ZLToolKit笔记6：Session

2024年12月22日 14:57

首先需要了解Session在网络通信中的作用，以下前驱内容是由chatgpt生成的：

网络通信中Session的作用

在网络通信中，**Session（会话）**是指客户端和服务器之间在特定时间内建立的一种连续交互状态。它在很多网络协议（如HTTP、FTP等）以及系统设计中有着重要作用。以下是Session的主要作用和设计原因：

1. 需要Session的原因

1.1 网络通信是无状态的

- **无状态性**：网络协议（如HTTP）本质上是无状态的，每个请求是独立的，服务器无法直接关联多个请求。
- **问题**：在无状态通信中，服务器无法识别是否是同一个客户端发起的请求，或请求间的关联关系。

1.2 需要维护会话状态

- 某些应用需要在多个请求之间保持上下文（如登录状态、购物车、游戏状态等）。
- 例如：
  - 用户登录后，服务器需要记住用户身份，以便后续操作无需重复认证。
  - 在文件传输协议中，服务器需要知道当前传输的进度。

2. Session的作用

2.1 标识客户端

Session为每个客户端分配一个唯一标识（如Session ID），服务器通过这个标识来区分不同的客户端。

- **实现方式**：
  - 在客户端（如浏览器）存储Session ID（通过Cookie、URL参数等）。
  - 服务器根据Session ID找到对应的会话数据。

2.2 维持状态

Session可以存储用户的临时状态信息，例如：

- 登录用户的身份信息。
- 用户的购物车内容。
- 当前操作的上下文（如当前页面、步骤）。

2.3 数据共享

- 在同一个会话中，客户端可以多次访问服务器，服务器根据Session记录共享的数据。
- 例如，一个用户在电商网站中选择了商品A，在支付页面，服务器通过Session获取用户的购物车数据。

2.4 安全性

Session可以增强通信的安全性：

- 避免敏感信息直接暴露在网络中（如在Session中存储用户ID而不是直接传递）。
- 实现权限控制，服务器根据Session内容判断用户是否有权限执行某操作。

2.5 资源管理

- **短连接场景**：在HTTP短连接中，Session可以帮助服务器在多次连接间保持状态。
- **长连接场景**：在WebSocket或TCP连接中，Session用于管理长时间的通信状态（如数据传输状态、用户上下线状态）。

3. Session的实现与管理

3.1 Session的存储

Session数据通常由服务器维护，可以存储在：

1. **内存**：高效，但不适合大规模分布式系统。
2. **数据库**：持久化，适合大规模分布式系统，但读取效率稍低。
3. **分布式缓存**（如Redis）：高效，适合需要高性能和分布式的场景。

3.2 Session的标识

每个Session通常用一个唯一的Session ID标识：

- 生成方式：随机数、UUID、加密的哈希值等。
- 存储方式：
  - 客户端存储：通过Cookie或URL参数传递。
  - 服务器存储：通过Session ID找到对应会话数据。

3.3 超时与销毁

- **超时**：Session通常会设置有效期，超时后会自动销毁以节省服务器资源。
- **销毁**：用户主动退出登录或会话结束时，Session会被销毁。

4. 应用场景

1. **用户认证**
   - 用户登录后，服务器生成一个Session存储用户信息，后续请求通过Session验证身份。
2. **购物车系统**
   - 在线购物时，用户的购物车状态存储在Session中，避免在每次请求时重复传递。
3. **实时通信**
   - 在WebSocket或TCP通信中，Session维护用户的在线状态及通信上下文。
4. **多步骤操作**
   - 在复杂流程（如支付）中，Session记录当前步骤及用户的输入数据。
5. **负载均衡和分布式系统**
   - Session可以用作用户请求的路由依据，使请求被路由到存储其Session的服务器。

5. 总结

Session的核心作用是**维持状态、管理会话和提供安全性**。它弥补了网络通信无状态性的缺陷，使得服务器能够高效、灵活地处理多请求的复杂业务场景。
通过合理设计和管理Session，可以提升系统的**性能**、**可靠性**和**用户体验**。

我们来看ZLToolKit项目中session是如何实现的：
session用于长时间管理TCP连接，服务器端accept之后，就会为该连接创建一个session（用户可以自定义相应逻辑），创建session的过程由回调函数_session_alloc完成。这在笔记5的最后部分已经介绍过了。

```
_session_alloc = [cb](const TcpServer::Ptr &server, const Socket::Ptr &sock) {
            auto session = std::shared_ptr<SessionType>(new SessionType(sock), []
(SessionType *ptr) {
                TraceP(static_cast<Session *>(ptr)) << "~" << cls_name;
                delete ptr;
            });
            if (cb) {
                cb(session);
            }
            TraceP(static_cast<Session *>(session.get())) << cls_name;
            session->setOnCreateSocket(server->_on_create_socket);
            return std::make_shared<SessionHelper>(server, std::move(session),
cls_name);
        };
```

_session_alloc回调函数返回的是一个智能指针对象，管理SessionHelper类资源。

代码中SessionType可由用户自定义，继承于session类。而session继承于SocketHelper类。