# **Python Socket 包和类用法详解**

Socket 是 Python 中用于网络编程的核心模块，它提供了访问 BSD 套接字的接口，允许你创建客户端和服务器应用程序。

## **1. 基本概念**

Socket 是网络通信的基本构建块，它允许不同主机或同一主机上的不同进程之间进行通信。Python 的 **socket** 模块提供了对底层网络通信的接口。

## **2. 主要 Socket 类型**

python

import socket

*# 创建 TCP socket (SOCK\_STREAM)*

tcp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

*# 创建 UDP socket (SOCK\_DGRAM)*

udp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

**AF\_INET**: IPv4 地址族

**AF\_INET6**: IPv6 地址族

**SOCK\_STREAM**: TCP 协议 (面向连接)

**SOCK\_DGRAM**: UDP 协议 (无连接)

## **3. TCP Socket 编程**

### **3.1 TCP 服务器**

python

import socket

*# 创建 TCP socket*

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

*# 绑定 IP 和端口*

server\_socket.bind(('0.0.0.0', 12345)) *# 0.0.0.0 表示监听所有可用接口*

*# 开始监听，设置最大连接数*

server\_socket.listen(5)print("服务器启动，等待连接...")

while True:

*# 接受客户端连接*

client\_socket, client\_address = server\_socket.accept()

print(f"接收到来自 {client\_address} 的连接")

try:

*# 接收数据*

data = client\_socket.recv(1024) *# 最多接收1024字节*

print(f"接收到数据: {data.decode('utf-8')}")

*# 发送响应*

response = "你好，客户端！".encode('utf-8')

client\_socket.send(response)

except Exception as e:

print(f"发生错误: {e}")

finally:

*# 关闭客户端连接*

client\_socket.close()

### **3.2 TCP 客户端**

python

import socket

*# 创建 TCP socket*

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

try:

*# 连接服务器*

client\_socket.connect(('127.0.0.1', 12345))

*# 发送数据*

message = "你好，服务器！".encode('utf-8')

client\_socket.send(message)

*# 接收响应*

response = client\_socket.recv(1024)

print(f"服务器响应: {response.decode('utf-8')}")

except Exception as e:

print(f"发生错误: {e}")finally:

*# 关闭连接*

client\_socket.close()

## **4. UDP Socket 编程**

### **4.1 UDP 服务器**

python

import socket

*# 创建 UDP socket*

udp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

*# 绑定 IP 和端口*

udp\_socket.bind(('0.0.0.0', 12345))print("UDP 服务器启动，等待数据...")

while True:

*# 接收数据*

data, client\_address = udp\_socket.recvfrom(1024)

print(f"接收到来自 {client\_address} 的数据: {data.decode('utf-8')}")

*# 发送响应*

response = "UDP 响应".encode('utf-8')

udp\_socket.sendto(response, client\_address)

### **4.2 UDP 客户端**

python

import socket

*# 创建 UDP socket*

udp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

try:

*# 发送数据 (UDP 不需要先连接)*

message = "UDP 测试".encode('utf-8')

udp\_socket.sendto(message, ('127.0.0.1', 12345))

*# 接收响应*

response, server\_address = udp\_socket.recvfrom(1024)

print(f"接收到来自 {server\_address} 的响应: {response.decode('utf-8')}")

except Exception as e:

print(f"发生错误: {e}")finally:

udp\_socket.close()

## **5. Socket 常用方法**

| **方法** | **描述** |
| --- | --- |
| **socket()** | 创建新的 socket |
| **bind((host, port))** | 绑定到特定地址和端口 |
| **listen(backlog)** | 开始监听传入连接 (TCP) |
| **accept()** | 接受连接并返回 (conn, address) (TCP) |
| **connect((host, port))** | 连接到远程 socket (TCP) |
| **send(data)** | 发送数据 (TCP) |
| **recv(bufsize)** | 接收数据 (TCP) |
| **sendto(data, address)** | 发送数据 (UDP) |
| **recvfrom(bufsize)** | 接收数据 (UDP) |
| **close()** | 关闭 socket |
| **settimeout(value)** | 设置超时时间 |
| **setsockopt(level, optname, value)** | 设置 socket 选项 |

## **6. 高级用法**

### **6.1 非阻塞 Socket**

import socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.setblocking(False) *# 设置为非阻塞模式*

### **6.2 Socket 选项**

python

*# 设置地址重用 (方便调试时快速重启服务器)*

server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

*# 设置接收缓冲区大小*

server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_RCVBUF, 8192)

### **6.3 处理多个连接 (select)**

python

import select

read\_list = [server\_socket]while True:

readable, \_, \_ = select.select(read\_list, [], [])

for s in readable:

if s is server\_socket:

*# 新连接*

client\_socket, addr = server\_socket.accept()

read\_list.append(client\_socket)

else:

*# 已有连接的数据*

data = s.recv(1024)

if data:

*# 处理数据*

pass

else:

*# 连接关闭*

s.close()

read\_list.remove(s)

## **7. 实际应用建议**

**异常处理**: 网络操作容易出错，务必添加适当的异常处理

**编码/解码**: 网络传输的是字节，记得编码(encode)和解码(decode)

**资源管理**: 使用 **with** 语句或 **try/finally** 确保 socket 被正确关闭

**性能考虑**: 对于高性能应用，考虑使用 **select**, **poll** 或异步框架如 **asyncio**

Python 的 socket 模块提供了强大的网络编程能力，理解其基本原理和使用方法对于开发网络应用至关重要。