

计算机网络

第三章 传输层

谢瑞桃

xie@szu.edu.cn

[rtxie.github.io](https://github.com/rtxie)

计算机与软件学院
深圳大学



第三章讲解内容

1. 传输层概述与UDP

- 需求/服务/协议、多路复用/分解、UDP协议

2. 可靠传输

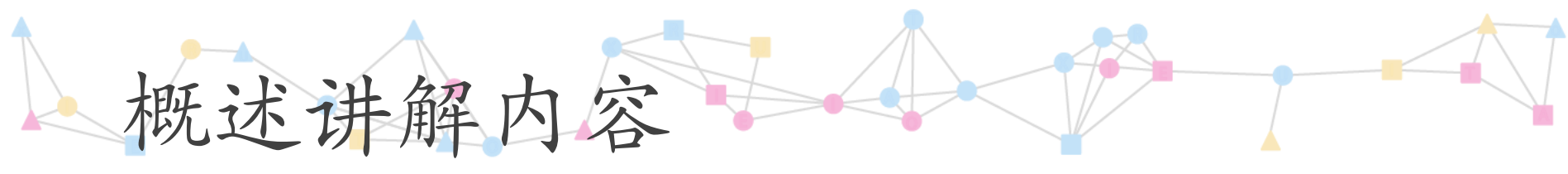
- 可靠传输基础知识、TCP可靠传输

3. TCP

- 报文段结构、超时间隔、流量控制、连接管理

4. TCP拥塞控制

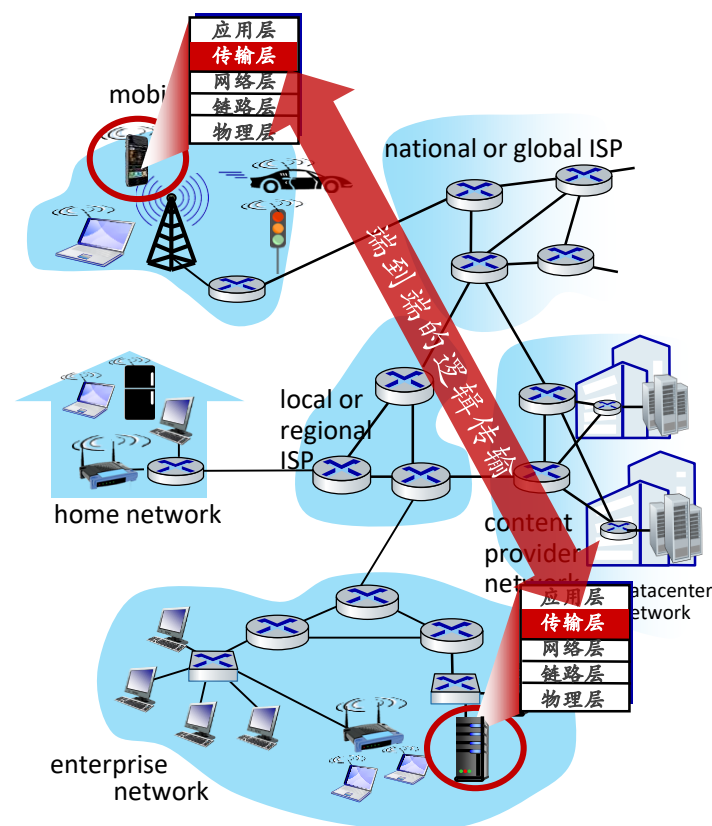
- 网络拥塞、TCP拥塞控制、吞吐量分析



- 传输层需求、服务、协议
- 多路复用与多路分解
- UDP协议

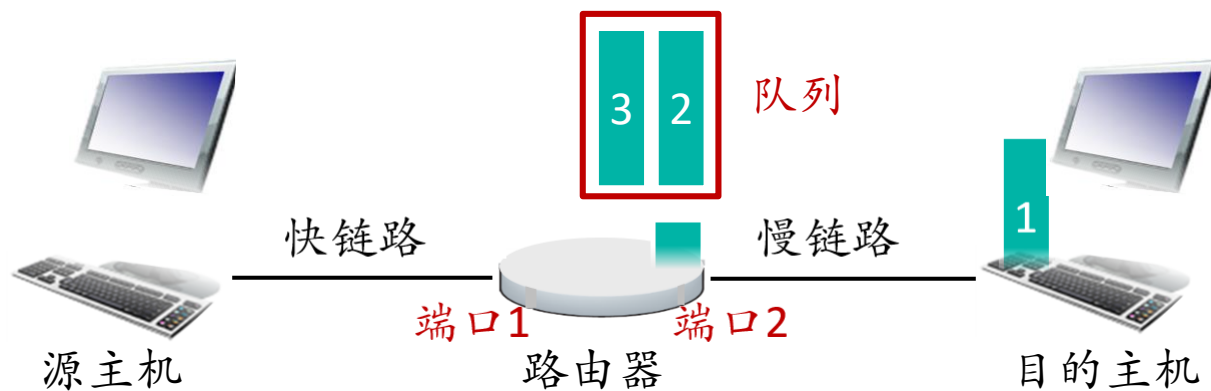
传输层服务

- 为运行在不同主机上的
进程之间提供逻辑通信
- 传输层协议运行在端系统里
 - 发送方：将应用层报文转换成报文段，交给网络层
 - 接收方：将报文段组装成报文，交给应用层
- 依赖网络层的服务



网络层服务

- 为**主机**提供逻辑通信
- 采用分组交换的网络层
 - 会发生**丢包**，因为链路速率不匹配，以及网络流量的突发性质
 - 会发生**乱序**，因为每个分组独立路由
 - 会发生**差错**，因为链路传输和路由器存储都可能引入比特差错。





需求	服务	协议	
为运行在不同主机上的进程之间提供逻辑通信	进程间交付		



需求	服务	协议	
为运行在不同主机上的进程之间提供逻辑通信	进程间交付		
检测报文段是否出错	差错检测		



传输层需求、服务和协议

需求	服务	协议	
为运行在不同主机上的进程之间提供逻辑通信	进程间交付		
检测报文段是否出错	差错检测		
解决丢包、差错问题	可靠传输		
解决乱序问题	按序交付		



传输层需求、服务和协议

需求	服务	UDP	TCP
为运行在不同主机上的进程之间提供逻辑通信	进程间交付	✓	✓
检测报文段是否出错	差错检测	✓	✓
解决丢包、差错问题	可靠传输	✗	✓
解决乱序问题	按序交付	✗	✓



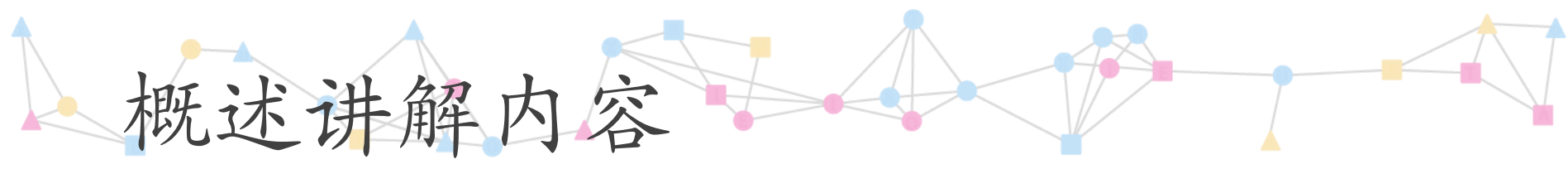
传输层需求、服务和协议

需求	服务	UDP	TCP
为运行在不同主机上的进程之间提供逻辑通信	进程间交付	✓	✓
检测报文段是否出错	差错检测	✓	✓
解决丢包、差错问题	可靠传输	✗	✓
解决乱序问题	按序交付	✗	✓
解决接收缓存溢出问题	流量控制	✗	✓
应对网络拥塞	拥塞控制	✗	✓



第三章知识点汇总

- 理解传输层的需求、服务与协议三者的关系



- 传输层需求、服务、协议
- 多路复用与多路分解
- UDP协议

多路复用/多路分解

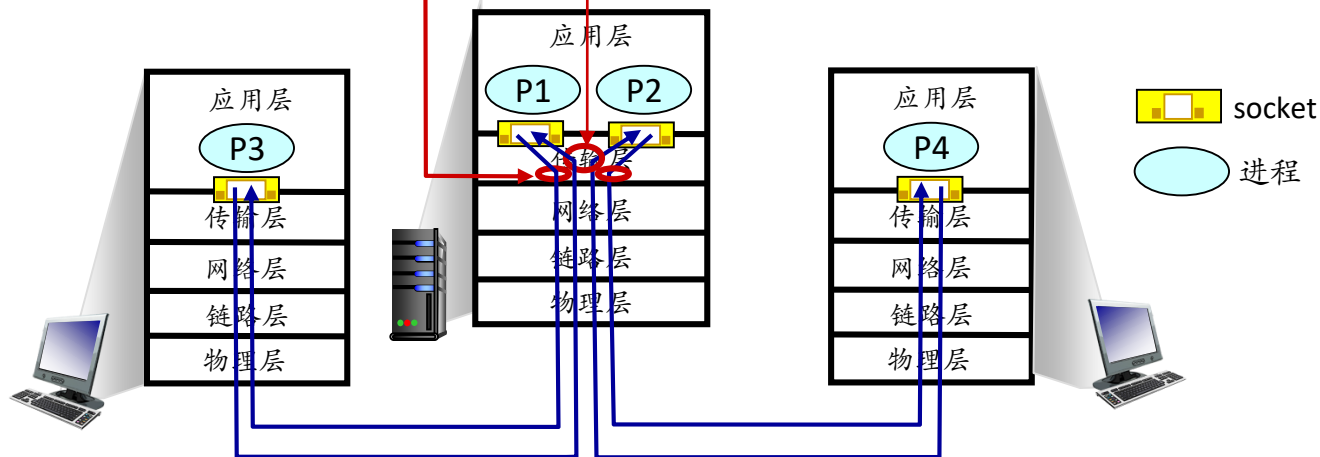
■ 解决进程间交付的技术

发送方多路复用

处理来自多个套接字的数据，添加传输层首部

接收方多路分解

利用报文段的首部，将数据交给正确的套接字



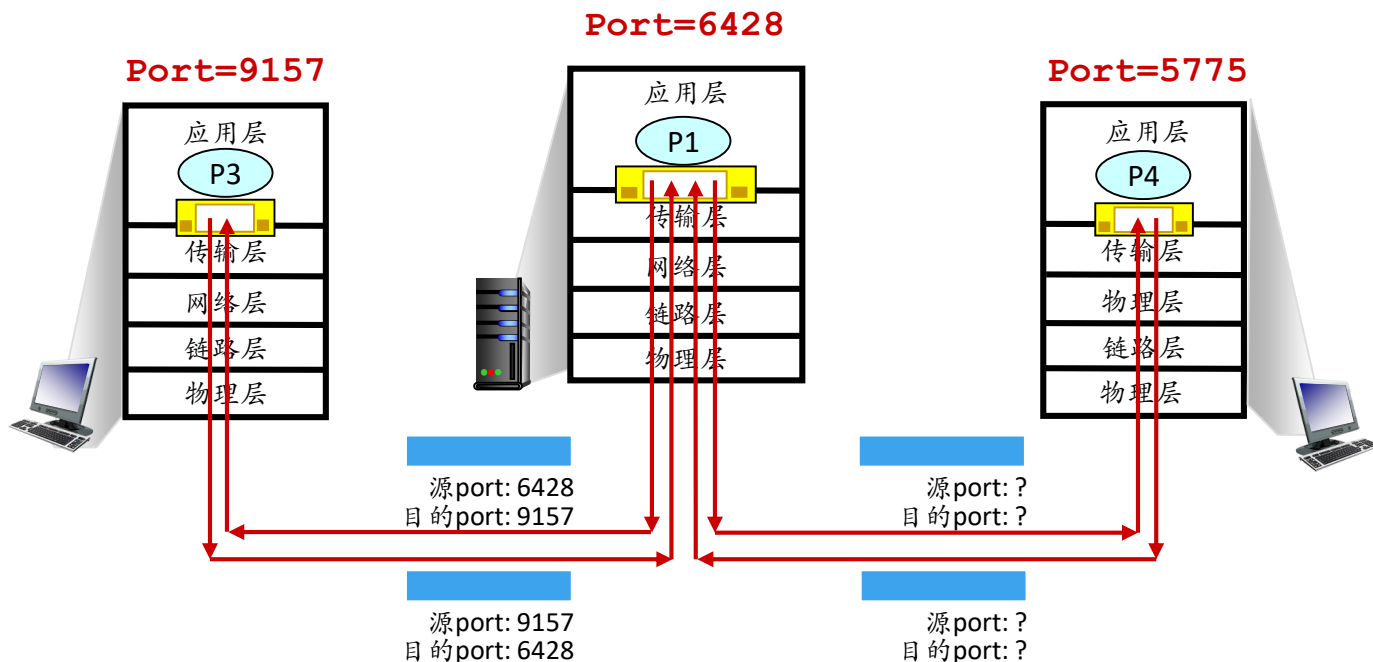


- IP地址标识主机
- 端口号标识进程
 - 16比特，0—65535之间
 - 其中0—1023是周知端口号（HTTP80，FTP21）

无连接 (UDP)

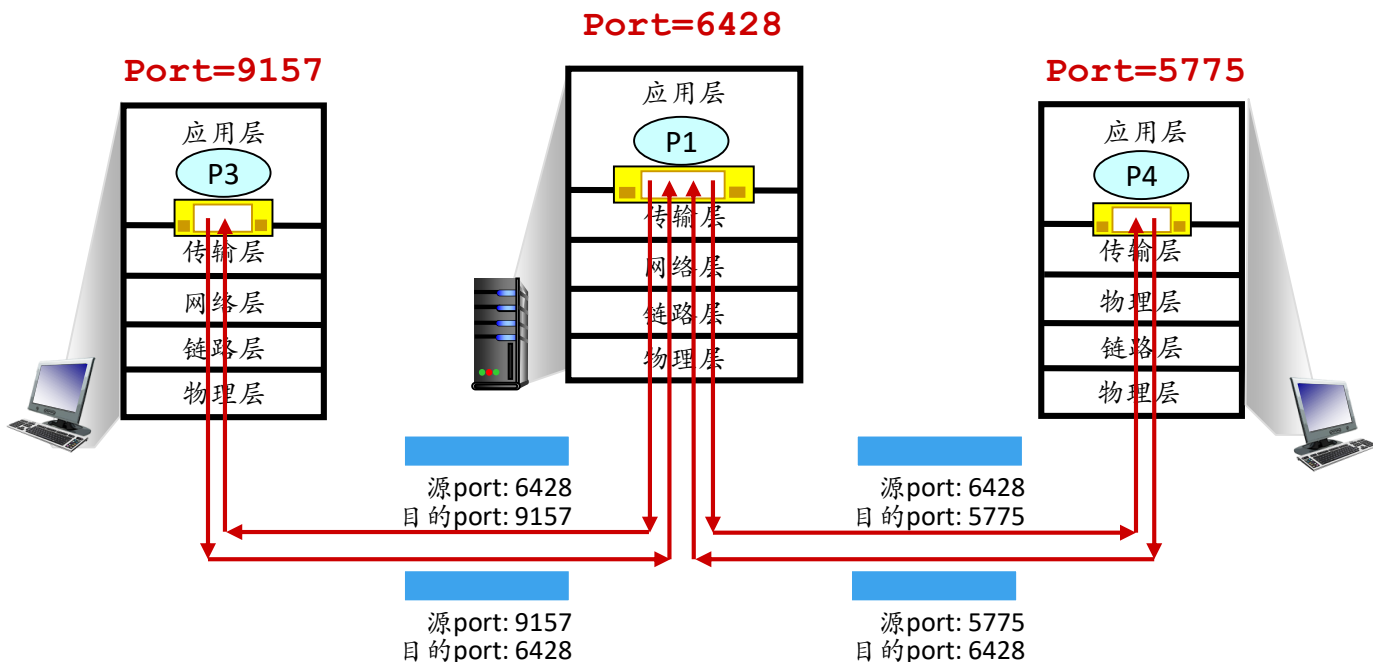
■ 创建套接字

- `clientSocket = socket (AF_INET, SOCK_DGRAM)`
- 函数运行时，传输层自动地为该套接字分配端口号



无连接 (UDP)

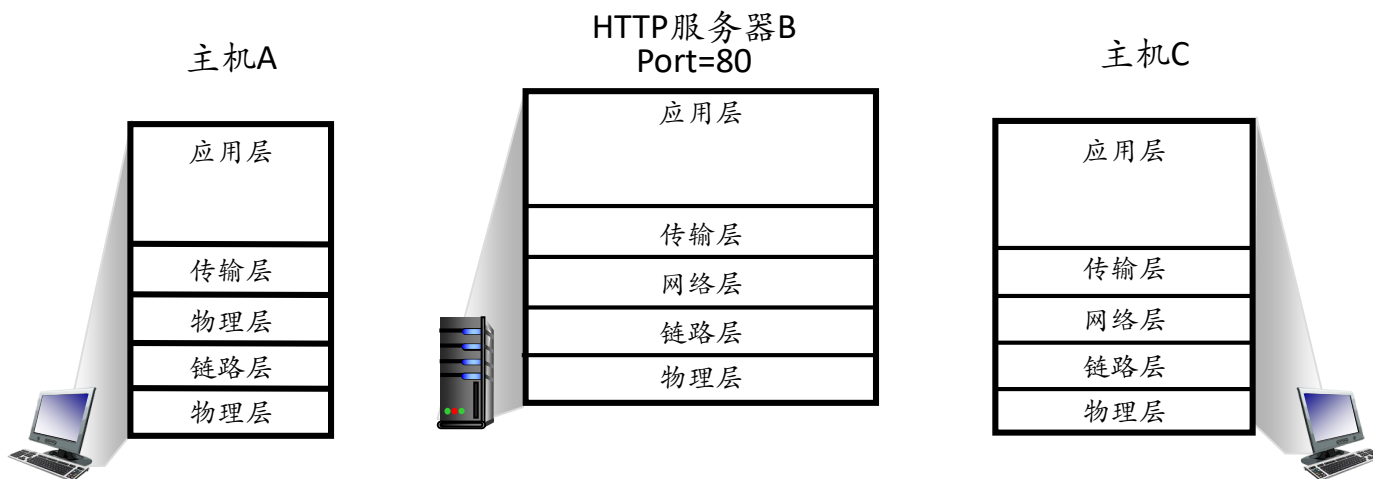
- **注意**：一个套接字可以和多个套接字通信
- 套接字由二元组 **<IP地址, 端口号>** 识别
- 所以，通过套接字发送数据的时候需要明示对方的IP地址和端口号



面向连接 (TCP)

■ 服务器创建监听套接字

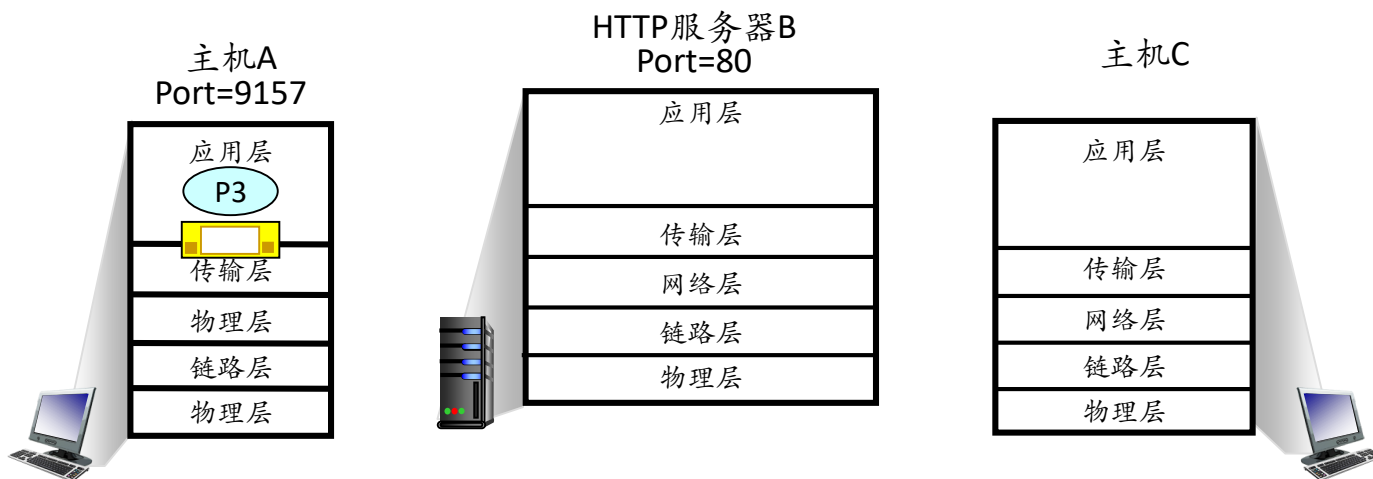
- `serverSocket = socket(AF_INET, SOCK_STREAM)`
- `serverSocket.bind('', serverPort)` #为套接字关联端口号
- `serverSocket.listen(1)`



面向连接 (TCP)

■ 客户端创建套接字

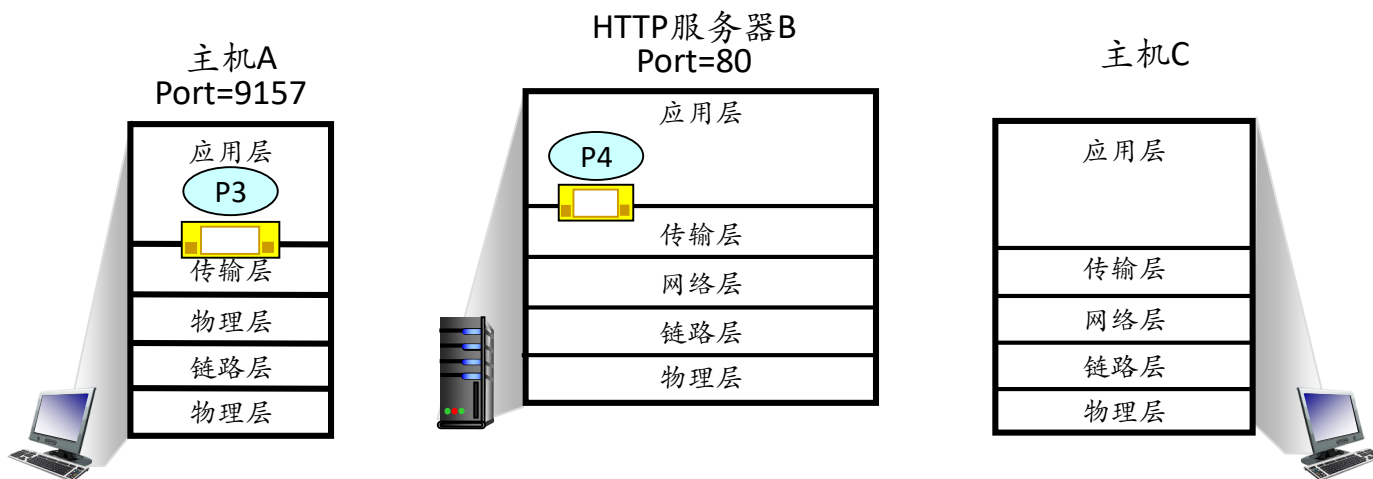
- `clientSocket = socket(AF_INET, SOCK_STREAM)`
- `clientSocket.connect((serverName, serverPort))` #发起连接



面向连接 (TCP)

■ 服务器创建通信套接字 (连接)

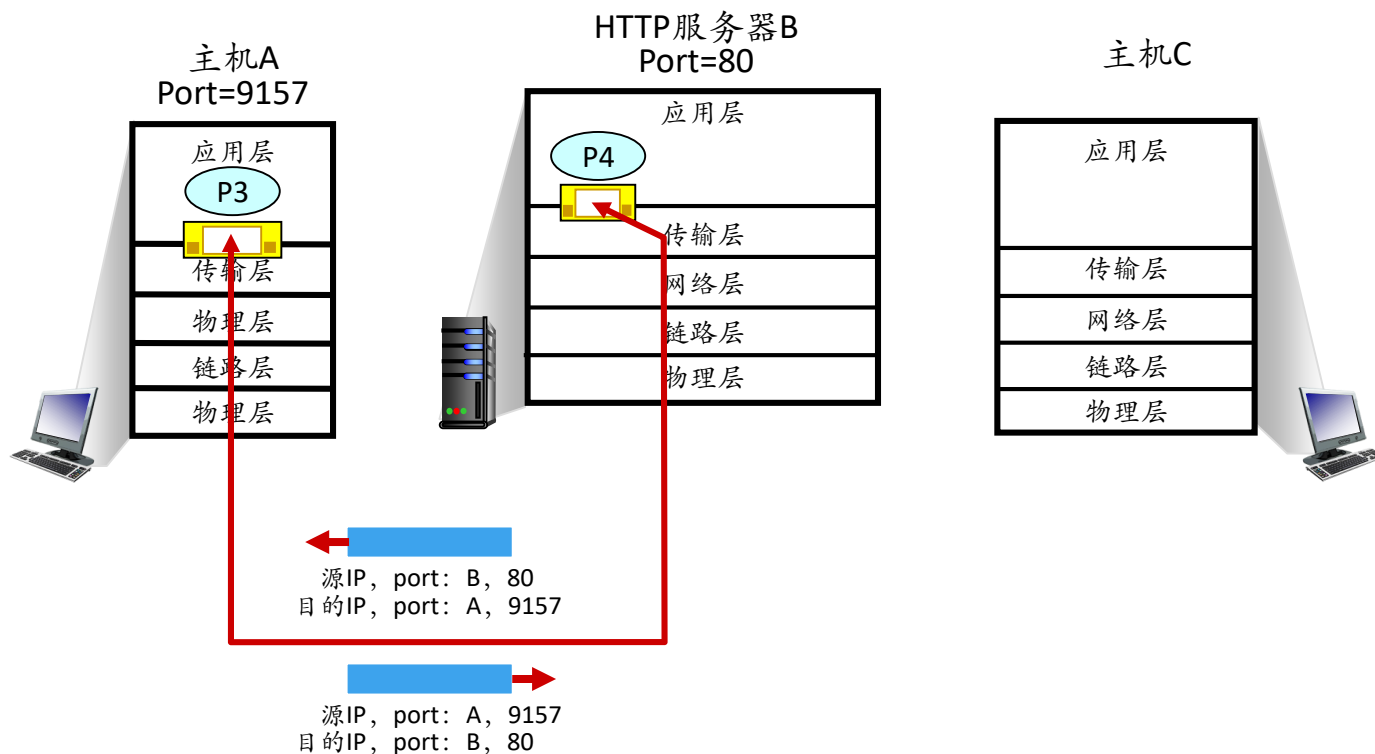
- `connectionSocket, addr = serverSocket.accept()`
- `connectionSocket`的端口号与`serverSocket`相同
- `addr`是客户端的地址



面向连接 (TCP)

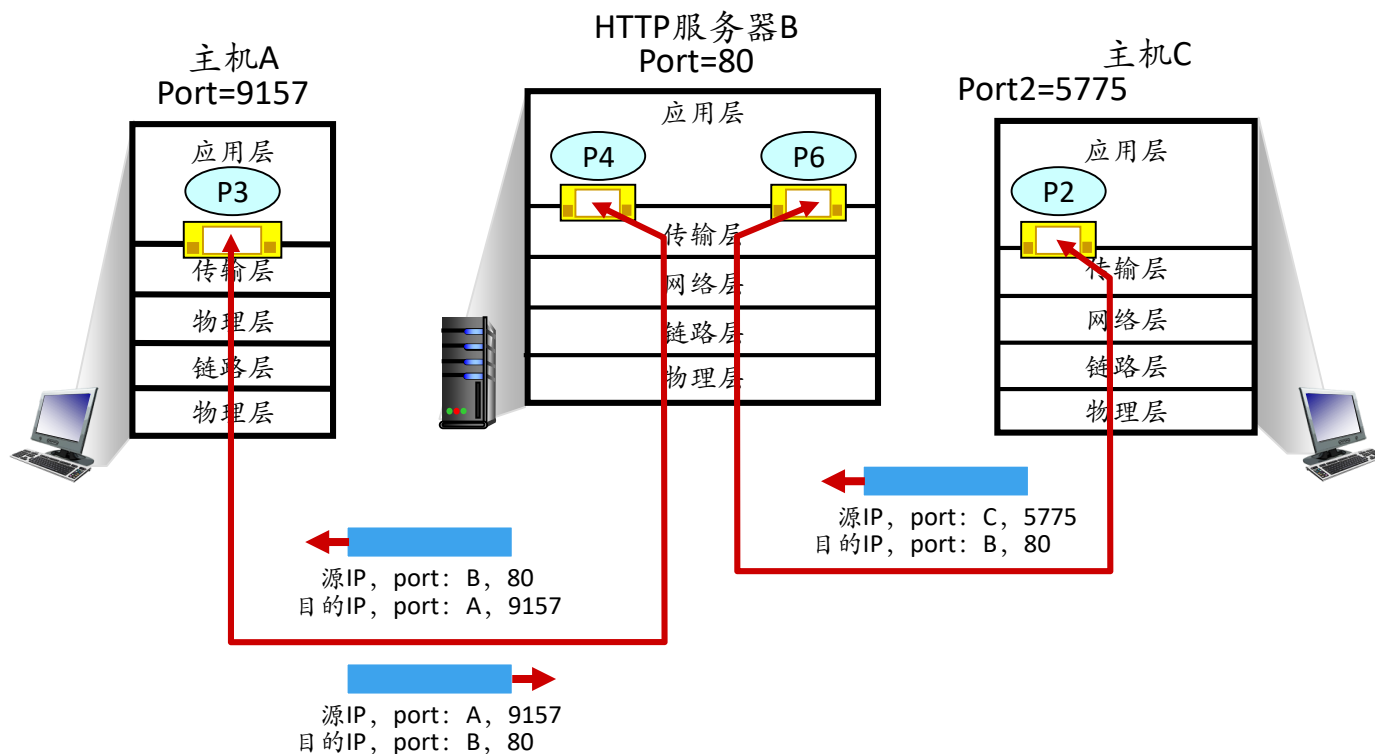
■ 服务器创建通信套接字

■ `connectionSocket, addr = serverSocket.accept()`



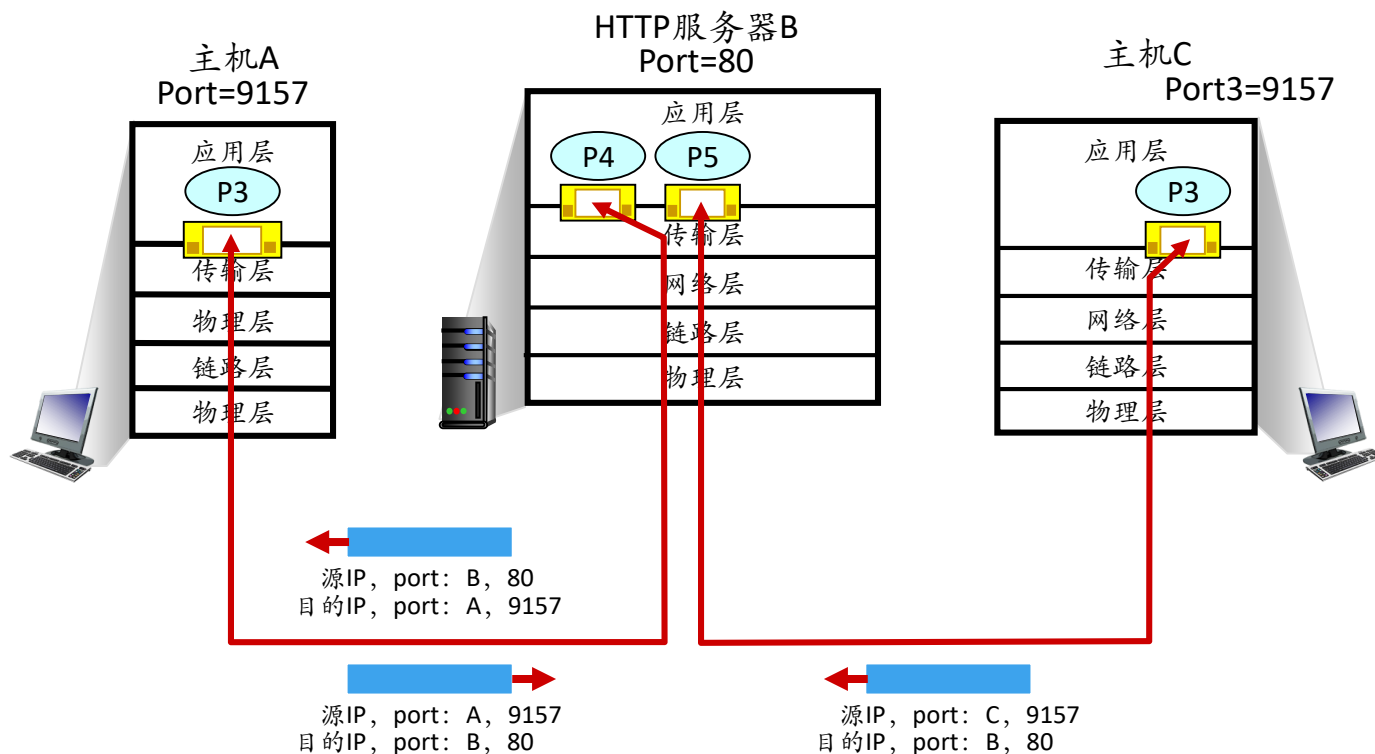
面向连接 (TCP)

■ 服务器创建通信套接字



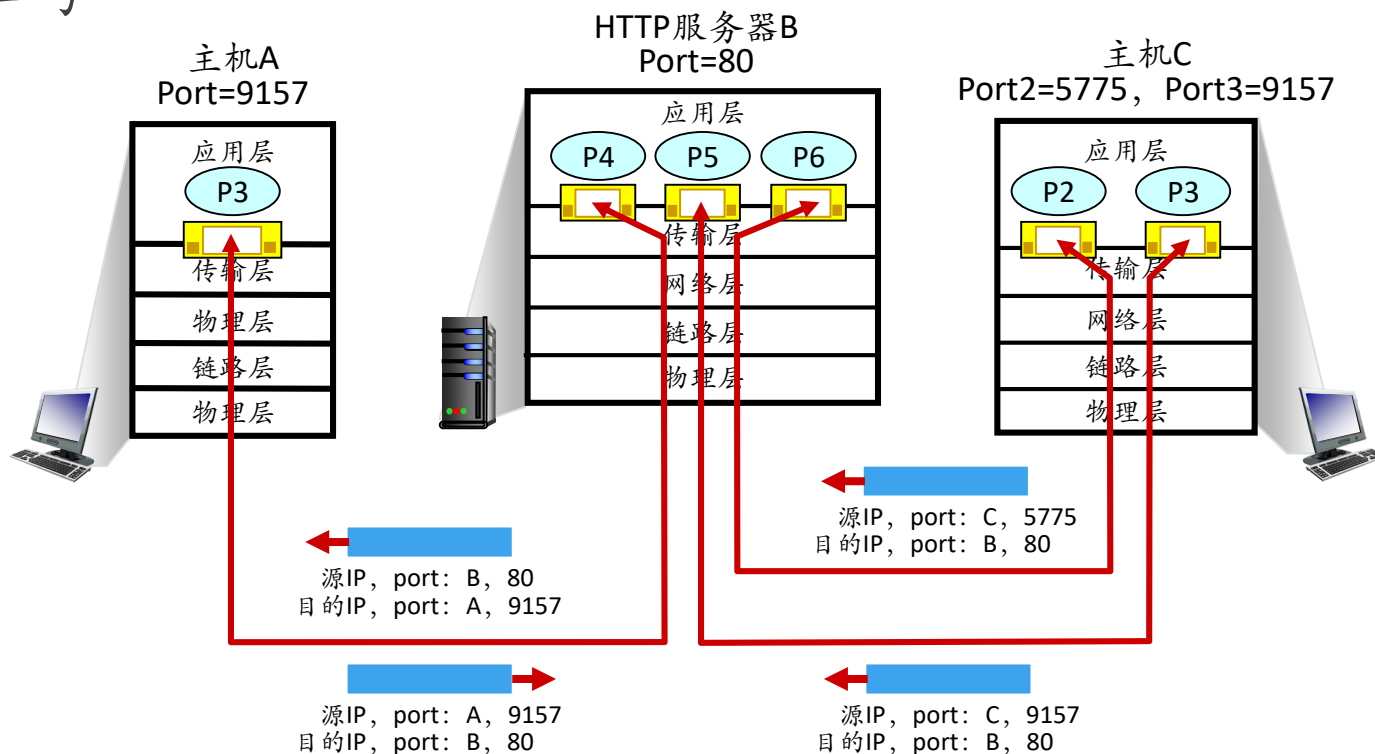
面向连接 (TCP)

- 客户端使用了同样的端口号，可以吗？
- 一个套接字对应一个连接，由四元组<源IP地址，源端口号，目的IP地址，目的端口号>识别



面向连接 (TCP)

- **注意**：一个套接字只能和一个套接字通信
- 一个套接字对应一个连接，由四元组<源IP地址，源端口号，目的IP地址，目的端口号>识别
- 所以，通过套接字发送数据的时候**不需要明示**对方的IP地址和端口号



TCP应用程序举例

- 客户端发送请求"Time", 服务器应答当前系统时间; 请求"Exit", 服务器应答"Bye"并结束连接。

服务器

```
Windows PowerShell
python .\system_time_inquiry_server.py

The server is ready to connect.
The server address: ('0.0.0.0', 12000)

Accepted a new connection.
The connection address: ('127.0.0.1', 12000)
The client address: ('127.0.0.1', 35637)
Received a request: Time.
Send a response: 2020-04-01 15:39:30.
Received a request: Exit.
Send a response: Bye.

Accepted a new connection.
The connection address: ('192.168.2.178', 12000)
The client address: ('192.168.2.119', 61632)
Received a request: Time.
Send a response: 2020-04-01 15:39:46.
Received a request: Exit.
Send a response: Bye.

Accepted a new connection.
The connection address: ('192.168.2.178', 12000)
The client address: ('192.168.2.119', 61633)
Received a request: Time.
Send a response: 2020-04-01 15:40:00.
Received a request: Exit.
Send a response: Bye.
```

本地客户端1

```
Windows PowerShell
python .\system_time_inquiry_client.py
A client is running.
The client address: ('127.0.0.1', 35637)
Connected to 127.0.0.1:12000.
Send a request: Time.
Received the current system time on the server: 2020-04-01 15:39:30.
Send a request: Exit.
Received a response: Bye.
```

远程客户端2

```
die@Doriss-Air ~/Downloads — zsh — 80x17
python system_time_inquiry_client.py
A client is running.
The client address: ('192.168.2.119', 61632)
Connected to 192.168.2.178:12000.
Send a request: Time
Received the current system time on the server: 2020-04-01 15:39:46.
Send a request: Exit
Received a response: Bye.

python system_time_inquiry_client.py
A client is running.
The client address: ('192.168.2.119', 61633)
Connected to 192.168.2.178:12000.
Send a request: Time.
Received the current system time on the server: 2020-04-01 15:40:00.
Send a request: Exit.
Received a response: Bye.
```


TCP应用程序举例

- 对于每个客户端，都创建一个新的连接套接字。

服务器

```
Windows PowerShell
python .\system_time_inquiry_server.py

The server is ready to connect.
The server address: ('0.0.0.0', 12000)

Accepted a new connection.
The connection address: ('127.0.0.1', 12000)
The client address: ('127.0.0.1', 35637)
Received a request: Time.
Send a response: 2020-04-01 15:39:30.
Received a request: Exit.
Send a response: Bye.

Accepted a new connection.
The connection address: ('192.168.2.178', 12000)
The client address: ('192.168.2.119', 61632)
Received a request: Time.
Send a response: 2020-04-01 15:39:46.
Received a request: Exit.
Send a response: Bye.

Accepted a new connection.
The connection address: ('192.168.2.178', 12000)
The client address: ('192.168.2.119', 61633)
Received a request: Time.
Send a response: 2020-04-01 15:40:00.
Received a request: Exit.
Send a response: Bye.
```

监听套接字端口号12000

连接套接字端口号12000

连接套接字端口号12000

连接套接字端口号12000

TCP应用程序举例

■ 为什么IP地址不同呢？

- ‘0.0.0.0’ 此处指匹配本地主机的所有IPv4地址；
- ‘127.0.0.1’ 指本地主机；
- ‘192.168.2.178’ 是本地主机在局域网中的IP地址。

服务器

```
Windows PowerShell
python .\system_time_inquiry_server.py

The server is ready to connect.
The server address: ('0.0.0.0', 12000)

Accepted a new connection.
The connection address: ('127.0.0.1', 12000)
The client address: ('127.0.0.1', 35637)
Received a request: Time
Send a response: 2020-04-01 15:39:30.
Received a request: Exit.
Send a response: Bye.

Accepted a new connection.
The connection address: ('192.168.2.178', 12000)
The client address: ('192.168.2.119', 61632)
Received a request: Time.
Send a response: 2020-04-01 15:39:46.
Received a request: Exit.
Send a response: Bye.

Accepted a new connection.
The connection address: ('192.168.2.178', 12000)
The client address: ('192.168.2.119', 61633)
Received a request: Time.
Send a response: 2020-04-01 15:40:00.
Received a request: Exit.
Send a response: Bye.
```

指匹配本地主机的所有IPv4地址，
指在本地主机的所有IPv4地址上的12000端口监听

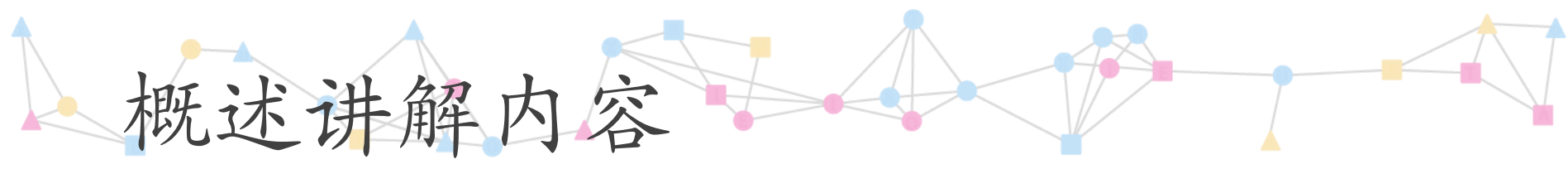
是本地主机在局域网中的IP地址，
客户端是网络中的其他主机

指本地主机，
客户端是本地主机自己



第三章知识点汇总

- 理解多路复用与多路分解的原理
- 理解无连接的复用/分解原理
- 理解面向连接的复用/分解原理



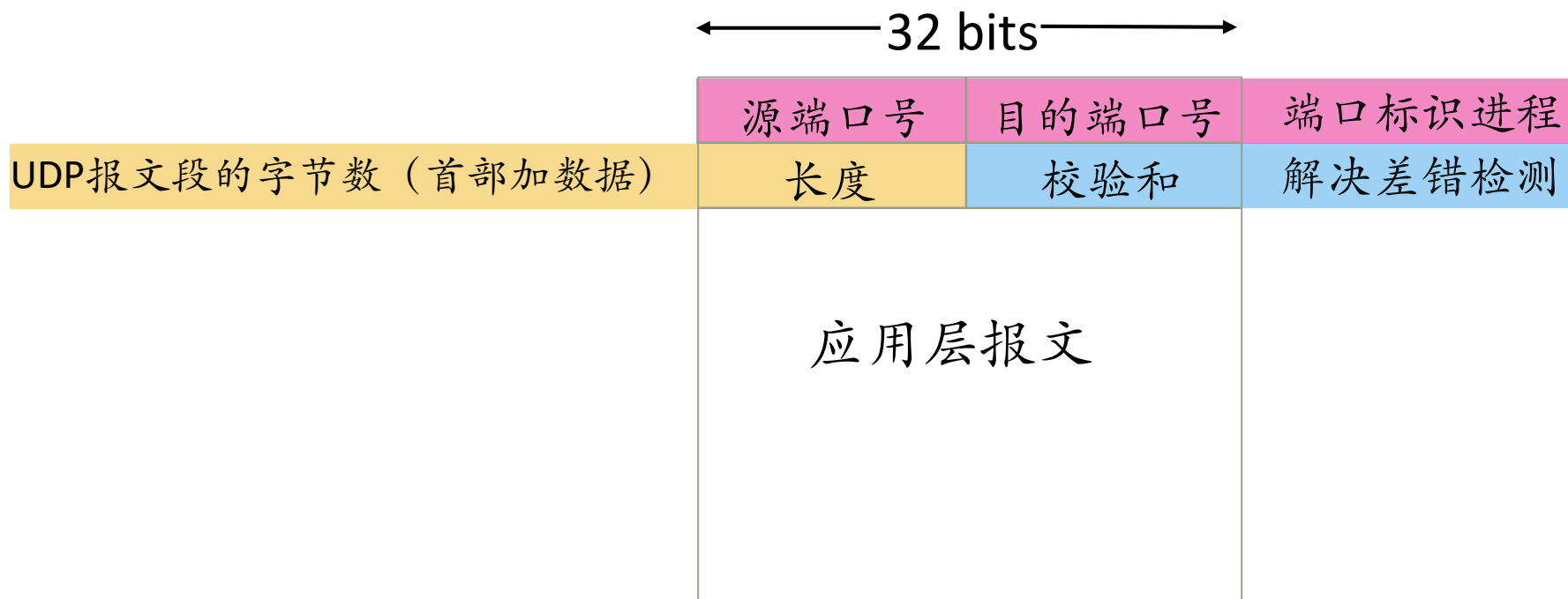
- 传输层需求、服务、协议
- 多路复用与多路分解
- UDP协议



UDP: User Datagram Protocol [RFC 768]

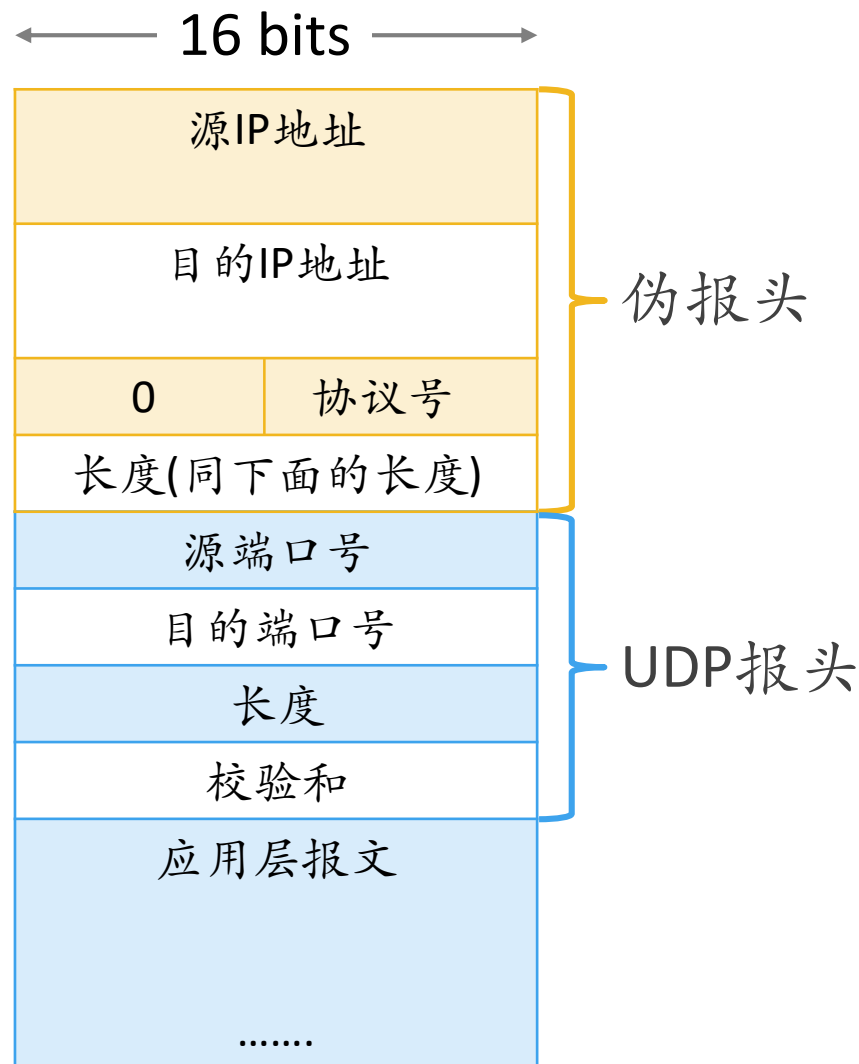
- 进程间交付
 - IP地址 + 端口号
- 差错检测
 - 校验和
- 不可靠的传输层协议
 - 无法恢复差错
 - 可能丢包
 - 可能乱序
- 使用UDP的应用层程序
 - 流媒体应用
 - DNS
 - 网络管理协议
 -

UDP报文段结构



校验和

- 检查报文段是否出错
- (回卷) 求和取反
- 伪报头
 - UDP在计算校验码时, 构造这部分用于计算, 但不传输。
 - (历史遗留问题)



UDP校验和

■ 举例：两个16位整数

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
求和	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
回卷																	1
求和		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
取反		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1



UDP校验和

- 接收方差错检测

- 如果传输未产生差错，则接收方将所有数据（包括校验和）按照16位求和，结果应该为？



UDP校验和

■ 接收方差错检测

- 如果传输未产生差错，则接收方将所有数据（包括校验和）按照16位求和，结果应该为**16位全幺**

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
求和	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
回卷																1
求和	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
校验和	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1
求和	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1



UDP校验和

- 接收方差错检测

- 如果传输未产生差错，则接收方将所有数据（包括校验和）按照16位求和，结果应该为**16位全0**

- 在接收方，如果求和结果不是16位全0，则说明传输产生了差错
- 否则，说明传输未产生差错

这种检测方法对吗？

UDP校验和

- 举例：红色加框数字表示传输出错了

	1	1	1	0	0	1	1	1	0	1	0	0	1	1	0
	1	1	0	1	0	1	0	0	0	1	0	1	0	1	0
求和	1	1	0	1	1	1	0	1	1	0	0	1	1	0	1
回卷															1
求和	1	0	1	1	1	0	1	1	1	0	0	1	1	1	0
校验和	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1
求和	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1



UDP校验和

■ 接收方差错检测

- 如果传输未产生差错，则接收方将所有数据（包括校验和）按照16位求和，结果应该为**16位全0**

- 在接收方，如果求和结果不是16位全0，则说明传输产生了差错
- 否则，不能说明传输未产生差错，只能说明未检测出差错



第三章知识点汇总

- 理解UDP协议的服务类型
- 了解UDP报文格式
- 掌握校验和的计算方法
- 理解校验和存在无法检测出差错的情况

*We read the world wrong and say that it deceives
us.*

——Tagore