Redis C++ 客户端开发全流程指南 （148 ~ 154）

在现代互联网和高性能服务开发中，Redis 已经成为不可或缺的内存数据库。对于 C++ 开发者来说，理解 Redis 客户端的使用、编译与开发流程，以及背后的设计思想，对于高效开发和工程化实践非常重要。本文从前置准备、编译构建、开发调试到核心设计思想做全面展开。

一、Redis 客户端开发的前置准备

1. GitHub 资源获取与代理方案

核心资源

- **官方源码**：Redis 本身及官方 C 客户端 `hiredis` 是 Redis 通信的核心。
- **C++ 封装客户端**：`redis-plus-plus` 是在 `hiredis` 基础上的 C++ 封装，提供面向对象接口，方便 C++ 开发者使用 Redis。
- **第三方客户端**：还有一些社区客户端，如 `cpp_redis` 等，但 `redis-plus-plus` 因稳定性和文档完善而更常用。

网络访问与优化

由于大部分源码托管在 GitHub 上，访问速度可能受限，尤其在国内：

- **代理工具**：
  - Steam++、Shadowsocks、VPN 等，可稳定加速 GitHub 访问。
  - 优点：可以保持最新版本同步，减少拉取失败概率。
- **镜像站**：
  - 国内 Gitee 镜像，例如：https://gitee.com/mirrors/redis-plus-plus
  - 适合拉取稳定版本，尤其在网络环境受限时。

开发建议

- **同步仓库**：在本地建立 Git 仓库，定期同步官方更新，避免网络阻塞影响开发。
- **文档管理**：将 README、示例、API 文档归档在个人博客或本地文档库，便于离线查阅。
- **版本锁定**：生产项目中建议固定客户端版本，避免因依赖更新引入不兼容问题。

---

2. `redis-plus-plus` 客户端特性与依赖

定位与优势

- **封装特点**：
  - 基于 `hiredis`，保留高性能协议解析。
  - 提供 C++ 面向对象接口，例如 `Redis` 类封装连接与命令。
  - 支持 C++11 及以上特性，如智能指针、异常处理。
- **简化开发**：
  - 自动管理连接。
  - 提供同步与异步接口。
  - 支持事务（`MULTI/EXEC`）、管道（pipeline）、发布订阅（Pub/Sub）。

依赖关系

- `hiredis` 是底层核心：
  - 提供 TCP 连接管理。
  - 实现 RESP 协议解析。
  - 由 C 语言编写，高效且稳定。
- 编译 `redis-plus-plus` 前必须安装 `hiredis`，并在 CMake 中正确指定其路径。

安装方式

- **源码编译**：
  - 下载 `redis-plus-plus` 和 `hiredis` 源码。
  - 使用 CMake 生成 Makefile，再执行 `make` 和 `make install`。
- **包管理器**：
  - Ubuntu：`sudo apt install libhiredis-dev`（安装 `hiredis`），再编译 `redis-plus-plus`。
  - CentOS：`yum install hiredis-devel`，然后编译。
- **注意**：
  - Ubuntu 的依赖安装更简洁。
  - CentOS 可能需要额外安装编译工具（gcc、make、cmake）。

二、C++ 客户端的编译与构建流程

1. 构建工具与工程结构

CMake 与 Makefile

- **CMake**：跨平台构建系统，生成适合目标系统的 Makefile 或 VS 工程。
- **Makefile**：具体执行编译、链接操作，将源代码生成可执行文件或库文件。

推荐工程结构

```
代码块
1  project-root/
2  │
3  ├── src/            # 源码文件
4  ├── include/        # 头文件
5  ├── third_party/    # 外部依赖，如 hiredis、redis-plus-plus
6  ├── build/          # 构建目录（存放中间文件和编译输出）
7  └── CMakeLists.txt  # 构建配置
```

- **构建目录隔离**：避免污染源码目录，便于多平台编译和调试。
- **依赖管理**：将第三方库放入 `third_party` 或通过系统路径引用，保证构建可复现。

2. 编译步骤

1. **创建构建目录**：

```
代码块
1  mkdir build
2  cd build
```

2. **生成构建文件**：

```
代码块
1  cmake ..
```

   - `..` 指向源码根目录。
   - CMake 会检测系统、编译器、库依赖并生成 Makefile。

3. **编译源码**：

```
代码块
```

```
make -j4
```

- `-j4` 指定并行编译，提高编译速度。

4. **安装库文件**：

```
sudo make install
```

- 将库文件、头文件安装到系统路径。
- 确保其他项目可以链接使用。

3. 编译注意事项

- **系统差异**：
  - Ubuntu：`apt` 安装依赖，如 `libhiredis-dev`。
  - CentOS：`yum` 安装依赖，可能需要额外编译工具。
- **路径排查**：
  - 编译失败时，可使用：

```
find / -name hiredis.h
```

  - 确认头文件位置。
- **代码规范**：
  - 命名清晰，避免模糊变量名如 `sb`。
  - 注释明确，提高团队可维护性。

三、C++ 客户端的开发与调试

1. 核心开发步骤

1. 引入头文件

```
#include <sw/redis++/redis++.h>
```

```
2   using namespace sw::redis;
```

2. 创建 Redis 连接

代码块

```
1   auto redis = Redis("tcp://127.0.0.1:6379");
```

- URL 格式为 `tcp://IP:Port`，不是 HTTP。
- 可设置连接选项，如超时、连接池大小。

3. 执行 Redis 命令

代码块

```
1   redis.ping();                    // 测试连接
2   redis.set("key", "value");       // 设置键值
3   auto val = redis.get("key");     // 获取键值
```

4. 编译运行

- Makefile 链接 `redis-plus-plus`、`hiredis`。
- 示例：

代码块

```
1   g++ main.cpp -o main -lredis++ -lhiredis -pthread
2   ./main
```

2. 调试与工具

- **VSCode 远程开发**
  - Remote-SSH 插件可直接在 Linux 服务器上编辑、编译、调试。
- **依赖库链接**
  - 必须确保：
    - `redis++`、`hiredis` 已安装
    - 系统线程库 `-pthread` 已链接
  - 否则会出现头文件或符号未定义错误。
- **常见问题排查**

- “头文件未找到” → 检查 CMake include 路径。
- “符号未定义” → 检查库文件链接顺序。

四、核心设计思想与工程实践

1. 分层封装的价值

- `hiredis`：
  - 专注高性能协议解析。
  - 提供基础 TCP 连接与 RESP 支持。
- `redis-plus-plus`：
  - 面向对象封装，C++ 风格。
  - 提供同步、异步、事务、管道接口。
- **优点**：底层性能与上层易用性兼顾。

2. 跨平台构建的必要性

- 使用 CMake 保证：
  - Linux / macOS / Windows 都可编译。
  - 编译配置可复用。
- 对企业项目至关重要，保证 CI/CD 和多环境兼容性。

3. 工程化细节

- **源码与构建目录分离**：
  - 避免中间文件污染源码。
- **命名规范**：
  - 类名、变量名清晰，便于团队协作。
- **依赖管理**：
  - 明确第三方库路径，保证编译可复现。
- **版本控制**：
  - 固定依赖版本，减少因更新引入的兼容性问题。

五、实践总结

1. **前置准备**：
   - 获取 GitHub 源码，保证网络顺畅或使用镜像。
   - 熟悉 `hiredis` 与 `redis-plus-plus` 的依赖关系。
2. **构建流程**：
   - CMake + Makefile，构建目录隔离，确保跨平台。
3. **开发调试**：
   - 引入头文件，建立连接，执行命令，调试连接问题。
   - 注意链接库顺序和线程库。
4. **设计思想**：
   - 分层封装，底层高性能，上层易用。
   - 跨平台构建，工程化规范。
   - 重视可维护性、团队协作和依赖管理。
