C++发组件：**网络库、RPC 框架、数据库、配置文件处理、消息队列、序列化**

Google 开源代码规范

https://github.com/zh-google-styleguide/zh-google-styleguide

C++ 是 Google 大部分开源项目的主要编程语言， C++ 有很多强大的特性，但这种强大不可避免地导致它走向复杂，使代码更容易产生 bug，难以阅读和维护。

本指南的目的是通过详细阐述 C++ 注意事项来驾驭其复杂性. 这些规则在保证代码易于管理的同时，也能高效使用 C++ 的语言特性.

数据库

cpp_redis

https://github.com/Cylix/cpp_redis/

C++11 Lightweight Redis client: async, thread-safe, no dependency, pipelining, multi-platform

这是一个 C++11 编写的轻量级 Redis 客户端，具有异步、线程安全、无依赖、pipelining、跨平台等特性。代码量不大，可以学习如何编写一个简洁高效的网络通信客户端程序，另外项目采用了 C++11 编写。

LevelDb

https://github.com/google/leveldb

由 Google 的两位工程师 Sanjay Ghemawat 和 Jeff Dean 开发的键/值对（Key/Value Pair）嵌入式数据库，具有很高的随机写，顺序读/写性能，采用 LSM 树 (Log Structured-Merge Tree)实现，LSM 树的核心思想就是放弃部分读的性能，换取最大的写入能力.

关于 LevelDb 详细内容学习可参考教程：

https://kevins.pro/leveldb_chinese_doc.html

https://leveldb-handbook.readthedocs.io/zh/latest/basic.html

文件解析器

在后台项目中需要经常处理配置文件解析工作，这些配置文件可以是 XML、JSON 或者是 YAML 等格式的标记语言文件，下面这几个项目就是几个不错的文件解析器，代码可读性好。

JSON 处理

C++ JSON 解析器

https://github.com/hjiang/jsonxx

用 C++写的轻量级 JSON 解析器，同时还可以将 jSON 文档转换成 XML 文档

迷你的 C++11 JSON 库

https://github.com/dropbox/json11

json11 is a tiny JSON library for C++11, providing JSON parsing and serialization.

mini JSON 库是一个 C++11 标准的小型 json 库，提供 json 的分析和序列化功能，仅 1 个 CPP 文件和一个头文件，方便地嵌入到自己的项目中。

TinyXML

http://sourceforge.net/projects/tinyxml/

是一个用 C++ 编写的，非常简单小巧的 XML 解析器，容易集成到项目中。

yaml-cpp

https://github.com/jbeder/yaml-cpp

YAML 也是一种类似 XML 和 JSON 一样的配置文件语言，YAML 的意思其实是："Yet Another Markup Language"（仍是一种标记语言），是专门用来写配置文件的语言，相比 JSON 更加简洁和方便阅读。

这个项目就是一个用 cpp 写的 yaml 文件解析器。

网络库

Muduo

https://github.com/chenshuo/muduo

它是一个基于 Reactor 模式的现代 C++ 网络库，采用非阻塞 IO 模型，基于事件驱动和回调，原生支持多核多线程，适合编写 Linux 服务端多线程网络应用程序。

muduo 基于 Reactor 模式，Redis 和 Java 的 Netty 库也是采用这种模式实现，具有如下特点：采用非阻塞 IO 模型、基于事件驱动和回调，原生支持多核多线程。

消息队列

后端开发常用到消息队列，消息队列是分布式系统中重要的组件，主要解决了应用耦合、异步处理、流量削锋等问题。

消息队列在实际应用中包括如下四个场景：

- 应用耦合：多应用间通过消息队列对同一消息进行处理，避免调用接口失败导致整个过程失败；
- 异步处理：多应用对消息队列中同一消息进行处理，应用间并发处理消息，相比串行处理，减少处理时间；

- 限流削峰：广泛应用于秒杀或抢购活动中，避免流量过大导致应用系统挂掉的情况；
- 消息驱动的系统：系统分为消息队列、消息生产者、消息消费者，生产者负责产生消息，消费者(可能有多个)负责对消息进行处理；

消息队列详细介绍参考：

https://cloud.tencent.com/developer/article/1006035

业界有名的 zeromq 核心代码也是用 C++ 编写，如果想深入研究消息队列，可以从这入手试试，下面给出项目主页和门户网站。

https://zeromq.org/

https://github.com/zeromq/libzmq

序列化

- 序列化： 将数据结构或对象转换成二进制串的过程
- 反序列化：将在序列化过程中所生成的二进制串转换成数据结构或者对象的过程

数据结构、对象与二进制串

不同的计算机语言中，数据结构，对象以及二进制串的表示方式并不相同。

数据结构和对象：对于类似 Java 这种完全面向对象的语言，工程师所操作的一切都是对象（Object），来自于类的实例化。在 Java 语言中

最接近数据结构的概念，就是 POJO（Plain Old Java Object）或者 Javabean - - 那些只有 setter/getter 方法的类。而在 C++这种半面向对象的语言中，数据结构和 struct 对应，对象和 class 对应。

二进制串：序列化所生成的二进制串指的是存储在内存中的一块数据。C++语言具有内存操作符，所以二进制串的概念容易理解，例如，C++语言的字符串可以直接被传输层使用，因为其本质上就是以’\0’结尾的存储在内存中的二进制串。在 Java 语言里面，二进制串的概念容易和 String 混淆。实际上 String 是 Java 的一等公民，是一种特殊对象（Object）。对于跨语言间的通讯，序列化后的数据当然不能是某种语言的特殊数据类型。二进制串在 Java 里面所指的是 byte[]，byte 是 Java 的 8 中原生数据类型之一（Primitive data types）。

序列化参考链接：

https://tech.meituan.com/2015/02/26/serialization-vs-deserialization.html

Protobuf

https://github.com/protocolbuffers/protobuf

Protocol Buffer (简称 Protobuf) 是 Google 出品的性能优异、跨语言、跨平台的序列化库。

RPC

RPC (Remote Procedure Call)远程过程调用是一个计算机通信协议。我们一般的程序调用是本地程序内部的调用，RPC 允许你像调用本地函数一样去调用另一个程序的函数，这中间会涉及网络通信和进程间通信，但你无需知道实现细节，RPC 框架为你屏蔽了底层实现。RPC 是一种服务器-客户端（Client/Server）模式，经典实现是一个通过**「发送请求-接受回应」**进行信息交互的系统。

rest_rpc

https://github.com/qicosmos/rest_rpc

c++11, high performance, cross platform, easy to use rpc framework.

It's so easy to love RPC.

rest_rpc 是一个高性能、易用、跨平台、header only 的 c++11 rpc 库，它的目标是让 tcp 通信变得非常简单易用，即使不懂网络通信的人也可以直接使用它。 可以快速上手，使用者只需要关注自己的业务逻辑即可。

业内成熟的后台开发 RPC 框架：

谷歌 gRPC

https://github.com/grpc/grpc

谷歌开源的高性能远程过程调用系统。

在 gRPC 里客户端应用可以像调用本地对象一样直接调用另一台不同的机器上服务端应用的方法，使得您能够更容易地创建分布式应用和服务。与许多 RPC 系统类似，gRPC 也是基于以下理念：定义一个服务，指定其能够被远程调用的方法（包含参数和返回类型）。在服务端实现这个接口，并运行一个 gRPC 服务器来处理客户端调用。在客户端拥有一个存根能够像服务端一样的方法

百度 brpc

https://github.com/apache/incubator-brpc/

百度内最常使用的工业级 RPC 框架，有 1,000,000+个实例(不包含 client)和上千种多种服务。"brpc"的含义是"better RPC"。

你可以使用它：

- 搭建能在**一个端口**支持多协议的服务，或访问各种服务
    - restful http/https, h2/gRPC。使用 brpc 的 http 实现比 libcurl 方便多了。从其他语言通过 HTTP/h2+json 访问基于 protobuf 的协议.
    - redis 和 memcached，线程安全，比官方 client 更方便。
    - rtmp/flv/hls，可用于搭建流媒体服务.
    - hadoop_rpc(可能开源)

    - 支持 rdma(即将开源)
    - 支持 thrift，线程安全，比官方 client 更方便
    - 各种百度内使用的协议：baidu_std, streaming_rpc, hulu_pbrpc, sofa_pbrpc, nova_pbrpc, public_pbrpc, ubrpc 和使用 nshead 的各种协议.
    - 基于工业级的 RAFT 算法实现搭建高可用分布式系统，已在 braft 开源。
- Server 能同步或异步处理请求。
- Client 支持同步、异步、半同步，或使用组合 channels 简化复杂的分库或并发访问。
- 通过 http 界面调试服务，使用 cpu, heap, contention profilers.
- 获得更好的延时和吞吐.
- 把你组织中使用的协议快速地加入 brpc，或定制各类组件，包括命名服务 (dns, zk, etcd)，负载均衡 (rr, random, consistent hashing)

腾讯 tars

https://github.com/TarsCloud/TarsCpp

腾讯开源的 RPC 框架

TARS 是 Linux 基金会的开源项目，它是基于名字服务使用 TARS 协议的高性能 RPC 开发框架，配套一体化的运营管理平台，并通过伸缩调度，实现运维半托管服务。

TARS 是腾讯从 2008 年到今天一直在使用的后台逻辑层的统一应用框架，覆盖腾讯 100 多个产品。目前支持 C++,Java,PHP,Nodejs,Go 语言。该框架为用户提供了涉及到开发、运维、以及测试的一整套解决方案，帮助一个产品或者服务快速开发、部署、测试、上线。 它集可扩展协议编解码、高性能 RPC 通信框架、名字路由与发现、发布监控、日志统计、配置管理等于一体，通过它可以快速用微服务的方式构建自己的稳定可靠的分布式应用，并实现完整有效的服务治理。

目前该框架在腾讯内部，各大核心业务都在使用，颇受欢迎，基于该框架部署运行的服务节点规模达到上万个。

搜狗 srpc

https://github**.com/sogou/srpc**

搜狗自研的 RPC 系统，主要功能和特点：

- 这是一个基于 Sogou C++ Workflow 的项目，兼具：
    - 高性能

    - 低开发和接入门槛
    - 完美兼容 workflow 的串并联任务流
    - 对于已有 pb/thrift 描述文件的项目，可以做到一键迁移
- 支持多种 IDL 格式，包括：
    - Protobuf
    - Thrift
- 支持多种数据布局，使用上完全透明，包括：
    - Protobuffer serialize
    - Thrift Binary serialize
    - json serialize
- 支持多种压缩，使用上完全透明，包括：
    - gzip
    - zlib
    - snappy
    - lz4
- 支持多种通信协议，使用上完全透明，包括：
    - tcp
    - http
    - sctp
    - ssl
    - https

- 用户可以通过 http+json 实现跨语言：
    - 如果自己是 server 提供方，用任何语言的 http server 接受 post 请求，解析若干 http header 即可
    - 如果自己是 client 调用方，用任何语言的 http client 发送 post 请求，添加若干 http header 即可
- 内置了可以与其他 RPC 框架的 server/client 无缝互通的 client/server，包括：
    - BPRC
    - Thrift Framed Binary
    - Thrift Http Binary
- 兼容 workflow 的使用方式：
    - 提供创建任务的接口来创建一个 rpc 任务
    - 可以把 rpc 任务放到任务流图中，回调函数里也可以拿到当前的任务流
    - workflow 所支持的其他功能，包括 upstream、计算调度、异步文件 IO 等

单元测试

https://github.com/google/googletest/

Google Test

程序写的好，单元测试少不了。现在流行的软件开发模式「测试驱动开发」，学习使用单元测试保证代码健壮性，Google 的开源 C++ 单元测试框架 Google Test 也称为 gtest ，提供了丰富的断言和各类比较操作。

断言

gtest 使用一系列断言的宏来检查值是否符合预期，主要分为两类：ASSERT 和 EXPECT。区别在于 ASSERT 不通过的时候会认为是一个 fatal 的错误，退出当前函数（只是函数）。而 EXPECT 失败的话会继续运行当前函数，所以对于函数内几个失败可以同时报告出来。通常我们用 EXPECT 级别的断言就好，除非你认为当前检查点失败后函数的后续检查没有意义。

项目主页：