



HP 소켓

고성능 네트워크 통신 프레임워크

버전 - 5.9.2

브루스 리양
2023-07-01

머리말

HP-Socket은 서버 구성 요소, 클라이언트 구성 요소 및 에이전트 구성 요소를 포함하는 일반적인 고성능 TCP/UDP/HTTP 통신 프레임워크로 다양한 애플리케이션 시나리오에서 TCP/UDP/HTTP 통신 시스템에 광범위하게 적용할 수 있으며 C/C++, C#, Delphi를 제공합니다. , E(쉬운 언어), Java, Python 및 기타 프로그래밍 언어 인터페이스.

HP-Socket은 통신 계층을 완전히 캡슐화하고 응용 프로그램은 통신 계층의 세부 사항에 주의를 기울일 필요가 없습니다. 그리고 오래된 응용 프로그램. 사용자가 HP-Socket을 쉽고 빠르게 배우고 사용할 수 있도록 프레임워크의 디자인 아이디어와 사용 방법을 빠르게 파악할 수 있도록 많은 데모 예제(예: PUSH 모델 예제, PULL 모델 예제, PACK 모델 예제 등), 성능 테스트 예제 및 기타 프로그래밍 언어 예제). HP-Socket은 현재 Windows 및 Linux 플랫폼을 지원합니다.

다목적성

HP-Socket의 유일한 책임은 바이트 스트림을 송수신하는 것이며 응용 프로그램의 프로토콜 분석에는 참여하지 않습니다.

HP-Socket은 인터페이스를 통해 애플리케이션과 상호 작용하고 완전히 분리됩니다. HP를 구현하는 모든 애플리케이션 소켓 인터페이스 사양은 HP-Socket을 원활하게 통합할 수 있습니다.

사용의 용이성

모든 공통 프레임워크는 사용의 용이성이 중요하며, 사용하기 어렵다면 처음부터 새로 작성하는 것이 좋습니다. 따라서 HP-Socket의 인터페이스는 매우 단순하고 통합되도록 설계되었습니다. HP-

Socket은 모든 기본 통신 세부 정보를 완전히 캡슐화하며 응용 프로그램은 기본 통신 작업에 개입할 필요가 없으며 개입할 수도 없습니다. 통신 연결은 연결 ID로 추상화되며 연결 ID는 다른 연결을 처리하기 위해 연결의 고유 식별자로 응용 프로그램에 제공됩니다. HP-Socket은

PUSH / PULL / PACK 등의 수신 모델을 제공하며, 응용 프로그램은 encapsulation과 unpacking을 수동, 반자동, 전자동 방식으로 유연하게 선택할 수 있으며, PULL / PACK 수신 모델은 복잡도를 줄일 수 있다. 캡슐화 및 포장 풀기 처리하면서 오류 가능성을 크게 줄입니다.

고성능 서버

구성 요소: IOCP / EPOLL 통신 모델을 기반으로 버퍼 풀, 개인 힙 및 기타 기술과 결합하여 높은 성능을 달성합니다.

초대형, 동시성 통신 시나리오를 지원하는 효율적인 메모리 관리.

에이전트 구성 요소: 에이전트 구성 요소는 본질적으로 다중 클라이언트 구성 요소이며 서버 구성 요소와 동일한 기술 아키텍처를 사용합니다. 에이전트 구성 요소 개체는 대규모 소켓 연결을 동시에 설정하고 효율적으로 처리할 수 있습니다. 클라이

언트 구성 요소: Event Select / POLL 통신 모델을 기반으로 각 구성 요소 개체는 통신 스레드를 생성하고 소규모 클라이언트 시나리오에 적합한 소켓 연결을 관리합니다.

확장성 응용 프

로그램은 다양한 용량 요구 사항, 통신 규모 및 자원 상태에 따라 HP-Socket의 다양한 성능 매개변수(예: 작업 스레드 수, 버퍼 풀 크기, 전송 모드 및 수신 모드 등)를 조정할 수 있습니다. 및 기타 현실적인 시나리오, 리소스를 과도하게 낭비하지 않고 애플리케이션 요구 사항을 충족하도록 리소스 할당을 최적화합니다.

목차

서문	1
목차.....	2
1. 개요.....	4
1.1 전체적인 구조.....	4
1.2 부품 분류.....	6
1.3 컴포넌트 인터페이스.....	7
1.4 리스너 인터페이스	10
2 프레임워크 세부 정보.....	14
2.1 주요 개념	14
2.1.1 수신 모델.....	14
2.1.2 전송 정책	17
2.1.3 OnSend 동기화 전략	17
2.1.4 주소 재사용 정책	18
2.1.5 연결 방법.....	19
2.1.6 연결 바인딩	20
2.1.7 정지 대기 중.....	스물 둘
2.2 서버 구성 요소.....	스물 셋
2.2.1 인터페이스 설명.....	스물셋
2.2.2 작업 과정.....	27
2.3 에이전트 구성 요소.....	29
2.3.1 인터페이스 설명.....	29
2.3.2 작업 흐름.....	32
2.4 클라이언트 구성요소.....	33
2.4.1 인터페이스 설명.....	33
2.4.2 작업 과정.....	36
2.5 UDP 노드 구성 요소.....	37
2.5.1 인터페이스 설명.....	37
2.5.2 작업 과정.....	39
3 SSL.....	41
3.1 컴포넌트 인터페이스.....	41
3.2 SSL 운영 환경	43
3.3 SSL 약수.....	45
4 HTTP.....	46
4.1 컴포넌트 인터페이스.....	46
4.2 리스너 이벤트.....	49
4.3 쿠키 관리.....	53
4.4 HTTP 통신 시작	57
5 UDP ARQ.....	58
5.1 컴포넌트 인터페이스.....	58
5.2 핸드셰이크 프로토콜	59
5.3 구성 매개변수.....	60
6 스레드 풀.....	62

6.1	컴포넌트 인터페이스.....	62
6.2	리스너 이벤트.....	63
7	압축/압축 해제.....	65
7.1	컴포넌트 인터페이스.....	65
7.1.1	압축기 어셈블리 인터페이스.....	65
7.1.2	압축해제기 구성요소 인터페이스	66
7.1.3	출력 데이터 콜백.....	67
7.2	개체 생성	67
8	리눅스.....	70
8.1	컴파일.....	70
8.2	설치하다.....	71
8.3	안드로이드 NDK.....	71
8.3.1	마치다.....	71
8.3.2	기능 스위치.....	72
8.3.3	다른 옵션.....	72
9	사용하는 방법.....	73
9.1	소스 코드.....	73
9.2	정적 라이브러리.....	73
9.3	HP 소켓 DLL.....	73
9.4	HPSocket4C DLL	74
9.5	확장 지원	75
10	부록.....	76
10.1	예제 데모.....	76
10.1.1	윈도우 예시.....	76
10.1.2	리눅스 예제.....	77
10.2	도우미 기능.....	79
10.3	자주하는 질문.....	81



개요 1 개

1.1 전체 아키텍처

HP-Socket은 기본 통신 세부 정보를 완전히 캡슐화하고 기본 통신과 완전히 독립적인 사용하기 쉬운 API 인터페이스 세트를 애플리케이션에 제공하여 애플리케이션이 처리할 필요 없이 고성능, 고확장성 통신을 얻을 수 있도록 합니다. 통신 세부 사항 부담. HP-Socket의 API 인터페이스 모델은 그림 1.1-1에 나와 있습니다.

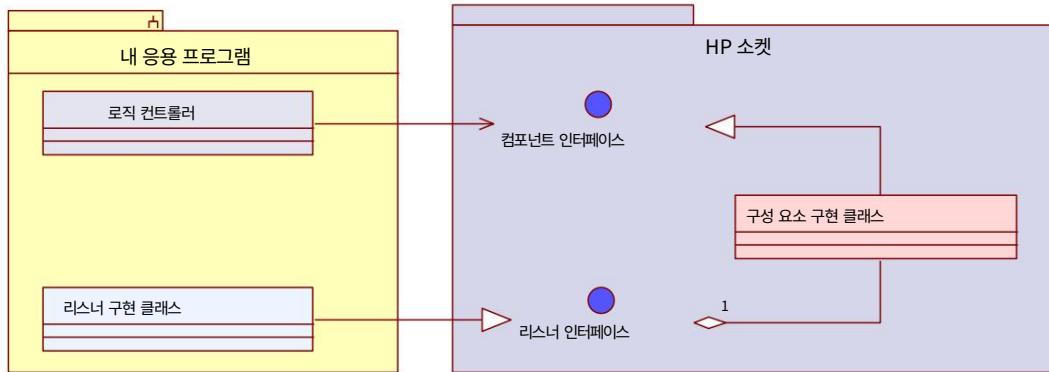


그림 1.1-1 HP-Socket API 인터페이스 모델

HP-Socket은 구성 요소 인터페이스(예: ITcpServer / IUdpClient), 구성 요소 구현 클래스(예: CTcpServer)를 정의합니다. / CUdpClient) 및 리스너 인터페이스(예: ITcpServerListener / IUdpClientListener), 여기서:

컴포넌트 인터페이스: 컴포넌트의 동작 방식을 선언하고, 애플리케이션은 이 인터페이스를 통해 컴포넌트 객체를 생성하여 컴포넌트를 사용
컴포넌트 구현 클래스: 컴포넌트 인터페이스를 구현하여 실제 통신 처리 작업을 수행하고, 통신 이벤트를 리스너에게 보고
리스너 인터페이스: 컴포넌트 선언 통신 이벤트 콜백 방식

각 컴포넌트 객체는 리스너 객체(리스너 객체의 구현 클래스는 애플리케이션에 의해 정의됨)와 연결되며, 컴포넌트 객체가 통신 이벤트를 트리거할 때 리스너 객체의 해당 콜백 메서드를 호출하고 애플리케이션이 프로세스를 처리합니다. 콜백 메소드 로직의 애플리케이션 비즈니스. 그림 1.1-2는 TCP 에이전트를 예로 들어 구성 요소와 애플리케이션 간의 상호 작용을 보여줍니다.

응용 프로그램은 먼저 수신기 객체와 TCP 에이전트 객체를 만들고 TCP 에이전트 객체를 만들 때 수신기 객체를 전달하고 TCP 에이전트 객체를 수신기 객체와 연결합니다. TCP Agent 객체가 생성된 후 응용프로그램은 TCP Agent 인터페이스 메소드를 호출하여 TCP Agent 객체를 동작시킨다(예: Start/Connect/Send/Stop 등). TCP 에이전트 객체가 통신 이벤트를 트리거하면 리스너 객체의 콜백 메서드(예: OnConnect / OnSend / OnReceive / OnClose 등)를 호출하여 애플리케이션에 알립니다.

참고: 수신기 객체의 비동기 콜백 메서드는 구성 요소의 통신 스레드에서 실행되므로 콜백 메서드는 시간이 많이 걸리는 비즈니스 논리 코드를 실행하지 않아야 하며 동시에 다른 스레드 동기화 문제에 주의하고 자율식을 사용하지 마십시오.

HP 소켓 "연결 바인딩"을 설정하면 응용 프로그램이 다중 스레드 동기화 및 상호 배제 잠금으로 인해 발생하는 복잡성 및 성능 문제를 피할 수 있습니다.

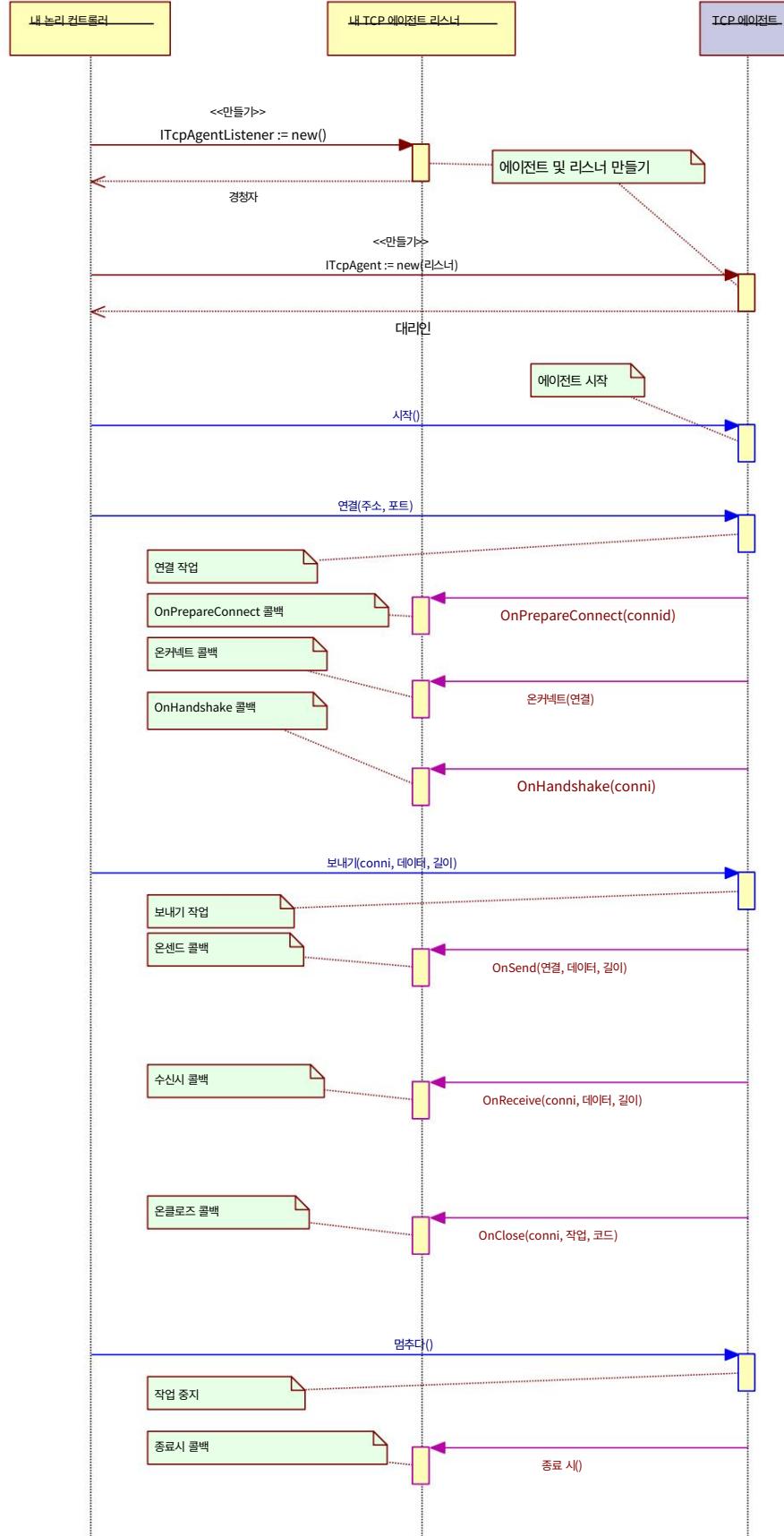
JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

그림 1.1-2 TCP Agent와 애플리케이션 간의 상호작용 예시

1.2 구성요소 분류

HP-Socket에는 31개의 구성 요소가 포함되어 있습니다(9개의 SSL 구성 요소는 3장, 8개의 HTTP 구성 요소에서 자세히 설명함). 통신 역할(클라이언트/ 서버), 통신 프로토콜(TCP/UDP/HTTP) 및 분류를 위해 모델(PUSH / PULL / PACK)을 받습니다. 표 1.2-1에는 이름, 인터페이스, 리스너 인터페이스, 구현 클래스 및 해당 분류:

이름	구성 요소 인터페이스 리스너 인터페이스	구현하다 수업	역할 프로토콜	수신 모델
TCP 서버	ITcp서버 ITcpServerListener	CTcp서버	섬기는 사람	TCP 푸시
TCP 풀 서버	ITcp풀서버 ITcpServerListener	CTcpPullServer	섬기는 사람	TCP 풀
TCP 팩 서버	ITcpPackServer ITcpServerListener	CTcpPack서버 서버		TCP 팩
UDP 서버	IUdpServer IUdpServerListener	CUdp서버	서버 UDP 푸시	
UDP ARQ 서버	IUdpArqServer IUdpServerListener	CUdpArqServer 서버	UDP PUSH	
TCP 에이전트	ITcpAgent ITcpServerListener	CTcpAgent	고객	TCP 푸시
TCP 풀 에이전트	ITcp풀 에이전트 ITcpAgentListener	CTcpPullAgent	고객	TCP 풀
TCP 팩 에이전트	ITcpPackAgent ITcpServerListener	CTcpPackAgent	고객	TCP 팩
TCP 클라이언트	ITcp클라이언트 ITcp클라이언트 리스너	CTcp클라이언트	고객	TCP 푸시
TCP 풀 클라이언트	ITcp풀클라이언트 ITcp클라이언트 리스너	CTcp풀클라이언트	고객	TCP 풀
TCP 팩 클라이언트	ITcpPack클라이언트 ITcp클라이언트 리스너	CTcpPack클라이언트	고객	TCP 팩
UDP 클라이언트	IUdp클라이언트 IUdpClientListener	CUdp클라이언트	클라이언트 UDP 푸시	
UDPARQ 클라이언트	IUdpArq클라이언트 IUdpClientListener	CUdpArq클라이언트	클라이언트 UDP 푸시	
UDP 캐스트	IUdpCast IUdpCastListener	CUdpCast	클라이언트 UDP 푸시	
UDP 노드	IUdp노드 IUdpNodeListener	CUdp노드	고객 섬기는 사람	UDP 푸시
스레드 풀	IHP스레드풀	CHP스레드풀	--	--

표 1.2-1 구성 요소의 분류

- ✓ Agent 구성 요소는 본질적으로 Client 구성 요소로서 하나의 Agent 개체가 동시에 여러 클라이언트 연결을 관리할 수 있음
- ✓ 실제 사용 시나리오에 따라 TCP 및 HTTP 기반 Agent 구성 요소만 HP-Socket에 구현됨 ✓ UDP Cast 구성 요소는 멀티캐스트 및 브로드캐스트 UDP 설계를 위한 것으로 특수 클라이언트 구성 요소로 간주될 수 있음 ✓ UDP 노드 구성 요소는 독립적인 연결 없는 UDP 인터페이스를 제공하고 동시에 유니캐스트, 멀티캐스트 및 브로드캐스트 모드를 지원합니다. ✓ TCP 기반 구성 요소는 PUSH, PULL 및 3개의 수신 모델을 각각 PACK ✓ Thread Pool 구성 요소는 HP-Socket에 의해 구현된 효율적이고 사용하기 쉬운 스레드 풀 구성 요소입니다.

1.3 컴포넌트 인터페이스

Server, Agent, Client의 컴포넌트 인터페이스 정의는 그림 1.3-1 ~ 1.3-3과 같으며, 컴포넌트 인터페이스는 컴포넌트가 제공하는 모든 동작 방식을 정의한다. 그 중 PULL 모델 인터페이스는 **Fetch(dwConnID, pData, iDataLength)** 메서드를 제공하는 **IPullSocket / IPullClient** 인터페이스에서 상속 되어 애플리케이션이 구성 요소에서 데이터를 가져올 수 있도록 합니다.



프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

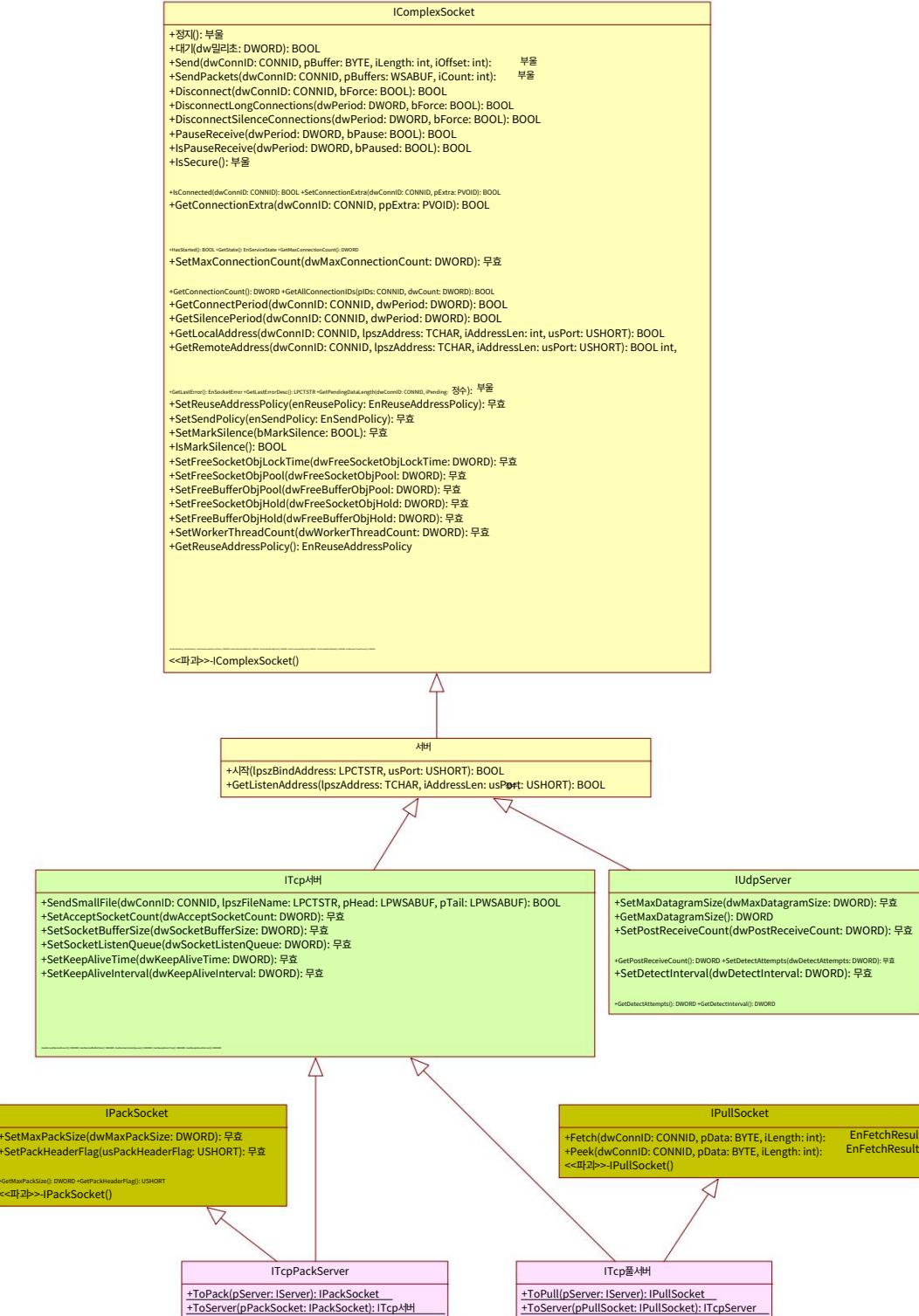


그림 1.3-1 서버 구성 요소 인터페이스



프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

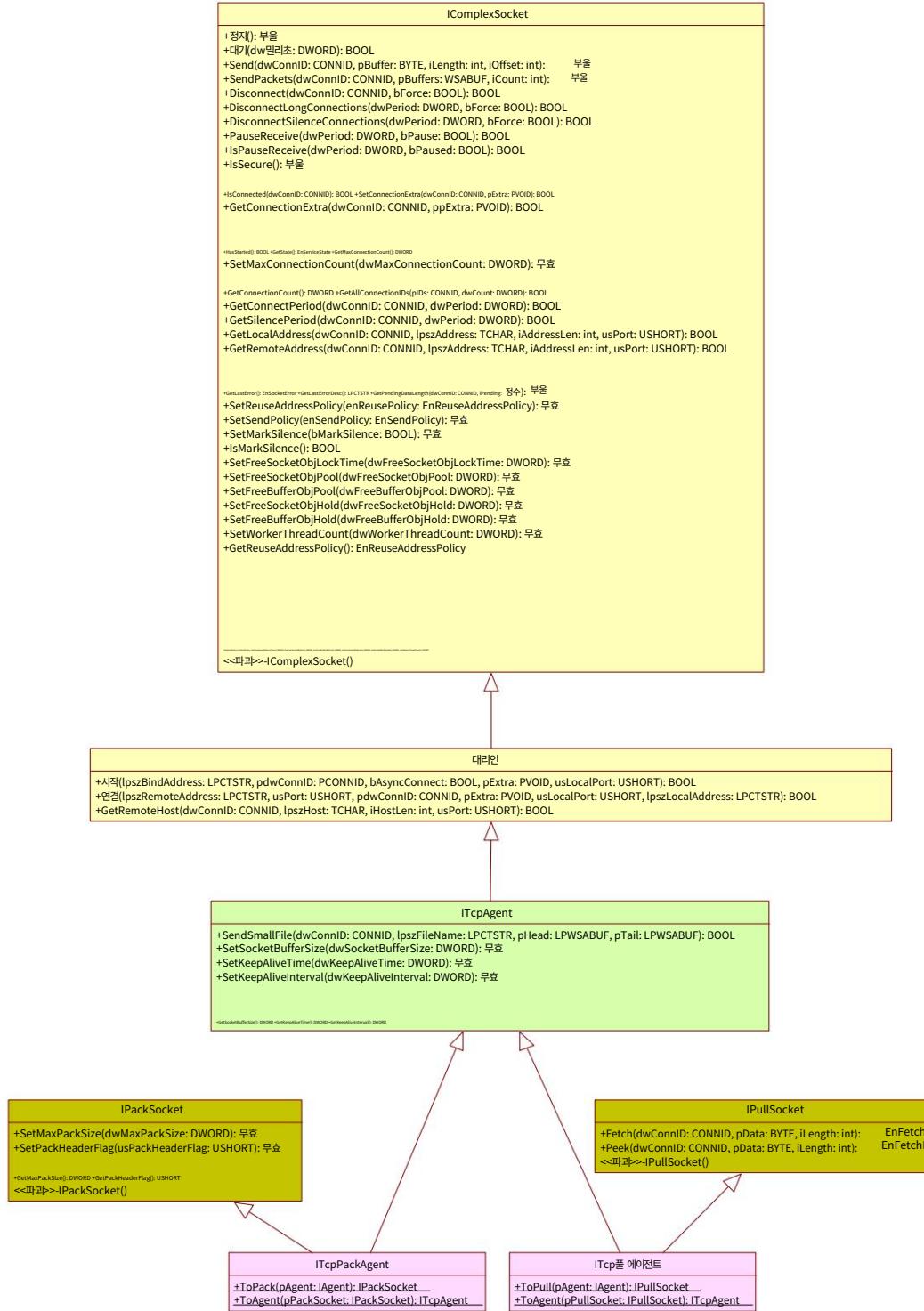


그림 1.3-2 에이전트 구성요소 인터페이스

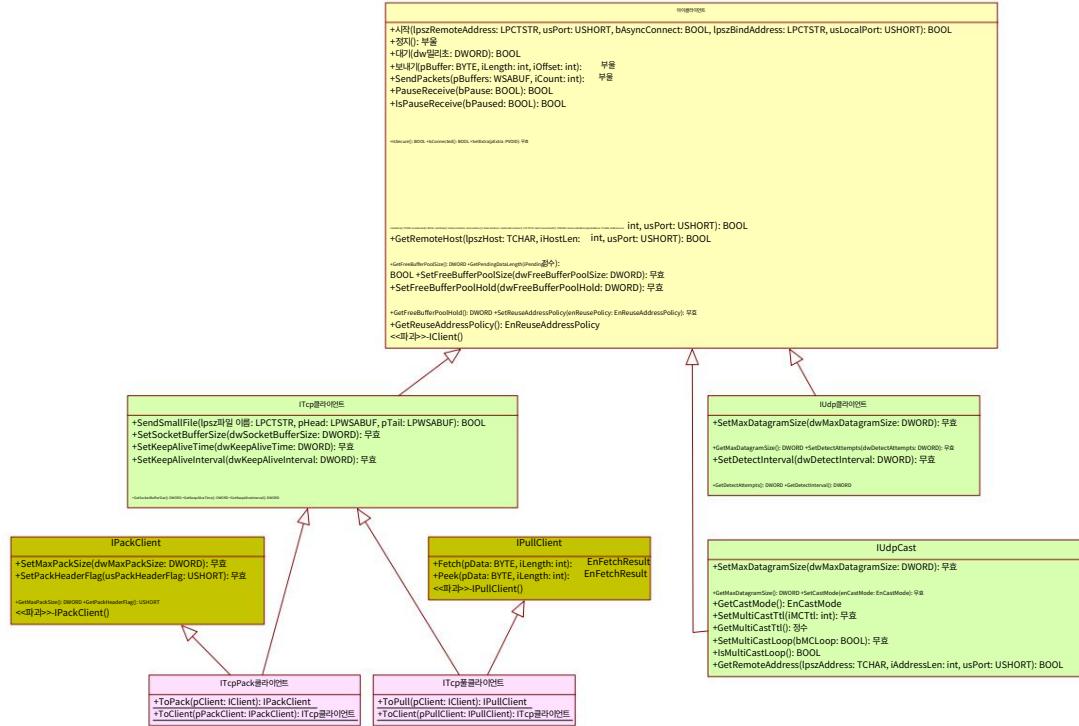


그림 1.3-3 클라이언트 구성 요소 인터페이스

1.4 리스너 인터페이스

서버, 에이전트 및 클라이언트의 수신기 인터페이스 정의는 그림 1.4-1 - 1.4-3에 나와 있습니다.

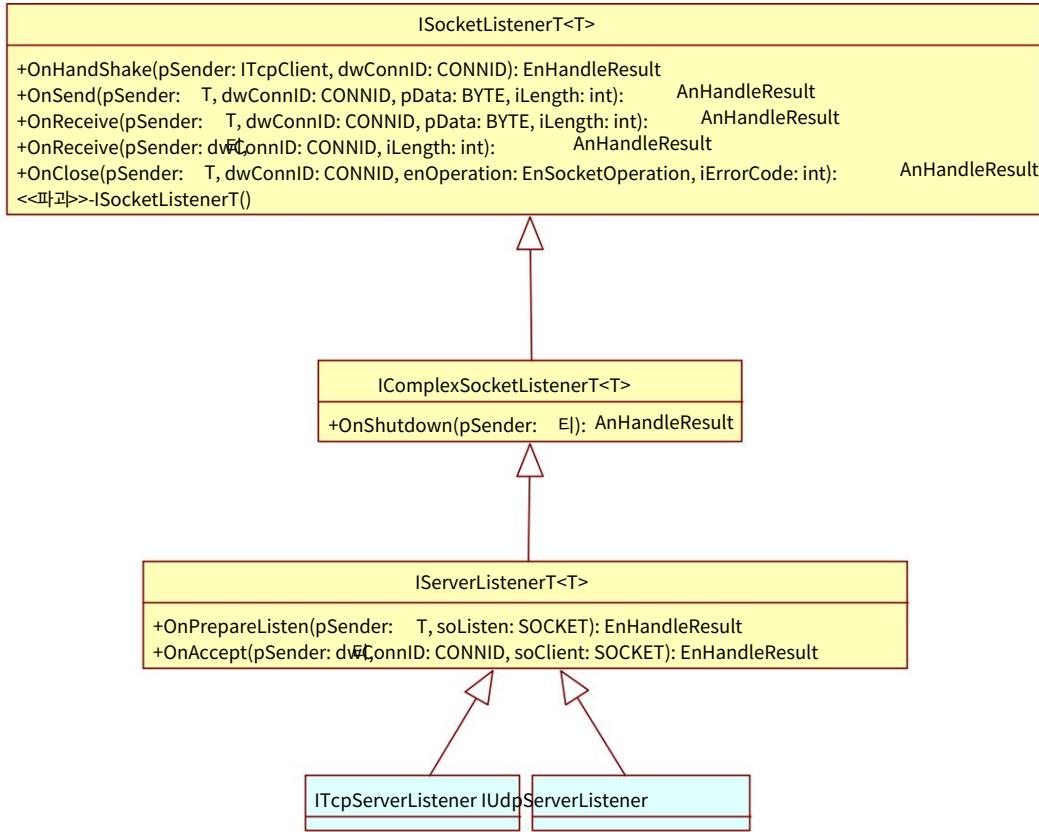


Gráfico 1.4-1: Interface de escuta do servidor

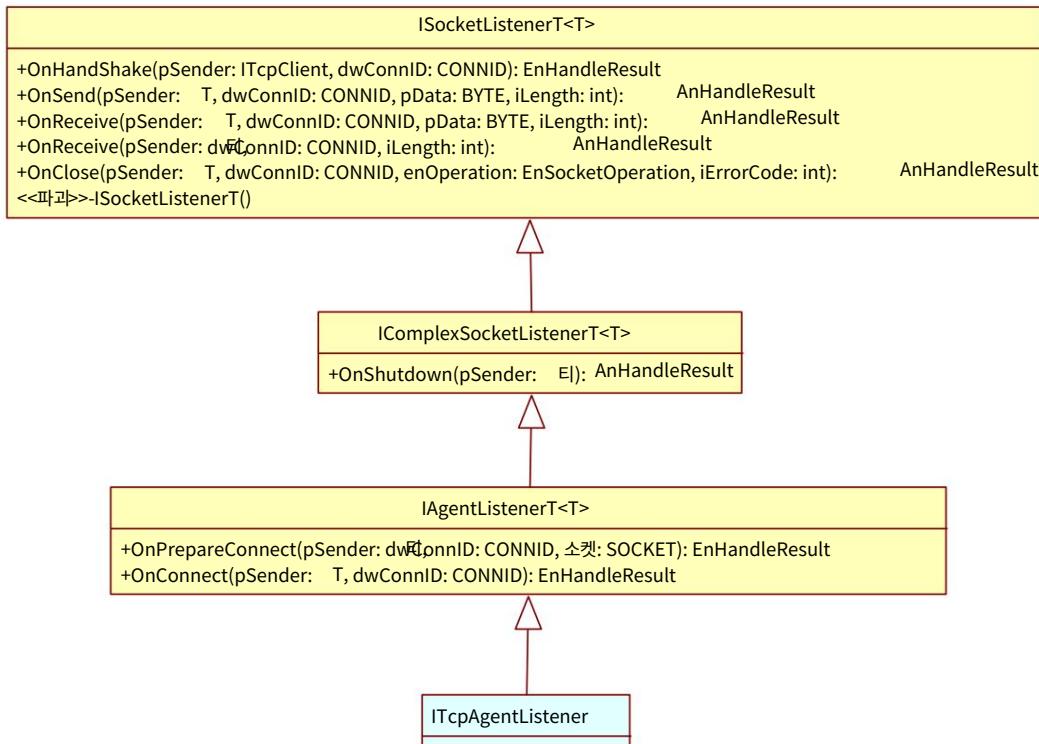


Gráfico 1.4-2: Interface de escuta do agente

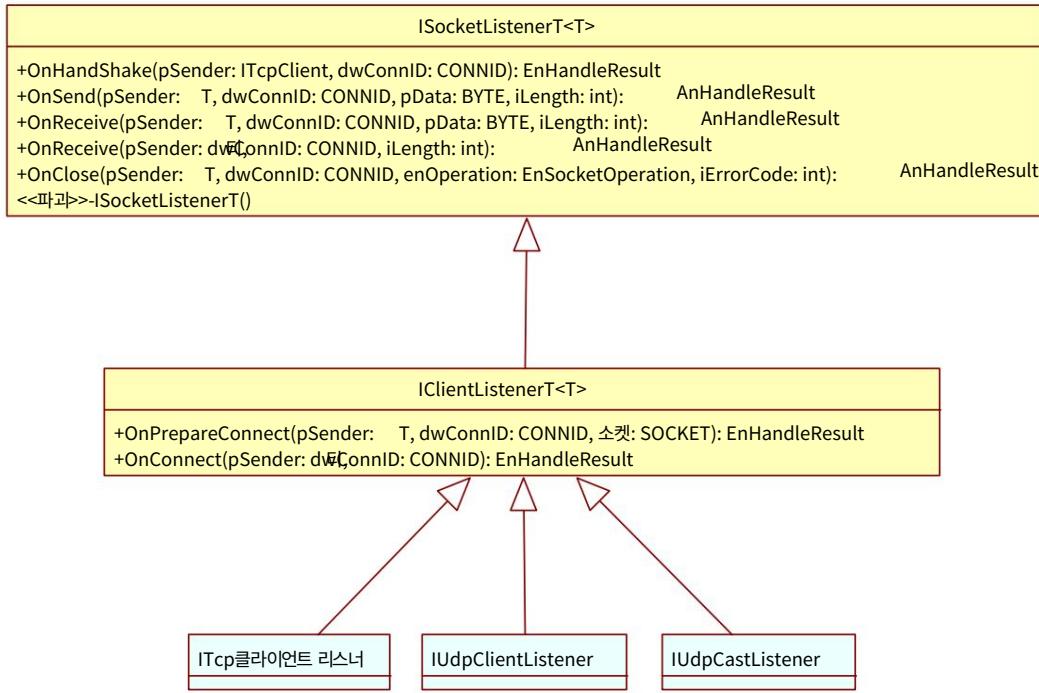


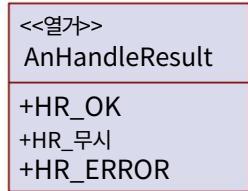
그림 1.4-3 클라이언트 리스너 인터페이스

HP-Socket은 PUSH/PULL/PACK 모델 구성 요소에 대해 별도의 리스너 인터페이스를 정의하지 않고 해당하는 동일한 리스너 인터페이스, 차이점은 다음과 같습니다. PUSH 및 PACK 모델 구성 요소는 데이터를 수신할 때 리스너 개체를 트리거합니다. OnReceive(pSender, dwConnID, pData, iLength) 이벤트, PULL 모델 구성 요소는 데이터를 수신할 때 트리거됩니다. 수신기 개체의 OnReceive(pSender, dwConnID, iLength) 이벤트를 보냅니다. 이벤트의 의미는 표 1.4-1에 나와 있습니다.

상호 작용	이벤트	설명하다
ISocketListenerT	OnHandShake(pSender, dwConnID)	악수 완료
	OnSend(pSender, dwConnID, pData, iLength)	전송된 데이터
	OnReceive(pSender, dwConnID, pData, iLength)	데이터 도착 (푸시)
	OnReceive(pSender, dwConnID, iLength)	데이터 도착 (당기다)
	OnClose(pSender, dwConnID, enOperation, iErrorCode)	연결이 닫힙니다.
IComplexSocketListenerT	OnShutdown(pSender)	닫기 구성 요소
IServerListenerT	OnPrepareListen(pSender, soListen)	들을 준비가
	OnAccept(pSender, dwConnID, soClient)	연결 수락
IAgentListenerT	OnPrepareConnect(pSender, dwConnID, 소켓)	연결 준비
	온커넥트(pSender, dwConnID)	완전한 연결
IClientListenerT	OnPrepareConnect(pSender, dwConnID, 소켓)	연결 준비
	온커넥트(pSender, dwConnID)	완전한 연결

표 1.4-1 리스너 인터페이스 이벤트

리스너 이벤트 콜백 메서드의 반환 값 유형은 EnHandleResult입니다.



- ✓ HR_OK : 성공적으로 처리됨
- ✓ HR_IGNORE : 처리 무시
- ✓ HR_ERROR : 처리 실패

참고: 이벤트 OnReceive / OnPrepareListen / OnAccept / OnPrepareConnect / OnConnect / 온핸드쉐이크 콜백 메서드가 반환되면 구성 요소가 종료 연결 해제됩니다.

비 구성 요소 및 SSL 구성 요소에 대한 처리를 일관되게 만들기 위해 시작, 비 구성 요소 HP 소켓 v4.0.x
이벤트도 발생합니다. 따라서 구성 요소가 이벤트를 수신하면 연결이 설정되었음을 알립니다.
그리고 소통을 시작합니다.

2 프레임워크 세부 정보

2.1 핵심 개념

2.1.1 수신 모델

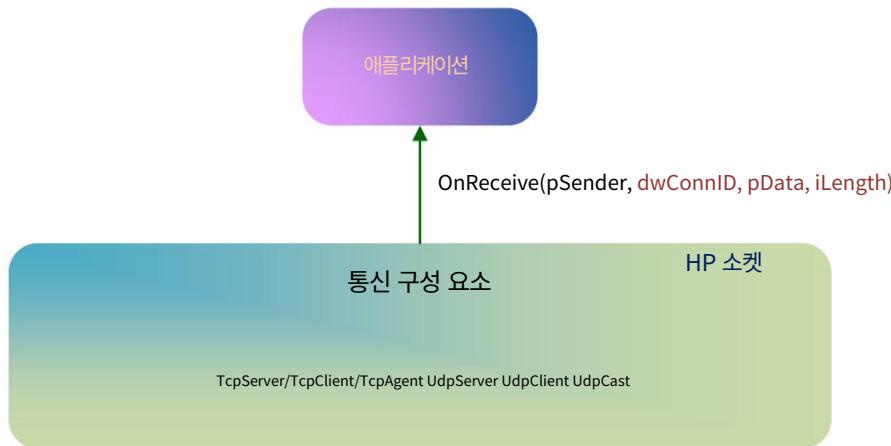
표 1.2-1과 같이 HP-Socket의 TCP 구성 요소는 PUSH, PULL 및 PACK의 세 가지 수신 모델을 지원합니다.

PUSH 모델: 구성 요소가 데이터를 수신하면 OnReceive(pSender, dwConnID, pData, iLength) 데이터를 응용 프로그램에 "푸시"하는 이벤트입니다.

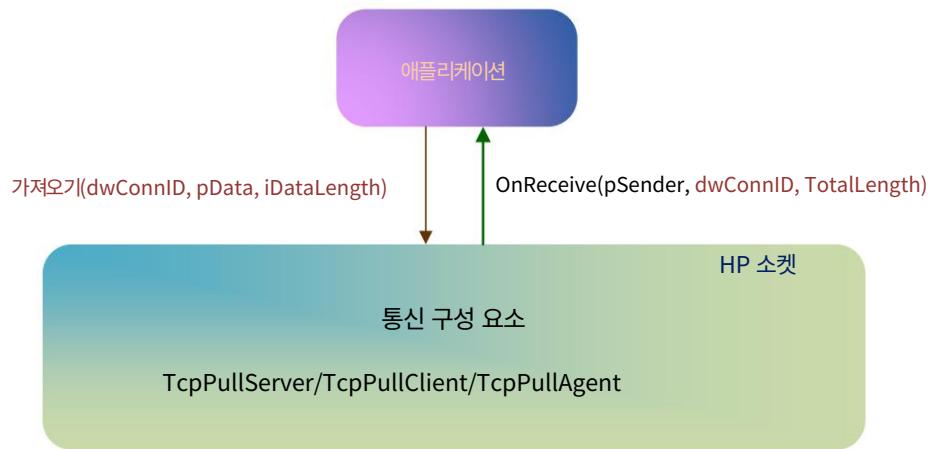
PULL 모델: 구성 요소가 데이터를 수신하면 수신기 객체의 OnReceive(pSender, dwConnID, iTotallength) 이벤트를 트리거 하여 지금까지 얼마나 많은 데이터를 수신했는지 애플리케이션에 알리고 애플리케이션은 데이터의 길이를 확인하고 요구 사항을 충족하는 경우 구성 요소의 Fetch를 호출합니다. (dwConnID, pData, iDataLength) 메서드는 필요한 데이터를 "가져옵니다". PACK 모델: PACK 모델 계열 구성 요소는 PUSH 및 PULL 모델의 조합입니다. 응용 프로그램은 하위 패키지(예: PUSH) 및 데이터 캡처(예: PULL)를 처리할 필요가 없습니다. 구성 요소는 각 OnReceive 이벤트를 보장합니다. 완전한 패킷을 제공합니다.

세 가지 모델의 비교는 그림 2.1.1-1에 나와 있습니다.

푸시 모델



풀 모델



팩 모델

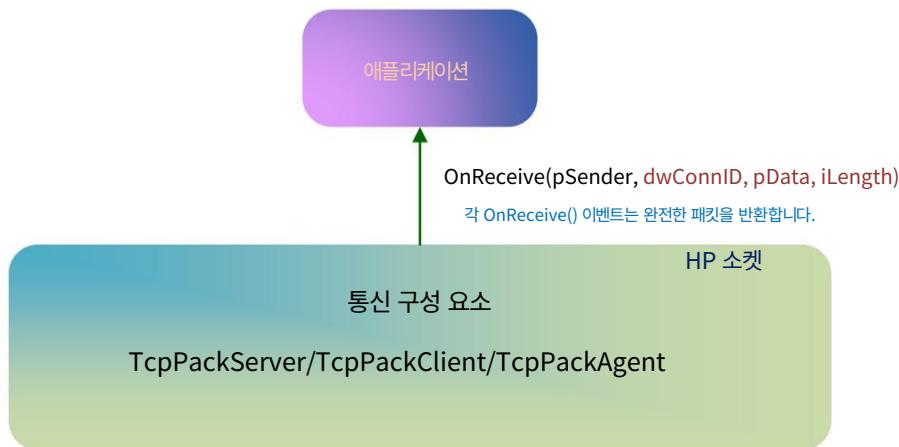


그림 2.1.1-1 수신기 모델

PUSH 모델 구성요소가 리스너 객체의 OnReceive(pSender, dwConnID, pData, iLength) 이벤트를 트리거 하면 애플리케이션은 고정 패킷 처리, 프로토콜 분석 등과 같이 수신된 데이터를 즉시 처리해야 합니다. 구성 요소는 응용 프로그램 계층에서 데이터 처리를 지원하지 않습니다.

PULL 모델 구성 요소가 수신기 객체의 OnReceive(pSender, dwConnID, iTotalLength) 이벤트를 트리거 하면 응용 프로그램은 수신된 데이터 길이(iTotalLength)가 응용 계층 프로토콜에 따라 처리 조건을 충족하는지 여부를 감지하고 선택적으로 처리합니다. iTotallength가 현재 예상 길이보다 작으면 이 이벤트를 무시할 수 있습니다. iTotallength 가 현재 예상 길이보다 크거나 같으면 구성 요소의 Fetch(dwConnID, pData, iDataLength) 메서드가 루프에서 호출되어 남은 데이터까지 필요한 데이터 길이가 현재 예상 길이보다 짧습니다.

Fetch(dwConnID, pData, iDataLength) 메서드의 반환 값 유형은 EnFetchResult입니다.

<code><<열가>></code>
<code>EnFetchResult</code>
<code>+FR_확인</code>
<code>+FR_LENGTH_TOO_LONG</code>
<code>+FR_DATA_NOT_FOUND</code>

✓ FR_OK ✓ : 가져오기 성공

HR_LENGTH_TOO_LONG : 가져온 길이가 실제 데이터 길이를 초과함 ✓ HR_DATA_NOT_FOUND : 가져올 데이터가 없으므로 연결이 끊어졌을 수 있음

참고: Fetch(dwConnID, pData, iDataLength) 메서드가 FR_OK 를 반환하는 경우에만 데이터가 추출됩니다. 또한 PULL 모델 구성 요소는 버퍼 데이터를 제거하지 않는 수신 버퍼에서 엿보기를 위한 Peek(dwConnID, pData, iDataLength) 메서드도 제공합니다.

PULL 모델은 응용 계층 프로토콜을 완전히 인식하는 데 적합하며 응용 계층 프로토콜은 다음을 알 수 있습니다.

패킷 길이의 시나리오. 일반적인 시나리오는 Head + Body이고, Head의 길이는 고정되어 있고, 첫 번째 데이터 패킷은 Head이고, Body의 길이는 Head를 통해 알고, 다음 데이터 패킷은 Body를 수신한 후 Head가 되어야 합니다.

참고: 모델과 응용 채팅 프로토콜 간의 상호 협력을 통해 응용 프로그램은 고정 패키지 처리 및 분할 패키지 작업에서 면제되어 응용 프로그램의 부담을 줄일 수 있습니다.

PACK 모델 구성 요소가 수신기 객체의 OnReceive(pSender, dwConnID, pData, iLength) 이벤트를 트리거하면 pData 가 완전한 데이터 패킷임을 확인합니다. PACK 모델 구성 요소는 응용 프로그램에서 보낸 각 데이터 패킷에 4바이트(32비트) 패킷 헤더를 자동으로 추가합니다. 구성 요소가 데이터를 수신하면 패킷 헤더 정보에 따라 자동으로 패킷을 분할합니다. 각 완전한 데이터 패킷은 OnReceive 이벤트를 통해 애플리케이션으로 전송됩니다..

PACK 헤더 형식:

XXXXXXXXXXXXYYYYYYYYYYYYYYYYYYYY

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcdsaa/HP-Socket>

처음 10개의 X는 데이터 패킷 확인을 위한 헤더 식별 비트입니다. 유효한 패킷 헤더 식별 값 범위는 0 ~ 1023(0x3FF)이며, 패킷 헤더 플래그가 0이면 패킷 헤더를 확인하지 않습니다. 마지막 22비트 Y는 패킷 본문의 길이를 기록하는 길이 비트입니다. 유효 패킷 최대값 길이는 4194303(0x3FFFFF)바이트를 초과할 수 없으며 기본 길이 제한은 262144(0x40000)바이트입니다. 답변 프로그램은 `SetPackHeaderFlag()` 및 `SetMaxPackSize()` 를 통해 각각 패킷 헤더 플래그와 최대 패킷 길이를 설정할 수 있습니다. 체계.

2.1.2 전송 전략

IClient 계열 컴포넌트 의 경우 애플리케이션이 컴포넌트의 `Send()`를 호출할 때 、 `SendPackets()` 、 작은 파일 보내기() 메서드가 데이터를 보낼 때 구성 요소는 데이터를 내부에 캐시하고 적절한 시간에 보냅니다.
IServer 및 **IAgent** 시리즈 구성 요소 의 경우 애플리케이션이 구성 요소의 `Send()`를 호출할 때 、 보내기패킷() 、 `SendSmallFile()` 메소드가 데이터를 전송할 때 서로 다른 전송 전략에 따라 서로 다른 처리 방법이 있습니다.

(다음을 통해 정책 보내기 `SetSendPolicy(enSendPolicy)` 설정 방법)

✓ SP_PACK : 패킹 전략(디폴트)

여러 전송 작업의 데이터를 결합하고 함께 전송하여 전송 효율성을 높이십시오.

✓ SP_SAFE : 보안 정책

여러 전송 작업의 데이터를 결합하여 함께 전송하고 버퍼 오버플로를 피하십시오.

✓ SP_DIRECT : 직접 전략

각 전송 작업은 직접 전달되므로 부하가 적지만 실시간 요구 사항이 높은 경우에 적합합니다.

메모: SP_DIRECT 일반적으로 TCP_NODELAY 소켓 가장 낮은 지연을 얻기 위한 옵션
시간. (통과하다 `SetSendPolicy(SP_DIRECT)` 협력 `SetNoDelay(TRUE)` 성취하다)

SP_PACK 및 SP_SAFE 전략 의 경우 구성 요소는 전송할 데이터를 내부적으로 버퍼링합니다. 또한 응용 프로그램은 구성 요소의 `GetPendingDataLength(dwConnID, iPending)` 메서드를 호출하여 지정된 연결의 미전송 데이터 가져오기 흐름 제어를 달성하기 위해 불륨.

참고: 모델의 특성에 따라 통신 구성 요소의 전송 전략은 리눅스 플랫폼의 통신 구성 요소는 전송 정책 설정을 지원하지 않으므로 오른리눅스 통신 구성 요소에 영향을 미치지 않습니다 SP_팩 . 즉 말하자면, `SetSendPolicy(enSendPolicy)` 방법 쪽 리눅스 다.

2.1.3 OnSend 동기화 전략

HP-Socket v5.4.2 부터 IServer 및 IAgent 시리즈 구성 요소는 `OnSend` 이벤트에 대한 동기화 정책 설정을 지원합니다. 약간.

온센드 이벤트 동기화 정책 통과 SetOnSendSyncPolicy(enSyncPolicy) 설정 방법)

<<열거>>
EnOnSendSyncPolicy
+OSSP_NONE
+OSSP_CLOSE
+OSSP_RECEIVE

✓ OSSP_NONE : 동기화 없음(기본값)

OnSend 이벤트는 동기화되지 않으며 OnReceive 및 OnClose 이벤트가 동시에 트리거될 수 있습니다.

✓ OSSP_CLOSE : 동기 OnClose

OnClose 이벤트만 동기화되며 OnReceive 이벤트가 동시에 트리거될 수 있습니다.

✓ OSSP_RECEIVE : 동기 OnReceive

(TCP 구성 요소에만 해당) OnReceive 및 OnClose 이벤트를 동기화합니다. 동시에 트리거할 수 없습니다.

OnReceive 또는 OnClose 이벤트.

OnSend 이벤트는 일반 응용 프로그램에 거의 중요하지 않으므로 기본 동기화 전략 OSSP_NONE이 채택 됩니다 .

예, 이 경우 OnSend 이벤트는 스레드로부터 안전하지 않으며 OnSend 이벤트를 처리하는 동안 동시에 트리거될 수 있습니다.

OnReceive 또는 OnClose 이벤트 보내기 , OSSP_CLOSE 동기화 전략은 OnSend 이벤트 가 처리 되도록 합니다.

OnClose 이벤트를 트리거하는 것은 불가능합니다 . OSSP_RECEIVE 동기화 전략은 OnSend 이벤트가 처리 되도록 합니다.

프로세스에서 OnReceive 또는 OnClose 이벤트를 트리거하는 것은 불가능합니다 .

참고: 모델 특성, 정책 설정 및 보안 버전을 기본으로 합 리눅스 플랫폼의 통신 구성 요소는 이벤트 동기화를 지원합니다. 윈도우입니다. 즉 말하자면, 리눅스 서버 대리인 통신 구성 요소의 이벤트 윈도우 전략은 다음과 같습니다.
OSSP_CLOSE SetOnSendSyncPolicy(enSyncPolicy) 방법이 통신 구성 요소에 유효하지 않습니다.

2.1.4 주소 재사용 정책

HP-Socket v5.7.x부터 모든 통신 구성 요소는 주소 재사용 정책 설정을 지원합니다.

(주소 재사용 정책을 통해 SetReuseAddressPolicy(enReusePolicy) 설정 방법)

<<열거>>
EnReuseAddressPolicy
+RAP_NONE
+RAP_ADDR_ONLY
+RAP_ADDR_AND_PORT

✓ 랩_없음

: 재사용하지 마십시오

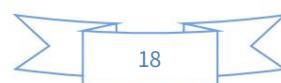
윈도우 :

✓ SO_EXCLUSIVEADDRUSE = 참

✓ SO_REUSEADDR = 거짓

리눅스/유닉스 :

✓ SO_REUSEADDR = 거짓



✓ SO_REUSEPORT = 거짓

✓ RAP_ADDR_ONLY : 주소만 재사용(기본값)

윈도우 :

✓ SO_EXCLUSIVEADDRUSE = 거짓

✓ SO_REUSEADDR = 거짓

리눅스/유닉스 :

✓ SO_REUSEADDR = 참

✓ SO_REUSEPORT = 거짓

✓ RAP_ADDR_AND_PORT : 주소 및 포트 재사용

윈도우 :

✓ SO_EXCLUSIVEADDRUSE = 거짓

✓ SO_REUSEADDR = 참

리눅스/유닉스 :

✓ SO_REUSEADDR = 참

✓ SO_REUSEPORT = 참

운영 체제 플랫폼마다 주소 재사용을 위한 설정 방법과 동작이 다르므로 설정하기 전에 서로 다른 운영 체제 플랫폼에서 이러한 옵션의 동작을 완전히 이해해야 합니다. 예: Windows 는 SO_EXCLUSIVEADDRUSE 및 SO_REUSEADDR 소켓 옵션을 통해 주소 재사용 을 설정 하고 SO_REUSEADDR 의 의미 는 Linux/Unix 의 SO_REUSEPORT 소켓 옵션과 유사 합니다 . Solaris 시스템은 를 지원하지 않으며 Linux 시스템은 커널 버전 3.9.0 이후만 지원합니다.

2.1.5 접속 방법

모든 HP-Socket 구성 요소의 통신 프로세스는 비동기식입니다. 예를 들어 구성 요소의 `Send()` 메서드를 호출하면 즉시 반환됩니다. 나중에 수신기는 얼마나 많은 데이터가 전송되었는지 알기 위해 `OnSend()` 이벤트를 수신하거나 수신 실패 `OnClose()` 이벤트를 사용하여 전송 실패 이유를 알 수 있습니다. 그러

나 HP-Socket의 `IClient` 및 `IAgent` 구성 요소를 서버에 연결하는 프로세스는 동기식 또는 비동기식일 수 있습니다. 동기화는 구성 요소의 연결 방법(`IClient - Start()`, `IAgent - Connect()`)이 연결이 성공적으로 설정되거나 실패할 때까지 기다렸다가 반환(`TRUE` 또는 `FALSE` 반환)하는 것을 의미합니다.

비동기 연결은 컴포넌트의 연결 메소드 `Start()` / `Connect()` 가 즉시 반환된다는 것을 의미하며, `Start()` / `Connect()` 가 성공(`TRUE`)을 반환하면 나중에 `OnConnect()` 또는 `OnClose()` 이벤트를 수신하고 , 전자가 수신되면 연결이 성공하고 후자를 수신하면 연결이 실패합니다. 참고: `Start()` / `Connect()`가 `FALSE`를 반환하면 `OnClose()` 이벤트가 나중에 수신되지 않을 수 있습니다 . 따라서 `Start()` / `Connect()` 의 반환 값 도 비동기 연결인지 확인해야 하며 , 반환 실패(`FALSE`)가 발생하면 즉시 연결이 실패한 것으로 결론을 내릴 수 있습니다.

`IClient` 연결 설정 방법:

```
BOOL 시작(lpszRemoteAddress, usPort, bAsyncConnect = TRUE,
           lpszBindAddress = nullptr, usLocalPort = 0)
```

매개 변수 `bAsyncConnect`는 `Start()` 메서드가 반환하는 경우 비동기 연결 모드(기본값: `TRUE`)를 사용할지 여부를 나타냅니다.

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcdsaa/HP-Socket>

실패한 경우 구성 요소의 `GetLastError()` 및 `GetLastErrorDesc()` 메서드를 호출하여 오류 코드 및 오류 설명을 얻을 수 있습니다.
`Start()` 메서드가 성공적으로 반환되면 구성 요소의 `GetConnectionID()` 메서드를 호출하여 현재 연결 ID를 얻을 수 있습니다.
 연결 ID。

알아채다: `IUpdCast` 요소 별() 방법 무시 `bAsyncConnect` 매개변수.

`IAgent`는 연결 방법을 설정합니다.

```
BOOL 시작(lpszBindAddress = nullptr, bAsyncConnect = TRUE)
BOOL 연결(lpszRemoteAddress, usPort, pdwConnID = nullptr,
           pExtra = nullptr, usLocalPort = 0)
```

`Start()` 메서드는 `IAgent` 구성 요소를 시작하고 연결 방법을 지정하며 `bAsyncConnect` 매개 변수는 비동기 사용 여부를 나타냅니다.
 연결 방법(기본값: TRUE), `Start()` 방법이 실패를 반환하는 경우 구성 요소의 `GetLastError()` 및
`GetLastErrorDesc()` 메서드는 오류 코드와 오류 설명을 가져옵니다. 참고: `Start()` 메서드는 전체 통신 주기에서만 사용됩니다.
 1회 호출이 필요합니다.

`Connect()` 메서드는 지정된 서버와의 연결을 설정합니다. `pdwConnID` 매개변수는 이 연결의 연결 ID를 얻기 위해 사용되며(기본값: nullptr, 획득하지 않음), `pExtra` 매개변수는 "연결 바인딩" 데이터를 설정합니다(기본값: nullptr, 설정되지 않음),
`Connect()` 메서드가 실패하면 Windows API 함수 `::GetLastError()`를 호출하여 Windows
 오류 코드: `pExtra`가 설정되어 있으면 이때 수동으로 해제해야 합니다.

동기식 연결이든 비동기식 연결이든 연결이 성공적으로 완료되면 리스너의 두 가지 이벤트가 연속적으로 트리거됩니다.

- ✓ `OnPrepareConnect(pSender, dwConnID, 소켓)`
- ✓ `온커넥트 pSender, dwConnID)`

이 중 `OnPrepareConnect(pSender, dwConnID, socket)`는 연결이 시작되기 전에 트리거되며 소켓은 로컬입니다.
 SOCKET 핸들, 이 이벤트에서 `setsockopt()` / `WSAioctl()` 및 기타 메서드를 통해 SOCKET 옵션을 설정할 수 있습니다.
 연결이 성공적으로 설정된 후 `OnConnect(pSender, dwConnID)` 가 트리거됩니다.

2.1.6 연결 바인딩

`IClient` 계열 컴포넌트의 경우 컴포넌트 객체는 Connection ID와 통신 연결에 해당하므로 매우
 통신 연결을 애플리케이션 계층 데이터와 쉽게 연관시킬 수 있습니다. 응용 프로그램이 구성 요소와 상호 작용할 때 구성 요소는 데이터를 처리하라는 알림을 직접 받습니다.
 예(예: 보내기(`pData`, `iLength`)). 그림 2.1.5-1과 같이:

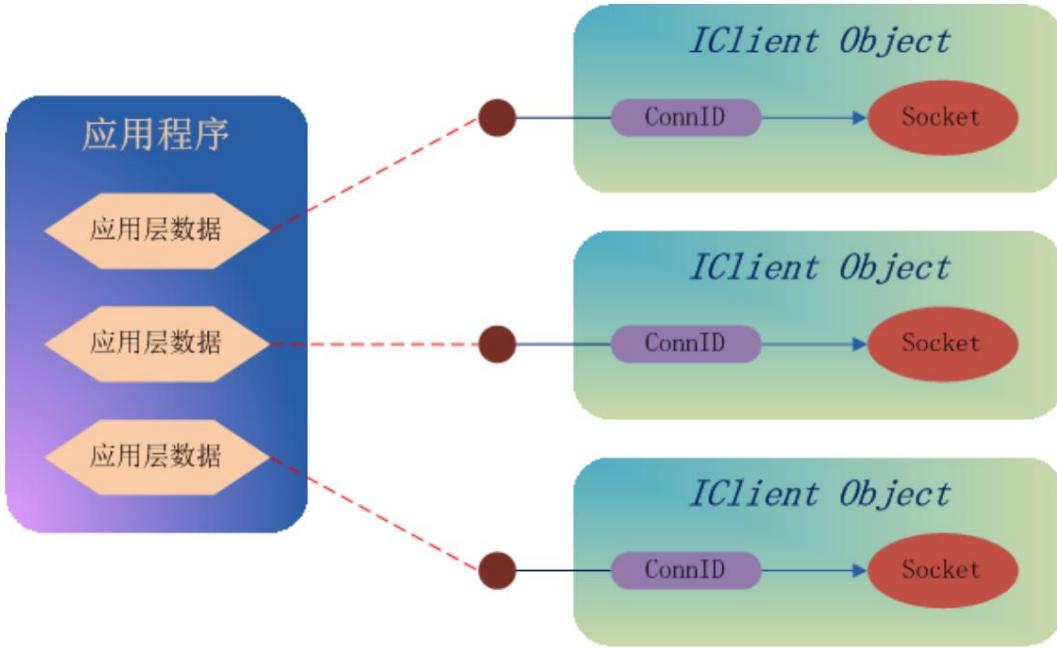


그림 2.1.5-1 IClient 구성 요소 연결의 개략도

참고: [아이클라이언트 구성요소, 사용 가능 세트 엑스트라\(\) / Get엑스트라\(\)](#) [메서드 바인딩, 추가 데이터에 대한 액세스.](#)

IServer 및 IAgent 시리즈 구성 요소의 경우 하나의 구성 요소 개체가 여러 통신 연결을 관리하고 HP-Socket 연결은 연결 ID로 추상화됩니다. 응용 프로그램이 구성 요소와 상호 작용할 때 구성 요소에 알리기 위해 연결 ID를 지정해야 합니다. 관리할 연결(예: Send(dwConnID, pData, iLength)). 그림 2.1.5-2와 같이:

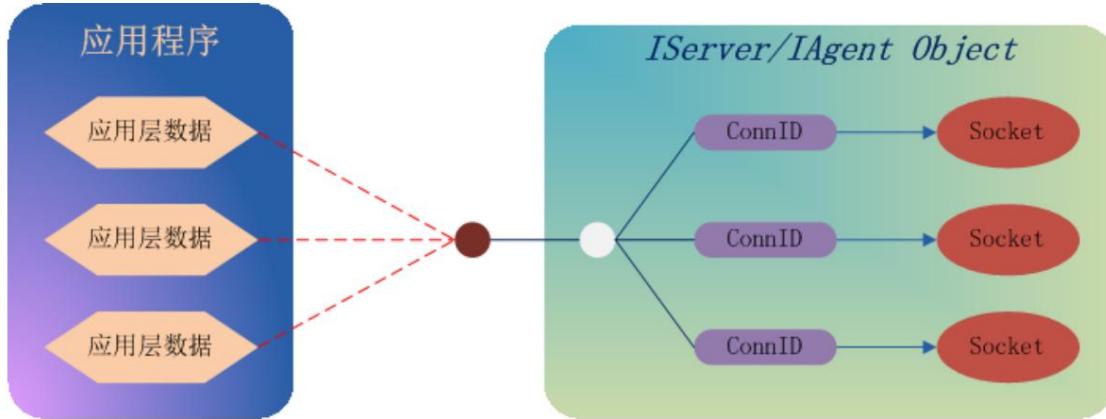


그림 2.1.5-2 IServer/IAgent 구성 요소 연결의 개략도

연결 ID와 애플리케이션 계층 데이터 간의 해당 관계를 설정하기 위해 애플리케이션은 일반적으로 매핑 테이블을 유지 관리해야 합니다. (예: map<CONNID, TMyAppData*>), 이는 애플리케이션에 대한 부담을 증가시킬 뿐만 아니라 실행으로 인해 다중 스레드 환경에서는 매핑 테이블에 대한 읽기 및 쓰기 작업을 동기식으로 처리해야 하므로 애플리케이션의 동시 성능이 저하됩니다.

HP-Socket은 IServer 및 IAgent 시리즈 구성 요소에 대해 다음과 같은 방법을 제공하여 연결 ID 및 응용 프로그램 계층 번호를 바인딩합니다. 데이터, 응용 프로그램이 매핑 테이블을 유지하지 않도록 하십시오.

`BOOL SetConnectionExtra(CONNID dwConnID, PVOID pExtra)`



프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcdsaa/HP-Socket>

BOOL GetConnectionExtra(CONNID dwConnID, PVOID* ppExtra)

일반적인 애플리케이션 시나리오

는 다음과 같습니다. 1) **OnAccept() / OnConnect()** 이벤트에서 **SetConnectionExtra(dwConnID, pExtra)** 를 호출합니다 . 응용 프로그램 계층 데이터와 연결 ID를 바인딩합니다.

2) **OnReceive() / OnSend()** 이벤트에서 **GetConnectionExtra(dwConnID, ppExtra)** 호출

Connection ID에 바인딩된 Application Layer 데이터를 꺼내 해당 비즈니스 로직 처리를 실행합니다.

3) **OnClose()** 이벤트에서 연결 ID와 애플리케이션 계층 데이터의 바인딩을 취소하고 애플리케이션 계층 데이터를 지우고 리소스를 해제합니다.

참고: 인해 HP 소켓 이미 보장 온리시브() / 온클로즈()

와 같은 이벤트의 스레드 안전성 덕분에 애플리케이션은 동기화 문제에 대한 걱정 없이 연결 바인딩 메커니즘을 안전하게 사용할 수 있습니다.

2.1.7 정지 대기

HP-Socket v5.7.x부터 모든 구성 요소(IHPThreadPool 포함)에 대해 대기 중지 메서드가 추가되었습니다. **Wait()**, 이 메서드의 동작은 매우 간단합니다. 구성 요소의 상태가 "실행 중"이면 실행 그렇지 않으면 즉시 반환합니다.

Wait()는 "구성 요소가 중지된 후 자동으로 프로그램 종료"와 같은 기능을 실현하기 위해 주로 명령줄 프로세스 또는 백그라운드 프로세스에 사용됩니다.

중지 대기 방법:

BOOL 대기(dwMilliseconds =INFINITE)

매개 변수 **dwMilliseconds**는 대기 시간 초과 시간(밀리초, 기본값: INFINITE, 시간 초과 없음)을 설정하고 시간 초과가 반환됩니다. FALSE를 반환하고 일반 종료는 TRUE를 반환합니다.

2.2 서버 구성요소

2.2.1 인터페이스 설명

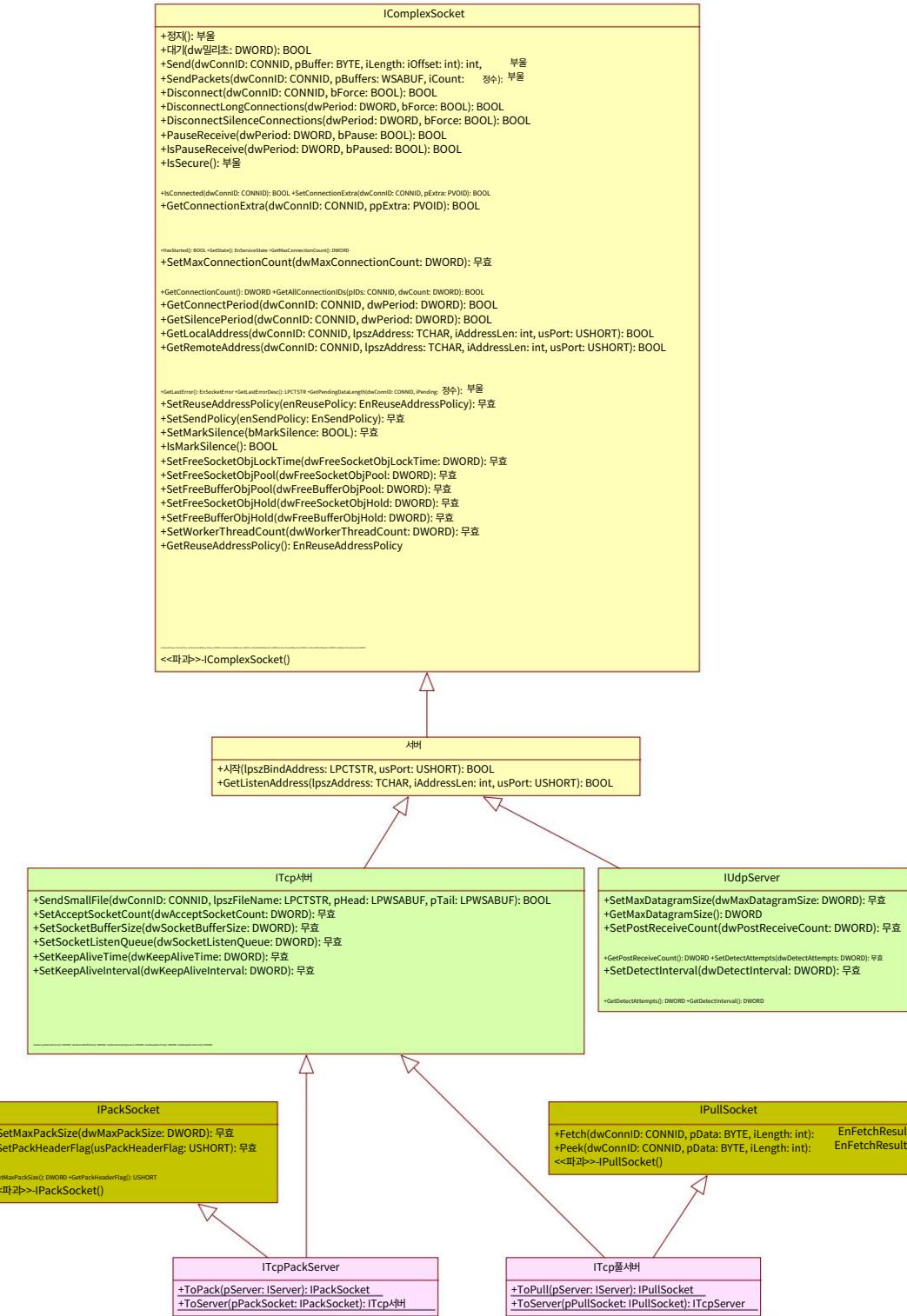


그림 2.2.1-1 서버 컴포넌트 인터페이스

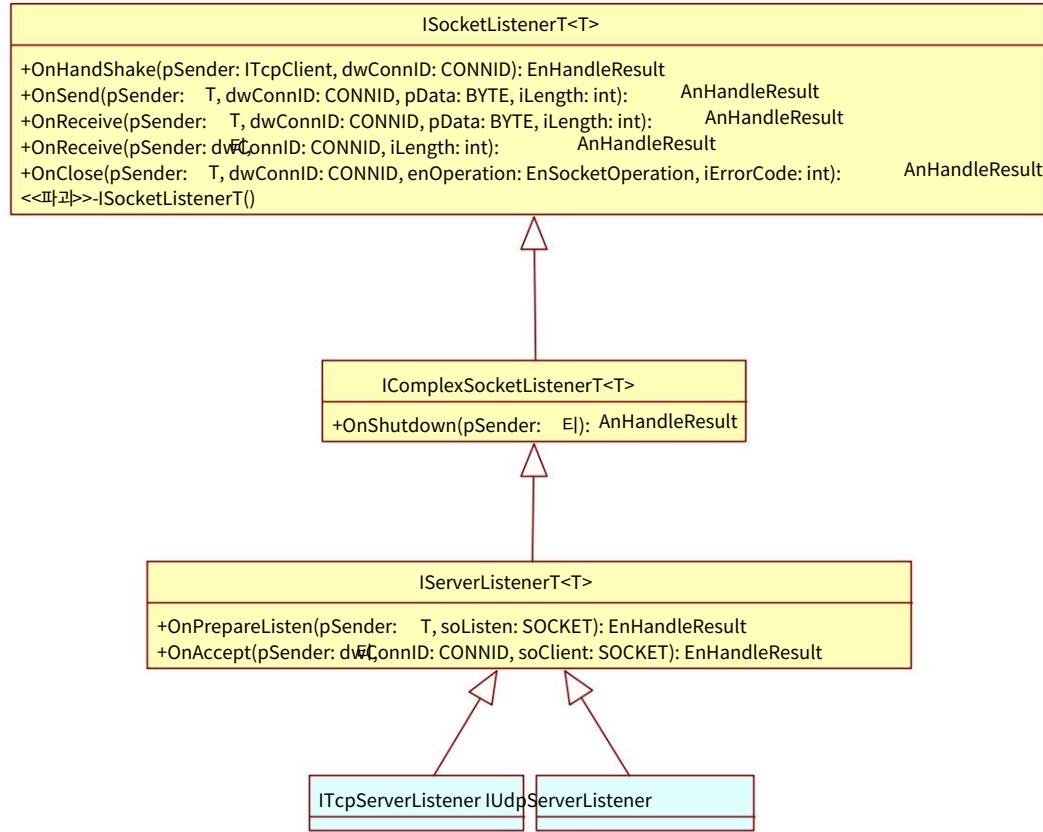


그림 2.2.1-2 서버 리스너 인터페이스

서버 구성 요소 인터페이스의 상속 계층 구조는 그림 2.2.1-1에 나와 있습니다. 여기서 ITcpServer 및 IUdpServer 는 IServer 의 경우 ITcpPullServer 및 ITcpPackServer는 ITcpServer 에서 상속됩니다. 주요 인터페이스 방법은 표 2.2.1에 나와 있습니다.
1, 기타 인터페이스 방법은 [include/hpsocket/SocketInterface.h](#) 파일의 관련 참고 사항을 참조하십시오 .

컴포넌트 인터페이스	작성방법	설명하다
서버	Start()는 구	시작 구성 요소
	Stop() 성 요소를 닫습니다.	
	Send()는 데이터 전송	
	SendPacket()는 연결을 보냅니다.	
	연결 끊기()	해제
	DisconnectLongConnections()는 긴 연결을 끊습니다.	
	DisconnectSilenceConnections()는 자동 연결을 끊습니다.	
	PauseReceive()는 연	데이터 수신 일시 중지
	IsConnected()는 통	결이 유효한지 확인합니다.
	HasStarted() 통	신 구성 요소가 시작되었는지 확인합니다.
	GetState()는 연	신 구성 요소의 현재 상태를 가져옵니다.
	연결 시간을 얻기 위한 GetConnectPeriod()	
	침묵 기간을 얻기 위한 GetConnectionCount()	
	GetSilencePeriod()GetConnectPeriod()	



프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcdsaa/HP-Socket>

	<code>GetAllConnectionIDs()</code> 연결	모든 연결의 CONNID 가져오기
	<code>GetLocalAddress()</code> 연결의 로컬 주소를 가져옵니다.	
	<code>GetRemoteAddress()</code> 수의 원격 주소를 가져옵니다.	
	<code>GetListenAddress()</code> 마지막 소켓의 주소를 가져옵니다.	
	<code>GetLastError()</code> 마지막으로 실패한 작업의 오류 코드를 가져옵니다.	
	<code>GetLastErrorDesc()</code> 마지막으로 실패한 작업의 오류 설명을 가져옵니다.	
	<code>SetWorkerThreadCount()</code> 스레드 수를 설정합니다.	
ITcp서버	<code>SetMaxConnectionCount()</code> 는 최대 연결 수를 설정합니다.	
	<code>SendSmallFile()</code> 은 대	작은 파일 보내기
	<code>SetSocketListenQueue()</code> 소켓의 대기 큐 크기를 설정합니다.	
	<code>SetAcceptSocketCount()</code>	Windows: Accept pre-delivery 수량 설정 Linux: 최대 EPOLL 이벤트 수 설정
	<code>SetSocketBufferSize()</code>	통신 데이터 버퍼 크기 설정
	<code>SetKeepAliveTime()</code>	하트비트 감지 패킷 전송 간격 설정
	<code>SetKeepAliveInterval()</code>	하트비트 감지 재시도 패킷 전송 간격 설정
ITcp풀서버	<code>SetNoDelay()</code>	TCP_NODELAY 소켓 옵션 설정
	<code>술책()</code>	풀 데이터
	<code>몰래 엿보다()</code>	스누핑 데이터
	<code>SetMaxPackSize()</code>	최대 패킷 길이 제한 설정
	<code>SetPackHeaderFlag()</code>	헤더 확인 플래그 설정
	<code>SetMaxDatagramSize()</code>	데이터그램의 최대 길이 설정
	<code>SetDetectAttempts()</code>	감지 재시도 횟수 설정
IUpdServer	<code>SetDetectInterval()</code>	탐지 패킷 전송 간격 설정
	<code>SetMaxMessageSize()</code>	ARQ 데이터 패킷의 최대 길이 설정
	<code>SetHandShakeTimeout()</code>	ARQ 핸드쉐이크 시간 초과 설정

표 2.2.1-1 서버 구성 요소 인터페이스

서버 수신기 인터페이스의 상속 계층 구조는 그림 2.2.1-2에 나와 있습니다. 여기서 `ITcpServerListener` 및 `IUpdServerListener`는 `IListener`에서 상속되며 인터페이스 콜백 이벤트는 표 2.2.1-2에 나와 있습니다.

리스너 인터페이스	콜백 이벤트	설명하다
<code>ISocketListenerT</code>	<code>온핸드쉐이크()</code>	악수 완료 핸드쉐이크가 완료되면 발생
	<code>온센드()</code>	전송된 데이터 데이터가 성공적으로 전송된 후 트리거됨
	<code>OnReceive()</code> [푸시]	데이터 도착(PUSH/PACK) 데이터 수신 시 트리거
	<code>OnReceive()</code> [풀]	데이터 도착(PULL) 데이터 수신 시 트리거
	<code>온클로즈()</code>	연결이 닫힘 연결이 정상적으로 닫히거나 비정상적으로 닫힐 때 트리거됨
	<code>IComplexSocketListenerT OnShutdown()</code>	통신 구성 요소 닫기

IServerListenerT	통신 구성 요소가 중지된 후 트리거
	됨, 수신 대기 준비됨, 수신 주소를 바인딩하기 전에 트리거됨, 연결 요청 수락됨, 클라이언트 연결 요청이 도착하면 트리거됨
온준비듣기()	됨, 수신 대기 준비됨, 수신 주소를 바인딩하기 전에 트리거됨, 연결 요청 수락됨, 클라이언트 연결 요청이 도착하면 트리거됨
온수락()	리거됨, 연결 요청 수락됨, 클라이언트 연결 요청이 도착하면 트리거됨

표 2.2.1-2 서버 리스너 인터페이스

2.2.2 작업 흐름

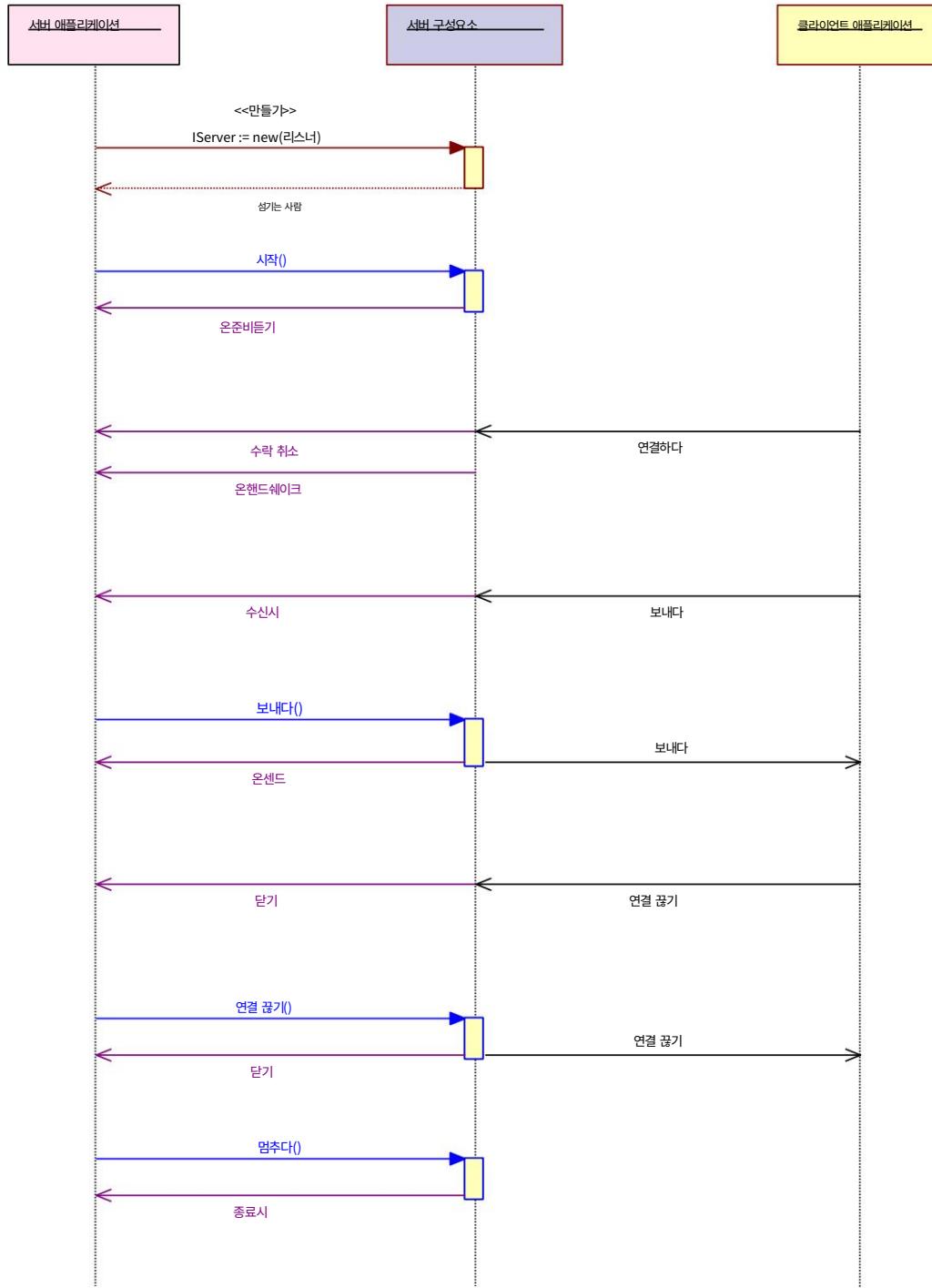


그림 2.2.2-1 서버 작업 흐름

그림 2.2.2-1은 서버, 클라이언트 응용 프로그램 및 서버 구성 요소의 상호 작용 흐름을 보여줍니다.

서버 응용 프로그램은 Server 구성 요소를 시작하기 위해 **Start()** 메서드를 호출합니다. 호출이 성공하면 **TRUE**를 반환하고



프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

`OnPrepareListen` 이벤트를 수신합니다.

클라이언트 응용 프로그램이 서버 응용 프로그램에 대한 연결 요청을 시작하면 서버 응용 프로그램은 `OnAccept` 및 `OnHandshake` 이벤

트를 수신합니다. 클라이언트 응용 프로그램이 서

버 응용 프로그램에 데이터를 보낼 때 서버 응용 프로그램은 `OnReceive`를 수신합니다.

이벤트.

서버 애플리케이션이 `Send()` 메서드를 호출하여 클라이언트 애플리케이션에 데이터를 보낸 후 서버 애플리케이션은

`OnSend` 이벤트를 수신합니다.

연결이 끊어지면 서버 응용 프로그램은 `OnClose` 이벤트를 받습니다. 서버 어플리케이

션은 Server 컴포넌트를 닫기 위해 `Stop()` 메소드를 호출하고, 호출이 성공하면 `TRUE`를 리턴하고 `OnShutdown` 이벤트를 수신합니다.



2.3 에이전트 구성 요소

2.3.1 인터페이스 설명

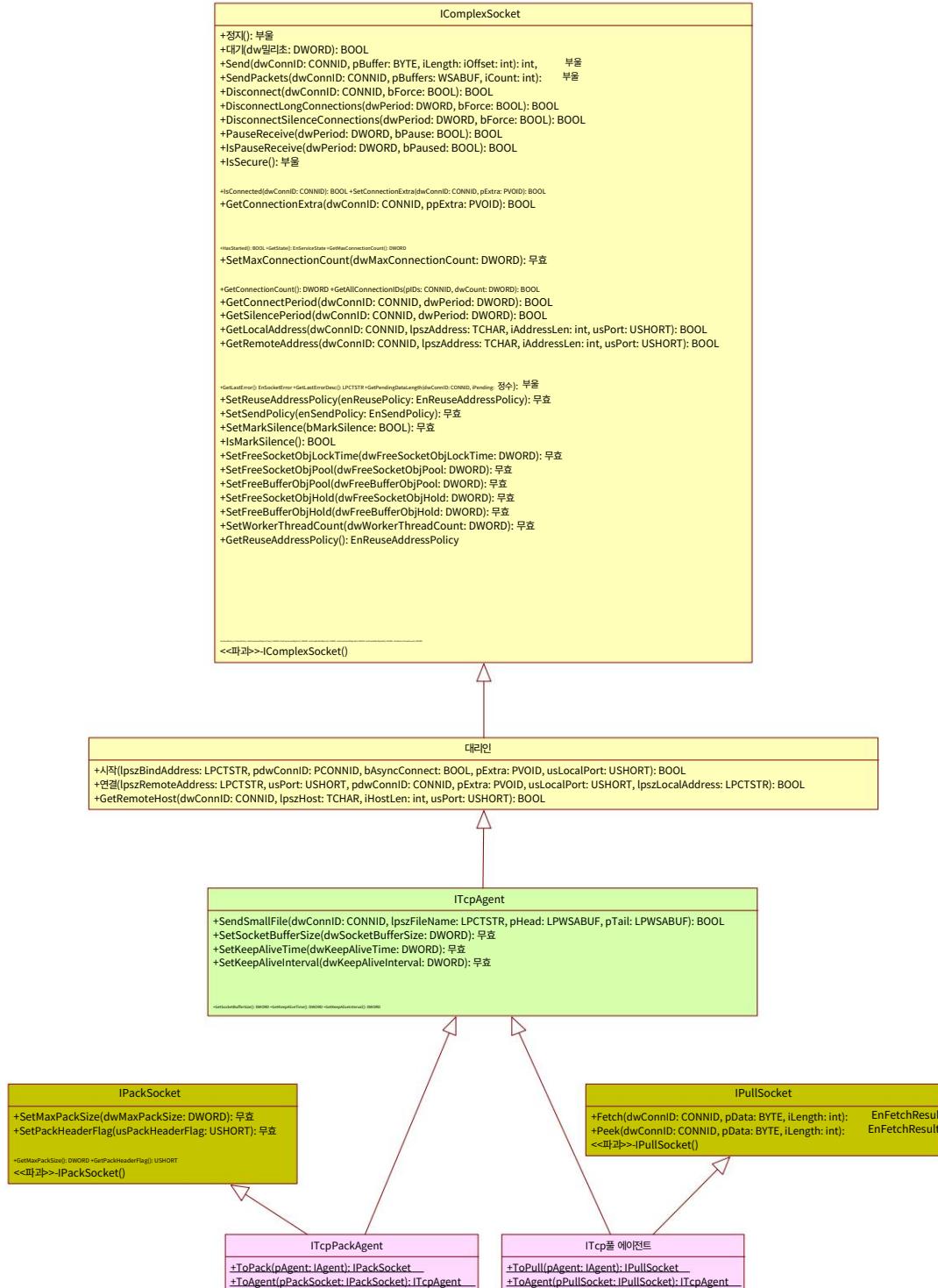


그림 2.3.1-1 에이전트 구성 요소 인터페이스

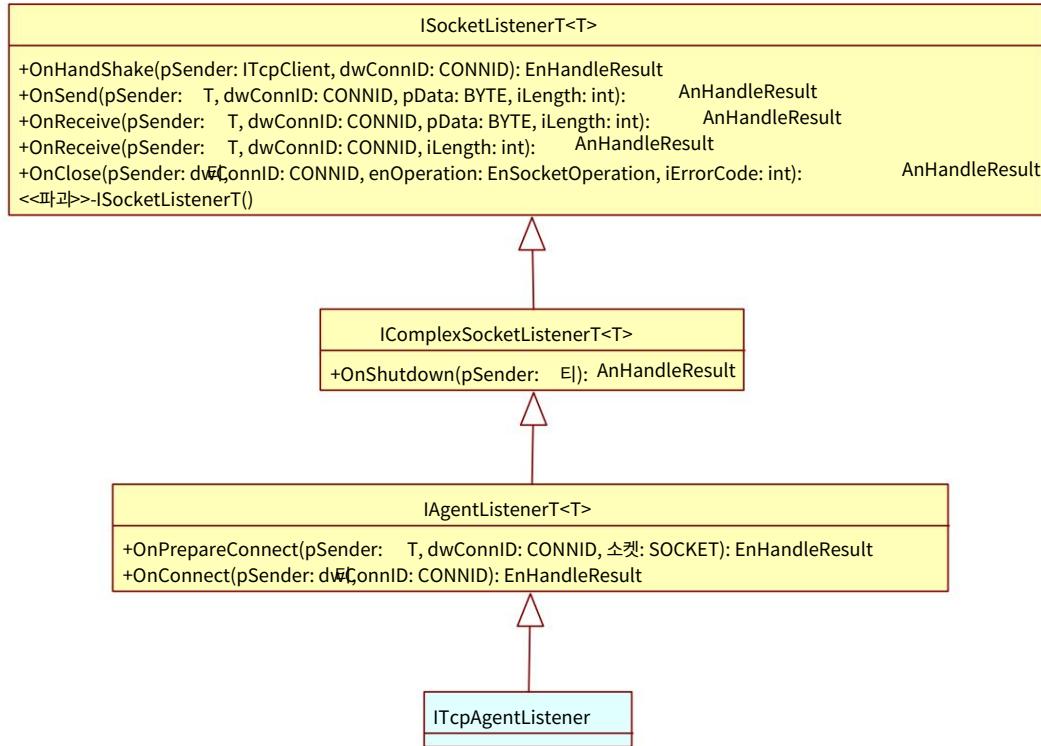


그림 2.3.1-2 에이전트 리스너 인터페이스

에이전트 구성 요소 인터페이스의 상속 계층 구조는 그림 2.3.1-1에 나와 있습니다. 여기서 ITcpAgent는 IAgent에서 상속합니다.

ITcpPullAgent 및 ITcpPackAgent는 ITcpAgent에서 상속됩니다. 주요 인터페이스 방법은 표 2.3.1-1에 나와 있습니다.

인터페이스 방법은 [include/hpsocket/SocketInterface.h](#) 파일의 관련 참고 사항을 참조하십시오.

컴포넌트 인터페이스	조작방법	설명하다
대리인	Start()는 구	시작 구성 요소
	Stop() 성 요소를 닫습니다.	
	데이터를 보내기 연결	
	Send()는 여 위한 Connect()	
	SendPackets()는 데이터 연결을 보냅니다.	
	연결 끊기() 해제	
	DisconnectLongConnections()는 긴 연결을 끊습니다.	
	DisconnectSilenceConnections()는 자동 연결을 끊습니다.	
	PauseReceive()는 연	데이터 수신 일시 중지
	IsConnected()는 통 결이 유효한지 확인합니다.	
	HasStarted() 통 신 구성 요소가 시작되었는지 확인합니다.	
	GetState()는 연 신 구성 요소의 현재 상태를 가져옵니다.	
	연결 시간을 얻기 위해 GetKeepAliveTime()	
	침묵 기간을 얻기 위한 GetConnectionCount()	
	GetSilencePeriod() 또는 ConnectPeriod()	
	GetAllConnectionIDs() 결의 CONNID를 가져옵니다.	

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

ITcpAgent	<code>GetLocalAddress()</code>	연결의 로컬 주소 가져오기
	<code>GetRemoteHost()</code> 마지막	연결의 원격 호스트 가져오기
	<code>GetLastError()</code> 마지막	막으로 실패한 작업의 오류 코드를 가져옵니다.
	<code>GetLastErrorDesc()</code>	마지막으로 실패한 작업의 오류 설명을 가져옵니다.
	<code>SetWorkerThreadCount()</code> 자 스레드 수를 설정합니다.	
	<code>SetMaxConnectionCount()</code> 는 최대 연결 수를 설정합니다.	
	<code>SendSmallFile()</code> 은 주소	작은 파일 보내기
	<code>SetReuseAddress()</code> 는	재활용 메커니즘을 활성화할지 여부를 설정합니다.
	<code>SetSocketBufferSize()</code> 는	선택된 데이터 버퍼 크기를 설정합니다.
ITcpPool 에이전트	<code>SetKeepAliveTime()</code> 은 하트 비트 감지 패킷 전송 간격을 설정합니다.	
	<code>SetKeepAliveInterval()</code> 은 비트 감지 재시도 패킷 전송 간격을 설정합니다.	
	<code>SetNoDelay()</code> 는 데이	TCP_NODELAY 소켓 옵션을 설정합니다.
ITcpPackAgent	<code>Fetch()</code> 스	터를 가져옵니다.
	<code>Peek()</code> 는	누핑 데이터
ITcpPackAgent	<code>SetMaxPackets()</code> 은	최대 패킷 수의 제한을 설정합니다.
	<code>SetPackHeaderFlag()</code>	헤더 확인 플래그를 설정합니다.

표 2.3.1-1 에이전트 구성 요소 인터페이스

에이전트 리스너 인터페이스의 상속 계층 구조는 그림 2.3.1-2에 나와 있습니다. 여기서 `ITcpAgentListener`는 `IAgentListener`, 인터페이스 콜백 이벤트는 표 2.3.1-2에 나와 있습니다.

리스너 인터페이스	콜백 이벤트	설명하다
ISocketListenerT	<code>OnHandleShake()</code>	약수 완료 핸드쉐이크가 완료되면 발생
	<code>OnSend()</code>	전송된 데이터 데이터가 성공적으로 전송된 후 트리거됨
	<code>OnReceive() [푸시]</code>	데이터 도착(PUSH/PACK) 데이터 수신 시 트리거
	<code>OnReceive() [풀]</code>	데이터 도착(PULL) 데이터 수신 시 트리거
	<code>OnClose()</code>	연결이 닫힘 연결이 정상적으로 닫히거나 비정상적으로 닫힐 때 트리거됨
IComplexSocketListenerT	<code>OnShutdown()</code>	통신 구성 요소 닫기 통신 구성 요소가 중지된 후 트리거됨
	<code>OnPreParareConnect()</code>	연결 준비 연결이 설정되기 전에 트리거됨
	<code>OnConnect()</code>	성공적으로 연결 설정 성공적인 연결이 설정된 후 트리거됨
IAgentListenerT		

표 2.3.1-2 에이전트 리스너 인터페이스

2.3.2 작업 흐름

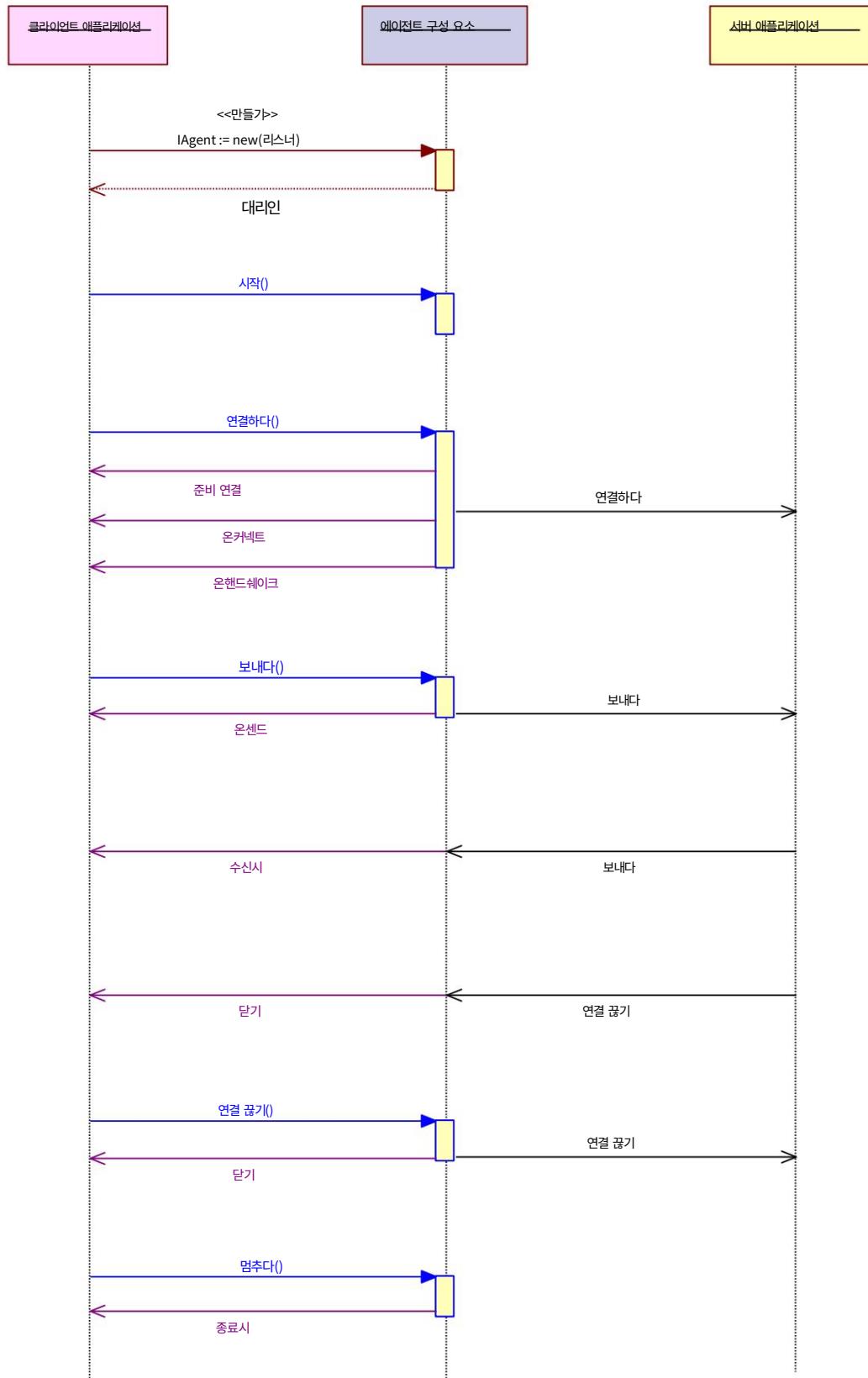


그림 2.3.2-1 에이전트 작업 흐름

그림 2.3.2-1은 서버, 클라이언트 응용 프로그램 및 에이전트 구성 요소의 상호 작용 흐름을 보여줍니다.

클라이언트 응용 프로그램은 Agent 구성 요소를 시작하기 위해 **Start()** 메서드를 호출하고 호출이 성공하면 TRUE를 반환합니다.

클라이언트 응용 프로그램은 **Connect()** 메서드를 호출하여 서버 응용 프로그램에 대한 연결 요청을 시작합니다.

성공하면 TRUE를 반환하고 **OnPrepareConnect**, **OnConnect** 및 **OnHandshake**를 연속적으로 수신합니다.

이벤트.

클라이언트 응용 프로그램이 **Send()** 메서드를 호출하여 서버 응용 프로그램에 데이터를 보낸 후 클라이언트 응용 프로그램은 **OnSend** 이벤트를 수신합니다.

서버 응용 프로그램이 클라이언트 응용 프로그램에 데이터를 보낼 때 클라이언트 응용 프로그램은 **OnReceive** 이벤트.

연결 해제 시 클라이언트 애플리케이션은 **OnClose** 이벤트를 수신합니다.

클라이언트 응용 프로그램은 **Stop()** 메서드를 호출하여 Agent 구성 요소를 닫고 호출에 성공하면 TRUE를 반환하고 **OnShutdown** 이벤트를 받습니다.

2.4 클라이언트 구성 요소

2.4.1 인터페이스 설명

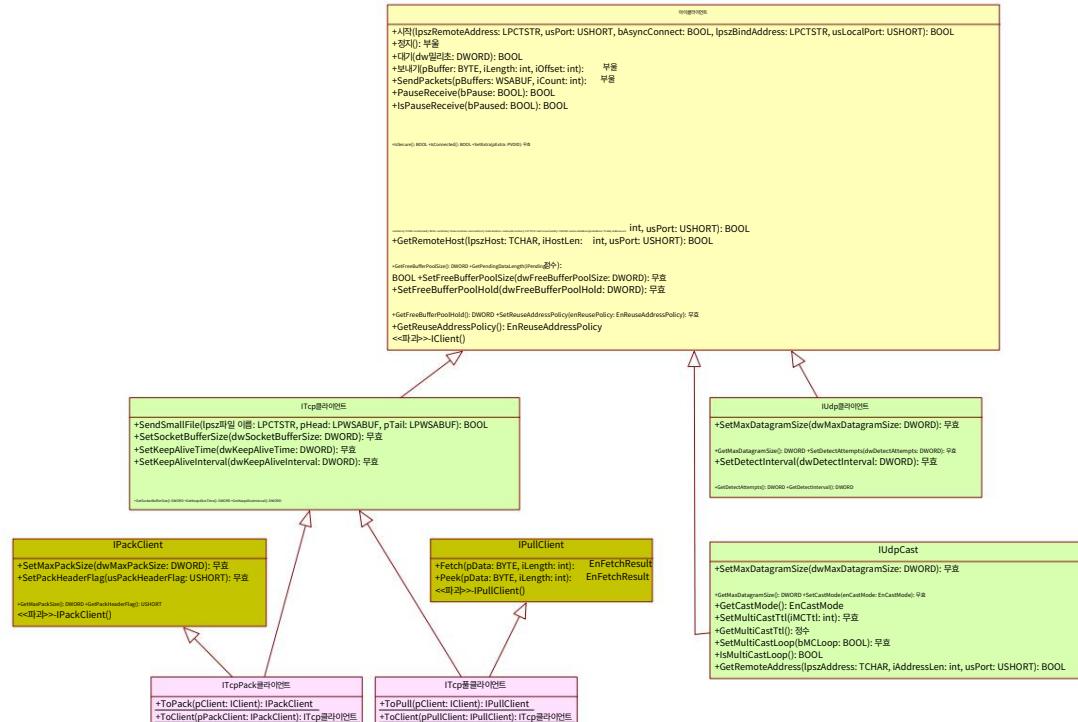


그림 2.4.1-1 클라이언트 구성 요소 인터페이스

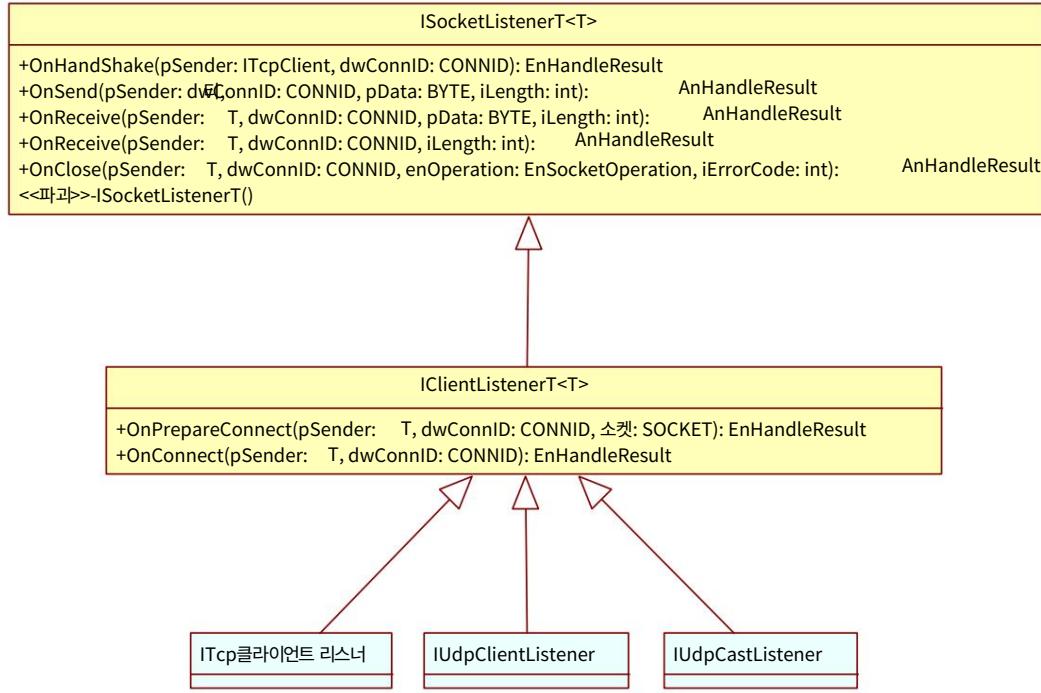


그림 2.4.1-2 클라이언트 리스너 인터페이스

클라이언트 구성 요소 인터페이스의 상속 계층 구조는 그림 2.4.1-1에 나와 있습니다. 여기서 **ITcpClient**, **IUdpClient** 및 **IUdpCast**는 **IClient**에서 상속되고 **ITcpPullClient** 및 **ITcpPackClient**는 **ITcpClient**에서 상속됩니다. 메인 인터페이스 방법은 표 2.4.1-1에 나와 있습니다. 다른 인터페이스 방법은 [include/hpsocket/SocketInterface.h](#) 파일의 관련 참고 사항을 참조하십시오.

컴포넌트 인터페이스	작동방법	설명하다
아이클라이언트	시작()	시작 구성 요소
	멈추다()	닫기 구성 요소
	연결하다()	서버에 연결
	보내다()	데이터 보내기
	보내기패킷()	여러 데이터 세트 보내기
	일시정지수신()	데이터 수신 일시 중지
	연결되었다()	유효한 연결 확인
	시작했다()	통신 구성 요소가 시작되었는지 확인
	GetState()	통신 구성 요소의 현재 상태 가져오기
	GetConnectionID()	구성 요소 개체의 CONNID 가져오기
	GetLocalAddress()	연결의 로컬 주소 가져오기
	GetRemoteHost()	연결된 원격 호스트 가져오기
	GetLastError()	마지막으로 실패한 작업의 오류 코드 가져오기
	GetLastErrorDesc()	마지막으로 실패한 작업의 오류 설명 가져오기
ITcp클라이언트	작은 파일 보내기()	작은 파일 보내기
	SetSocketBufferSize()	통신 데이터 버퍼 크기 설정
	SetKeepAliveTime()	하트비트 간지 패킷 전송 간격 설정

ITcp풀클라이언트 ITcpPack클라이언트 IUpd클라이언트 IUpdArq클라이언트 IUpdCast	SetKeepAliveInterval()	하트비트 감지 재시도 패킷 전송 간격 설정
	SetNoDelay()	TCP_NODELAY 소켓 옵션 설정
	술책()	풀 데이터
	몰래 엿보다()	스누핑 데이터
	SetMaxPackSize()	최대 패킷 길이 제한 설정
	SetPackHeaderFlag()	헤더 확인 플래그 설정
	SetMaxDatagramSize()	데이터그램의 최대 길이 설정
	SetDetectAttempts()	감지 재시도 횟수 설정
	SetDetectInterval()	탐지 패킷 전송 간격 설정
	SetMaxMessageSize()	ARQ 데이터 패킷의 최대 길이 설정
	SetHandShakeTimeout()	ARQ 핸드셰이크 시간 초과 설정
	SetMaxDatagramSize()	데이터그램의 최대 길이 설정
	SetReuseAddress()	주소 재사용 메커니즘 활성화 여부 설정
	세트캐스트모드()	브로드캐스트 모드 설정(멀티캐스트 또는 브로드캐스트)
	SetMultiCastTtl()	멀티캐스트 패킷의 TTL 설정
	SetMultiCastLoop()	멀티캐스트 루프 활성화 여부 설정
	GetRemoteAddress()	현재 패킷의 원격 주소 얻기

표 2.4.1-1 클라이언트 구성 요소 인터페이스

클라이언트 수신기 인터페이스의 상속 계층 구조는 그림 2.4.1-2에 나와 있습니다. 여기서 **ITcpClientListener** 및 **IUpdClientListener**는 **IClientListener**에서 상속되며 인터페이스 콜백 이벤트는 표 2.4.1-2에 나와 있습니다.

리스너 인터페이스	콜백 이벤트	설명하다
ISocketListenerT	온핸드쉐이크()	약수 완료 핸드쉐이크가 완료되면 발생
	온센드()	전송된 데이터 데이터가 성공적으로 전송된 후 트리거됨
	OnReceive() [푸시]	데이터 도착(PUSH/PACK) 데이터 수신 시 트리거
	OnReceive() [풀]	데이터 도착(PULL) 데이터 수신 시 트리거
	온클로즈()	연결이 닫힘 연결이 정상적으로 닫히거나 비정상적으로 닫힐 때 트리거됨
	온프레파레카넥트()	연결 준비 연결이 설정되기 전에 트리거됨
IClientListenerT	온커넥트()	성공적으로 연결 설정 성공적인 연결이 설정된 후 트리거됨

표 2.4.1-2 클라이언트 리스너 인터페이스

2.4.2 작업 흐름

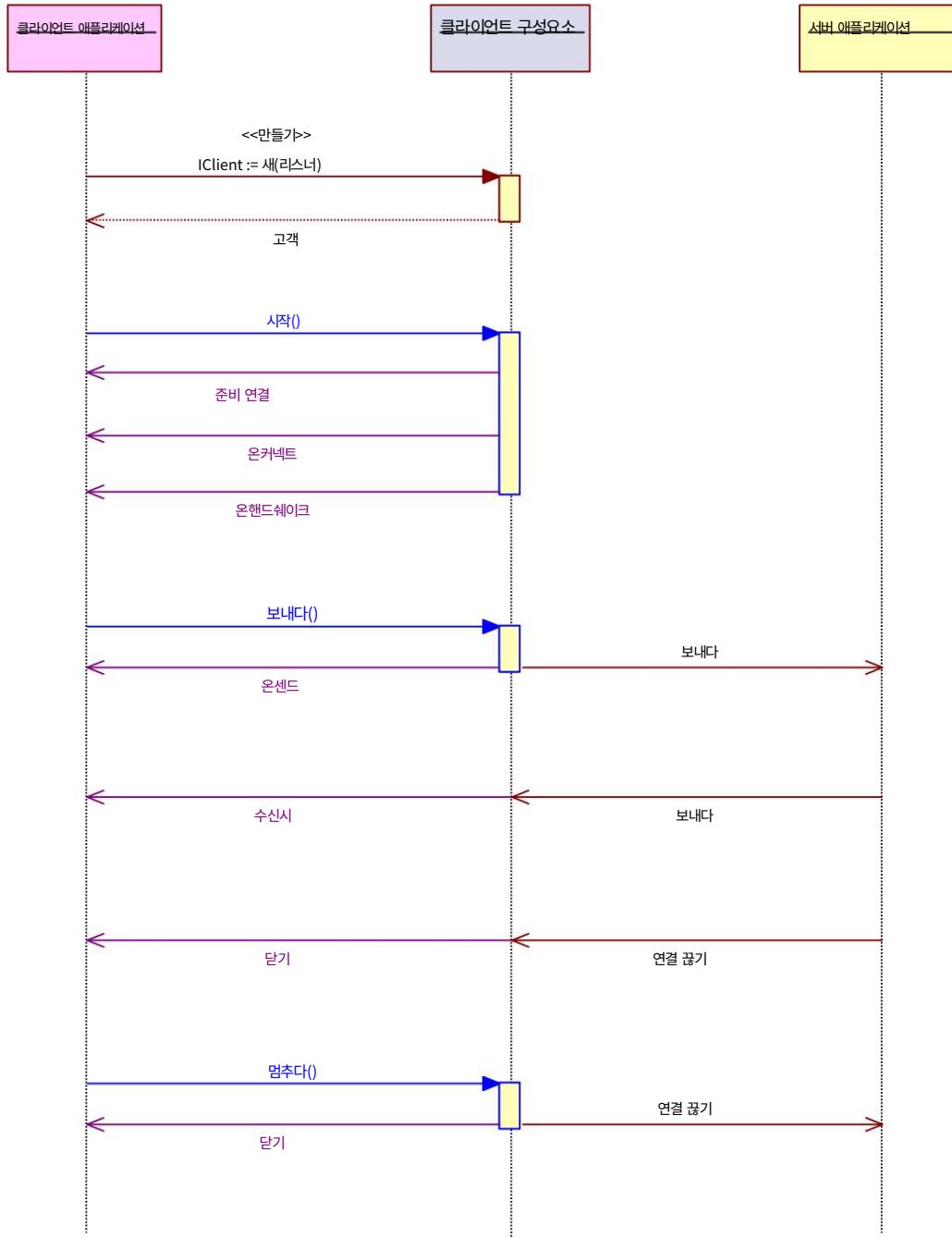


그림 2.4.2-1 클라이언트 작업 흐름

그림 2.4.2-1은 서버, 클라이언트 응용 프로그램 및 클라이언트 구성 요소의 상호 작용 흐름을 보여줍니다.

클라이언트 응용 프로그램은 `Start()` 메서드를 호출하여 서버 응용 프로그램에 연결 요청을 시작하고 연결에 성공하면 `TRUE`를 반환하고 `OnPrepareConnect`, `OnConnect` 및 `OnHandshake` 이벤트를 차례로 수신합니다.

JessMA

프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

클라이언트 응용 프로그램이 `Send()` 메서드를 호출하여 서버 응용 프로그램에 데이터를 보낸 후 클라이언트 응용 프로그램은 `OnSend` 이벤트를 수신합니다.

서버 응용 프로그램이 클라이언트 응용 프로그램에 데이터를 보낼 때 클라이언트 응용 프로그램은 `OnReceive` 이벤트.

연결 해제 시 클라이언트 애플리케이션은 `OnClose` 이벤트를 수신합니다. 클라이언트

애플리케이션은 클라이언트 구성 요소를 닫기 위해 `Stop()` 메서드를 호출하고 호출이 성공하면 `TRUE`를 반환하고 `OnClose` 이벤트를 받습니다.

2.5 UDP 노드 구성요소

2.5.1 인터페이스 설명

IUpdp노드

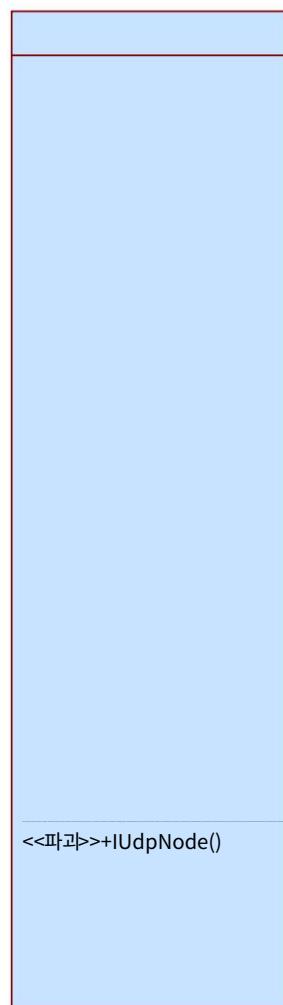


그림 2.5.1-1 UDP 노드 구성 요소 인터페이스

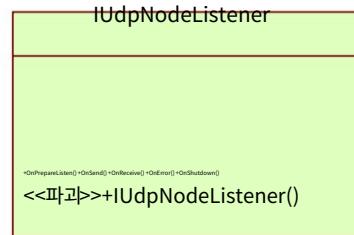


그림 2.5.1-2 UDP 노드 리스너 인터페이스

HP-Socket v5.7.x부터 UDP 노드(UDP 노드) 구성 요소인 **IUpdNode**가 제공됩니다. UDP 노드

HP-Socket의 다른 통신 구성 요소와 달리 "연결" 개념이 없습니다(따라서 연결 ID가 없음).

동작 모드는 일반 UDP 소켓과 유사하며 모든 UDP 응용 프로그램과 동시에 통신할 수 있습니다. 이 구성 요소는 유니캐스트로 브로드캐스트할 수 있습니다.

멀티캐스트 및 브로드캐스트 작동에는 세 가지 모드가 있습니다. 그림 2.5.1-1과 같이 UDP Node 구성 요소 인터페이스는 독립적인 인터페이스이며 상속되지 않습니다.

다른 인터페이스에서. 주요 인터페이스 방법은 표 2.5.1-1과 같으며, 기타 인터페이스 방법은 다음을 참조하십시오.

[include/hpsocket/SocketInterface.h](#) 파일에 대한 관련 주석:

컴포넌트 인터페이스	조작방법	설명하다
IUpd노드	시작()	시작 구성 요소
	멈추다()	닫기 구성 요소
	보내다()	지정된 목적지 주소로 데이터 보내기
	보내기파킷()	결합된 데이터를 지정된 목적지 주소로 전송
	센드캐스트()	멀티캐스트 또는 브로드캐스트 데이터 보내기
	SendCastPackets()	멀티캐스트 또는 브로드캐스트 결합 데이터 보내기
	SetMaxDatagramSize()	데이터그램의 최대 길이 설정
	시작했다()	통신 구성 요소가 시작되었는지 확인
	GetState()	통신 구성 요소의 현재 상태 가져오기
	GetLocalAddress()	연결의 로컬 주소 가져오기
	GetCastAddress()	연결된 멀티캐스트 또는 브로드캐스트 주소 가져오기
	GetLastError()	마지막으로 실패한 작업의 오류 코드 가져오기
	GetLastErrorDesc()	마지막으로 실패한 작업의 오류 설명 가져오기

표 2.5.1-1 UDP 노드 구성 요소 인터페이스

그림 2.5.1-2와 같이 UDP 노드 수신기 인터페이스도 다른 인터페이스에서 상속되지 않는 독립 인터페이스입니다.

인터페이스 콜백 이벤트는 표 2.5.1-2에 나와 있습니다.

리스너 인터페이스	콜백 이벤트	설명하다
IUpdNodeListener	온준비듣기()	들을 준비가 수신 주소를 바인딩하기 전에 트리거됨
	온센드()	전송된 데이터 데이터가 성공적으로 전송된 후 트리거됨
	온리시브()	데이터 도착 데이터 수신 시 트리거

종료 시()	데이터 수신 또는 전송 실패 시 트리거가 통신 구성 요소를 종료하고 통신 구성 요소가 중지된 후 트리거합니다.
온오류()	데이터 수신 또는 전송 실패 시 트리거가 통신 구성 요소를 종료하고 통신 구성 요소가 중지된 후 트리거합니다.

표 2.5.1-2 UDP 노드 리스너 인터페이스

2.5.2 작업 흐름

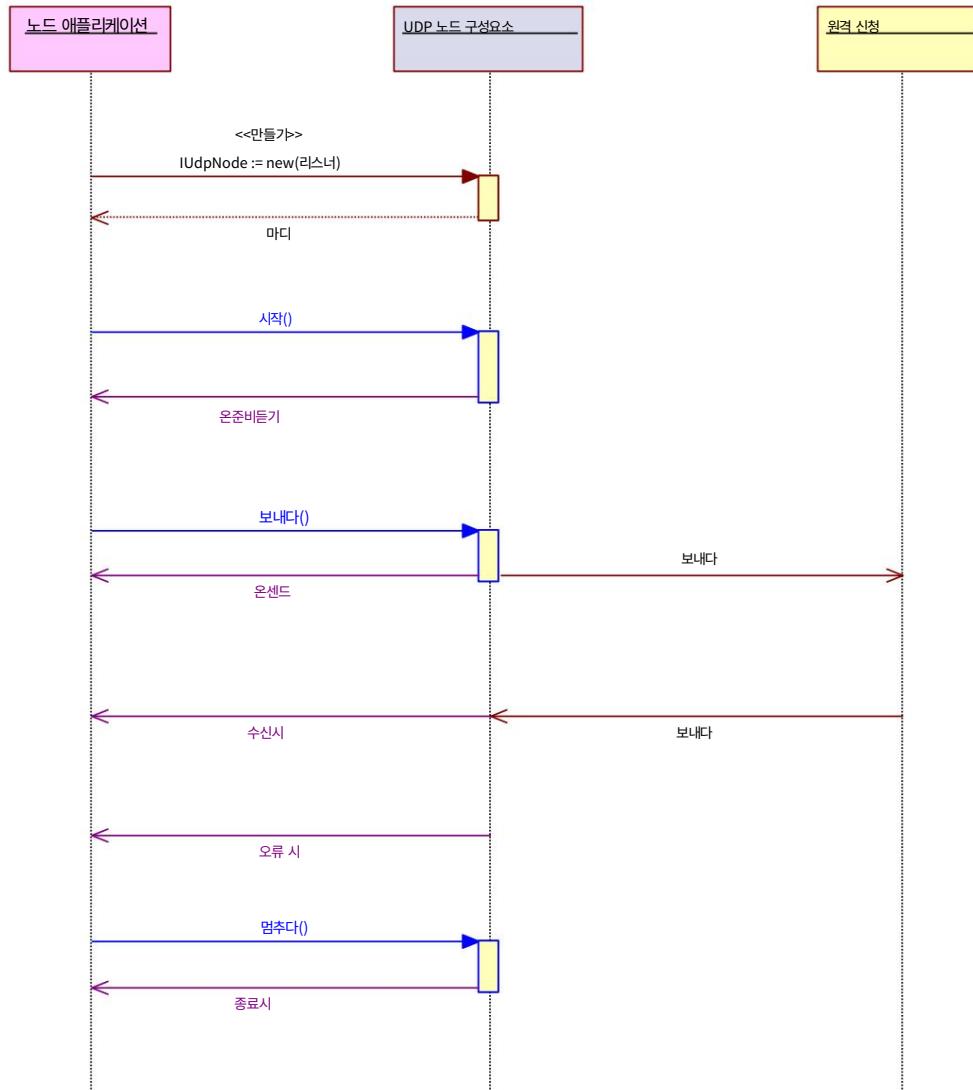


그림 2.5.2-1 UDP 노드 작업 흐름

그림 2.5.2-1은 응용 프로그램과 UDP 노드 구성 요소 간의 상호 작용 프로세스를 보여줍니다.

응용 프로그램은 UDP 노드 구성 요소를 시작하기 위해 `Start()` 메서드를 호출하고 호출이 성공하면 `TRUE`를 반환하고 `OnPrepareListen` 이벤트를 수신합니다. 애플리케이션은 데

이터가 전송된 후 `Send()`, `SendCast()` 및 기타 메서드를 호출하여 유니캐스트, 멀티캐스트 또는 브로드캐스트 데이터를 전송합니다.

JessMA

프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcdsaa/HP-Socket>

애플리케이션은 **OnSend** 이벤트를 수신합니다.

다. 원격 애플리케이션이 애플리케이션에 데이터를 보낼 때 애플리케이션은 **OnReceive** 이벤트를 수신합니다.

데이터 송수신에 예외가 발생하면 애플리케이션은 **OnError** 이벤트를 수신합니다. 애플리케이션

은 **Stop()** 메서드를 호출하여 UDP 노드 구성 요소를 닫습니다. 호출이 성공하면 **TRUE**를 반환하고 수신 합니다.

OnShutdown 이벤트에 .

3 SSL

3.1 컴포넌트 인터페이스

HP-Socket v3.5.x부터 모든 TCP 구성 요소는 SSL을 완벽하게 지원합니다. SSL 구성 요소는 SSL이 아닌 구성 요소와 동일한 인터페이스를 구현하며 동일한 방식으로 사용됩니다. 표 3.1-1에는 모든 SSL 구성 요소의 이름, 인터페이스, 수신기 인터페이스, 구현 클래스 및 분류가 나열되어 있습니다.

이름	구성 요소 인터페이스 리스너 인터페이스	클래스 역할 프로토콜 구현		
SSL 서버	ITcpServer ITcpServerListener	CSSL서버	서버 푸시	
	ITcpPullServer ITcpServerListener	CSSLPullServer	서버 풀	
	ITcpPackServer ITcpServerListener	CSSLPackServer 서버 팩		
	ITcpAgent ITcpServerListener	CSSLAgent	클라이언트 푸시	
	ITcpPullAgent ITcpAgentListener	CSSLPullAgent	클라이언트 풀	
	ITcpPackAgent ITcpServerListener	CSSLPackAgent 클라이언트 팩		
	ITcpClient ITcpClientListener	CSSL 클라이언트	클라이언트 푸시	
	ITcpPullClient ITcpClientListener	CSSLPullClient	클라이언트 풀	
	ITcpPackClient ITcpClientListener	CSSLPackClient	클라이언트 팩	

표 3.1-1 컴포넌트 분류

각 SSL 구성 요소의 계층 구조는 그림 3.1-1에 나와 있으며 모든 SSL 구성 요소는 해당 TCP 구성 요소에서 상속됩니다.

```
CSSLServer >> CTcpServer   CSSLAgent
>> CTcpAgent   CSSLClient >> CTcpClient
```

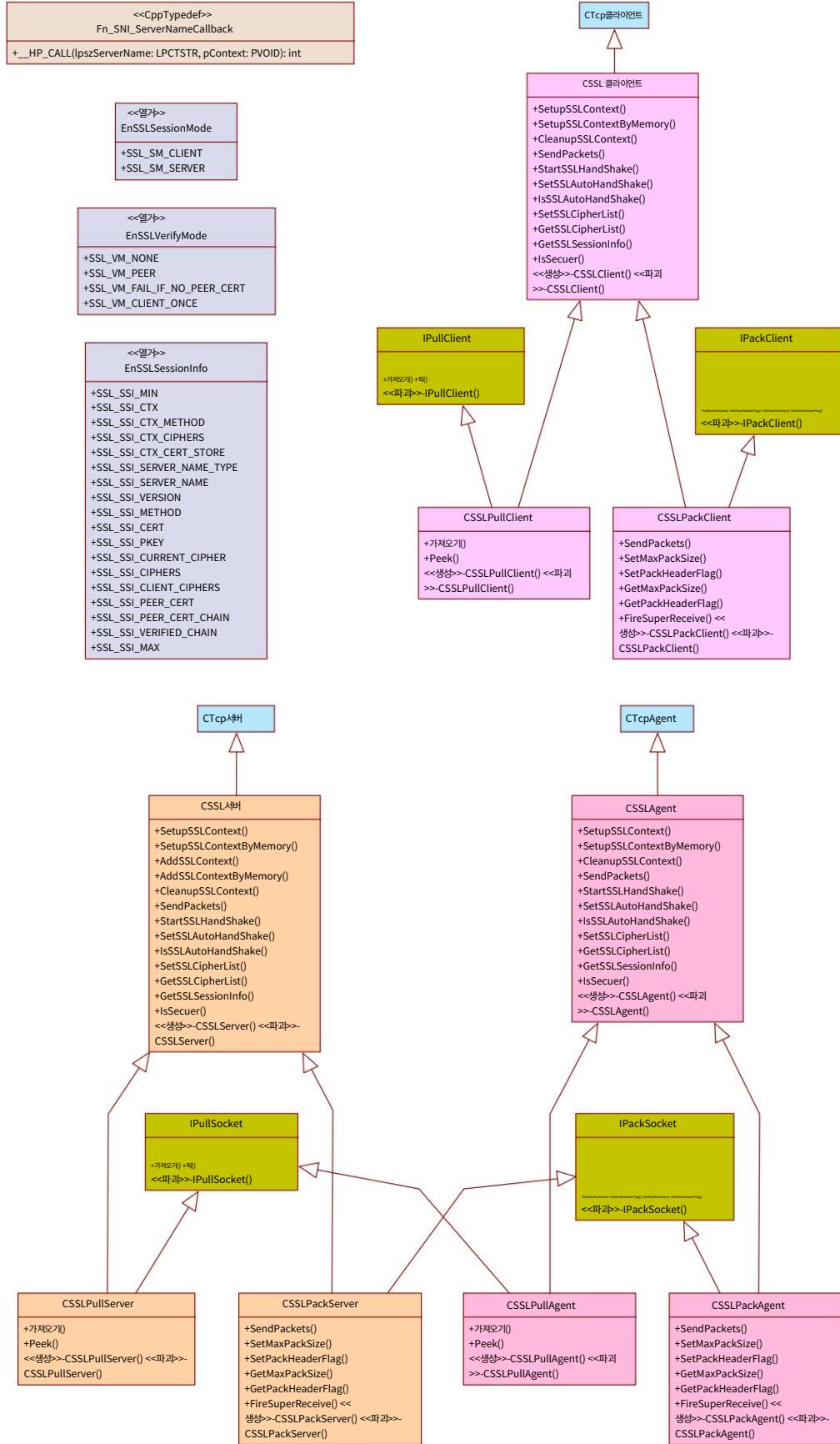
JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

그림 3.1-1 구성요소 계층

3.2 SSL 운영 환경

SSL 구성요소는 통신을 시작하기 전에 SSL 환경 매개변수를 초기화해야 하며 통신이 완료되면 SSL 실행 환경을 정리해야 합니다.

HP-Socket v4.xx 및 이전 버전은 모든 SSL 구성 요소에서 공유하는 전역적으로 고유한 SSL 운영 환경을 사용합니다.

환경 및 `HP_SSL_Initialize()` / `HP_SSL_AddServerContext()` / `HP_SSL_Cleanup()` / 을 통해

`HP_SSL_IsValid()` 와 같은 전역 기능은 전역 SSL 환경을 운영합니다. HP-Socket v5.0.x부터 각 SSL은

구성 요소는 별도의 SSL 런타임을 사용하므로 위의 기능이 제거되었습니다. 대신 SSL 구성 요소는 해당하는

자체 SSL 런타임 환경을 운영하기 위한 인스턴스 메소드:

SSL 암호화 스위트 설정

무효 `SetSSLCipherList(lpszCipherList)`

추가를 수정해야 하는 경우 HP-Socket의 기본 암호화 제품군은 `DEFAULT:laNULL:!eNULL:!SSLv2:!SSLv3`입니다.

암호 제품군(예: 이전 응용 프로그램 시스템을 지원하기 위해 SSLv3 활성화)은 `SetSSLCipherList()`를 호출하여 필요한 암호를 설정 해야 합니다.

암호 제품군. 암호 제품군에 대한 자세한 내용은 [OpenSSL 암호를 참조하십시오. 문서](#).

알아채다 : 암호 그룹 설정을 적용하려면 메소드 전에 호 **호출해야합니다** `SetupSSLContext[ByMemory]()` 、

`AddSSLContext[ByMemory]()` 출하십시오. `SSLCipherList()` 설정 암호 스위트를 설정합니다.

SSL 환경 매개변수 초기화 :

```
BOOL SetupSSLContext(enSessionMode, iVerifyMode = SSL_VM_NONE, lpszPemCertFile =
    nullptr, lpszPemKeyFile = nullptr, lpszKeyPasswod = nullptr, lpszCAPemCertFileOrPath = nullptr,
    [fnServerNameCallback = nullptr])
```

```
BOOL SetupSSLContextByMemory(enSessionMode, iVerifyMode = SSL_VM_NONE, lpszPemCert = nullptr,
    lpszPemKey = nullptr, lpszKeyPasswod = nullptr, lpszCAPemCert = nullptr, [fnServerNameCallback = nullptr])
```

매개변수 `iVerifyMode`는 SSL 확인 모드를 지정하며 매개변수 `lpszPemCert[파일]`, `lpszPemKey[파일]`,
`lpszKeyPasswod` 및 `lpszCAPemCert[FileOrPath]`는 인증서 파일, 개인 키 파일, 개인 키 암호 및
CA 인증서 파일/디렉토리, `fnServerNameCallback` 매개변수는 HTTPS 서버에만 사용되며 이 매개변수는 SNI 콜백을 지정합니다.

함수 포인터 호출, 이 매개변수의 값이 `nullptr` 인 경우 HP-Socket의 기본 SNI 콜백 함수가 사용됩니다.

`HP_SSL_DefaultServerNameCallback(lpszServerName, pContext)`에 따른 기본 SNI 콜백 함수
다음과 같은 인증서를 찾기 위해 `BindSSLServerName(lpszServerName, iContextIndex)`에 의해 설정된 도메인 이름-인증서 바인딩 관계
찾을 수 없으면 기본 인증서가 사용됩니다. 초기화가 성공하고 TRUE를 반환하고 실패하면 FALSE를 반환하며 초기화를 통과하지 못합니다.
`SYS_GetLastError()`는 오류 코드를 가져옵니다. (매개변수에 대한 자세한 내용은 `include /hpsocket/HPSocket-SSL.h` 또는
포함/`hpsocket/HPSocket4C-SSL.h` 헤더 파일)

SNI 호스트 인증서 추가 (SSL 서버 구성 요소):

```
int AddSSLContext(iVerifyMode, lpszPemCertFile, lpszPemKeyFile, lpszKeyPasswod = nullptr,
    lpszCAPemCertFileOrPath = nullptr)
```

```
int AddSSLContextByMemory(iVerifyMode, lpszPemCert, lpszPemKey, lpszKeyPasswod = nullptr, lpszCAPemCert
= nullptr)
```

SSL 서버 구성 요소에만 사용됩니다. `iVerifyMode` 매개변수는 SSL 확인 모드를 지정하고 매개변수 `lpszPemCert[File]`, `lpszPemKey[File]`, `lpszKeyPasswod` 및 `lpszCAPemCert[FilePath]`는 각각 인증서 파일과 개인 키를 지정합니다. 파일, 개인 키 비밀번호 및 CA 인증서 파일/디렉토리. 실행에 성공하면 SNI 호스트 인증서에 해당하는 인덱스가 반환됩니다. SNI 콜백 함수에서 SNI 호스트를 찾는 데 사용되며, 실패 시 -1을 반환하고 오류 코드는 `SYS_GetLastError()`를 통해 얻을 수 있습니다. 암호.

서버 호스트의 도메인 이름을 SNI 인증서(SSL 서버 구성 요소)에 바인딩합니다.

```
BOOL BindSSLServerName(lpszServerName, iContextIndex)
```

SSL 서버 구성 요소에만 사용됩니다. 서버 호스트 도메인 이름 `lpszServerName`을 `iContextIndex` 인증서에 바인딩합니다. 인덱스, 기본 인증서 인덱스는 0이고 다른 인증서 인덱스는 `AddSSLContext()` / `AddSSLContextByMemory()`에 의해 생성됩니다. 설립하다. 바인딩 실패는 `FALSE`를 반환하며 오류 코드는 `SYS_GetLastError()`를 통해 얻을 수 있습니다.

기본 SNI 콜백 기능(SSL 서버 구성요소):

```
int HP_SSL_DefaultServerNameCallback(lpszServerName, pContext)
```

SSL 서버 구성 요소에만 사용됩니다. 서버 호스트 도메인 이름 `lpszServerName`에 해당하는 인증서 색인을 얻으십시오. 호스트 도메인 이름이 인증서에 바인딩되지 않은 경우 기본 인증서 인덱스 0이 반환됩니다.

참고: 서버 응용 프로그램은 일반적으로 초기화 시 모든 호SSL 환경을 실행할 때 서버 호스트 `AddSSLContext[ByMemory]()` 스트 인증서를 해제하며 일부 특수 응용 프로그램은 `BindSSLServerName()` 의 도메인 이름을 해당 인증서에 바인딩하기 위해 호출됩니다. 다중 스레딩으로 인한 액세스 충돌을 피하기 위해 동 SNI 호스트 인증서를 호출해야 합니다. `AddSSLContext()`BindSSLServerName() 적으 `HP_SSL_DefaultServerNameCallback()` 동기식 처리를 위한 코드 스냅, 로 호출될 수 있습니다.

SSL 세션 정보 얻기 :

```
BOOL GetSSLSessionInfo([dwConnID, ]enInfo, lppInfo)
```

SSLServer 및 Agent 구성요소는 대상 연결을 지정하기 위해 `dwConnID` 매개변수가 필요하고 `enInfo` 매개 변수는 정보 유형; 획득에 성공하면 `TRUE`를 반환하고 `lppInfo` 매개변수는 획득한 정보 값을 저장합니다. 가져오기 실패 `FALSE`를 반환하며 오류 코드는 `SYS_GetLastError()`를 통해 얻을 수 있습니다. (파라미터 `enInfo`의 자세한 설명을 참조하십시오. `include/hpsocket/HPTTypeDef.h` 헤더 파일)

SSL 환경 정리 :

```
무효 CleanupSSLContext()
```

구성 요소가 통신을 중지하면(Stop() 호출) SSL 환경이 자동으로 정리되므로 애플리케이션은 SetupSSLContext()는 이 함수를 수동으로 호출하지 않고 구성 요소의 SSL 환경 매개변수를 초기화합니다.

스레드 로컬 SSL 환경 리소스 정리 (글로벌 기능):

무효 HP_SSL_RemoveThreadLocalState()

SSL을 운영하는 모든 스레드는 종료 시 스레드의 로컬 환경에서 SSL 리소스를 정리해야 합니다. 메인 스레드와 HP-Socket 작업자 스레드는 통신이 종료되면 자동으로 스레드의 로컬 환경에서 SSL 리소스를 정리합니다. 끝납니다. 따라서 일반적으로 이 함수를 수동으로 호출 할 필요는 없습니다. 특수한 경우에는 사용자 지정 스레드가 HP-Socket 통신 작업에 참여하는 경우(예: 통신 구성 요소 전송 전략이 SP_DIRECT이고 사용자 지정 스레드가 Send()를 호출 하는 경우) 데이터 전송 방법) 확인하고 SSL 메모리 누수가 있는 경우 구성 요소가 종지될 때마다 이 사용자 지정 스레드에서 이 함수를 호출해야 합니다.

3.3 SSL 악수

기본적으로 SSL 구성 요소(Https 구성 요소 포함)는 연결이 설정된 직후 SSL 핸드셰이크를 시작합니다(OnHandShake 이벤트는 OnConnect / OnAccept 이벤트 직후에 트리거됨). 그러나 일부 시나리오에서는 SSL 통신을 시작하기 전에 몇 가지 사전 작업을 수행해야 하며, 예를 들어 프록시 서버를 통해 대상 서버와 통신하는 것이 일반적인 시나리오입니다. 이 시나리오에서는 SSL 통신을 시작하기 전에 프록시 서버에 대한 연결을 설정해야 합니다.

HP-Socket v5.4.2부터 SSL 핸드셰이크를 수동으로 시작하기 위한 지원을 제공합니다. SSL 구성 요소의 SetSSLAutoHandShake(FALSE) 메서드를 호출하여 구성 요소를 수동 핸드셰이크 모드로 설정하고 StartSSLHandShake()를 호출하여 사전 작업 수행 후 SSL 통신을 시작합니다.

SSL 핸드셰이크를 수동으로 시작합니다.

BOOL StartSSLHandShake(dwConnID)

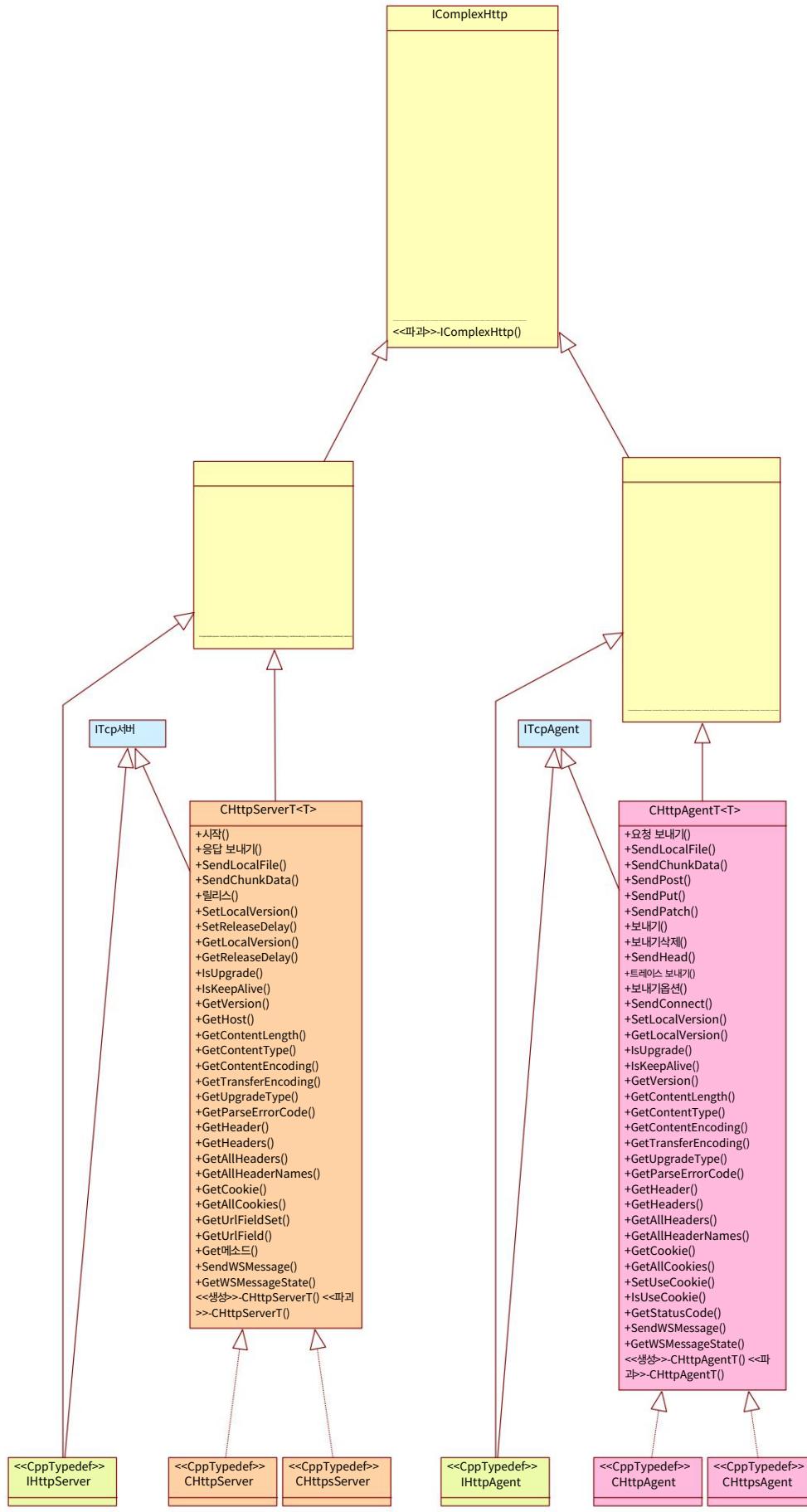
4 HTTP

4.1 컴포넌트 인터페이스

HP-Socket v4.0.x는 HTTP 구성 요소를 추가하기 시작했습니다. HTTP 컴포넌트는 해당 TCP 컴포넌트를 상속받아 HTTP 관련 동작 메소드를 추가하고, HTTP 컴포넌트 리스너도 해당 TCP 컴포넌트 리스너를 상속받아 HTTP 관련 통신 이벤트를 제공한다. 표 4.1-1은 모든 HTTP 구성 요소의 이름, 인터페이스, 수신기 인터페이스, 구현 클래스 및 TCP 구성 요소의 상위 클래스를 나열합니다.

이름	구성 요소 인터페이스 리스너 인터페이스	클래스 역할 구현		기본 클래스
HTTP 서버	IHttpServer IHttpServerListener	CHttpServer	서버 CTcp서버	
https 서버	IHttpServer IHttpServerListener	CHttpsServer	서버 CSSL서버	
HTTP 에이전트	IHttpAgent IHttpAgentListener	CHttpAgent	클라이언트 CTcpAgent	
https 에이전트	IHttpAgent IHttpAgentListener	CHttpsAgent	클라이언트 CSSLAgent	
HTTP 클라이언트	IHttpClient IHttpClientListener	CHttpClient	고객	CTcp클라이언트
Https 클라이언트	IHttpClient IHttpClientListener	CHttpsClient	클라이언트 CSSL클라이언트	
HTTP 동기화 클라이언트	IHttpSyncClient IHttpClientListener	CHttpSyncClient	고객	CTcp클라이언트
Https 동기화 클라이언트	IHttpSyncClient IHttpClientListener	CHttpsSyncClient	클라이언트 CSSLClient	

표 4.1-1 구성 요소 분류



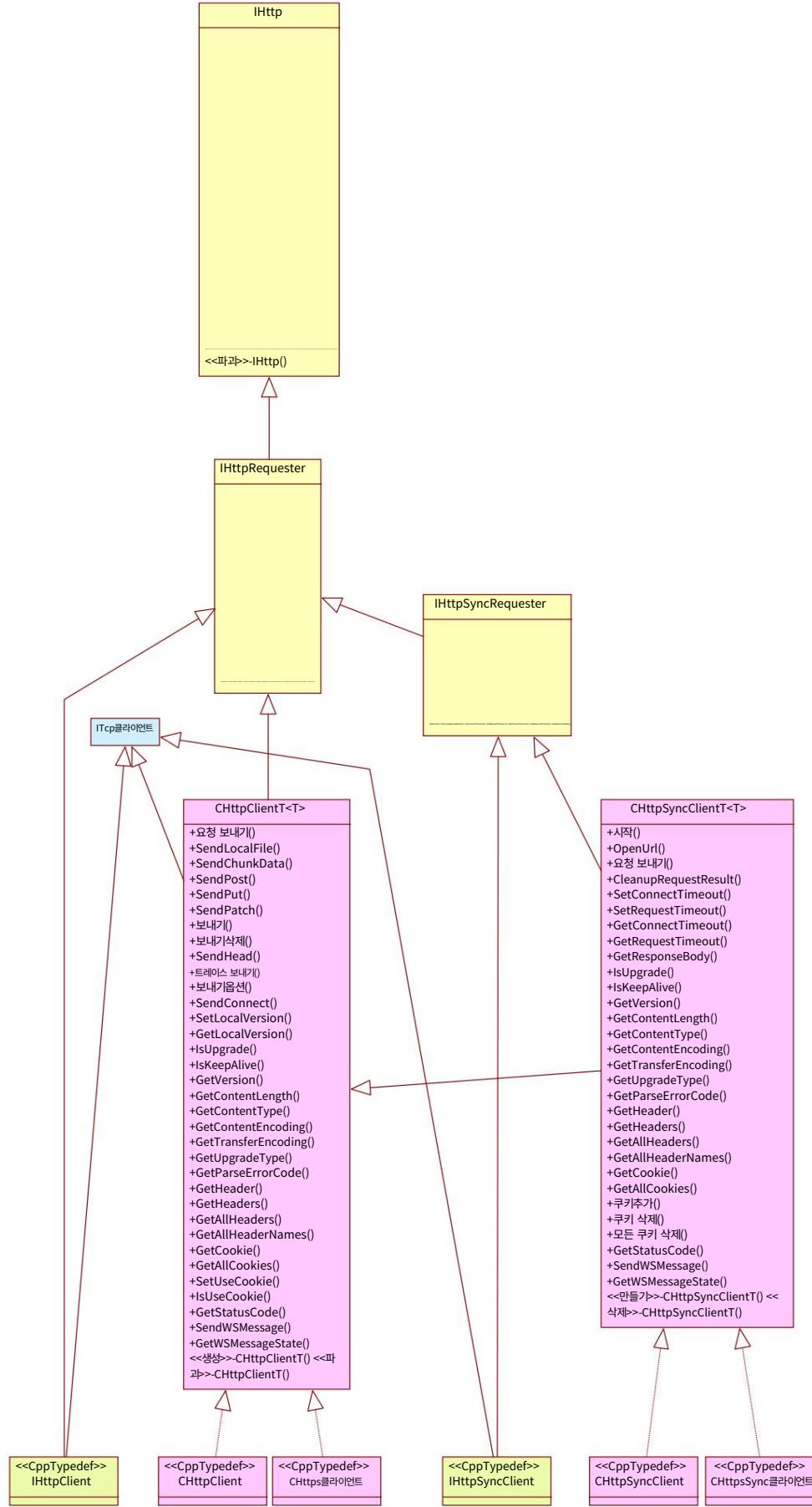


그림 4.1-1 HTTP 구성 요소 계층

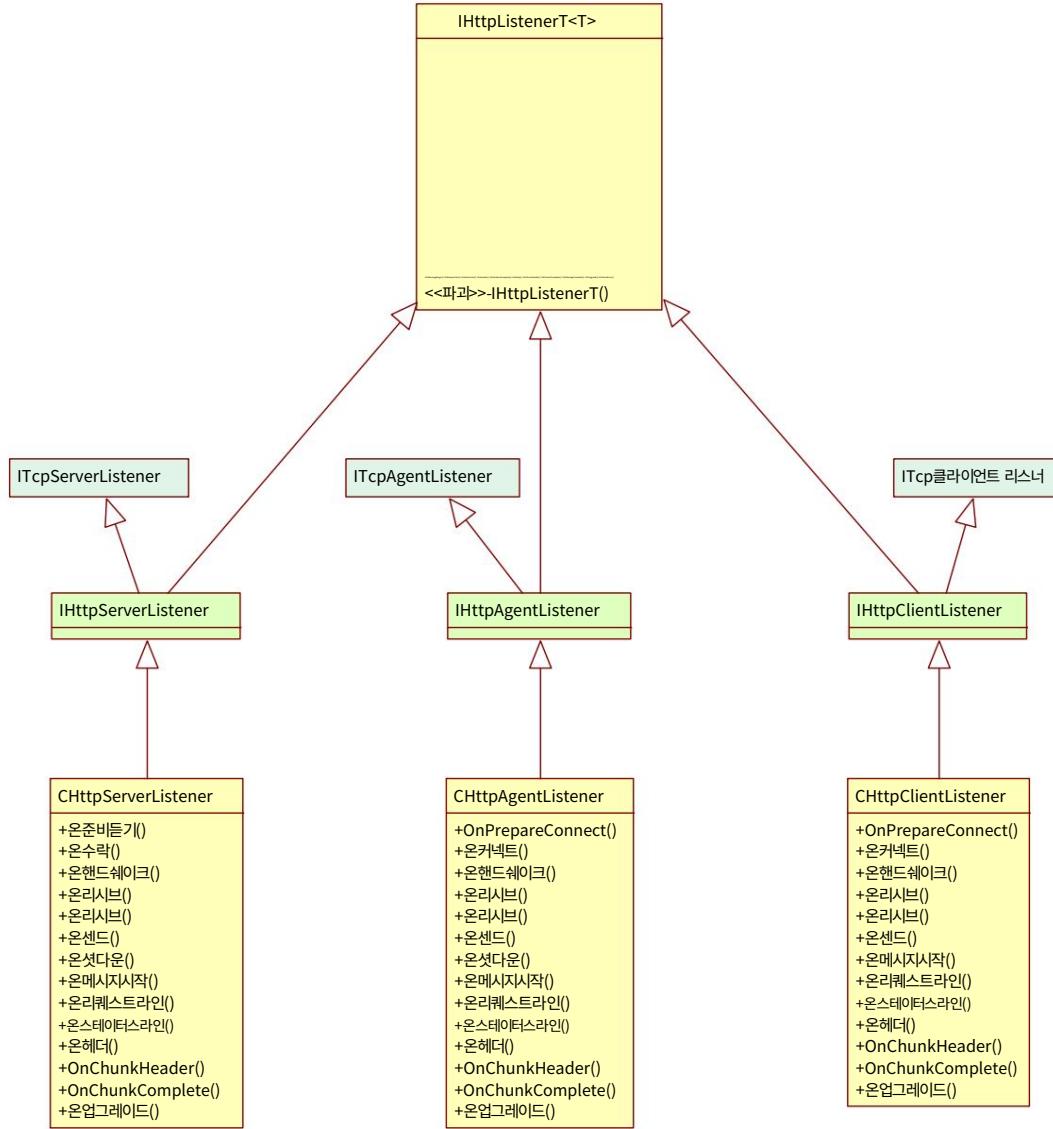


그림 4.1-2 HTTP 구성 요소 수신기 계층

4.2 리스너 이벤트

HTTP 구성 요소 수신기 인터페이스는 **IHttpListenerT** 및 해당 TCP 구성 요소 수신기 인터페이스에서 상속됩니다.

IHttpListenerT의 HTTP 이벤트는 TCP 컴포넌트 리스너의 **OnReceive** 이벤트 분해로 이해할 수 있습니다. HTTP 이벤트 반환 값의 유형은 **EnHttpParseResult**입니다.

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

- ✓ **HPR_OK** : 구문 분석 성공, 계속 실행
- ✓ **HPR_SKIP_BODY** : 현재 요청 BODY를 건너뛰고 이 요청을 완료합니다.
(OnHeadersComplete 이벤트 예만 해당)
- ✓ **HPR_UPGRADE** : 프로토콜 업그레이드, 이 요청 완료, 후속 HTTP 분석 없음
(OnHeadersComplete 이벤트 예만 해당)
- ✓ **HPR_ERROR** : 파싱 오류, 파싱 종료, 연결 해제

이벤트 구문 분석 시작:

EnHttpParseResult OnMessageBegin(pSender, dwConnID)

- ✓ **pSender** -- 이벤트 소스 객체
- ✓ **dwConnID** -- 연결 ID

요청 라인 파싱 완료 이벤트 (예만 사용 HTTP 섬기는 사람) :

EnHttpParseResult OnRequestLine(pSender, dwConnID, lpszMethod, lpszUrl)

- ✓ **pSender** -- 이벤트 소스 객체
- ✓ **dwConnID** -- 연결 ID
- ✓ **lpsz방법** ✓ -- 요청 메서드 이름
- ✓ **lpszUrl** -- 요청 라인의 URL 필드

상태 표시줄 구문 분석 완료 이벤트 (예만 사용 HTTP 고객) :

EnHttpParseResult OnStatusLine(pSender, dwConnID, usStatusCode, lpszDesc)

- ✓ **pSender** -- 이벤트 소스 객체
- ✓ **dwConnID** -- 연결 ID
- ✓ **usStatusCode** -- HTTP 상태 코드
- ✓ **lpszDesc** -- 상태 설명

요청 헤더 이벤트:

EnHttpParseResult OnHeader(pSender, dwConnID, lpszName, lpszValue)

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

✓ pSender	-- 이벤트 소스 객체
✓ dwConnID	-- 연결 ID
✓ lpsz이름 ✓	-- 요청 헤더 이름
lpsz값	-- 요청 헤더 값

요청 헤더 완료 이벤트:

EnHttpParseResult OnHeadersComplete(pSender, dwConnID)

✓ pSender	-- 이벤트 소스 객체
✓ dwConnID	-- 연결 ID

BODY 메시지 이벤트:

EnHttpParseResult OnBody(pSender, dwConnID, pData, iLength)

✓ pSender	-- 이벤트 소스 객체
✓ dwConnID	-- 연결 ID
✓ pData ✓ i	-- 데이터 버퍼
길이	-- 데이터 길이

첨크 헤더 이벤트:

EnHttpParseResult OnChunkHeader(pSender, dwConnID, iLength)

✓ pSender	-- 이벤트 소스 객체
✓ dwConnID	-- 연결 ID
✓ 길이	-- 첨크 메시지 본문 데이터 길이

첨크 패킷 종료 이벤트:

EnHttpParseResult OnChunkComplete(pSender, dwConnID)

✓ pSender	-- 이벤트 소스 객체
✓ dwConnID	-- 연결 ID

전체 구문 분석 이벤트:

EnHttpParseResult OnMessageComplete(pSender, dwConnID)

✓ pSender	-- 이벤트 소스 객체
✓ dwConnID	-- 연결 ID

업그레이드 프로토콜 이벤트:

EnHttpParseResult OnUpgrade(pSender, dwConnID)

- ✓ pSender -- 이벤트 소스 객체
- ✓ dwConnID -- 연결 ID
- ✓ enUpgradeType -- 프로토콜 유형

구문 분석 오류 이벤트:

EnHttpParseResult OnParseError(pSender, dwConnID, iErrorCode, lpszErrorDesc)

- ✓ pSender -- 이벤트 소스 객체
- ✓ dwConnID -- 연결 ID
- ✓ iErrorCode -- 에러 코드
- ✓ lpszErrorDesc -- 오류 설명

웹 소켓 헤더 이벤트:

EnHandleResult OnWSMessageHeader(pSender, dwConnID, bFinal, iReserved, iOperationCode, lpszMask, ullBodyLen)

- ✓ pSender ✓ -- 이벤트 소스 객체
- dwConnID -- 연결 ID
- ✓ 최종 -- 프레임 종료 여부
- ✓ iReserved -- RSV1/RSV2/RSV3에 대해 각각 1비트
- ✓ iOperationCode -- 연산 코드: 0x0 - 0xF
- ✓ lpszMask ✓ --mask (`nullptr` 또는 4바이트 마스크, `nullptr` 인 경우 마스크 없음)
- ullBodyLen -- 메시지 본문 길이

웹 소켓 패킷 본문 이벤트:

EnHandleResult OnWSMessageBody(pSender, dwConnID, pData, iLength)

- ✓ pSender -- 이벤트 소스 객체
- ✓ dwConnID -- 연결 ID
- ✓ pData ✓ i -- 메시지 본문 데이터 버퍼
- 길이 -- 메시지 본문 데이터 길이

웹 소켓 패킷 완료 이벤트:

EnHandleResult OnWSMessageComplete(pSender, dwConnID)



프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

- | | |
|------------|--------------|
| ✓ pSender | -- 이벤트 소스 객체 |
| ✓ dwConnID | -- 연결 ID |

[OnHeadersComplete](#), [OnBody](#), [OnMessageComplete](#) 및 [OnParseError](#)는 가장 기본적인 4가지입니다.

HTTP 이벤트, 모든 애플리케이션은 이러한 이벤트를 처리해야 하며 기타 이벤트는 애플리케이션의 실제 상황을 기반으로 합니다.

과정. 예: 응용 프로그램이 청크 패킷을 처리하지 않는 경우 [OnChunkHeader](#) 및 [OnChunkHeader](#)를 무시할 수 있습니다.

[OnChunkComplete](#) 이벤트, 응용 프로그램이 프로토콜 업그레이드를 수행하지 않는 경우 [OnUpgrade](#) 이벤트는 무시할 수 있습니다.

HP-Socket은 웹 소켓 및 Http 터널 프로토콜 업그레이드를 지원하며 프로토콜이 업그레이드되면 [OnUpgrade](#) 이벤트가 트리거됩니다.

이벤트에서 이 이벤트의 [enUpgradeType](#) 매개변수는 업그레이드 유형을 나타냅니다.

<<열기>>
EnHttpUpgradeType
+헛_없음
+HUT_WEB_SOCKET
+HUT_HTTP_TUNNEL
+헛_알 수 없음

- ✓ HUT_WEB_SOCKET :웹소켓
- ✓ HUT_HTTP_TUNNEL :HTTP 터널

프로토콜 업그레이드가 완료되면 HTTP 구문 분석이 수행되지 않으며 일반 HTTP 이벤트가 트리거되지 않습니다. 업그레이드 유형이 웹 소켓, [OnWSMessageHeader](#), [OnWSMessageBody](#) 및 [OnWSMessageComplete](#) 이벤트 트리거 웹 소켓 데이터 처리, 업그레이드 유형이 HTTP 터널인 경우 수신되는 모든 후속 데이터는 TCP로 간주됩니다. 데이터, [OnReceive](#) 이벤트를 트리거합니다.

알아채다:

동기화 클라이언트 :동기화 HTTP 클라이언트 구성 요소([CHttpClient](#) 그리고 [CHttpSyncClient](#))
는 모든 이벤트를 내부적으로 처리하므로 리스너를 바인딩할 필요가 없습니다(생성자의 리스너 매개변수가 전달됨). ; 루이면 수신기는 구성 요소의 통신 프로세스를 추적할 수 있습니다.

4.3 쿠끼 관리

HP-Socket v4.2.x부터 프로세스 수준을 제공합니다.

기타 쿠끼 관리자인 관리자는 표준 HTTP 쿠끼 기능을 구현하고 Max-Age httpOnly expires secure 등을 지원합니다. ↴

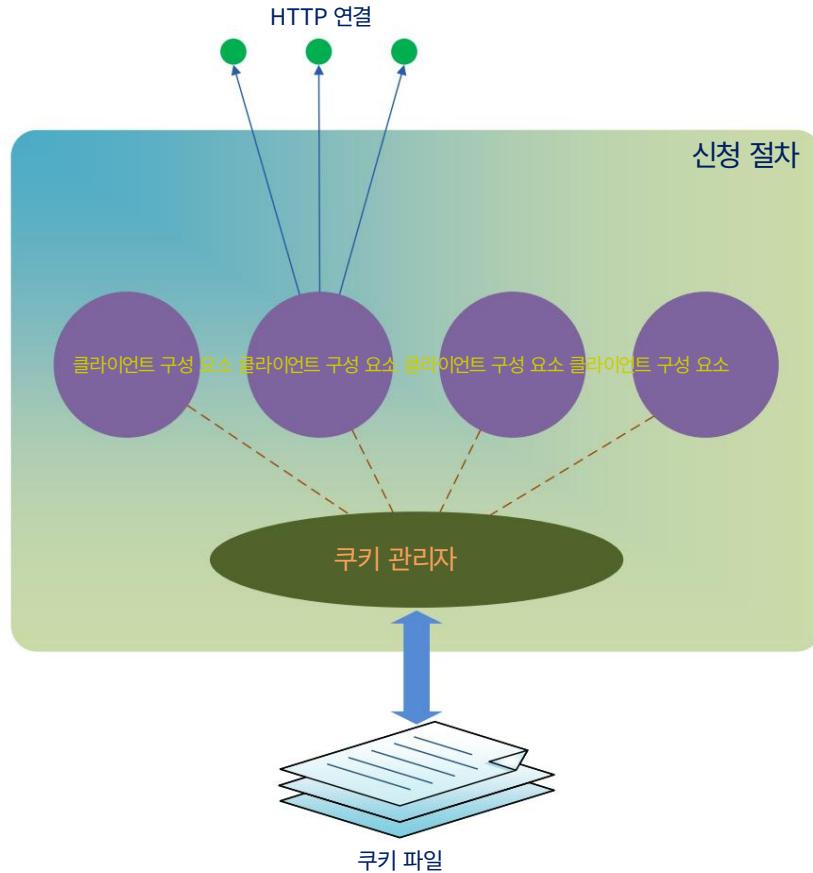


그림 4.3-1 쿠키 관리자

그림 4.3-1에서 볼 수 있듯이 관리자는 모든 HTTP 클라이언트 구성 요소에서 생성된 쿠키를 메모리에 유지하며 다른 위치에 저장할 수 있습니다.

쿠키는 연결과 다른 구성 요소 간에 공유되며 쿠키 직렬화 및 역직렬화가 지원됩니다.

클라이언트 구성 요소는 기본적으로 쿠키를 사용하도록 설정되어 있으며, 클라이언트 구성 요소가 쿠키를 사용하도록 설정되어 있는 한 수신된 쿠키는 관리자에 자동으로 저장되며 HTTP 요청이 있을 때 관리자에서 자동으로 로드됩니다.

구성 요소가 쿠키를 사용하지 않도록 설정한 경우 (설정 방법: SetUseCookie(FALSE)) 구성 요소 [가 받는 쿠키는](#) 구문 분석되지 않고 관리자에 저장되지 않으며 HTTP 요청을 보낼 때 관리자에서 쿠키를 로드하지 않습니다.

참고: 관리자는 내부적으로 읽기-쓰기 잠금 제어를 사용하여 구성 요 **쿠키 액세스**, 높은 동시성은 약간의 성능 손실을 가져옵니다. 만약에 쿠키가 사용되지 않도록 결정하면 사용하**지 않도록** 설정할 수 있습니다.

HP-Socket은 관리자를 운영하기 위한 일부 관리 기능을 제공하며, 응용 프로그램 처리를 용이하게 하기 위한 쿠키 헬퍼 기능도 제공합니다.
쿠키:

파일에서 쿠키 로드 :

```
BOOL HP_HttpCookie_MGR_LoadFromFile(lpszFile, bKeepExists)
```

✓ lpszFile ✓	-- 문서
bKeepExists	-- 원래 쿠키를 관리자에 보관할지 여부

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcsaa/HP-Socket>

파일에 쿠키 저장 :

BOOL HP_HttpCookie_MGR_SaveToFile(lpszFile, bKeepExists)

✓ lpszFile ✓	-- 문서
bKeepExists	-- 원본 쿠키를 파일에 보관할지 여부

쿠키 정리 :

BOOL HP_HttpCookie_MGR_ClearCookies(lpszDomain, lpszPath)

✓ lpsz도메인 ✓ lpsz	-- 도메인, 비어 있음은 모든 도메인을 의미합니다.
경로	-- 경로, 비어 있음은 모든 경로를 의미합니다.

만료된 쿠키 삭제 :

BOOL HP_HttpCookie_MGR_RemoveExpiredCookies(lpszDomain, lpszPath)

✓ lpsz도메인 ✓ lpsz	-- 도메인, 비어 있음은 모든 도메인을 의미합니다.
경로	-- 경로, 비어 있음은 모든 경로를 의미합니다.

쿠키 설정 :

BOOL HP_HttpCookie_MGR_SetCookie(lpszName, lpszValue, lpszDomain, lpszPath, iMaxAge, bHttpOnly, bSecure, enSameSite, bOnlyUpdateValueIfExists)

✓ lpszName ✓	-- 이름
lpszValue ✓	-- 값
lpszDomain ✓	-- 영역
lpszPath ✓	-- 길
iMaxAge ✓	-- 라이프 사이클: > 0 -> 라이브 초, = 0 -> 즉시 삭제, < 0 -> 애플리케이션이 끝날 때까지
bHttpOnly	-- HttpOnly 속성이 있습니까?
✓ 비시큐어	-- 보안 속성이 있습니까?
✓ enSameSite	-- SameSite 속성: 0 -> 없음, 1 -> 엄격, 2 -> LAX
✓ bOnleUpdateValueIfExists	-- 쿠키가 이미 존재하는 경우에만 쿠키 값을 업데이트할지 여부

쿠키 삭제:

BOOL HP_HttpCookie_MGR_DeleteCookie(lpszDomain, lpszPath, lpszName)

✓ lpsz도메인 ✓ lpsz	-- 영역
경로 ✓ lpsz이름	-- 길
	-- 이름

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/ldcasa/HP-Socket>

타사 쿠키 허용 여부 설정:

무효 HP_HttpCookie_MGR_SetEnableThirdPartyCookie(bEnableThirdPartyCookie)

✓ **bEnableThirdPartyCookie** -- TRUE -> 허용, FALSE -> 비활성화

타사 쿠키 허용 여부 확인:

BOOL HP_HttpCookie_MGR_IsEnableThirdPartyCookie()

✓

쿠키 만료 문자열을 정수로 변환합니다.

BOOL HP_HttpCookie_HLP_ParseExpires(lpszExpires, ptmExpires)

✓ **lpszExpires** ✓ -- 만료 문자열
ptmExpires -- 정수 포인터 만료

쿠키로의 정수 변환 만료 문자열:

BOOL HP_HttpCookie_HLP_MakeExpiresStr(lpszBuff, piBuffLen, tmExpires)

✓ **lpszBuff** ✓ -- 문자열 버퍼
piBuffLen ✓ -- 버퍼 길이
tmExpires -- 만료 정수

쿠키 문자열 생성 :

BOOL HP_HttpCookie_HLP_ToString(lpszBuff, piBuffLen, lpszName, lpszValue, lpszDomain, lpszPath, iMaxAge, bHttpOnly, bSecure, enSameSite)

✓ **lpszBuff** -- 문자열 버퍼
✓ **piBuffLen** -- 버퍼 길이
✓ **lpszName** ✓ -- 이름
lpszValue ✓ -- 값
lpszDomain ✓ -- 영역
lpszPath -- 길
✓ **iMaxAge** -- 라이프 사이클: > 0 -> 라이브 초, = 0 -> 즉시 삭제, < 0 -> 애플리케이션이 끝날 때까지
✓ **bHttpOnly** -- HttpOnly 속성이 있습니까?
✓ **비시큐어** -- 보안 속성이 있습니까?
✓ **enSameSite** -- SameSite 속성: 0 -> 없음, 1 -> 엄격, 2 -> LAX

현재 UTC 시간 가져오기:

JessMA

프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/l0c0saa/HP-Socket>

`__time64_t HP_HttpCookie_HLP_CurrentUTCTime()`

✓

최대 연령 -> 만료 :

`__time64_t HP_HttpCookie_HLP_MaxAgeToExpires(iMaxAge)`

✓ `iMaxAge` -- 라이프 사이클: > 0 -> 라이브 초, = 0 -> 즉시 삭제, < 0 -> 애플리케이션이 끝날 때까지

만료 -> 최대 연령 :

`int HP_HttpCookie_HLP_ExpiresToMaxAge(tmExpires)`

✓ `tm만료` -- 만료 정수

4.4 HTTP 통신 시작

기본적으로 Http 구성 요소(Htts 구성 요소 포함)는 연결이 설정된 직후 HTTP 통신을 시작합니다 (`OnReceive` 이벤트는 일련의 HTTP 통신 이벤트로 분해됨). 그러나 일부 시나리오에서는 HTTP 통신을 시작하기 전에 몇 가지 사전 작업을 수행해야 하며, 예를 들어 SOCKS 프록시 서버를 통해 대상 서버와 통신하는 것이 일반적인 시나리오입니다. 이 시나리오에서는 HTTP 통신을 시작하기 전에 SOCKS 프록시 서버에 대한 연결을 설정해야 합니다.

HP-Socket v5.4.3부터 수동으로 HTTP 통신을 시작하는 지원을 제공합니다. HTTP 구성 요소의 `SetHttpAutoStart(FALSE)` 메서드를 호출하여 HTTP 통신 모드를 수동으로 시작하도록 구성 요소를 설정하고, 사전 작업이 수행된 후 HTTP 통신을 시작하려면 `StartHttp()`를 호출합니다.

수동으로 HTTP 통신 시작 :

`BOOL 시작Http(dwConnID)`

5 UDP ARQ

HP-Socket v5.5.x부터 UDP ARQ(Automatic Repeat Request UDP, Reliable UDP) 구성 요소인 **IUpdArqServer** 및 **IUpdArqClient**를 제공합니다. 대역폭이 낮고 대기 시간이 길거나 패킷 손실이 심각한 네트워크 환경에서 TCP보다 효율적인 네트워크 전송 성능을 제공할 수 있습니다.

5.1 컴포넌트 인터페이스

UDP ARQ 컴포넌트 **IUpdArqServer** / **IUpdArqClient**의 동작 방식 및 리스너 인터페이스는 기존 UDP 컴포넌트 **IUpdServer** / **IUpdClient**와 동일 하며 기존 UDP 컴포넌트를 기반으로 ARQ 통신과 관련된 일부 파라미터 설정 및 획득 방식만 추가된다.

이름	구성 요소 인터페이스 리스너 인터페이스	구현 클래스	역할	기본 클래스
UDP 서버	IUpdServer IUpdServerListener	CUpd서버	섬기는 사람	
UDP 클라이언트	IUpd클라이언트 IUpdClientListener	CUpd클라이언트	고객	
UDP 캐스트	IUpdCast IUpdCastListener	CUpdCast	고객	
UDP ARQ 서버	IUpdArqServer IUpdServerListener	CUpdArqServer	서버 CUpdServer	
UDPARQ 클라이언트	IUpdArq클라이언트 IUpdClientListener	CUpdArq클라이언트	고객	CUpd클라이언트

표 5.1-1 UDP 구성 요소의 분류

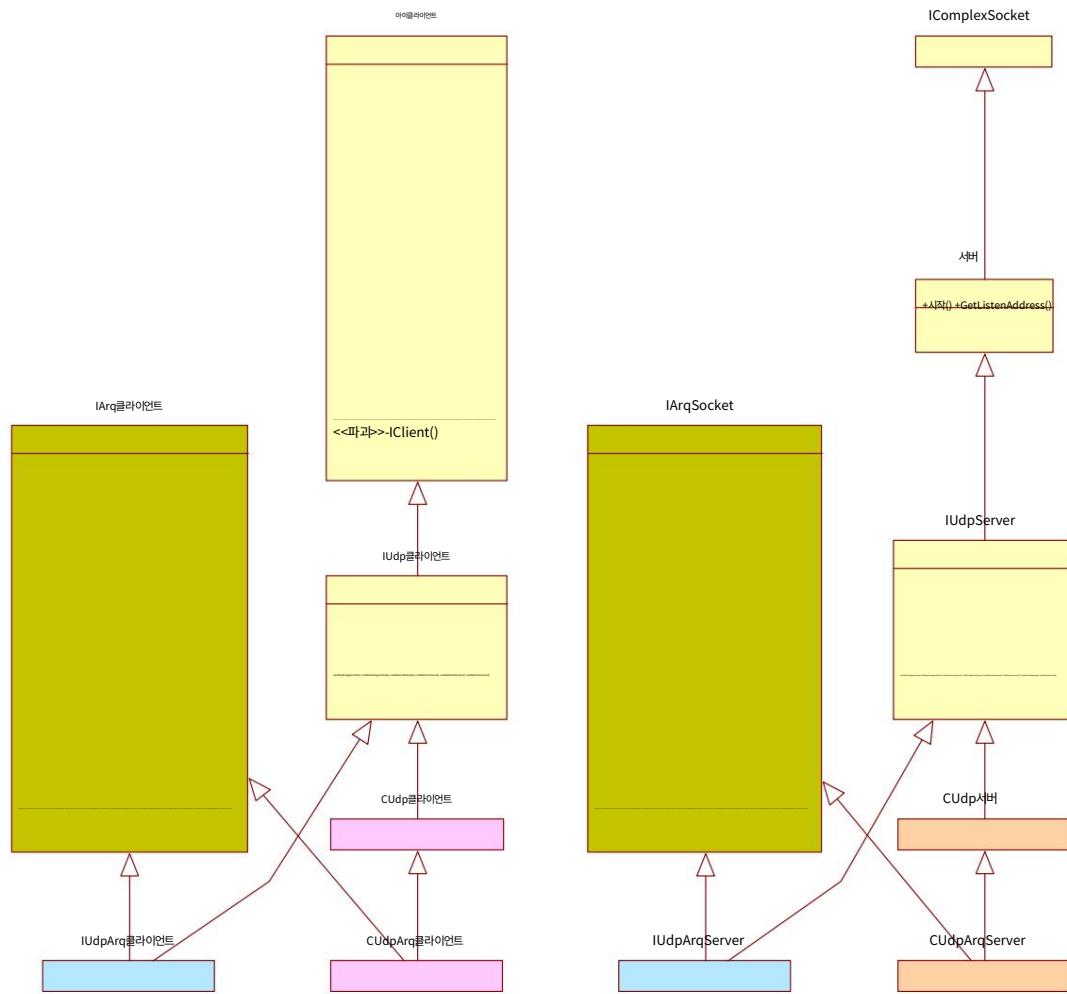


그림 5.1-1 UDP 구성 요소 계층

5.2 핸드셰이크 프로토콜

HP 소켓은 [KCP를 사용합니다.](#) ARQ 통신을 구현하지만 통신 협상 방식이 기존 KCP와 다릅니다. 두 통신 당사자 모두 자신의 독립 세션 ID를 사용할 수 있습니다.

그림 5.2-1과 같이 HP-Socket의 ARQ 구성 요소는 통신을 시작하기 전에 통신 핸드셰이크를 수행해야 합니다. [IUdpArqServer](#) 구성 요소가 [OnAccept](#) 이벤트를 수신하거나 [IUdpArqClient](#) 구성 요소가 [OnConnect](#) 이벤트를 수신하면 ARQ 핸드셰이크를 시작합니다. 시간 초과로 완료 또는 종료되었습니다. [OnHandshake](#) 이벤트는 핸드셰이크가 완료되면 수신되고 [OnClose](#) 이벤트는 핸드셰이크가 시간 초과되고 오류 코드가 [ERROR_TIMEOUT](#) 일 때 수신됩니다.

협상 메시지 형식:

MM C F XXXX YYYY

✓ 메시지 길이 : 12 바이트

JessMA

프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcdsaa/HP-Socket>

- ✓ 처음 1-2바이트: 고정 매직 넘버 0xBB4F
 - ✓ 세 번째 바이트: 명령 유형, 현재 핸드셰이크 명령 0x01 만 사용
 - ✓ 4번째 바이트: 명령 식별, 0x00 - 완료되지 않음, 0x01 - 완료됨
 - ✓ 바이트 5-8: 로컬 세션 ID
 - ✓ 9~12바이트 : 상대방의 세션 ID, 상대방의 세션 ID를 받지 못한 경우 0x00000000 으로 채워짐

알아채다: IUpdArqServer 클라이언트별" IP 주소 + 포트 + 대화 ID

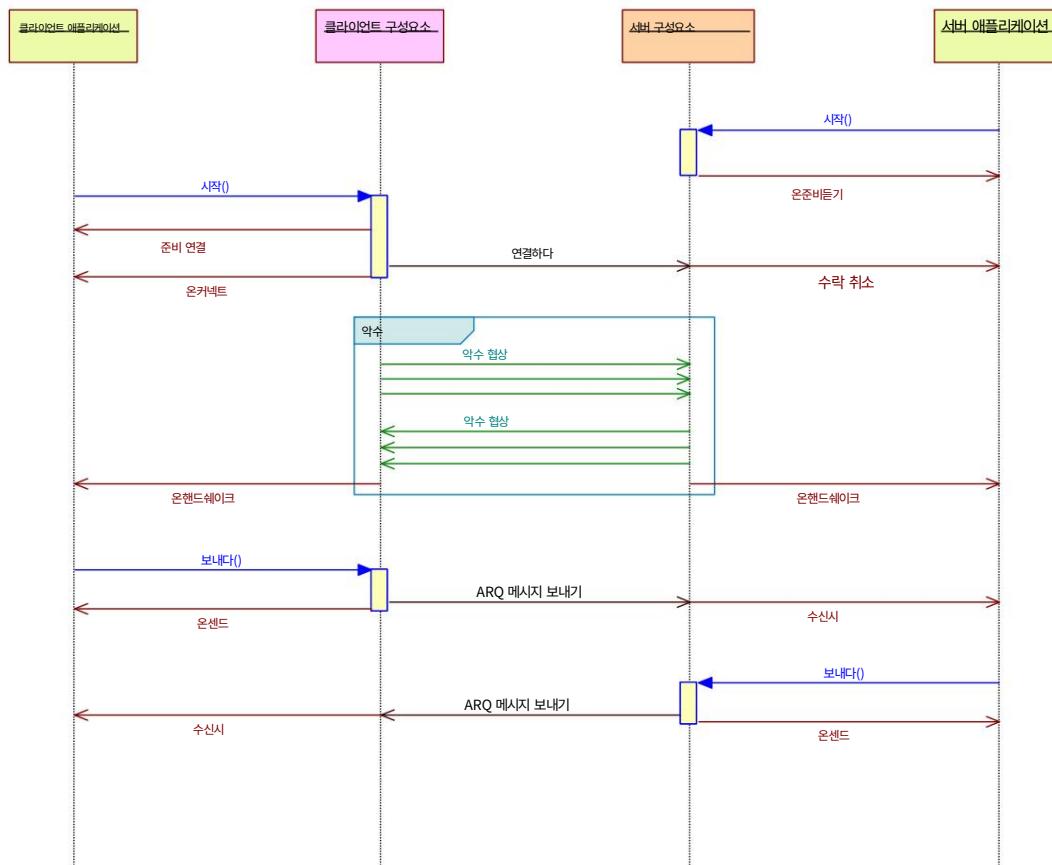


그림 5.2-1 ARO 통신 과정

5.3 구성 매개변수

SetNoDelay(BOOL)

열지 여부 지체없이 모드(기본값: 거짓 , 열지 마)

SetTurnoffCongestCtrl(BOOL)

혼잡 제어 비활성화 여부(기본값: 거짓, 닫히지 않음)

SetFlushInterval(DWORD)

데이터 새로 고침 간격(밀리초, 기본값: 60)

JessMA

프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/l0c0aa/HP-Socket>

SetResendByAcks(DWORD)

빠른 재전송 ACK 스패 수(기본값: 0, 빠른 재전송 끄기)

SetSendWndSize(DWORD)

전송 창 크기(패킷 수, 기본값: 128)

SetRecvWndSize(DWORD)

수신 창 크기(패킷 수, 기본값: 512)

SetMinRto(DWORD)

최소 재전송 제한 시간(밀리초, 기본값: 30)

SetMaxTransUnit(DWORD)

최대 전송 단위(기본값: 0, 그리고 UDP 매개변수 [SetMaxDatagramSize\(DWORD\)](#) 일관됨)

SetMaxMessageSize(DWORD)

최대 패킷 크기(기본값: 4096)

SetHandShakeTimeout(DWORD)

핸드셰이크 시간 초과(밀리초, 기본값: 5000)

[SetMaxMessageSize\(DWORD\)](#) 및 [SetHandShakeTimeout\(DWORD\)](#)를 제외한 기타 구성 매개변수 [KCP_관련](#) 정보는 KCP의 관련 파라미터 설명을 참조하십시오.

6 스레드 풀

6.1 컴포넌트 인터페이스

HP-Socket v5.4.x부터 스레드 풀 구성 요소 IHPThreadPool이 제공되어 사용자가 통신 논리와 비즈니스 논리의 분리를 실현하고 응용 프로그램의 전반적인 실행 효율성을 향상시킬 수 있습니다. IHPThreadPool은 다음과 같은 주요 동작 방법을 제공합니다. 이러한 메서드는 성공 시 TRUE, 실패 시 FALSE를 반환하며 시스템 오류 코드는 실패 시 SYS_GetLastError()를 통해 얻을 수 있습니다.

IHP스레드풀

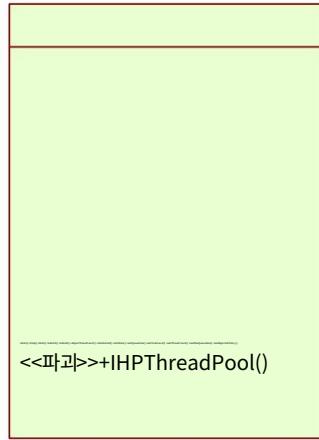


그림 6.1-1 스레드 풀 구성 요소 인터페이스

스레드 풀 시작

BOOL 시작(dwThreadCount = 0, dwMaxQueueSize = 0, enRejectedPolicy = TRP_CALL_FAIL, dwStackSize = 0)

- ✓ **dwThreadCount** 스레드 수, (기본값: 0)
 - >0 : dwThreadCount
 - =0 : (CPU 코어 번호 * 2 + 2)
 - <0 : (CPU 코어 수 * (-dwThreadCount))
- ✓ **dwMaxQueueSize** 작업 대기열 최대 용량(기본값: 0, 무제한) ✓
- enRejectedPolicy** 작업 거부 처리 정책
 - TRP_CALL_FAIL** : (기본값) 즉시 실패 반환
 - TRP_WAIT_FOR** : 대기(성공, 시간 초과 또는 스레드 풀 종료로 인해 실패할 때까지)
 - TRP_CALLER_RUN** : 호출자 스레드가 직접 실행
- ✓ **dwStackSize** 스레드 스택 공간 크기(기본값: 0 -> 운영 체제 기본값)

스레드 풀 닫기 지정

된 시간 내에 스레드 풀 구성 요소를 닫음 최대 대기 시간 내에 작업 중인 스레드가 정상적으로 닫히지 않으면 강제 종료를 시도하며 이 경우 시스템 자원 유출이 발생할 수 있습니다.

BOOL 중지(dwMaxWait = **INFINITE**)

- ✓ dwMaxWait 최대 대기 시간(밀리초, 기본값: INFINITE, 항상 대기)

작업 제출

BOOL 제출(fnTaskProc, pvArg, dwMaxWait = **INFINITE**)

- ✓ fnTaskProc 태스크 처리 기능
- ✓ pvArg 작업 매개변수
- ✓ dwMaxWait 작업 제출 최대 대기 시간(밀리초, TRP_WAIT_FOR 유형 스레드 예만 해당)
풀이 적용됩니다. 기본값: INFINITE, 항상 대기).

** SYS_GetLastError() 오류 코드 ERROR_DESTINATION_ELEMENT_FULL은 다음을 의미합니다.

작업 대기열이 가득 찬 것입니다.

소켓 작업 제출

BOOL 제출(pTask, dwMaxWait = **INFINITE**)

- ✓ pTask 작업 매개변수
 - ✓ dwMaxWait 작업 제출 최대 대기 시간(밀리초, TRP_WAIT_FOR 유형 스레드 예만 해당)
풀이 적용됩니다. 기본값: INFINITE, 항상 대기)
- ** SYS_GetLastError() 오류 코드 ERROR_DESTINATION_ELEMENT_FULL은 다음을 의미합니다.
- 작업 대기열이 가득 찬 것입니다.

참고: 시간 ~~스레드~~ 풀은 HP_Create_SocketTaskObj() pTask 성공적으로 제출되었을 때 ~~제출(pTask)~~

자동 소멸을 담당합니다.

캐시, 커밋이 실패하면 응용 프로그램에서 수동으로 호출해야 함

작업

HP_Destroy_SocketTaskObj() pTask 자는 캐시를 파괴합니다.

스레드 풀 크기를 동적으로 조정

BOOL AdjustThreadCount(dwNewThreadCount)

- ✓ dwNewThreadCount 스레드 수
 >0 : dwNewThreadCount
 =0 : (CPU 코어 번호 * 2 + 2)
 <0 : (CPU 코어 수 * (-dwNewThreadCount))

6.2 리스너 이벤트

HP-Socket v5.8.5부터 스레드 풀 수신기 인터페이스 IHPThreadPoolListener, 스레드 풀 그룹

구성 요소는 수신기를 바인딩하여 다음 수명 주기 이벤트를 처리할 수 있습니다.

JessMA

프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcscsa/HP-Socket>

Thread Pool 시작 이벤트

무효 OnStartup(pThreadPool)

✓ pThreadPool -- 이벤트 소스 객체

스레드 풀 종료 이벤트

무효 OnShutdown(pThreadPool)

✓ pThreadPool -- 이벤트 소스 객체

작업자 스레드 시작 이벤트 (작업자 스레드당 한 번 트리거됨)

무효 OnWorkerThreadStart(pThreadPool, dwThreadId)

✓ pThreadPool -- 이벤트 소스 객체 -- 작업자 스

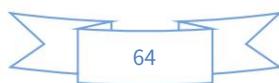
✓ dwThreadId 레드 ID

작업자 스레드 종료 이벤트 (작업자 스레드당 한 번 트리거됨)

무효 OnWorkerThreadEnd(pThreadPool, dwThreadId)

✓ pThreadPool -- 이벤트 소스 객체

✓ dwThreadId -- 작업자 스레드 ID



7 압축 / 압축 해제

7.1 컴포넌트 인터페이스

이전 버전의 HP-Socket은 일회성 압축/압축 해제와 관련된 보조 기능을 제공했는데, 소위 "일회성"은 압축/압축 해제 기능으로 전달되는 입력 데이터가 데이터 조각이 아닌 완전한 데이터여야 함을 의미합니다. 압축/압축 해제 기능은 입력 데이터의 원샷 압축/압축 해제를 수행합니다. 따라서 단일 압축/압축 해제 함수는 적은 양의 입력 데이터만 처리할 수 있습니다.

<code>int SYS_Compress()</code>	: ZLib 압축
<code>int SYS_CompressEx()</code>	: ZLib 고급 압축
제 <code>int SYS_Uncompress()</code> :	: ZLib 압축 해
제 <code>SYS_UncompressEx()</code> : ZLib 압축 해제 <code>int</code>	<code>int</code> 결과 길이 추측
<code>int SYS_GZipCompress()</code>	: GZip 압축
제 <code>int SYS_GZipUncompress()</code>	: GZip 압축 해
<code>int SYS_GZipGuessUncompressBound()</code> : Gzip 압축 해제 결과 길이 추측	
<code>int SYS_BrotliCompress()</code>	: Brotli 압축
<code>int SYS_BrotliCompressEx()</code>	: Brotli 고급 압축
제 <code>int SYS_BrotliUncompress()</code>	: Brotli 압축 해
<code>int SYS_BrotliGuessCompressBound()</code> : Brotli 압축 결과 길이 추측	

HP-Socket v5.8.6부터 대량의 입력 데이터 또는 불확실한 양의 스트림 데이터를 처리할 수 있는 스트림 데이터 압축/압축 해제 인터페이스를 제공합니다.

7.1.1 압축기 컴포넌트 인터페이스



그림 7.1.1-1 압축기 컴포넌트 인터페이스

프로세스 입력 데이터

`BOOL Process(pData, iLength, bLast, pContext = nullptr)`

`BOOL ProcessEx(pData, iLength, bLast, bFlush, pContext = nullptr)`

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcscsa/HP-Socket>

- ✓ **pData** to-be-compressed 데이터 버퍼 ✓
- iLength** to-be-compressed 데이터 길이 ✓
- bLast** 플래그: 압축할 마지막 데이터인지 여부 ✓ **bFlush** 강제 새로 고침
- 여부(강제 새로 고침은 압축 효율성을 감소시키지만 데이터는 분할 및 압축됨) ✓ **pContext** 컨텍스트 매개변수: 콜백 함수
- Fn_CompressDataCallback**에 전달

애플리케이션은 이 메서드를 주기적으로 호출하여 스트리밍 또는 세그먼트화된 데이터를 압축하고 실행에 성공하면 TRUE를 반환할 수 있습니다.

FALSE를 반환하며, 실패 시 **SYS_GetLastError()**를 통해 오류 코드를 얻을 수 있습니다.

압축기 재설정

불 리셋()

새 스트림 처리를 시작하기 전에 압축기를 재설정해야 합니다.

(참고: 압축기는 데이터 스트림 후 자동으로 재설정되기 때문에 응용 프로그램은 일반적으**프로세스()** 메서드가 처리를 완료하거나 처리를 중단합니다.

로 이 메서드를 명시적으로 호출할 필요가 없습니다.)

압축기 사용 가능 여부 확인

BOOL 유효 여부()

7.1.2 압축해제기 컴포넌트 인터페이스

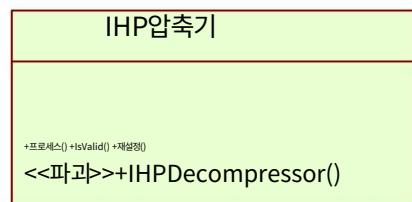


그림 7.1.2-1 Decompressor 컴포넌트 인터페이스

프로세스 입력 데이터

BOOL 프로세스(**pData**, **iLength**, **pContext** = **nullptr**)

- ✓ 압축 해제할 **pDat** 데이터 버퍼 ✓ 압축 해제할
- iLength** 데이터 길이 ✓ **pContext** 컨텍스트 매
- 개변수: 콜백 함수 **Fn_DecompressDataCallback**에 전달

애플리케이션은 이 메서드를 주기적으로 호출하여 스트리밍 또는 세그먼트화된 데이터의 압축을 풀고 실행에 성공하면 TRUE를 반환할 수 있습니다.

FALSE를 반환하며, 실패 시 **SYS_GetLastError()**를 통해 오류 코드를 얻을 수 있습니다.

압축해제기 재설정

불 리셋()

압축 해제기는 새 스트림 처리를 시작하기 전에 재설정해야 합니다.

(참고: 데이터 스트림 후 압축 해제기가 자동으로 재설정되기 때문에 응용 프로그램은 일반적프로세스() 메서드가 처리를 완료하거나 처리를 중단합니다.

으로 이 메서드를 명시적으로 호출할 필요가 없습니다.)

압축해제기 사용 가능 여부 확인

BOOL 유효 여부()

7.1.3 출력 데이터 콜백

출력 데이터 콜백 기능

```
typedef BOOL (*Fn_DataCallback)(pData, iLength, pContext); typedef
Fn_DataCallback Fn_CompressDataCallback; typedef Fn_DataCallback
Fn_DecompressDataCallback;
```

- ✓ 압축 해제할 **pData** 데이터 버퍼 ✓ 압축 해제
할 **iLength** 데이터 길이 ✓ **pContext** 컨텍스
트 매개변수: **Process()** 메서드에 의해 전달됨

압축기/압축해 제기의 **Process(pData, iLength, [bLast,] pContext)** 메서드를 실행하는 동안 출력이 생성되면 출력 데이터와 **pContext**를 입력 매개변수로 사용하여 "출력 데이터 콜백 함수"가 호출됩니다. 이 콜백 함수는 압축기/압축 해제 개체를 생성할 때 지정됩니다(참조: 개체 생성).

콜백 함수는 처리를 계속하려면 **TRUE**를 반환하고, 처리를 종단하려면 **FALSE**를 반환하고, **Process()**는 **FALS**를 반환하고,
SYS_GetLastError()는 **ERROR_CANCELLED** 오류 코드를 반환합니다.

7.2 객체 생성

현재 ZLib/GZip 및 Brotli 압축 알고리즘을 지원합니다.

ZLib 압축기 객체 생성

IHPCompressor* **HP_Create_ZLibCompressor(fnCallback, iWindowBits, iLevel, iMethod, iMemLevel, iStrategy)**

- ✓ **fnCallback** 출력 데이터 콜백 기능
- ✓ 매개변수 기본값 **WindowBits** = 15, **수준** = -1, **방법** = 8, **MemLevel** = 8, **전략** = 0

JessMA

프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/ldcasa/HP-Socket>

GZip 압축기 객체 생성

IHPCompressor* HP_Create_GZipCompressor(fnCallback, iLevel, iMethod, iMemLevel, iStrategy)

- ✓ fnCallback 출력 데이터 콜백 기능
- ✓ 매개변수 기본값 iLevel = -1, iMethod = 8, iMemLevel = 8, iStrategy = 0

Brotli 압축기 객체 생성

IHPCompressor* HP_Create_BrotliCompressor(fnCallback, iQuality, iWindow, iMode)

- ✓ fnCallback 출력 데이터 콜백 함수 iQuality = 11,
- ✓ 매개변수 기본값 iWindow = 22, iMode = 0

ZLib 압축 풀기 객체 만들기

IHPDecompressor* HP_Create_ZLibDecompressor(fnCallback, iWindowBits)

- ✓ fnCallback 출력 데이터 콜백 기능
- ✓ 매개변수 기본값 iWindowBits = 15

GZip 압축 풀기 객체 생성

IHPDecompressor* HP_Create_GZipDecompressor(fnCallback)

- ✓ fnCallback 출력 데이터 콜백 기능

Brotli 압축해제기 객체 생성

IHPDecompressor* HP_Create_BrotliDecompressor(fnCallback)

- ✓ fnCallback 출력 데이터 콜백 기능

압축기 객체 파괴

무효 HP_Destroy_Compressor(pCompressor)

- ✓ pCompressor 압축기 객체 포인터

압축해제기 객체 파괴

무효 HP_Destroy_Decompressor(pDecompressor)

JessMA

프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/ldcsaa/HP-Socket>

✓ pDecompressor 압축 해제 개체 포인터

8 리눅스

Linux용 HP-Socket에서 제공하는 API 인터페이스는 Windows 버전과 일치하지만 구현 코드는 완전히 독립적입니다.
Linux용 HP-Socket은 C++14 표준의 새로운 기능을 사용하여 컴파일을 위해 GCC 6.x 이상의 컴파일러가 필요합니다.

번역했습니다.

알아채다: Linux용 HP 소켓 **컴파일 및 실행 요구 사항:**

- 1) 리눅스 커널 버전: 버 2.6.32 이상
- 2) GCC 3) 전: 버 6.x 이상
- glibc 4) 전: 종속 2.14.x 이상
- 성 라이브러리책 libdl libpthread

8.1 컴파일

HP-Socket 릴리스 패키지는 x86 및 x64 플랫폼에서 컴파일된 바이너리 라이브러리 파일과 샘플 데모 실행 파일을 제공했습니다.
라인 파일(컴파일러 환경: 리눅스 2.6.32 、 GCC 6.3.1). 다음을 사용하여 직접 컴파일할 수도 있습니다.

- 1, 원도우 원격 편집: HP-Socket 및 해당 [샘플 데모는](#) HP-Socket 릴리스 패키지에서 [제공됩니다.](#)
Visual Studio 프로젝트 프로젝트에서 해당 프로젝트 프로젝트를 열어 HP-Socket 라이브러리 또는 샘플 데모를 컴파일할 수 있습니다.
또한 [script/ 디렉토리](#)에서 [ms-build-libs.bat](#), [ms-build-demos.bat](#) 및 [ms-build all.bat](#)을 실행하여 [각각](#) HP-Socket 라이브러리 또는 (및) 샘플 데모를 컴파일 할 수 있습니다.
2. `compile.sh`: HP-Socket 릴리스 패키지에서 제공하는 `script/compile.sh`를 사용하여 스크립트를 컴파일 할 수 있습니다.
[HP-Socket 라이브러리 파일을 번역합니다.](#) 참고: `compile.sh`는 샘플 데모가 아닌 HP-Socket 라이브러리만 [컴파일합니다.](#)

```
$ ./compile.sh -h
Usage: compile.sh [...O.P.T.I.O.N.S...]
-----
-d|--with-debug-lib : compile debug libs (default: true)
-m|--mem-allocator : memory allocator (default: mimalloc)
                   : (options: mimalloc / jemalloc / system)
-u|--udp-enabled : enable UDP components (default: true)
-t|--http-enabled : enable HTTP components (default: true)
-s|--ssl-enabled : enable SSL components (default: true)
-z|--zlib-enabled : enable ZLIB related functions (default: true)
-b|--brotli-enabled : enable BROTLI related functions
                   : (x86/x64 default: true, arm/arm64 default: false)
-i|--iconv-enabled : enable ICONV related functions (default: true)
-c|--compiler : compiler (default: g++)
-p|--platform : platform: x86 / x64 / arm / arm64
                : (default: current machine arch platform)
-e|--clean : clean compilation intermediate temp files
-r|--remove : remove all compilation target files
-v|--version : print hp-socket version
-h|--help : print this usage message
-----
```

그림 8.1-1 `compile.sh` 컴파일 스크립트

8.2 설치

릴리스 패키지에서 제공하는 `script/install.sh` 스크립트는 HP-Socket을 설치(또는 제거)하는 데 사용되며 설치(또는 제거)됩니다. 현재 플랫폼용 라이브러리 파일 `install.sh` 스크립트는 다음 매개변수를 지원합니다.

```
$ sudo ./install.sh -h
Usage: install.sh [...O.P.T.I.O.N.S...]
-----+
-p|--prefix   : install/uninstall path (default: /usr/local)
-l|--libdir    : lib dir (x86/arm default: 'lib', x64/arm64 default: 'lib64')
-d|--with-demo : install demos or not (default: false)
-u|--uninstall : execute uninstall operation from install path
-v|--version   : print hp-socket version
-h|--help      : print this usage message
-----+
```

그림 8.2-1 `install.sh` 설치 스크립트

참고: 설치 컴파일 시 예제를 설치하려는 경우 컴파일 스크립트는 `/src/포함/종속/lib/` 디렉토리에 `install.sh`
 스크립트는 디렉토리에 의존해야 하는 배포 패키지 스크립트/ / 포함하다/ / 포 패키지의 디렉토리에 따라 다릅니다. 데모 리, 실행 파일, 또한
 의 필요성에 따라 다른 디렉토리/출시/

8.3 안드로이드 NDK

HP-Socket은 Android NDK 라이브러리 파일 빌드 스크립트 `script/build-android-ndk.sh`(Windows build-android-평행한 탑: ndk.bat)를 제공합니다. NDK를 설치 및 구성한 후 `build-android-ndk.sh`를 실행하여 라이브러리를 빌드합니다. 파일 기본적으로 빌드 스크립트는 현재 NDK에서 지원하는 모든 ABI의 동적 및 정적 라이브러리를 빌드하고 빌드 대상 라이브러리 파일을 출력합니다. `lib/android-ndk/` 디렉토리로 이동합니다. 특별한 요구 사항이 있는 경우 빌드 스크립트에 해당하는 명령줄 매개 변수를 제공하십시오.

(기본 빌드 실행)

```
$ cd HP-Socket/Linux
$ ./build-android-ndk.sh
```

8.3.1 ABIS

빌드 스크립트는 기본적으로 현재 NDK에서 지원하는 오래되지 않은 모든 ABI 라이브러리 파일을 빌드합니다. 빌드 스크립트에 다음을 제공할 수 있습니다.
`APP_ABI` 명령줄 인수에 대한 ABI 관련 라이브러리 파일입니다.

(빌드만 armeabi-v7a 그리고 x86 ABI 라이브러리 파일)

```
$ ./build-android-ndk.sh APP_ABI=armeabi-v7a,x86
```

8.3.2 기능 스위치

빌드 스크립트는 기본적으로 모든 기능(UDP /SSL/HTTP/ZLIB/BROTLI/ICONV)을 빌드합니다.

특정 기능을 제거하기 위해 스크립트를 빌드 하려면 `_XXX_DISABLED=true` 인수를 제공하십시오 .

- ✓ `_UDP_DISABLED=true` : SSL 제거 : UDP 제거
- ✓ `_SSL_DISABLED=true` : SSL 제거
- `_HTTP_DISABLED=true` : 移除 HTTP
- ✓ `_ZLIB_DISABLED=true` : zlib 제거
- `_BROTLI_DISABLED=true` : 移除 BROTLI
- ✓ `_ICONV_DISABLED=true` : ICONV 제거
- ✓ `_MIMALLOC_DISABLED=true` : mimalloc 메모리 할당자 비활성화

(제거하다 SSL 그리고 ICONV)

```
$ ./build-android-ndk.sh _SSL_DISABLED=true _ICONV_DISABLED=true
```

참고: 라이브러리를 빌드할 때 특정 기능이 제거되면 라이브러리를 사용하는 애플리케이션을 컴파일할 때 해당 매크로를 사용하여 앱을 빌드할 때 정의해야 합니다.

`_SSL_BUILDSUPPORT` 그리고 `_ICONV_DISABLED` 매크로.

8.3.3 기타 옵션

빌드 스크립트 `build-android-ndk.sh` 는 빌드 스크립트 인 Android NDK 명령 `ndk-build` 의 단순한 래퍼입니다.

이는 `ndk-build` 의 모든 명령줄 매개변수를 지원합니다. `ndk-build` 에 대한 자세한 지침은 공식 문서를 참조하세요.

다음 예제는 다음 빌드 옵션을 보여줍니다.

- ✓ armeabi-v7a 및 x86_64 ABI 라이브러리 파일 빌드
- ✓ UDP, ZLIB 및 ICONV 기능 제거
- ✓ 라이브러리 파일은 lib/android-ndk/ 디렉토리로 내보내집니다.
- ✓ lib/android-ndk/obj 디렉토리에 obj 중간 파일 출력

```
$ ./build-android-ndk.sh APP_ABI=armeabi-v7a,x86_64 \
    _UDP_DISABLED=true \
    _ZLIB_DISABLED=true \
    _ICONV_DISABLED=true \
    NDK_LIBS_OUT=./lib/android-ndk \
    NDK_OUT=./lib/android-ndk/obj
```

9 가지 사용 방법

HP-Socket은 MBCS 및 유니코드 문자 집합을 지원하고 32비트 및 64비트 응용 프로그램을 지원합니다. 소스 생성

HP-Socket을 사용하기 위한 코드, DLL 또는 LIB. HPSocket DLL 및

HPSocket4C DLL。

참고: 제공된 라 **HP 소켓 v5.2.x** 더 이상 별도로 제공되지 않는 배포 패키지부터 시작 SSL 그리고 SSL 릴리스 패키지에 언급된 라이브러리 파일
라이브러리 파일에는 다음이 포함되어 있습니다. SSL 그리고 HTTP 매크로를 제거하고 다시 컴파일하 SSL 아님 HTTP 별도로 정의할 수 있는 구성요소
_SSL_비활성화됨 또는 _HTTP_비활성화됨 려는 경우 구성 요소.

9.1 소스 코드

HP-Socket은 **Dependent/** 디렉토리의 타사 라이브러리에 의존합니다. 따라서 소스 코드로 HP-Socket을 사용하십시오.

, HP-Socket의 **Src/**, **Include/** 및 **Dependent/** 디렉토리 아래에 해당 코드 파일 또는 라이브러리 파일을 추가해야 합니다.

엔지니어링 프로젝트로 전송합니다(참조: [TestEcho](#) / [TestEcho-UDP](#) 예제 데모).

9.2 정적 라이브러리

HP-Socket 배포 패키지의 Project/HPSocketLIB 및 Project/HPSocketLIB4C 프로젝트는 컴파일에 사용됩니다.

HPSocket LIB 및 HPSocket4C LIB를 번역하면 출력 디렉토리는 [Lib\x86\(x64\)/static](#)입니다. 원하는 경우 직접 만들 수 있습니다.

번역하다. 정적 라이브러리는 동적 라이브러리와 같은 방식으로 사용됩니다. 다음 장을 참조하십시오.

(정적 라이브러리 참조 방법 참조: [TestEcho-SSL-4C](#) / [TestEcho-SSL-PFM](#) 예제 데모).

참고: Windows 플랫폼에서 프로젝트가 HPSocket LIB 또는 HPSocket4C LIB를 사용하는 경우 다음 위치에 있어야 합니다.

프로젝트 속성 -> **HPSOCKET_STATIC_**에서 전처리 매크로를 정의합니다.

9.3 HP소켓 DLL

HPSocket DLL은 C++ 프로그램이 HP-Socket을 사용하는 데 선호되는 방법인 C++ 프로그래밍 인터페이스를 내보냅니다. HPSocket DLL은 HP-Socket 배포 패키지의 Project/HPSocketDLL 프로젝트를 통해 컴파일 및 생성되며 다음 DLL이 출력됩니다.

✓ Lib\x86\HPSocket.dll ✓	(32비트/MBCS/릴리즈)
Lib\x86\HPSocket_D.dll ✓	(32비트/MBCS/디버그)
Lib\x86\HPSocket_U.dll ✓	(32位/Unicode/Release)
Lib\x86\HPSocket_UD.dll ✓	(32위/Unicode/Debug)
Lib\x64\HPSocket.dll	(64비트/MBCS/릴리즈)
✓ Lib\x64\HPSocket_D.dll ✓	(64비트/MBCS/디버그)
Lib\x64\HPSocket_U.dll ✓	(64位/Unicode/Release)
Lib\x64\HPSocket_UD.dll	(64위/Unicode/Debug)



프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcdsaa/HP-Socket>

HPSocket DLL을 사용할 때 [/hpsocket/SocketInterface.h](#)를 포함 해야 합니다.

[include/hpsocket/HPSocket.h](#) 및 DLL에 해당하는 *.lib 파일이 프로젝트에 추가됩니다.

[Include/hpsocket/HPSocket.h](#)는 구성 요소의 생성 및 소멸 방법과 구성 요소 인터페이스를 내보낼 뿐만 아니라 각 구성 요소를 정의합니다.

HP 소켓 구성 요소를 보다 편리하게 사용할 수 있는 스마트 포인터(예: CTcpServerPtr / CTcpClientPtr) . (참조: [TestEcho-Pull / TestEcho-PFM 예제](#) 데모).

HPSocket DLL에는 SSL 구성 요소와 비SSL 구성 요소가 포함되어 있습니다. SSL 구성 요소를 사용해야 하는 경우

[include/hpsocket/HPSocket-SSL.h](#), [include/hpsocket/SocketInterface.h](#) 및 DLL에 해당하는 *.lib 파일

구성 요소가 엔지니어링 프로젝트에 추가됩니다. (참조: [TestEcho-SSL-Pack](#) 샘플 데모).

HP-Socket 업데이트 또는 업그레이드가 필요할 때 DLL을 통한 HP-Socket 사용, DLL 인터페이스가 발생하는 경우

DLL 인터페이스가 변경되면 응용 프로그램을 다시 컴파일해야 하며, DLL 인터페이스가 변경되지 않으면 재컴파일 없이 DLL을 직접 교체할 수 있습니다.

애플리케이션을 컴파일합니다.

9.4 HPSocket4C DLL

HPSocket4C DLL은 C 프로그래밍 인터페이스를 내보내 HP-Socket을 사용할 수 있도록 C 언어 또는 기타 프로그래밍 언어를 제공합니다.

HPSocket4C DLL은 HP-Socket 릴리스 패키지의 [Project/HPSocketDLL4C](#) 프로젝트를 통해 컴파일 및 생성됩니다.

다음 DLL을 출력합니다.

✓ Lib\x86\HPSocket4C.dll	(32비트/MBCS/릴리즈)
✓ Lib\x86\HPSocket4C_D.dll	(32비트/MBCS/디버그)
Lib\x86\HPSocket4C_U.dll	(32位/Unicode/Release)
Lib\x86\HPSocket4C_UD.dll	(32위/Unicode/Debug)
Lib\x64\HPSocket4C.dll	(64비트/MBCS/릴리즈)
✓ Lib\x64\HPSocket4C_D.dll	(64비트/MBCS/디버그)
Lib\x64\HPSocket4C_U.dll	(64位/Unicode/Release)
Lib\x64\HPSocket4C_UD.dll	(64위/Unicode/Debug)

HPSocket4C DLL을 사용하는 경우 DLL에 해당하는 [/hpsocket/HPSocket4C.h](#) 및 *.lib 파일을 포함해야 합니다.

구성 요소가 엔지니어링 프로젝트에 추가됩니다. (참조: [TestEcho-4C 예제](#) 데모).

HPSocket4C DLL에는 SSL 구성 요소와 비SSL 구성 요소가 포함되어 있습니다. SSL 구성 요소를 사용해야 하는 경우

[include/hpsocket/HPSocket4C-SSL.h](#) 및 DLL에 해당하는 *.lib 파일이 프로젝트에 추가됩니다. (인용하다:

[TestEcho-SSL-Pack](#) 데모).

HP-Socket을 업데이트하거나 업그레이드해야 할 때 DLL 인터페이스에서 전송하는 경우 4C DLL을 통해 HP-Socket을 사용합니다.

변경 사항이 있는 경우 응용 프로그램을 다시 컴파일해야 하며 DLL 인터페이스가 변경되지 않은 경우 DLL을 직접 교체하면 다시 컴파일할 필요가 없습니다.

애플리케이션을 새로 컴파일합니다.

9.5 확장 지원

HP-Socket 배포 패키지는 쉬운 언어 모듈과 지원 라이브러리 SDK를 제공합니다. SDK를 제공하지 않는 프로그래밍 언어의 경우 다음을 수행할 수 있습니다.

HPSocket4C DLL에서 제공하는 C 프로그래밍 인터페이스와 함께 HP-Socket을 사용합니다.

• 오픈소스 프로젝트 [HPSocket.net](#) HP-Socket용 .NetCore SDK를 제공합니다. • [macOS용 오픈 소스 프로젝트 HPSocket](#) HP-Socket의 Mac OSX 플랫폼 포팅 버전을 제공합니다.

10 부록

10.1 예제 데모

10.1.1 윈도우 예제

프로젝트 이름	구성 요소 사용	인용 설명	
Http프록시 (서버-1)	TCP 서버 TCP 에이전트	DLL HTTP 프록시 서버(TCP 구현)	
Http프록시 (서버-2)	HTTP 서버 TCP 에이전트	DLL HTTP 프록시 서버(HTTP 구현)	
테스트에코	TCP 서버 TCP 클라이언트	SRC	에코 서버 및 클라이언트
테스트에코-4C	TCP 풀 서버 TCP 풀 클라이언트	4C DLL 에코 서버 및 클라이언트	
TestEcho-Agent (요원-4C)	TCP 풀 에이전트	4C DLL 에코 클라이언트	
TestEcho-Agent (에이전트-PFM)	TCP 에이전트	SRC	에코 성능 테스트 클라이언트
TestEcho-Agent (에이전트 풀)	TCP 풀 에이전트	DLL 에코 클라이언트	
테스트Echo-PFM	TCP 서버 TCP 클라이언트	DLL Echo 성능 테스트 서버 및 클라이언트	
테스트에코 풀	TCP 풀 서버 TCP 풀 클라이언트	DLL Echo 서버 및 클라이언트	
테스트에코팩	TCP 팩 서버 TCP 팩 클라이언트	4C DLL DLL	에코 서버 및 클라이언트
TestEcho-UDP	UDP 서버 UDP 클라이언트	SRC	에코 서버 및 클라이언트
TestEcho-UDP-PFM	UDP 서버 UDP 클라이언트	DLL Echo 성능 테스트 서버 및 클라이언트	
테스트UDP캐스트	UDP 캐스트	SRC 멀티캐스트(브로드캐스트) 네트워크 멤버십	
테스트UDP노드	UDP 노드	SRC UDP 노드	
TestEcho-SSL	SSL 서버 SSL 클라이언트	SRC	에코 서버 및 클라이언트
TestEcho-SSL-4C	SSL 풀 서버 SSL 풀 클라이언트	4C LIB Echo 서버 및 클라이언트	
TestEcho-SSL-Pack	SSL 팩 서버 SSL 팩 클라이언트	4C DLL DLL	에코 서버 및 클라이언트
TestEcho-SSL-PFM SSL 서버		LIB	Echo 성능 테스트 서버 및 클라이언트

	SSL 에이전트		
TestEcho-Http	HTTP 서버	SRC	에코 서버 및 클라이언트
	HTTP 클라이언트		
	HTTP 동기화 클라이언트		
TestEcho-Http-4C	HTTP 서버	4C LIB	에코 서버 및 클라이언트
	HTTP 클라이언트	4C DLL	
	HTTP 동기화 클라이언트	4C DLL	
TestEcho-ARQ	트 UDP ARQ 서버 UDP	SRC	에코 서버 및 클라이언트
	ARQ 클라이언트 UDP		
TestEcho-ARQ-PFM	ARQ 서버 UDP ARQ 클라이언트表 10.1.1-1	DLL Echo 성능 테스트 서버 및 클라이언트	

Windows 예

10.1.2 리눅스 예제

프로젝트 이름	구성 요소 사용	인용 설명	
시험	TCP 서버	SRC	에코 서버 및 클라이언트
	TCP 에이전트		
	TCP 클라이언트		
testecho-pfm	TCP 서버	SRC	Echo 성능 테스트 서버 및 클라이언트
	TCP 에이전트		
	TCP 클라이언트		
testecho 풀	TCP 풀 서버	SRC	에코 서버 및 클라이언트
	TCP 풀 에이전트		
	TCP 풀 클라이언트		
testecho 팩	TCP 팩 서버	SRC	에코 서버 및 클라이언트
	TCP 팩 에이전트		
	TCP 팩 클라이언트		
testo-udp	UDP 서버	SRC	에코 서버
	UDP 클라이언트		Echo 클라이언트 멀티
	UDP 캐스트		캐스트(브로드캐스트) 네트워크 구성원
	UDP 노드		UDP 노드
testecho-udp-pfm	UDP 서버	SRC	Echo 성능 테스트 서버 및 클라이언트
	UDP 클라이언트		
testecho-lib	TCP 풀 서버	4C SO	에코 서버 및 클라이언트
	TCP 풀 에이전트		
	TCP 풀 클라이언트		
testecho-ssl	SSL 서버	SRC	에코 서버 및 클라이언트
	SSL 에이전트		
	SSL 클라이언트		
testecho-ssl-pfm	SSL 서버	그래서	Echo 성능 테스트 서버 및 클라이언트
	SSL 에이전트		
	SSL 클라이언트		

testecho-ssl-풀	SSL 풀 서버 SSL 풀 에이전트 SSL 풀 클라이언트	그래서	에코 서버 및 클라이언트
testecho-ssl-pack	SSL 팩 서버 SSL 팩 에이전트 SSL 팩 클라이언트	4C SO Echo	서버 및 클라이언트
Testecho-http	HTTP 서버 HTTP 에이전트 HTTP 클라이언트 HTTP 동기화 클라이언트	SRC	에코 서버 및 클라이언트
Testecho-http-4c	HTTP 서버 HTTP 에이전트 HTTP 클라이언트 HTTP 동기화 클라이언트	4C SO Echo	서버 및 클라이언트
testecho-arq	UDP ARQ 서버 UDP ARQ 클라이언트	SRC	에코 서버 및 클라이언트
testecho-arq-pfm	UDP ARQ 서버 UDP ARQ 클라이언트	+	Echo 성능 테스트 서버 및 클라이언트

표 10.1.2-1 리눅스 예시

10.2 도우미 기능

LPCTSTR HP_GetSocketErrorDesc() : HP Socket 오류 코드의 해당 설명을 가져옵니다.

DWORD SYS_GetLastError() : 캡슐화 API 함수 `GetLastError()` / `errno`

LPCSTR SYS_GetLastErrorStr() int [L] : API 함수 캡슐화 `strerror()`

SYS_WSAGetLastError() int [W] : API 함수 캡슐화 `WSAGetLastError()`

SYS_SetSocketOption() int : API 함수 `setsockopt()` 캡슐화

SYS_GetSocketOption() int : 캡슐화 API 함수 `getsockopt()`

SYS_ioctlSocket () int : 캡슐화 API 함수 `ioctlsocket()`

SYS_WSAIoctl () int SYS_ [W] : API 함수 `WSAIoctl()` 캡슐화

fcntl_SETFL() int SYS_SSO_ [L] : API 함수 `fcntl()`을 캡슐화하여 `F_SETFL` 설정

Block() :置 socket : [L] : FD 옵션 설정: `O_NONBLOCK`

int SYS_SSO_NoDelay() :TCP_NOSPLIT

int SYS_SSO_DontLinger() : 소켓 옵션 설정: SO_DONTLINGER

int SYS_SSO_Linger() int : 선 설정: SO_LINGER

SYS_SSO_RecvBuffSize() : 소켓 옵션 설정: `SO_RCVBUF`

int SYS_SSO_SendBuffSize() : 소켓 옵션 설정: SO_SNDBUF

int SYS_SSO_RecvTimeout() : 소켓 옵션 설정: SO_RCVTIMEO

int SYS_SSO_SendTimeout() : 소켓 주선 설정: SO SNDTIMEO

int SYS_SSO_ReuseAddress() : 소 재사용 전략 설정

BOOL SYS_GetSocketLocalAddress() : 소켓 로컬 주소 얻기

BOOL SYS_GetSocketRemoteAddress() : 소켓 원격 주소 얻기

ULONG SYS_EnumHostIPAddresses() : 호스트 IP 주소 열거

BOOL SYS_FreeHostIPAddresses() : 자유 호스트 IP 주소 구조

BOOL SYS_IsIPAddress() : 호스트 : 문자열이 IP 주소 형식과 일치하는지 확인

BOOL SYS_GetIPAddress() : 64비트 배열을 통해 IP 주소 획득

BOOL SYS_NToH64() : 64비트 호 트워크 바이트 순서를 호스트 바이트 순서로 변환

BOOL SYS_HToN64() : 짧은 정수 스트 바이트 순서를 네트워크 바이트 순서로 변환

BOOL SYS_SwapEndian16() : LowByte와 HighByte의 바이트와 하위 바이트 교환

BOOL SYS_SwapEndian32() : 바이트 의 상위 바이트와 하위 바이트 교환

BOOL SYS_IsLittleEndian() : BOOL 순서가 little endian인지 확인

SYS_CodePageToUnicode() [W] :CP_XXX -> 유니코드

BOOL SYS_CodePageToUnicodeEx() [W] :CP_XXX -> 유니코드

BOOL SYS_UnicodeToCodePage() [W] :UNICODE -> CP_XXX

BOOL SYS_UnicodeToCodePageEx() [W] :UNICODE -> CP_XXX

BOOL SYS_CharsetConvert() BOOL [L] :문자셋-A -> 문자셋-B

SYS_GbkToUnicode() BOOL :GBK -> 유니코드

SYS_GbkToUnicodeEx() BOOL :GBK -> 유니코드

SYS_UnicodeToGbk() BOOL :유니코드 -> GBK

SYS_UnicodeToGbkEx() BOOL :유니코드 -> GBK

SYS_Utf8ToUnicode() BOOL :UTF8 -> 유니코드

SYS_Utf8ToUnicodeEx () BOOL :UTF8 -> 유니코드

SYS_UnicodeToUtf8() :유니코드 -> UTF8

BOOL SYS_UncodeToUtf8Ex()	BOOL	:유니코드 -> UTF8	
SYS_GbkToUtf8 ()	BOOL	:GBK -> UTF8	
SYS_GbkToUtf8Ex()	BOOL	:GBK -> UTF8	
SYS_Utf8ToGbk()	BOOL	:UTF8 -> GBK	
SYS_Utf8ToGbkEx()	int	:UTF8 -> GBK	
SYS_GuessBase64EncodeBound() : 계산 Base64 인코딩 길이			
int SYS_GuessBase64DecodeBound() : Base64 디코딩 후 길이 계산			
int SYS_Base64Encode()			: Base64 인코딩
int SYS_Base64Decode()			: Base64 디코딩
int SYS_GuessUrlEncodeBound()			인코딩 길이 계산
int SYS_GuessUrlDecodeBound()			URL의 길이 계산
int SYS_UrlEncode()			: URL 인코딩
int SYS_UrlDecode()			: URL 디코딩
LPBYTE SYS_Malloc()			메모리 할당
LPBYTE SYS_Realloc()			메모리 재할당
VOID SYS_Free()			메모리 해제
LPVOID SYS_Calloc()			스택 할당
LPBYTE SYS_Alloca()			모리 할당

10.3 FAQ

Q-01 연 : 연결 ID 생성 규칙은 무엇입니까? 숫자 오버플로는 어떻습니까?

에이 : 결 ID는 32비트 프로그램에서 4바이트, 64비트 프로그램에서 8바이트입니다. 클라이언트 구성 요소

에이전트/서버 구성 요소에는 서로 다른 빌드 규칙이 있습니다.

1) 에이전트/서버 구성 요소: 연결 ID 값 범위: 1 - N*256, 여기서 N은 가장 큰 연결입니다.

번호를 가져 가라. HPSocket은 내부 알고리즘을 통해 안전한 임의의 연결 ID를 합리적으로 할당합니다.

2) 클라이언트 구성 요소: 연결 ID가 오버플로되면 다시 1부터 증가하기 시작합니다. 이론상 32세

비트 프로그램에서 연결 ID가 중복될 수 있지만 걱정하지 마십시오.

불가능한 시나리오에서: 클라이언트 프로세스가 동시에 여러 구성 요소를 시작하 고객 구성 요소 및

그리고 "생성 40 1억 연결 후 첫 시작 고객 고 연결이 끊기지 않았습니다." .

Q-02 에는 : 이벤트 핸들러에서 Start() / Stop()을 호출할 수 있습니까?

A, : 사용할 수 없습니다. 리스너 이벤트(OnReceive /OnClose 등)는 일반적으로 통신 스레드에서 트리거되기 때문에

Stop() 메서드는 통신 스레드가 끝날 때까지 기다려야 합니다. 그러면 끝을 기다리는 무한 루프가 발생합니다.

따라서 리스너 이벤트 처리 코드에서는 Start() / Stop() 컨트롤 메서드를 호출할 수 없습니다.

Q-03 에 대 : Server 나 Agent를 종료할 때 Stop() 메서드에서 멈추는 이유는 무엇입니까?

A 1) : 한 몇 가지 가능성:

이벤트 처리 함수에서 Stop() 메서드를 호출합니다 (참조: Q-02).

2) 하나 이상의 통신 스레드가 교착 상태에 있어 Stop() 메서드가 모든 통신을 기다립니다.

스레드가 끝납니다. 모든 통신 스레드가 교착 상태에 빠지면 또 다른 현상이 발생합니다.

모든 통신 요청을 수신하고 처리합니다.

3) 사용자 지정 Windows DLL에서 HP-Socket 서버 구성 요소를 사용하지만 DLL 언로드

서버 구성 요소는 이전에 수동으로 닫히지 않았으므로 서버 구성 요소가 자동 제거 중에 Stop() 메서드에서 중단되었습니다. 해결 방법: DLL을 언로드하기 전에 명시적으로 HP-Socket 서버 구성 요소를 종료합니다.

4) 일부 시스템은 Stop() 메서드를 원활하게 실행할 수 있고 일부 시스템은 그렇지 않은 경우 Winsock 프로토콜이 잘못된 것일 수 있습니다.

프로토콜 스택이 파괴되었습니다. 해결 방법: 명령줄 도구에서 관리자로 실행: "netsh

winsock 재설정", 컴퓨터를 다시 시작하십시오.

Q-04 에는 : 이벤트 핸들러에서 사용자 인터페이스를 업데이트할 수 있습니까?

A 인터페 : 사용할 수 없습니다. 사용자가 이벤트 핸들러에서 업데이트되면 이벤트 핸들러는 소켓 IO 스레드에 의해 트리거됩니다.

이스는 응용 프로그램의 성능을 크게 저하시키고 쉽게 교착 상태를 유발할 수 있으므로 다른 방법을 사용하여 비동기적으로 사용자를 업데이트해야 합니다.

사용자 인터페이스.

Q-05 소위 : 매우 긴 연결을 끊는 방법은 무엇입니까?

: 초장거리 연결이란 연결 시간이 정상 시간을 초과하는 연결을 말합니다. 서버 및 에이전트 구성 요소는 다음을 제공합니다.

DisconnectLongConnections () 메서드는 긴 연결을 모두 끊고 **GetConnectPeriod()** 도 제공합니다.

이 메서드는 연결 기간을 가져오는 데 사용됩니다.

Q-06 소위 : 자동 연결을 끊는 방법은 무엇입니까?

: 무음 연결은 오랫동안 데이터 상호 작용이 없는 연결을 의미합니다. 서버 및 에이전트 구성 요소는 다음을 제공합니다.

DisconnectSilenceConnections () 메서드는 모든 자동 연결을 끊고 **GetSilencePeriod()** 도 제공합니다.

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcdsaa/HP-Socket>

메소드는 연결의 무음 시간을 얻는 데 사용됩니다. 참고: 구성 요소가 자동 플래그를 켰을 때 위의 두 가지 방법 유효한 경우에만 "SetMarkSilence(TRUE)" 메서드를 호출하여 구성 요소가 시작되기 전에 무음 표시를 키 수 있습니다. 자동 플래그는 HP 소켓 v3.5.x 이상 버전에서 기본적으로 활성화됩니다.

Q-07 : 연결 해제 후 다시 연결하는 방법은 무엇인가?

A **자** Agent 컴포넌트는 연결 해제 알림 이벤트(OnClose) 수신 시 즉시 Connect() 호출을 시작할 수 있습니다.

접속 클라이언트 컴포넌트는 연결 끊김 알림 이벤트(OnClose)를 수신하지 못한 경우 즉시 Start() 메소드 호출 다시 연결하는 방법. 따라서 클라이언트 구성 요소는 다음 방법을 선택하여 다시 연결할 수 있습니다.

1) 모니터링 스레드 또는 타이머를 시작하고 주기적 으로 구성 요소 개체의 GetState() 메서드를 호출하여 그룹을 확인합니다.

구성 요소 개체의 상태가 SS_STOPED 이면 재연결을 수행합니다.

2) 모니터링 스레드를 시작하고 컴포넌트의 OnClose 이벤트에서 모니터링 스레드로 연결 해제 재접속 전송

(이벤트)는 모니터링 스레드를 활성화하고 모니터링 스레드는 루프에서 구성 요소 개체의 GetState() 메서드를 호출합니다.

상태가 SS_STOPED가 될 때까지 구성 요소 개체의 상태를 확인한 다음 다시 연결하십시오.

3) ::PostMessage() / ::PostThreadMessage() API 함수 와 결합된 창 메시지 메커니즘 사용

2)의 모니터링 스레드와 알림(이벤트)을 교체합니다.

4) HP-Socket v5.7.x부터 Wait() 메서드를 사용하여 재연결을 실현할 수 있습니다. component Start()

성공 후 재접속 확인 쓰레드를 시작하고, 재접속 확인 쓰레드에서 Wait()를 호출하여 기다린다.

Wait()이 TRUE를 반환 하면 재연결을 수행합니다 .

Q-08 의 경 : HP-Socket에 하트비트 감지 메커니즘이 있습니까?

아이 : 우 예 (UdpCast 구성 요소 제외). TCP 구성 요소는 TCP 프로토콜의 기본 제공 하트비트 감지 메커니즘과 UDP를 사용합니다.

구성 요소는 0바이트 패킷을 서로 전송하여 하트비트 감지를 구현합니다.

1) TCP 하트비트 감지: SetKeepAliveTime() / SetKeepAliveInterval(), 단위 - 밀리초

시간 초과 계산 공식: KeepAliveTime + (KeepAliveInterval * N)

그 중에서 N은 고정 값입니다: WinXP 이하 시스템의 경우 N=5, Win7 이상의 시스템의 경우 N=10

2) UDP 하트비트 감지: SetDetectInterval() / SetDetectAttempts(), 단위 - 밀리초

시간 초과 계산 공식: DetectInterval * (DetectAttempts + X)

여기서 X는 0과 1 사이의 값일 수 있습니다.

3) 서버 또는 에이전트 구성 요소의 경우 "자동 연결 끊기"를 통해 간접적으로 하트비트를 실현할 수 있습니다.

발각. 예: 타이머 또는 독립 스레드를 사용하여 DisconnectSilenceConnections()를 정기적으로 호출합니다.

자동으로 연결을 끊는 방법.

Q-09 **HP-** : HP-Socket 의 UDP 구성 요소가 내 UDP 프로그램과 통신하고 종종 연결이 끊어지는 이유는 무엇입니까 ?

UDP : Socket의 UDP 서버 및 클라이언트 구성 요소는 기본적으로 하트비트 감지 메커니즘을 활성화했습니다.

프로그램 통신에는 두 가지 옵션이 있습니다.

1) SetDetectInterval(0) / SetDetectAttempts(0) 를 호출하여 UDP 구성 요소의 하트비트 감지를 끕니다.

기구.

2) 자신의 프로그램이 HP-Socket을 사용하여 UDP 하트비트 감지 핸드셰이크를 구현합니다. 구체적인 방법은 다음과 같습니다.

IUdpClient를 클라이언트로 사용: 서버가 0바이트의 UDP 하트비트를 수신한 경우

패킷을 수신하면 즉시 0바이트 핸드셰이크 패킷에 응답합니다.

IUdpServer를 서버로 사용: 클라이언트는 주기적으로 서버에 0바이트를 보내야 합니다.

UDP 하트비트 패킷.

Q-10 : HP-Socket 의 UDP 연결이 닫히면 피어 끝이 OnClose 이벤트를 트리거하지 않습니까?

JessMA프로젝트 홈페이지 <http://www.oschina.net/p/hp-socket> 다운로드 주소 <https://github.com/lcdsaa/HP-Socket>

에이 : UDP 자체는 연결이 없는 통신 프로토콜이며 한쪽 끝이 닫히면 다른 쪽 끝이 인식할 수 없습니다. HP 소켓 v5.5.x

그리고 이전 버전은 하트비트 메커니즘(Q-07) 또는 애플리케이션 계층을 통해서만 인식을 실현할 수 있습니다. HP-Socket v5.6.x 버전 처음에 한쪽 끝이 닫히면 특수 "종료 알림" 데이터그램을 다른 쪽 끝으로 적극적으로 보내고 다른 쪽 끝은 수신합니다.

이 데이터그램을 수신하는 즉시 자체를 닫습니다. 그러나 참고: UDP의 신뢰할 수 없는 특성으로 인해 피어 앤드는 "종료 알림" 데이터그램을 수신해야 합니다. 이 경우 여전히 하트비트 메커니즘 등을 통해 수행해야 합니다.

보상하다. "닫기 알림" 데이터그램의 길이는 16바이트이며 내용은 다음과 같습니다.

```
{0xBE, 0xB6, 0x1F, 0xEB, 0xDA, 0x52, 0x46, 0xBA, 0x92, 0x33, 0x59, 0xDB, 0xBF, 0xE6, 0xC8,
0xE4}
```

Q-11에 대 : HP-Socket 의 TCP 구성 요소가 고정 패킷을 처리합니까?

A 1) : 한 세 가지 옵션 :

1) PUSH 모델: 애플리케이션이 스티키 패킷을 수동으로 처리합니다.

2) PULL 모델: 응용 계층 프로토콜과 협력하여 고정 패킷을 반자동으로 처리합니다.

3) PACK 모델: 통신 구성 요소는 고정 패킷을 자동으로 처리합니다.

Q-12 는 : HP-Socket은 타사 소켓 응용 프로그램과 어떻게 통신합니까?

에이 : HP-Socket 구성 요소 수신 모델에 따라 별도로 처리됩니다.

1) PUSH 모델: 응용 계층 프로토콜과 관련이 없으며 직접 통신할 수 있습니다.

2) PULL 모델: 피어와 애플리케이션 계층 프로토콜을 협상합니다.

3) PACK 모델: 피어 앤드는 HP-Socket PACK 의 패킷 형식을 따라야 합니다.

Q-13은 그 : 여러 스레드가 동시에 데이터를 보낼 때 송신자 또는 수신자가 데이터 패킷을 잘못된 순서로 보내게 됩니까?

A 는 : 렇지 않습니다. 발신자를 위해 HP-Socket은 각 Send() 메서드 호출 에서 보낸 데이터가 올바른지 확인합니다.

완전하고 질서정연하며 다른 Send() 메서드에 의해 방해받지 않습니다.

여러 OnReceive 이벤트가 동시에 트리거되므로 수신자의 데이터 패킷이 순서를 벗어나지 않습니다.

Q-14 는 괜 : 여러 통신 구성 요소가 동일한 수신기 객체를 공유할 수 있습니까?

에이 : 찮습니다. 리스너 콜백 이벤트의 pSend 매개변수는 현재 통신 구성 요소를 식별합니다.

Q-15 HP- : HP-Socket이 10,000개의 연결 만 설정할 수 있는 이유는 무엇입니까?

최대 : Socket의 Server 및 Agent 구성 요소에 대한 기본 최대 연결 수는 10,000이라고 할 수 있습니다.

연결 수를 수정하는 SetMaxConnectionCount() 메서드 .

Q-16은 데 : HP-Socket은 흐름 제어를 어떻게 구현합니까?

에이 : 이터 수신 및 데이터 전송의 두 가지 측면을 통해 흐름 제어를 실현할 수 있습니다.

1) 데이터 수신: PauseReceive() 메서드를 호출하여 데이터 수신을 일시 중지하거나 재개합니다.

2) 데이터 수신: GetPendingDataLength() 메서드를 호출하여 누적된 미전송 데이터 양을 구하고, 데이터 전송 속도를 제어합니다.

Q-17 : 데이터 패킷이 작고 실시간 요구 사항이 높은 TCP 통신 시나리오에 대한 통신 속성을 설정하는 방법은 무엇입니까?

A 1) SP_DIRECT 전송 전략을 통해 TCP_NODELAY 소켓 옵션을 설정합니다 .

SetSendPolicy(SP_DIRECT) 메소드를 호출하여 직접발송 정책을 설정한다.

2) SetNoDelay(TRUE) 메서드를 호출하여 TCP_NODELAY 소켓 옵션을 설정합니다 .

Q-18은 **다** : HP-Socket 용 프록시 서버를 설정하는 방법은 무엇입니까?

에이 : 른 구성 요소 유형 및 프록시 서버 유형에 따라 다른 설정 방법을 가지고 있습니다.

1) TCP 구성 요소는 SOCKS 프록시를 설정합니다. SOCKS 프로토콜을 통해 프록시 서버에 연결한 후

정상적인 데이터 통신을 시작하십시오.

2) SSL 구성 요소는 SOCKS 프록시를 설정합니다. SetSSLAutoHandShake(FALSE) 메서드를 호출하여 설정합니다.

구성 요소는 "수동으로 SSL 핸드셰이크 시작" 모드로 설정되고 SOCKS 프로토콜을 통해 프록시 서버에 연결됩니다.

서버가 SSL 핸드셰이크를 시작하고 일반 데이터 통신을 시작하기 위해 StartSSLHandShake() 메서드를 호출한 후.

3) Http/Https 구성 요소는 SOCKS 프록시를 설정합니다. SetHttpAutoStart(FALSE) 메서드를 호출하여 설정합니다.

구성 요소는 SOCKS 프로토콜을 통해 프록시에 연결하는 "수동으로 HTTP 통신 시작" 모드로 설정됩니다.

그런 다음 서버는 StartHttp() 메서드를 호출하여 HTTP 통신을 시작합니다.

4) Http 구성 요소는 HTTP 프록시를 설정합니다. 추가 작업을 수행할 필요 없이 연결 주소를 다음으로 설정하기만 하면 됩니다.

프록시 서버 주소, 요청을 보낼 때 "호스트" 요청 헤더를 대상 서버 주소로 지정하십시오.

5) Https 구성 요소는 HTTP 터널 프록시를 설정합니다. SetSSLAutoHandShake(FALSE) 메서드를 호출합니다.

구성 요소를 "수동으로 SSL 핸드셰이크 시작" 모드로 설정하고 시작할 방법이 없습니다.

"CONNECT" 요청, 요청이 성공한 후 StartSSLHandShake() 메서드를 호출하여 SSL을 시작합니다.

핸드셰이크, 정상적인 데이터 통신을 시작합니다.

Q-19 **기본** : SSL/HTTPS 클라이언트는 서버의 SNI(서버 이름 표시)를 어떻게 지정합니까?

A 의 : 적으로 HP-Socket의 SSL/HTTPS 클라이언트 구성 요소(클라이언트 또는 에이전트)는

도메인 이름은 SNI로 서버에 전송되며 연결이 IP 주소인 경우 SNI는 전송되지 않습니다. HP 소켓

버전 v5.8.8부터 클라이언트 구성 요소는 특수 연결 주소 형식을 통해 SNI를 수동으로 지정할 수도 있습니다.

<hostaddress>^<hostname>, 여기서 <hostaddress>는 실제 연결 주소이고 <hostname>은

서버로 전송된 SNI입니다. 이 형식의 한 가지 사용 사례는 프록시 서버를 통해 대상에 연결할 때입니다.

SSL/HTTPS 서버.

Q-20 **HP-** : 라이브러리 파일의 크기를 줄이는 방법은 무엇입니까?

에이 : Socket 릴리스 패키지에서 제공하는 라이브러리 파일에는 모든 구성 요소와 기능이 포함되어 있습니다.

구성 요소 또는 기능의 경우 include/hpsocket/HPTypDef.h에서 해당 pragma 매크로를 정의 할 수 있습니다.

라이브러리 파일을 컴파일합니다(참고: 수정된 include /hpsocket/HPTypDef.h를

include/hpsocket/hpsocket/ HPTypDef.h):

1) _UDP_DISABLED : UDP 구성 요소를 제외합니다.

2) _SSL_DISABLED : SSL 구성 요소(Htts 구성 요소 포함)를 제외합니다.

3) _HTTP_DISABLED : HTTP 구성 요소(Htts 구성 요소 포함)를 제외합니다.

4) _ZLIB_DISABLED : zlib 관련 함수를 제외합니다.

5) _BROTLI_DISABLED : brotli 관련 기능을 제외한다.

6) _ICONV_DISABLED : iconv 관련 기능을 제외합니다(Linux에서만 유효).