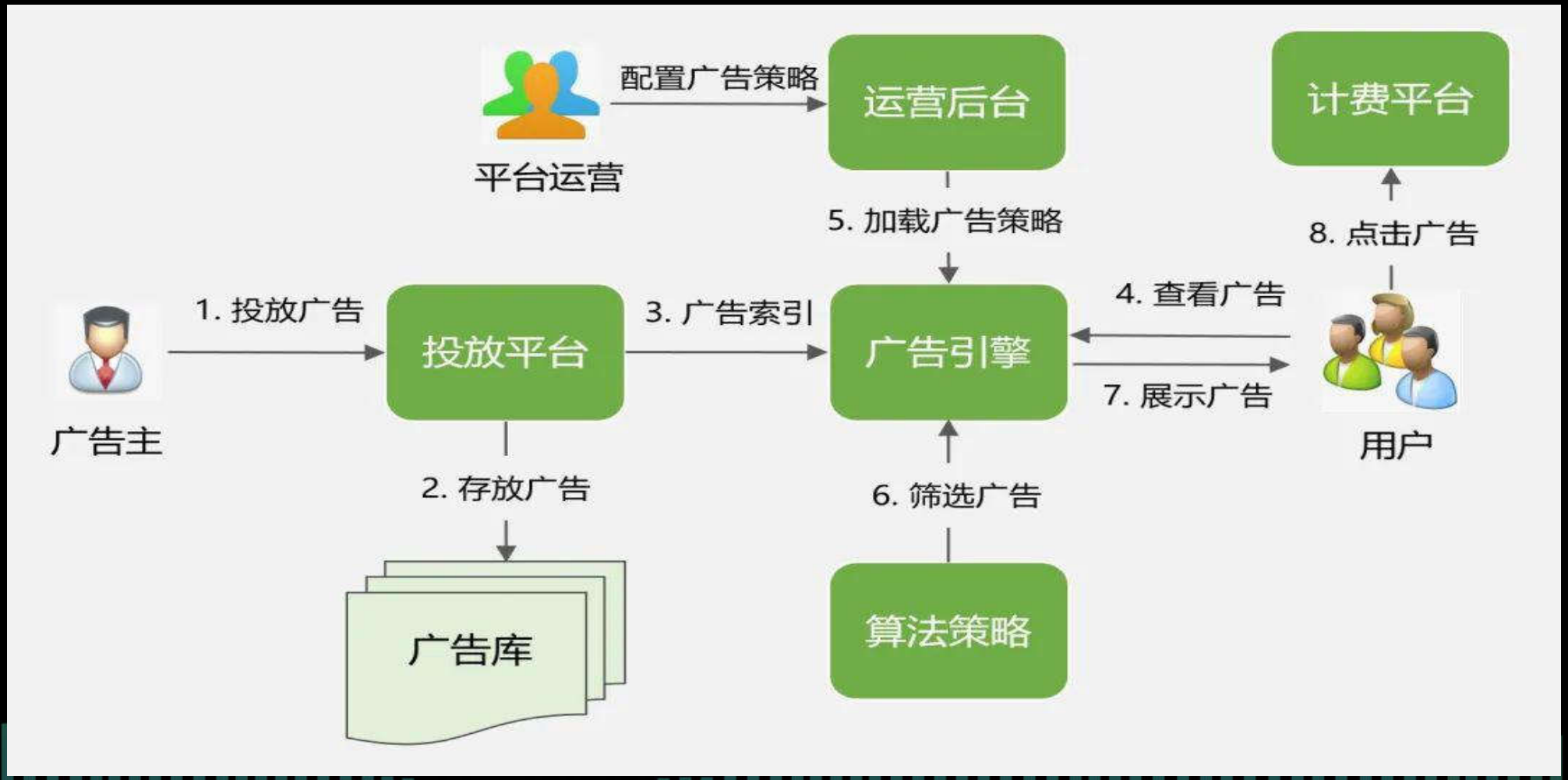
腾讯广告大规模C++工程实践

陈峰
腾讯广告工程效能负责人
高路
腾讯广告高级工程师


微信及WeChat
合并月活跃账户数
约10.98亿
腾讯广告联盟
拥有10万个行业头部APP
联盟每月活跃人数超5亿
联盟
QQ
月活跃账户8.07亿
70%用户为95及0
QQ浏览器
中国安卓用户覆盖率
第一
QQ空间
每天上传至QQ
照片数量达1亿
腾讯新闻
业界领先的
新闻资讯平台
腾讯视频
月度活跃用户
超5亿
腾讯音乐
中国在线音乐
娱乐服务领航者

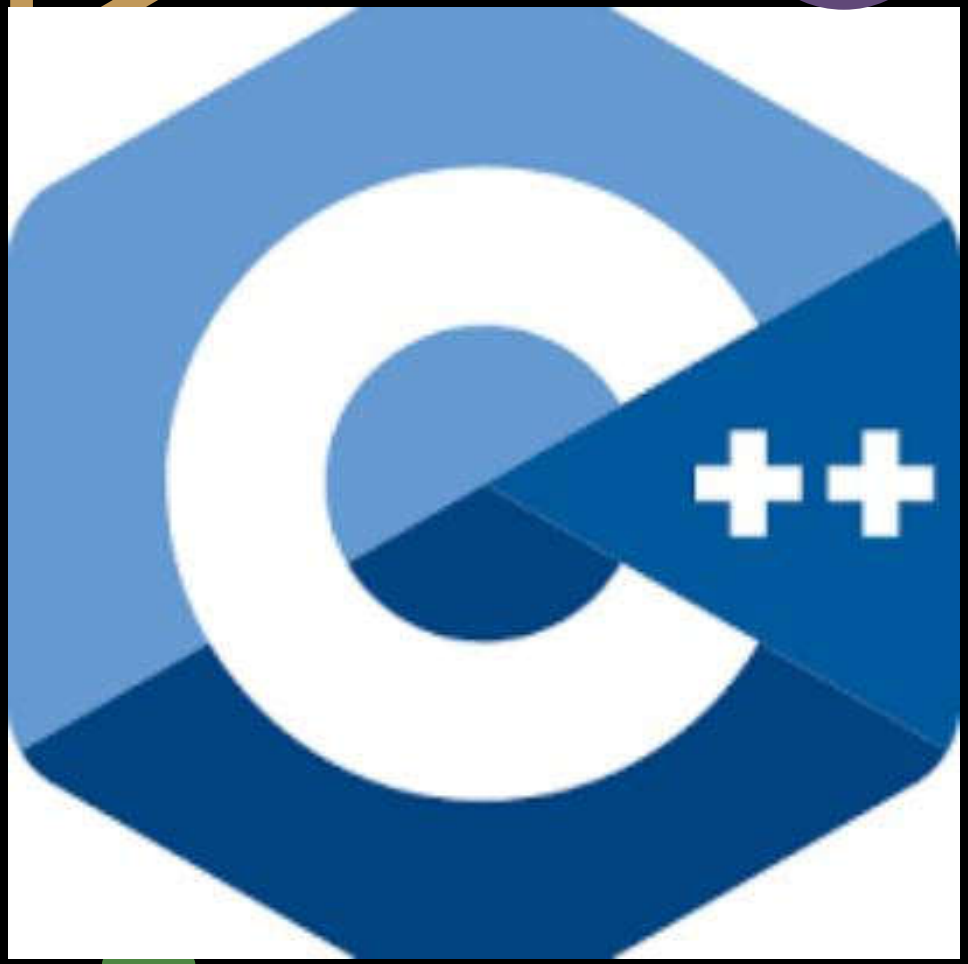
什么是腾讯广告

广告系统基本架构


配置广告策略
运营后台
计费平台
平台运营
5. 加载广告策略
8. 点击广告
1. 投放广告
投放平台
3. 广告索引
广告引擎
4. 查看广告
7. 展示广告
广告主
用户
2. 存放广告
6. 筛选广告
广告库
算法策略

议题

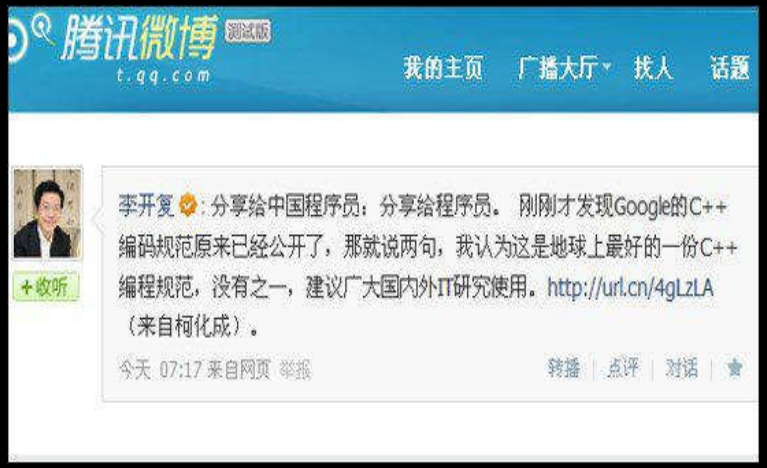
- 代码库管理
- 平台工具建设
- 现代C++实践



代码库管理

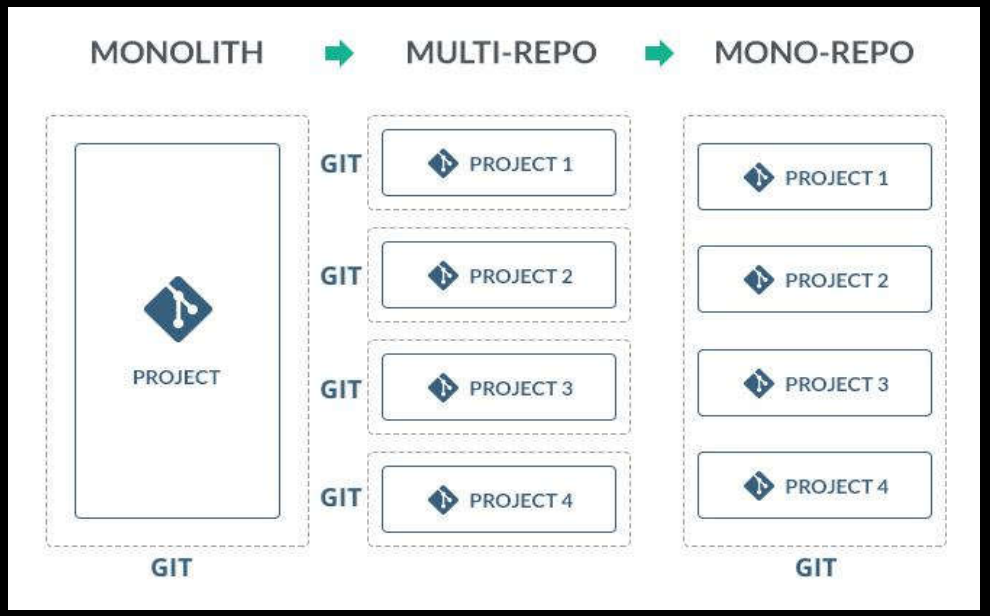
代码库管理——统一组织方式和代码规范

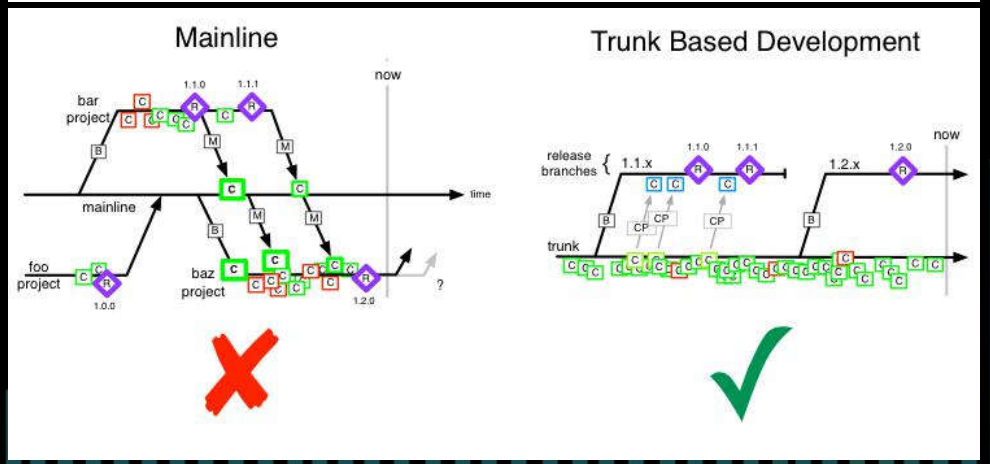
- 基于Google C++代码规范
- 制定《代码组织规范》
- 通过工具检查和代码评审来实施



代码库管理——单一代码库

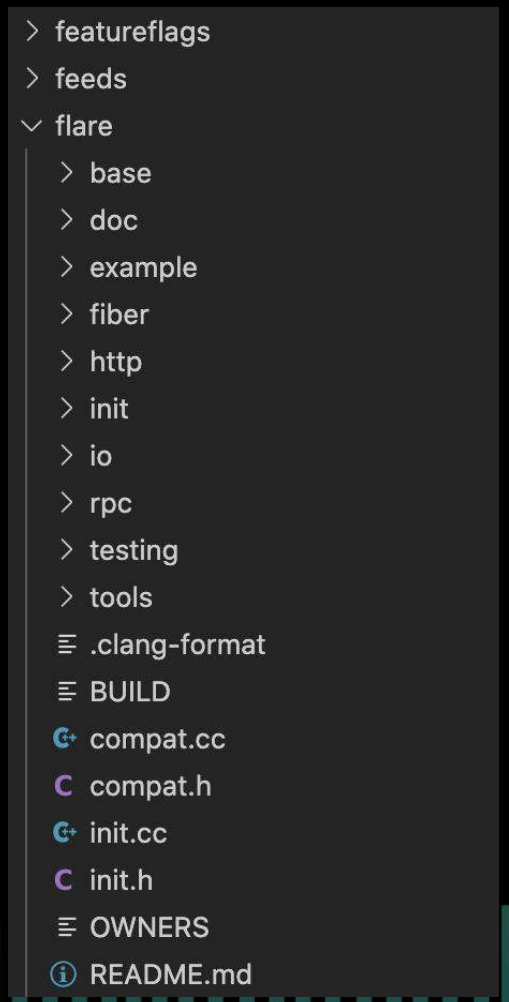
- 绝大多数代码对所有人可以访问
- 有利于倡导代码公有的理念，方便代码复用
- 方便跨语言共享代码
- 让有问题的代码尽早暴露
- 方便实施原子提交
- 方便实施代码重构
- 不用考虑兼容旧版本
- 方便实施统一的工具检查
- 方便实施主干开发模式





代码库管理——基本组织

- 按项目划分目录，并设置评审人，要有README
- 目录名只和项目关联，不和组织架构关联，因为后者容易变动
- 各级目录名都要有含义，不用include，src 等目录名
- 头文件必须自足性，使用时不需要考虑包含顺序
- 每个功能模块的头文件，实现文件和测试代码都放在同一个目录下，方便发现缺少测试的情况



代码库管理——权限管理

- /common/proto/OWNERS

```
set noparent  # 不继承父目录设置（默认为继承）
martin; mark; michael


# 支持逻辑表达式运算
per-file *.proto ([zhang3;lisi], [wang5,zhao6])


# 三人中任意两人通过即可
per-file config.json (terry;peter;saren){2}


CC=alex;john;mike  # 抄送，无评审通过权
```

- .github/CODEOWNERS

```
/common/net/ @pony @martin @mark

/common/net/*.proto ?

CC?
```

代码库管理——统一构建系统

- 声明式语法
- 自动传递依赖关系
- 支持多种编程语言，方便跨语言调用
- 多任务/分布式构建
- 最小化增量构建
- 内置测试支持

```
cc_library(
    name = 'ip_address',
    srcs = ['ip_address.cpp'],
    hdrs = ['ip_address.h'],
)

cc_library(
    name = 'socket',
    srcs = 'socket.cc',
    hdrs = 'socket.h',
    deps = ':socket_address'
)

cc_library(
    name = 'socket_address',
    srcs = 'socket_address.cc',
    hdrs = 'socket_address.h',
    deps = [
        ':ip_address',
        '//common/base/string:string',
        '//thirdparty/glog:glog',
    ]
)
```

代码库组织——头文件用全路径包含

- 传统的非全路径包含

```
#include "http_server.h"
#include "rpc_server.h"

#include "gflags.h"
#include "logging.h"

#include "ad_rewriter.h"
#include "auction_log_report.h"
#include "query_rewrite_service_impl.h"
```

- 全路径包含

```
#include "common/net/http/server/http_server.h"
#include "common/rpc/rpc_server.h"

#include "thirdparty/gflags/gflags.h"
#include "thirdparty/glog/logging.h"

#include "beeble/ad/ad_rewriter.h"
#include "beeble/query/auction_log_report.h"
#include "beeble/query/query_rewrite_service_impl.h"
```

代码库组织——区分公开头文件和私有头文件

- 通过构建系统控制头文件的可见性
- 不依赖某个库，就不能包含其头文件
- 不能包含库的私有头文件

```
cc_library(
    name = "http",
    srcs = [
        "http.cc",
        "http_impl.h"],
    hdrs = ["http.h"],
)
```

代码库组织——命名空间

- 为了避免潜在的名字冲突项目必须采用命名空间
- 命名空间的名字基于项目名
- 必要时可以使用嵌套命名空间
- 如果必须用宏，宏也需要有前缀

```
// In mixer/mixer_server.h
namespace gdt::mixer {
class MixerServer {
  // ...
};

}  // namespace gdt::mixer

#define GDT_MIXER_DEFINE_PROPERTY(...)
```

代码库组织——库也用全路径

```
cc_binary(
    name = "mixer_server",
    ...
    deps = [
        ":mixer_proto",
        "//common/base/string:string",
        "//common/net:ip_address",
        "//common/rpc:rpc_server",
    ],
)
```

```
# 非全路径链接
LINKFLAGS= \
    -L. -lmixer_proto \
    -L common/base/string -lstring \
    -L common/net –l ip_address \
    -L common/rpc –l rpc_server
```

代码库组织——控制库的使用范围

- 通过构建系统控制一些库的使用范围
- 业务模块的库也可以控制自己的可见性，避免被别的项目意外使用

```
cc_library(
    name = 'boost',
    deps = [
        '//thirdparty/boost_1_69_0:boost',
    ],
    visibility = [
        '//thirdparty/thrift:thrift',
        '//thirdparty/click_house:click_house',
        '//exp/radar:radar',
    ],
)
```

代码库组织——统一管理第三方库

- 制定《第三方库管理规范》
- 统一放在一个目录下
- 禁止项目私有第三方库
- 统一版本
- 专人评审、维护、升级

```
thirdparty
|-- boost
|-- curl
|-- flatbuffers
|-- fmt
|-- gflags
|-- glog
|-- googletest
|-- hdfs
|-- jemalloc
|-- jsoncpp
|-- leveldb
```

代码库组织——基础库建设

- string
- random/hash
- compress/crypto
- datetime
- hex/base64/json/xml/protobuf
- Threads/ThreadPool
- FileStorage
- Network
- HTTP/URL/QueryParams
- RPC

```
#ifndef COMMON_SYSTEM_NET_IP_ADDRESS_H_
#define COMMON_SYSTEM_NET_IP_ADDRESS_H_
#pragma once

#include <limits.h>
#include <stdio.h>
#include <string>
#include <vector>
#include "common/base/string/string_piece.h"
#include "common/system/net/os_socket.h"

namespace gdt {

/// IP v4 address
class Ipv4Address {
 public:
  static const size_t kByteSize = 4;
  constexpr Ipv4Address() : m_ip() {}
  constexpr explicit Ipv4Address(uint32_t ip);
  constexpr explicit Ipv4Address(in_addr addr);
  constexpr Ipv4Address(uint8_t b1, uint8_t b2,
                        uint8_t b3, uint8_t b4);
  explicit Ipv4Address(const char* src);
  explicit Ipv4Address(const std::string& src);
```

代码工具平台

代码平台工具——挑战和应对

- 挑战

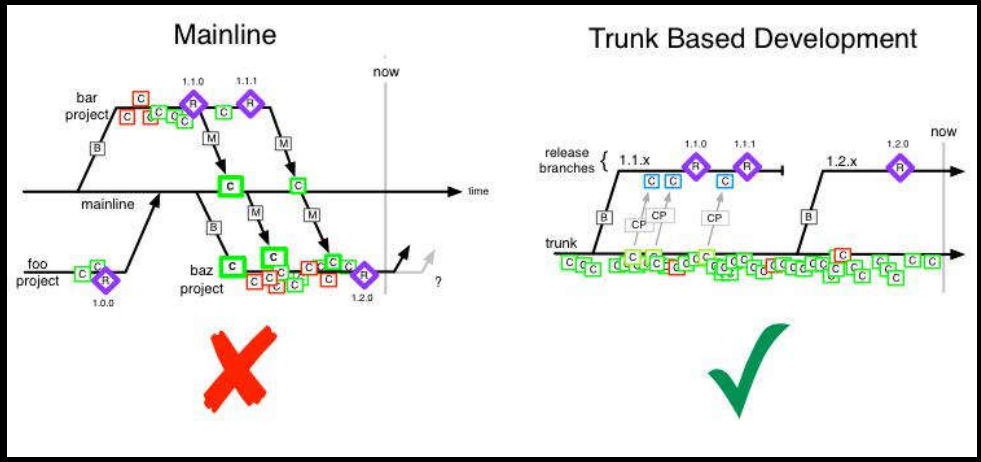
- 77377个C++文件，55911个Java文件
- 576万行C++代码
- 数百人在一个代码库上开发，直接发生相互影响
- master分支每天合并200多次，对CI压力很大
- 代码中存在着大量还在开发中的特性

- 应对

- 代码提交前增量检查
- 提交后的模块级代码质量检查
- 单元测试要求和覆盖率目标跟进
- 代码评审制度
- 特性开关系统
- 构建系统优化
- 持续集成系统
- 回归测试系统

代码平台工具——FeatureFlags控制特性变更

- 适配主干开发模式，减少合并冲突
- 方便实施自动化回归测试
- 方便对每个开关单独启用，灰度开启验证
- 方便有问题时单独关闭而不用回滚整个发布
- 可以精确统计和度量每个特性的访问情况
- 平台自动跟踪开关生命周期，并驱动清理



版本	flag	flag描述	下线时间
20201202 V1	fe_video_analysis	视频分析flag	2020-12-17
20201202 V1	switch_union_dev_api	切换联盟应用dev-api	2021-01-30
20201201 V2	fe_multiple_scene	通投版位多场景	2021-01-31
20201201 V1	fe_creative_table_new	创意工作台-新建视频	2021-11-01

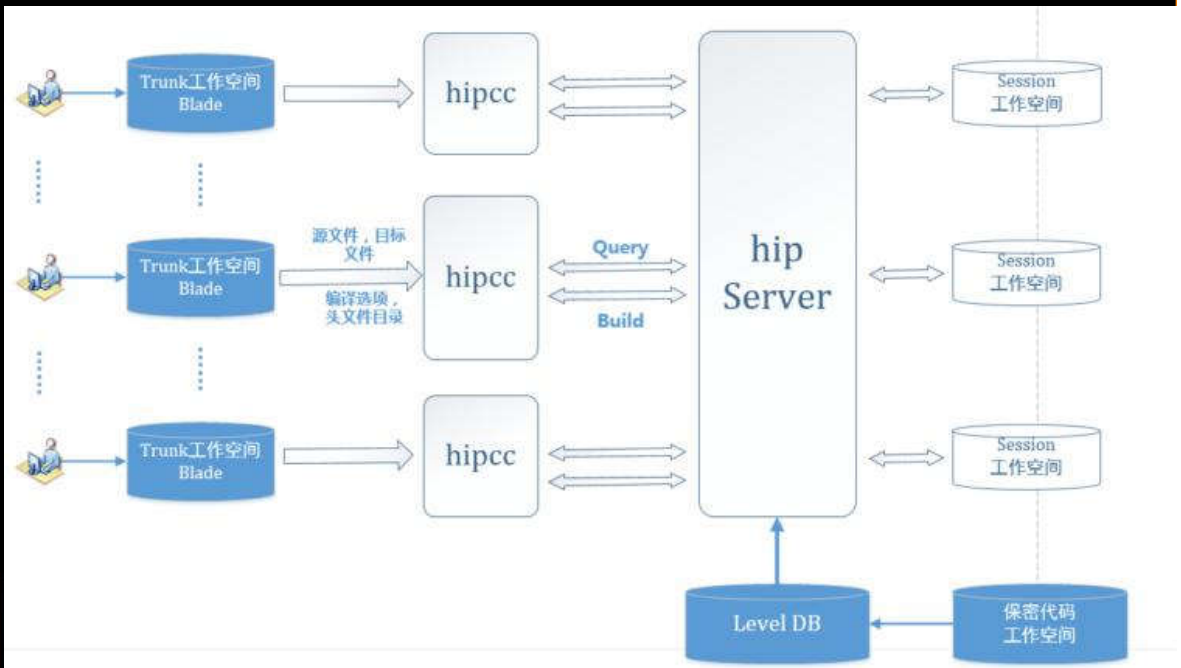
代码工具平台——再谈构建系统

- 尽量从源代码构建
  - 减少二进制兼容问题
  - 方便编译器升级
- 不要用预编译头文件
  - 脆弱的实现
  - 增大不必要的耦合
- 静态链接发布和部署
  - 减少依赖
  - 方便编译器升级

代码工具平台——保密代码编译器

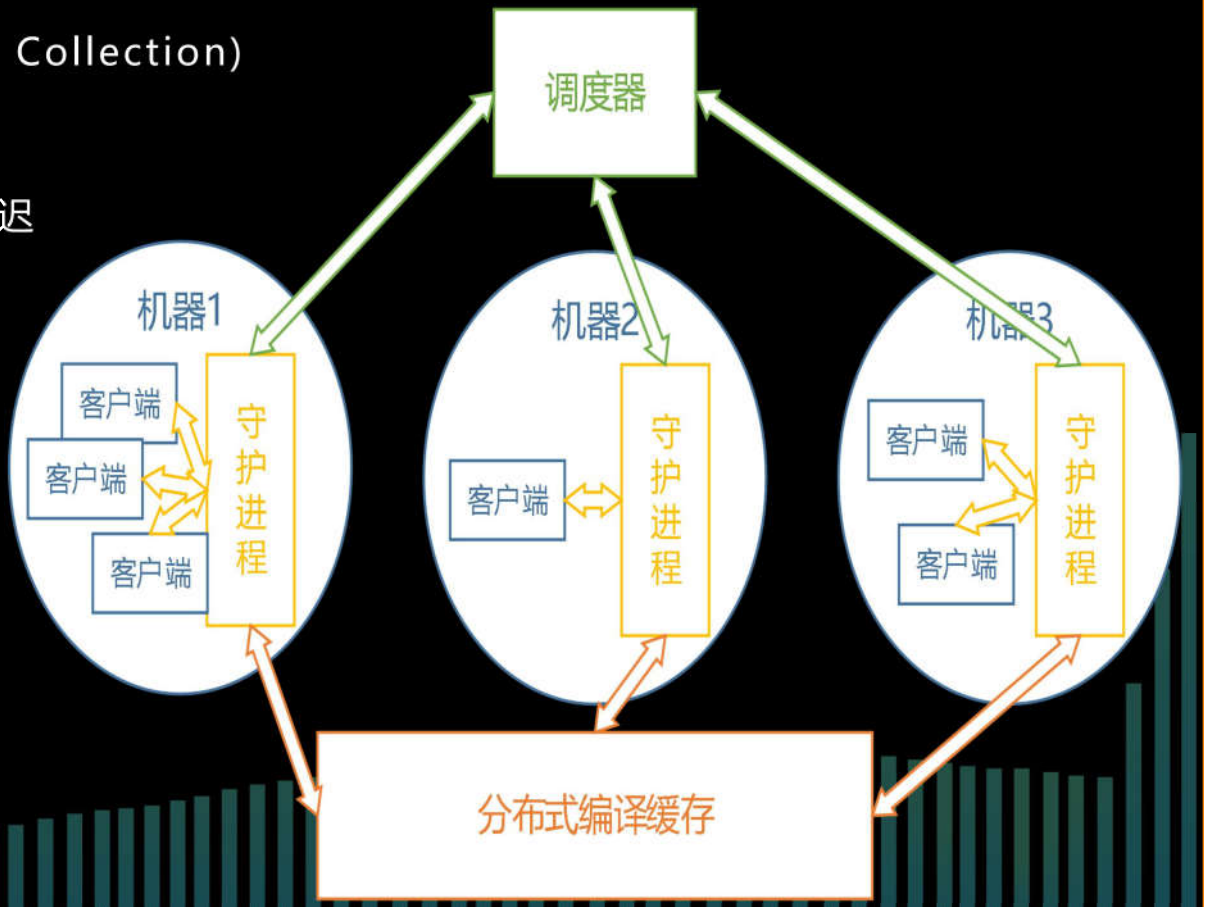
```
cc_library(
    name = 'sunfish',
    srcs = ['foo.cc', 'bar.cc', 'baz.cc'],
    hip = True,
    hip_version = 10777,
)
```

- 99.9%的代码都是全员可以访问的
- 极少量代码，只有相关同事才能访问
- 建设保密代码构建系统构建这些代码



代码工具平台——分布式构建系统

- Yadcc (Yet Another Distributed Compiler Collection)
  - 中心化任务调度
  - 分布式缓存+布隆过滤器减少无效查询的延迟
  - 本地任务并发度控制
  - 用-fdirectives-only加速预处理
  - 使用高效率的压缩/哈希算法
  - “P2P”方式共享CPU资源
  - 支持多版本编译器共存



代码工具平台——分布式构建系统

- 8C 2.5GHz Cascade Lake (KVM), 16 GB RAM，集群1400核
- CentOS 7 / Devtoolset-9
- llvm-project 11.0.0
- CXX=~/.yadcc/symlinks/g++ CC=~/.yadcc/symlinks/gcc cmake3 -G "Ninja" -DLLVM_ENABLE_PROJECTS="clang;libcxx;libcxxabi;libunwind;lldb;compiler-rt;lld" -DCMAKE_BUILD_TYPE=Release ../llvm
- time ninja -k256

```
[6124/6124] Creating executable symlink bin/clang

real    3m30.717s
user    18m44.844s
sys     4m50.628s
```

```
[6124/6124] Generating ../../bin/llvm-readelf

real    49m2.797s
user    366m6.104s
sys     22m3.883s
```

CPP-Summit 2020

现代C++

现代C++——编译器升级

- 单一代码库和静态链接使得升级更容易
- 升级过程中创建两个CI，确保都代码对两个编译器都兼容
- gnu++2a，全面支持C++17及部分C++20特性
- 提供代码检查工具引导使用新特性

年份	GCC版本	C++标准
2013	4.1.2	03
2015	4.4.6	0x
2016	4.9.4	14
2018	8.2	2a
2021?	11.2?	20

```
-------

_test.cpp:16:tr1: Don't use std::tr1, use std library, see https
_test.cpp:17:tr1: Don't use std::tr1, use std library, see https
_test.cpp:20:tr1: Don't use std::tr1::unordered_map, use std::un

_test.cpp:21:nullptr: Using nullptr instead of NULL, see https:/
------
```

现代C++——Flare服务开发框架

- 基础库
- RPC框架

现代C++——Flare服务开发框架

- 基础库
  - 广泛使用标准库新增的类型改善可读性及性能
    - optional, chrono, string_view, move
  - 尚未被编译器支持的新标准库功能/好的提案：引入第三方库或仿造
    - fmt, expected
  - 补齐常用但标准库尚未支持的功能
    - base64, object pool, compression,...
  - 现代C++设计风格，通过Traits自定义/扩展库的行为

现代C++——Flare服务开发框架

```
// 旧式接口，用返回值报告错误，出参数返回实际结果
template <class T>
bool StringToNumeric(
    const std::string& s, T* result, int32_t base = 0);
```

```
template <class T, class = void>
struct TryParseTraits;

template <class T, class... Args>
inline std::optional<T> TryParse(const std::string_view& s,
                                 const Args&... args) {
  return TryParseTraits<T>::TryParse(s, args...);
}

template <class T>
struct TryParseTraits<T, std::enable_if_t<std::is_integral_v<T>>> {
  static std::optional<T> TryParse(
      const std::string_view& s, int base = 10);
};
```

现代C++——Flare服务开发框架

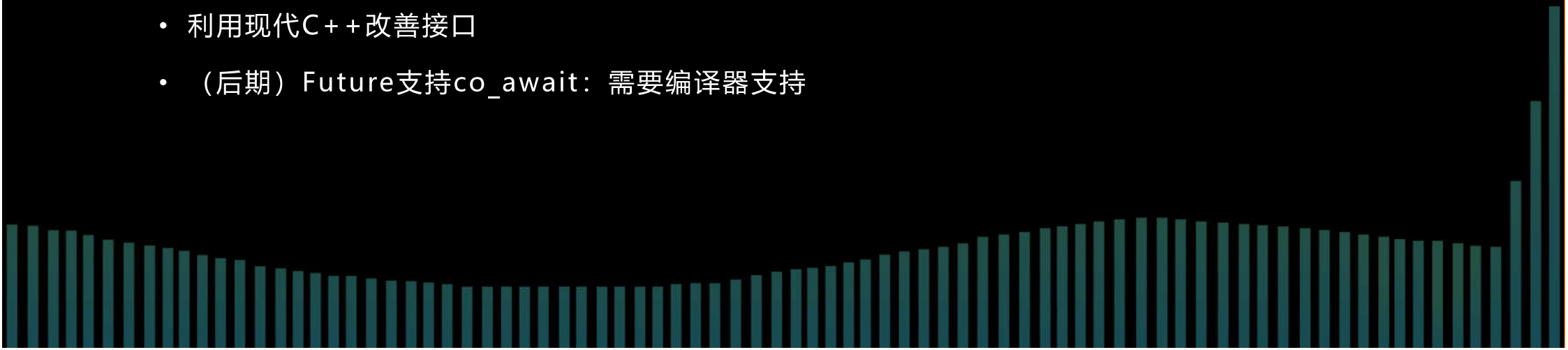
```
inline constexpr struct from_ipv4_t {
  constexpr explicit from_ipv4_t() = default;
} from_ipv4;
inline constexpr struct from_ipv6_t {
  constexpr explicit from_ipv6_t() = default;
} from_ipv6;

template <class T, class>
struct TryParseTraits;

template <>
struct TryParseTraits<Endpoint, void> {
  static std::optional<Endpoint> TryParse(const std::string_view& s,
                                          from_ipv4_t);
  static std::optional<Endpoint> TryParse(const std::string_view& s,
                                          from_ipv6_t);
  static std::optional<Endpoint> TryParse(const std::string_view& s);
};
```

现代C++——Flare服务开发框架

- RPC框架
  - 依赖注入支持多协议、调用链、监控、etc.
  - 底层基于用户态线程（Fiber）
  - 上层使用Future支持异步/异构/并发RPC
  - 利用现代C++改善接口
  - （后期）Future支持co_await：需要编译器支持



现代C++——Flare服务开发框架

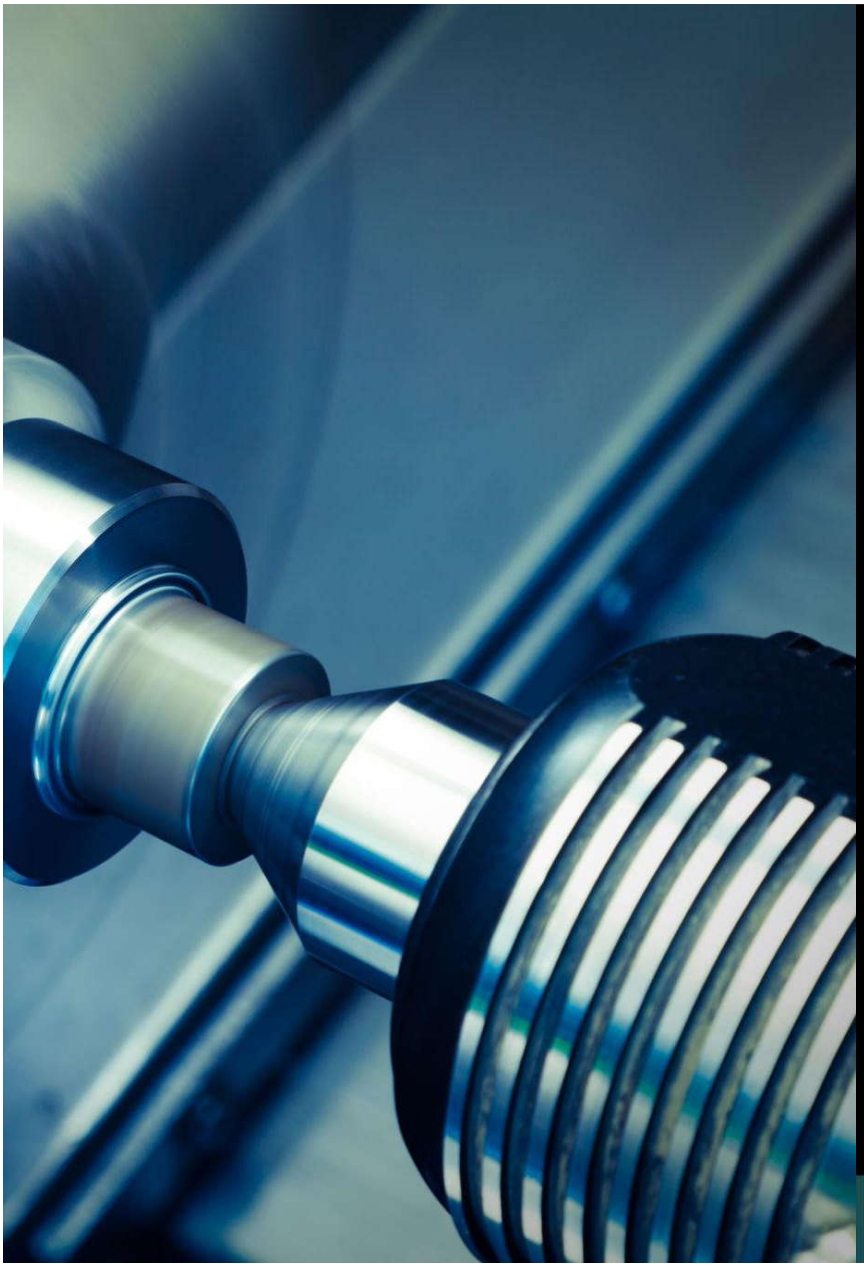
```cpp
EchoService_AsyncStub stub1("flare://some.polaris.address-1");
EchoService_AsyncStub stub2("flare://some.polaris.address-2");

EchoRequest req1, req2;
req1.set_body("body1");
req2.set_body("body2");

flare::RpcClientController ctlr1, ctlr2;
ctlr1.SetTimeout(1s);
ctlr2.SetTimeout(std::chrono::steady_clock::now() + 2s);
auto f1 = stub1.Echo(req1, &ctlr1);  // Future<Expected<EchoResponse, Status>>
auto f2 = stub2.Echo(req1, &ctlr2);

auto&& [res1, res2] = flare::fiber::BlockingGet(flare::WhenAll(&f1, &f2));

if (res1 && res2) {
  FLARE_LOG_INFO("Received: {}, {}", res1->body(), res2->body());
}
```



总结

- 单一代码库和主干开发模式提高开发效率
- 通过工具化解决效率和质量之间的矛盾
- 跟进C++语言发展，走现代C++之路
- 面向现代C++建设基础库和框架，提升研发效率


Q & A