工具/服务概述

工作中涉及了好多工具以及服务等框架。。。。

虽然都是已经环境配置好功能实现好，暂时不需要掌握使用。。。

但是也还是需要一定了解，不能一直云里雾里，万一以后也要用呢。。。

gRPC

gRPC（gRPC Remote Procedure Calls）是一个高性能、开源和通用的 RPC 框架，由 Google 开发。它基于 HTTP/2 协议，并使用 Protocol Buffers 作为接口定义语言（IDL）来定义服务和消息格式。gRPC 支持多种编程语言，使得跨语言服务调用变得更加容易。

RPC 是一种进程间通信方式，允许像调用本地服务一样调用远程服务，通信协议大多采用二进制方式。

（服务间必须使用进程间通信机制来交互，实现跨语言跨平台的交互。常见的通信协议主要有 RPC 和 REST 协议，二者的介绍如下https://blog.csdn.net/a745233700/article/details/122445199）

通信的原理

gRPC 的通信基于 HTTP/2 协议，这带来了一些优势：

1. **多路复用**：允许在单个 TCP 连接上同时发送多个请求和响应。
2. **二进制帧**：数据以二进制格式发送，减少了传输数据的大小，提高了传输效率。
3. **流式通信**：支持客户端流、服务端流和双向流。

使用 gRPC 在 C++ 中开发客户端和服务端应用程序时，你需要以下步骤：

1. 定义 `.proto` 文件

创建一个 Protocol Buffers 文件，定义服务和消息格式。

Code block

```
syntax = "proto3";

package example;

```

```
service Greeter {
    rpc SayHello (HelloRequest) returns (HelloReply) {}
}

message HelloRequest {   string name = 1; }

message HelloReply {   string message = 1; }
```

2. 生成 gRPC 和 Protocol Buffers 代码

使用 `protoc` 编译器生成 C++ 代码。

Code block

```
protoc -I=. --grpc_out=. --plugin=protoc-gen-grpc=`which grpc_cpp_plugin` example.proto
protoc -I=. --cpp_out=. example.proto
```

3. 实现服务端

在 C++ 中实现服务端逻辑。

```cpp
#include <iostream>
#include <memory>
#include <string>
#include <grpcpp/grpcpp.h>
#include "example.grpc.pb.h"

using grpc::Server;
using grpc::ServerBuilder;
using grpc::ServerContext;
using grpc::Status;
using example::Greeter;
using example::HelloRequest;
using example::HelloReply;

class GreeterServiceImpl final : public Greeter::Service {
    Status SayHello(ServerContext* context, const HelloRequest* request, HelloReply
        std::string prefix("Hello ");
        reply->set_message(prefix + request->name());
        return Status::OK;
    }
};

void RunServer() {
    std::string server_address("0.0.0.0:50051");
    GreeterServiceImpl service;

    ServerBuilder builder;
    builder.AddListeningPort(server_address, grpc::InsecureServerCredentials());
    builder.RegisterService(&service);
    std::unique_ptr<Server> server(builder.BuildAndStart());
    std::cout << "Server listening on " << server_address << std::endl;
    server->Wait();
}

int main(int argc, char** argv) {
    RunServer();
    return 0;
}
```

4. 实现客户端

在 C++ 中实现客户端逻辑。

```cpp
class GreeterClient {
public:
    GreeterClient(std::shared_ptr<Channel> channel) : stub_(Greeter::NewStub(chann

    std::string SayHello(const std::string& user) {
        HelloRequest request;
        request.set_name(user);

        HelloReply reply;
        ClientContext context;

        Status status = stub_->SayHello(&context, request, &reply);

        if (status.ok()) {
            return reply.message();
        } else {
            std::cout << status.error_code() << ": " << status.error_message() <<
            return "RPC failed";
        }
    }

private:
    std::unique_ptr<Greeter::Stub> stub_;
};

int main(int argc, char** argv) {
    GreeterClient greeter(grpc::CreateChannel("localhost:50051", grpc::InsecureCha
    std::string user("world");
    std::string reply = greeter.SayHello(user);
    std::cout << "Greeter received: " << reply << std::endl;
    return 0;
}
```

5. 编译和运行

Protocol buffers

Protobuf 的核心思想是**使用协议（Protocol）来定义数据的结构和编码方式**。使用 Protobuf，可以先定义数据的结构和各字段的类型、字段等信息，**然后使用 Protobuf 提供的编译器生成对应的代码，用于序列化和反序列化数据**。由于 Protobuf 是基于二进制编码的，因此可以在数据传输和存储中实现更高效的数据交换，同时也可以**跨语言**使用。

- **protocol buffers** 是一种**语言无关、平台无关、可扩展的序列化结构数据**的方法，它可用于（数据）通信协议、数据存储等。
- **可类比 XML和JSON**，但是比 XML 更小（3 ~ 10倍）、更快（20 ~ 100倍）、更为简单。
- 可自定义数据的结构，使用各种语言进行编写和读取结构数据。也可以更新数据结构，而不破坏由旧数据结构编译的已部署程序。

版本说明：proto3比2支持更多语言，更简洁，一般使用proto3

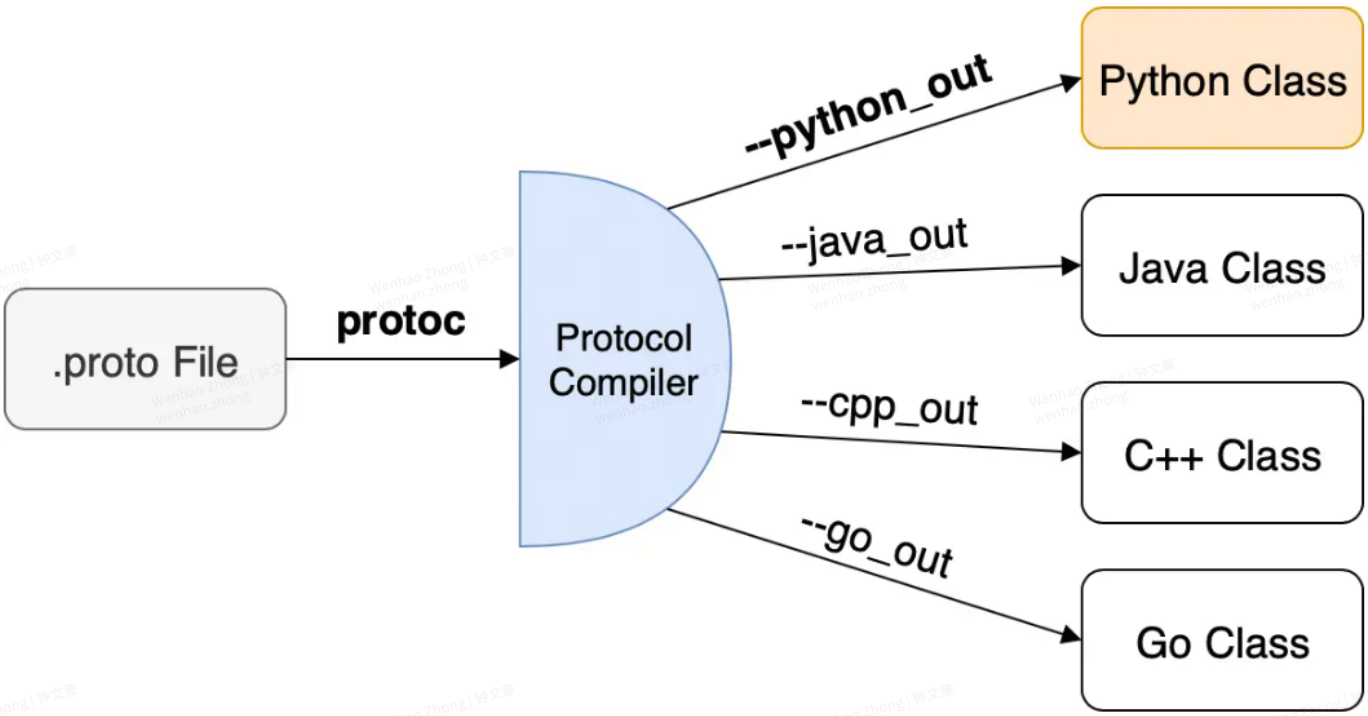
Code block

```
syntax = "proto3"  //不加上默认使用2

// 每一个字段都有一个字段编号，用于二进制格式标识字段。
// 1-15编号需要1个byte编码，16-2047编号需要两个字节。所以高频使用字段放在1~15编号中
message SearchRequest {
    string query = 1;      // 字段编号为1的字符串类型
    int32 page_number = 2;
    int32 result_per_page = 3;
    enum Corpus {
        UNIVERSAL = 0;
        WEB = 1;
        IMAGES = 2;
        LOCAL = 3;
        NEWS = 4;
        PRODUCTS = 5;
        VIDEO = 6;
    }
    Corpus corpus = 4;
}


// 说明:
// 在C++中，当你定义的Protobuf消息包含字段时，Protobuf会自动生成一组访问器函数。
// 例如，对于你给出的 GNSSData 消息，其中的 wgs_position 字段将生成以下方法：
// - wgs_position() const: 返回 wgs_position 的当前值。
// - mutable_wgs_position(): 返回 wgs_position 的可修改指针。
// 因此可以使用 mutable_wgs_position() 方法来获取 wgs_position 字段的可修改版本并直接进行操作。
// ----------------
// 在Protobuf生成的C++代码中，通常只有对于标量类型的字段才会自动生成 set_ 方法
// 例如 set_heading(float heading)。
// 对于复杂类型（如 message 类型）的字段，通常不会生成 set_ 方法。
// 相反，你应该使用 mutable_ 方法来获取字段的可修改指针，然后修改字段的内容。
```

使用 Protobuf 提供的编译器，可以将 `.proto` 文件编译成各种语言的代码文件（如 Java、C++、Python 等）。



Code block

```
protoc --java_out=./java ./resources/addressbook.proto
# --java_out 指定输出 java 格式文件，输出到 ./java 目录
# ./resources/addressbook.proto 为 proto 文件位置
```

ProtoBuf 是一种**轻量、高效**的数据交换格式，它具有以下优点：

- **语言中立**，可以支持多种编程语言；
- 数据结构清晰，易于维护和扩展；
- 二进制编码，数据**体积小，传输效率高**；
- 自动生成代码，开发效率高。

但是，ProtoBuf 也存在以下缺点：

- 学习成本较高，需要掌握其语法规则和使用方法；
- 需要先定义数据结构，然后才能对数据进行序列化和反序列化，增加了一定的开发成本；
- 由于二进制编码，**可读性较差，这点不如 JSON 可以直接阅读**。

总体来说，**Protobuf 适合用于数据传输和存储等场景，能够提高数据传输效率和减少数据体积**。但对于需要人类可读的数据，或需要实时修改的数据，或者对数据的传输效率和体积没那么在意的场景，选择更加通用的 JSON 未尝不是一个好的选择。

很不错的RPC、proto的整体介绍！！！！

https://www.sudosu.cn/grpc/01-proto.html#%E4%B8%80%E3%80%81%E5%90%8D%E8%AF%8D%E8%A7%A3%E9%87%8A

WebSocket

WebSocket 是一种协议，用于在 Web 应用程序中创建实时、双向的通信通道。

传统的 HTTP 请求通常是一次请求一次响应的，而 WebSocket 则可以建立一个持久连接，允许服务器即时向客户端推送数据，同时也可以接收客户端发送的数据。WebSocket 相比于传统的轮询或长轮询方式，能够显著减少网络流量和延迟，提高数据传输的效率和速度。它对实时 Web 应用程序和在线游戏的开发非常有用。

WebSocket 最主要的特点就是建立了一个可持久化的 TCP 连接，这个连接会一直保留，直到客户端或者服务器发起中断请求为止。WebSocket 建立在 HTTP 协议之上，所有的 WebSocket 请求都会通过普通的 HTTP 协议发送出去，然后在服务器端根据 HTTP 协议识别特定的头信息 Upgrade，服务端也会判断请求信息中 Upgrade 是否存在。

具体使用留待后续需要时再学习

openXR

OpenXR是一个由Khronos Group开发的开源跨平台标准，旨在为虚拟现实（VR）和增强现实（AR）设备提供一个统一的API接口。它简化了开发过程，使得开发者能够编写一次代码，即可在多种硬件和平台上运行，从而减少了碎片化的问题。OpenXR通过抽象硬件层，提供了一致的开发环境，并支持多种XR设备，包括头显、手柄等。

OpenXR的主要特点：

1. **跨平台兼容**：支持各种VR和AR硬件和软件平台。
2. **模块化设计**：允许扩展和自定义，以满足特定设备的需求。
3. **性能优化**：通过底层优化提供高性能XR体验。

基本使用步骤：

1. **安装OpenXR SDK**：获取并安装OpenXR的开发工具包。
2. **配置开发环境**：将OpenXR SDK集成到你的开发环境中。
3. **编写应用程序**：使用OpenXR的API编写跨平台XR应用。
4. **测试和部署**：在支持OpenXR的硬件上测试并部署应用。

示例代码：

以下是一个简单的使用OpenXR的C++示例，展示了初始化和基本的XR会话管理：

Code block

```cpp
#include <openxr/openxr.h>
#include <iostream>

int main() {
    XrInstance instance;
    XrInstanceCreateInfo createInfo = {XR_TYPE_INSTANCE_CREATE_INFO};
    createInfo.applicationInfo = {"SampleApp", 1, "", 0, XR_CURRENT_API_VERSION};
    xrCreateInstance(&createInfo, &instance);

    if (instance == XR_NULL_HANDLE) {
       std::cerr << "Failed to create OpenXR instance." << std::endl;
        return -1;
    }

    std::cout << "OpenXR instance created successfully!" << std::endl;

    // Clean upxrDestroyInstance(instance);

    return 0;
}
```

这段代码创建了一个OpenXR实例，并输出创建成功的信息。

IPC

Runtime

非常详细的描述：https://cloud.tencent.com/developer/article/2322237

runtime 是一个通用抽象的术语，指的是计算机程序运行的时候所需要的一切代码库，框架，平台等。它包括了程序的执行环境和执行状态，以及程序在运行时所产生的各种数据和结果。runtime主要是为了实现程序设计语言的执行模型，在每种语言有着不同的实现。

runtime 或者 run-time是指**运行环境**，又称为“运行时系统”，简称“运行时”，是执行码在目标机器上运行的环境。它有可能是由操作系统提供，或由执行此程序的父程序提供。通常由操作系统负责处理程序的载入：利用加载器（loader）读入程序执行码，进行基本的内存配置，并视需要链接此程序指定的所有动态链接库。有些编程语言也会由此语言提供的运行环境处理上述工作。

运行环境可以解决许多问题，包括应用程序内存的管理、程序如何访问变量、程序之间传递参数的机制、与操作系统的接口等问题。编译器根据具体的运行时系统做出假设，以生成正确的代码。通常情况下，运行时系统将承担一些设置和管理堆栈的责任，并可能包括诸如垃圾回收、线程或其他内置于语言中的动态功能。

对于很多传统高级语言尤其是动态语言而言，⌧运行时系统基本上就是程序和硬件的桥梁。一般地，运行时系统实际上提供了一个动态语言得以执行的统一的虚拟计算机，这样一个虚拟机完全屏蔽了硬件的具体特征⌧从而极大的提高了软件的开发效率。但是正因为这个虚拟机完全屏蔽了程序员接触硬件特性的可能，虚拟机本身必须承担起充分利用硬件特性来高效执行动态语言程序的责任。

运行时的实现与分类

在编程语言中，runtime通常是由编译器和运行时库共同实现的。编译器负责将源代码编译成可执行代码，而运行时库则负责在程序运行时提供各种运行时支持和服务。运行时库通常包括了各种系统库和标准库，以及一些特定于编程语言的库和框架。

在C语言中，runtime的实现通常是由C库和操作系统共同提供的。C库提供了各种常用的函数和数据类型，而操作系统则提供了一些底层的系统调用和服务。在Java语言中，runtime的实现则是由Java虚拟机（JVM）和Java类库共同提供的。JVM负责将Java字节码编译成可执行代码，并提供各种运行时支持和服务，而Java类库则提供了各种常用的类和函数。

根据运行时系统实现的不同，可以将运行时系统初步分成三类：

- 第一种是是原生运行时，例如C/C++/Rust，这些语言的运行时系统是依赖操作系统的，操作系统也可以认为是一种“运行时环境”。
- 第二种是轻运行时，例如Golang，它的运行时是和代码打包到一起的，相对轻量一些。
- 第三种是重运行时，例如Java(JVM)，Python(CPython)，还有 C#(.NET Runtime)。它们的运行时环境是需要单独安装的，而且一般还不小，几十兆上百兆都是正常的。

其中，在C/C++里你可以不引用任何库，Rust里有一个专门的特性叫 no_std，脱离标准库提供了一个很牛的能力，就是直接和硬件交互。也就是说，C/C++/Rust是可以用来编写操作系统内核的。

VR眼镜的Runtime

VR眼镜的**Runtime**（运行时）是指运行虚拟现实（VR）应用所需的核心软件组件。它在VR设备和PC或主机之间起到桥梁作用，负责处理和管理VR硬件设备的功能，确保应用程序能够流畅运行并与设备进行交互。具体来说，Runtime 涉及设备追踪、渲染、控制器输入、声音处理以及优化系统性能等方面。

VR眼镜Runtime的主要功能

1. **设备追踪**：
   - Runtime 负责跟踪VR眼镜和控制器的位置和方向。通过传感器和摄像头的数据，Runtime 确保用户在虚拟空间中的运动能够准确映射到虚拟环境中。
2. **渲染管理**：
   - Runtime 管理图像渲染，确保图像在VR设备上以足够的帧率呈现，减少延迟和图像撕裂，提高沉浸感。它会优化渲染流程，确保高效利用计算资源。
3. **控制器输入处理**：
   - Runtime 处理VR控制器的输入信号，并将这些输入信号转化为VR应用中的动作。例如，当用户按下按钮或挥动控制器时，Runtime 会将这些动作转换为应用程序中的相应操作。
4. **声音处理**：
   - 音频是VR体验的重要组成部分，Runtime 负责空间音效的处理和传输，以确保声音与用户的视觉体验同步并且具有方向感。
5. **性能优化**：
   - 为了防止在VR体验中产生眩晕，Runtime 通常包含许多性能优化功能，如异步时间扭曲（Asynchronous Timewarp, ATW）和异步空间扭曲（Asynchronous Spacewarp, ASW）。这些技术可以在帧率不足的情况下通过插值来补偿渲染，提供更平滑的视觉体验。
6. **设备管理**：
   - Runtime 负责管理和配置VR设备，包括VR头盔、控制器、外部传感器、摄像头等硬件设备的驱动和更新，确保设备之间的互操作性。
7. **应用兼容性和交互**：
   - Runtime 为开发者提供了与VR硬件交互的API（应用程序接口），确保开发的VR应用能够与不同硬件设备兼容并实现最佳性能。

常见的VR Runtime

1. **Oculus Runtime**：
   - 用于 Meta（前身为 Oculus）系列的 VR 设备，如 Oculus Rift、Quest 和 Quest 2。Oculus Runtime 负责设备的跟踪、渲染优化、输入处理等核心功能。

2. **SteamVR**：
    - Valve 提供的 VR 平台，支持各种 VR 设备（如 HTC Vive、Valve Index、Oculus Rift 及其他兼容设备）。SteamVR 的 Runtime 提供跨设备的兼容性，允许开发者通过统一接口开发应用，并支持丰富的追踪和交互功能。
3. **Windows Mixed Reality Runtime**：
    - 微软为 Windows Mixed Reality（WMR）设备提供的运行时，支持微软的自家设备如 HoloLens 和第三方 WMR 头盔。它负责处理 WMR 设备的输入、渲染、空间映射等功能。
4. **OpenXR**：
    - OpenXR 是由 Khronos Group 主导的一个开放标准，旨在统一不同VR/AR平台的API，提供跨平台兼容性。许多VR设备厂商和平台都逐步支持 OpenXR，如 Oculus、SteamVR、Windows Mixed Reality 等。

ADB

ADB（Android Debug Bridge）是 Android SDK 中的一个命令行工具，用于在 Android 设备与计算机之间进行调试和数据交互。它提供了多种功能，可以执行安装应用、文件传输、日志查看等操作，非常适合开发、测试、调试和设备管理。

一、ADB的安装和配置

1. **安装 ADB**：ADB 通常随 Android SDK 一起安装。可以直接从 Android Developer 官网下载 Android SDK。
2. **设置环境变量**：确保 ADB 所在路径已添加到系统的环境变量中，这样就可以在命令行中直接使用 `adb` 命令。

二、ADB 基本用法

3. **1. 连接设备**

- **通过 USB 连接**：将设备连接到电脑，启用设备上的 USB 调试功能（设置路径通常为：设置 > 开发者选项 > USB 调试）。
- **无线连接**：使用 `adb tcpip 5555` 将设备端口设为 5555，然后使用 `adb connect <设备IP地址>` 来连接设备。

4. **2. 基本命令**

- **查看已连接的设备**：`adb devices` 显示所有连接的 Android 设备的列表。
- **进入设备 shell**：`adb shell` 进入设备终端，可以在其中执行 Linux 命令。
- **安装 APK**：`adb install <apk路径>` 将 APK 文件安装到设备上。

- **卸载 APK**：`adb uninstall <包名>` 卸载设备中的应用。
- **重启设备**：`adb reboot` 重启设备，`adb reboot bootloader` 重启到 bootloader 模式。
- **文件传输**：
  - **传输文件到设备**：`adb push <本地文件路径> <设备路径>`
  - **从设备传输文件到电脑**：`adb pull <设备文件路径> <本地路径>`

5. **3. 设备调试命令**

- **查看设备日志**：`adb logcat` 显示设备的实时日志，用于查看运行时输出和调试信息。
- **抓取 bugreport**：`adb bugreport` 抓取设备的完整状态报告，包含日志、内存使用等详细信息。
- **查看设备信息**：`adb shell getprop` 获取设备的属性信息，比如型号、版本等。

三、常用实例

1. **检查设备是否连接**

```
adb devices
```

2. **安装应用**

```
adb install example.apk
```

3. **重启到 Recovery 模式**

```
adb reboot recovery
```

4. **从设备上获取日志**

```
adb logcat > logs.txt
```

5. **使用 ADB 无线调试**

```
adb tcpip 5555  # 在设备上开启 5555 端口
adb connect 192.168.1.101  # 替换为设备的 IP 地址
```

6. **终端方式操作设备**

```
adb shell  # 进入设备 shell
```

四、ADB调试常见问题

- **设备无法识别**：确认已开启 USB 调试、安装了驱动（特别是 Windows 系统），并且 USB 连接模式为“文件传输”。
- **无线调试连接失败**：检查设备与电脑是否在同一网络中，或者尝试重启 ADB 服务（`adb kill-server` 和 `adb start-server`）。

ADB 是 Android 开发和调试过程中非常重要的工具，通过 ADB 能极大地提升操作 Android 设备的灵活性和便捷性。