南開大學
Nankai University

基于 UDP 服务可靠传输协议编程实现（3-4）

姓 名：　边　笛

学 号：　2012668

学 院：计算机学院

目 录

一、 实验内容说明 1
二、 实验内容说明 1
1. 实验 3-1 1
2. 实验 3-2 1
3. 实验 3-3 1
三、 测试内容说明 1
四、 测试结果分析 1
1． 停等机制与滑动窗口机制性能对比 1
2． 滑动窗口机制不同窗口大小对性能的影响 3
3． 有拥塞控制和无拥塞控制性能对比 5
五、 传输协议说明 7
1. rdt3.0 超时重传 7
2. 停等机制 7
3. 累计确认 7
4. 滑动窗口 GBN 7
5. 拥塞控制——reno 算法 8
6. 丢包设置 8
六、 思考与感悟 ..... **错误！未定义书签。**

一、实验内容说明

本实验对第三次作业中实现的三个基于 UDP 服务设计的可靠传输协议进行性能测试与分析。

二、实验内容说明

1. 实验 3-1

实现功能：三次握手建立连接、四次挥手断开连接、校验和差错检验、rdt3.0 超时重传、停等机制。

2. 实验 3-2

实现功能：三次握手建立连接、四次挥手断开连接、校验和差错检验、rdt3.0 超时重传、累计确认、基于滑动窗口 GBN 的流量控制。

3. 实验 3-3

实现功能：三次握手建立连接、四次挥手断开连接、校验和差错检验、rdt3.0 超时重传、累计确认、基于滑动窗口 GBN 的流量控制、reno 算法拥塞控制。

三、测试内容说明

实验以传输数据样例中的 3.jpg（11.4 MB）为例，进行传输测试。在传输测试中通过改变丢包率或延时，以控制变量法测试停等机制与滑动窗口机制、滑动窗口机制不同窗口大小、有无拥塞控制传输协议下的文件传输时间，多次测量取平均值，并对所得性能结果进行分析。（重传时间始终为 500ms）。

四、测试结果分析

1．停等机制与滑动窗口机制性能对比（滑动窗口大小=10）

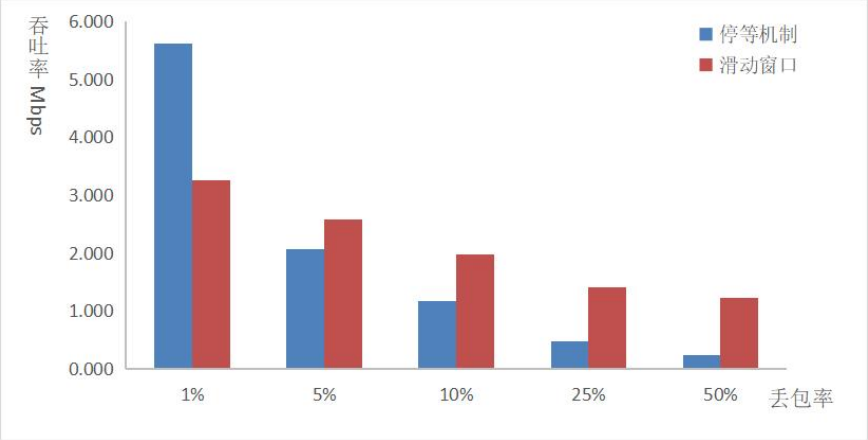
停等机制与滑动窗口相比，停等机制是每发一个数据包就会进行一次确认，通信性能受制于 RTT 的大小；而滑动窗口是以更大的单位进行确认，即使 RTT 较长，也能控制网络性能的下降。

① 延迟时间恒为 0，改变丢包率

在不同的丢包率下运行实验 3-1、3-2 的程序代码进行文件传输，得到如下测试数据：

测试结果 / 丢包率	文件传输时间（s）		吞吐量（Mbps）	
	停等机制	滑动窗口机制	停等机制	滑动窗口机制
1%	16.24	28.01	5.62	3.26
5%	44.26	35.30	2.06	2.58
10%	78.62	46.43	1.16	1.96
25%	195.26	65.35	0.47	1.40
50%	377.97	74.86	0.24	1.22

将测试结果绘制成条形图如下：



通过图表可以看到，对于停等机制与滑动窗口协议，在相同的延迟时间下，改变丢包率：

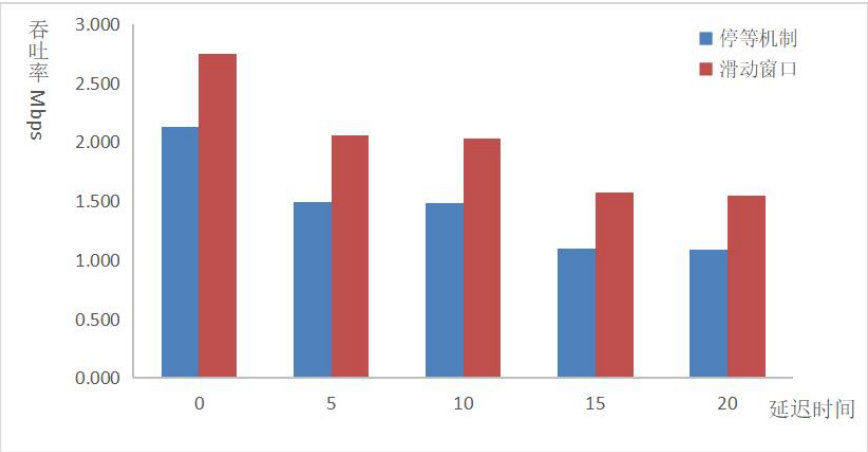
- **当丢包率很小时，滑动窗口吞吐率低于停等机制，分析导致这种现象的原因是丢包率较小的情况下停等机制的 RTT 比较小，停等机制性能并未受到明显牵制，但由于滑动窗口协议下重传的是整个窗口中的数据包，导致滑动窗口协议重传的时间开销大。**
- **随着丢包率增大，滑动窗口的性能由于停等机制，且丢包率越大，滑动窗口协议的性能优势越明显。这是因为丢包率增大，停等机制 RTT 增大，性能受制于 RTT 大小，而滑动窗就机制相比之下受制程度更小。**

② 丢包率恒为 5%，改变延迟时间

在不同的延迟时间下运行实验 3-1、3-2 的程序代码进行文件传输，得到如下测试数据：

测试结果 / 延迟时间	文件传输时间（s）		吞吐量（Mbps）		比值
	停等机制	滑动窗口机制	停等机制	滑动窗口机制	
0	42.79	33.25	2.13	2.74	1.29
5	61.23	44.34	1.49	2.06	1.38
10	61.69	44.90	1.48	2.03	1.37
15	83.54	58.04	1.09	1.57	1.44
20	84.06	58.94	1.08	1.55	1.43

将测试结果绘制成条形图如下：



通过图表可以看到，对于停等机制与滑动窗口协议，在相同的丢包率下，改变延迟时间：

停等机制性能始终比滑动窗口协议性能差，并且随着延迟时间的增大，滑动窗口协议与停等机制吞吐率的比值整体呈上升趋势。

这是由于本身在 5%的丢包率下，停等机制的性能下降本身就比滑动窗口协议，增加延迟时间相当于进一步增大了 RTT，加剧了停等机制的性能下降，而滑动窗口协议由于窗口这一整体的存在，受到的性能影响会更小，因而滑动窗口协议性能始终更优且优势随着延迟时间的增加呈增大趋势。

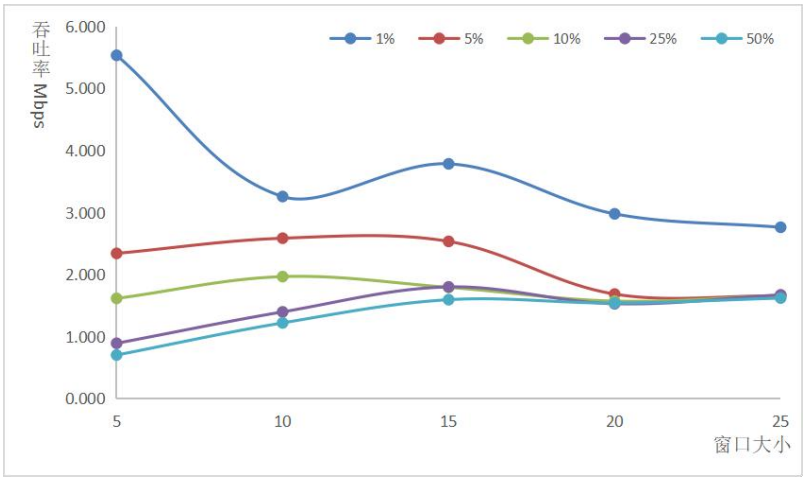
2．滑动窗口机制不同窗口大小对性能的影响

① 延迟时间恒为 0，改变丢包率

在不同的丢包率下分别设置实验 3-2 中窗口大小为 5、10、15、20、25 进行文件传输，得到如下测试数据：

测试结果 / 丢包率	不同窗口大小下的文件传输时间（s）				
	5	**10**	**15**	**20**	**25**
1%	16.49	28.01	24.11	30.64	33.05
5%	39.00	35.30	36.02	54.19	55.13
10%	56.53	46.43	50.92	58.08	54.90
25%	102.55	65.35	50.72	59.72	54.61
50%	130.22	74.86	57.27	59.29	56.35

将测试结果绘制为折线图如下：



通过图表可以看到，对于不同窗口大小，在相同的延迟时间下，改变丢包率：

- **当丢包率较低时的吞吐率明显高于其他吞吐率。**

  对于并不是非常小的丢包率，在滑动窗口较小时，相同的窗口大小下丢包率大的性能差，随着丢包率的不断增大，当丢包率较大时不同丢包率下的性能差异相较于丢包率小些时已经不明显了。

  这是由于丢包率太大，重传数据量已经不相上下，吞吐量已经很低了。特别是在窗口较大时，不同丢包率下性能差异已经非常小了。

- **随着窗口大小的增大，吞吐率既存在上升的情况也存在下降的情况，当窗口大小较大时，吞吐率变化幅度逐渐降低，吞吐率趋于稳定。**

  在相同丢包率下窗口大小对于传输性能的影响主要体现在两方面，一方面，窗口越大传输性能在 RTT 增大时下降的越小；另一方面窗口越大数据重传的时间代价越大。这两方面一个提升传输性能，一个降低传输性能。

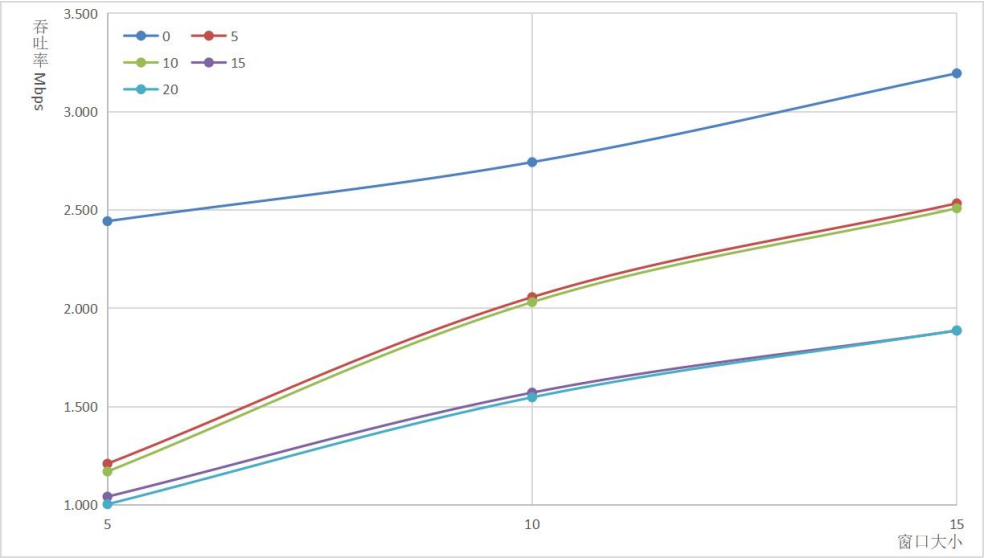
  因此在相同丢包率下，随窗口大小增大，吞吐率是波动的，有可能增大，有可能减小，具体增大还是减小取决于窗口大小增大对于上述两方面哪方面影响更大。但在限定条件下，如果窗口无限大，窗口中实际容纳数据数量是有上限的，因此当随着窗口大小增大，性能变化幅度减小，并呈现一个趋于稳定的趋势。

② 丢包率恒为 5%，改变延迟时间

在不同的延迟时间下分别设置实验 3-2 中窗口大小为 5、10、15 进行文件传输，得到如下测试数据：

测试结果 / 延迟时间	不同窗口大小下的文件传输时间（s）		
	5	**10**	**15**
0	37.33	33.25	28.55
5	75.38	44.34	36.00
10	77.93	44.90	36.36
15	87.48	58.04	48.34
20	90.84	58.94	48.31

将测试结果绘制为折线图如下：



通过图表可以看到，对于不同窗口大小，在相同的丢包率下，改变延迟时间：

- **0 延迟性能明显优于有延迟的性能。随着延迟时间增加，性能降低。**

  延迟时间增加,RTT 增大，性能降低。值得注意的是，延迟时间增加相同的数值，吞吐率变化差异比较大，推断可能在延迟时间在某个范围内的传输性能

都比较相近，整体呈现阶跃式的变化趋势。

- 在限定范围内随着窗口大小的增大，吞吐率增大。不同延迟时间下的变化趋势比较一致。

3．有拥塞控制和无拥塞控制性能对比（滑动窗口大小=初始 ssthresh=5，初始 cwnd=1）

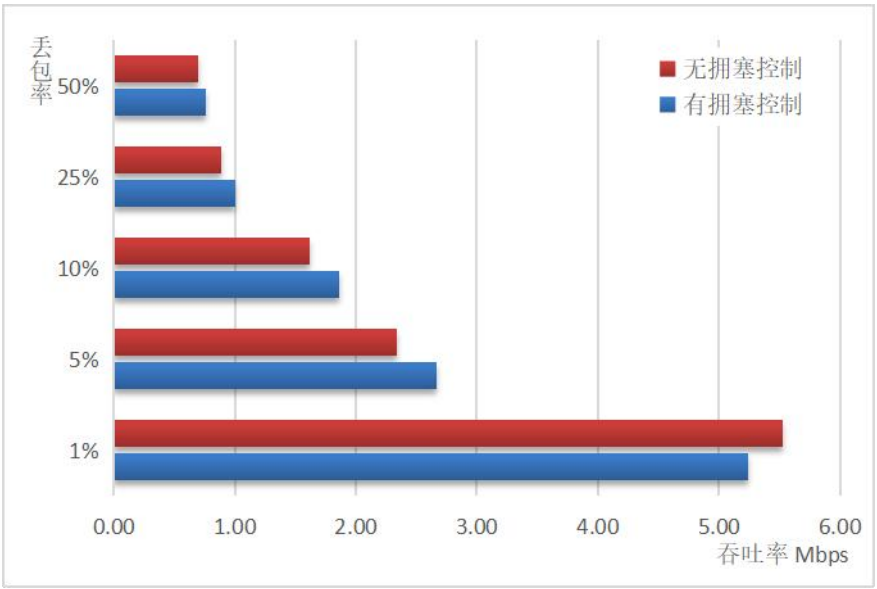
在某段时间，若对网络资源的需求超过了该资源所能提供的可用部分就会出现拥塞。在滑动窗口控制下，发送端一次发送的数据量为窗口大小，但如果网络不好，窗口过大可能会导致网络瘫痪。因此设计拥塞控制算法对发送数据量也就是窗口大小进行控制，通过拥塞窗口的增大或减小控制发送速率。

① 延迟时间恒为 0，改变丢包率

在不同的丢包率下运行实验 3-2、3-3 的程序代码进行文件传输，得到如下测试数据：

测试结果 / 丢包率	文件传输时间（s）		吞吐量（Mbps）		比值
	有拥塞控制	无拥塞控制	有拥塞控制	无拥塞控制	
1%	17.41	16.49	5.24	5.53	0.95
5%	34.16	39.00	2.67	2.34	1.14
10%	48.91	56.53	1.86	1.61	1.16
25%	91.03	102.55	1.00	0.89	1.13
50%	120.35	130.22	0.76	0.70	1.08

将测试数据绘制成图形结果



通过图表可以看到，对于有无拥塞控制，在相同的延迟时间下，改变丢包率：

- 当丢包率很低时无拥塞控制性能更好。

  这是由于丢包率比较小，拥塞控制发挥的作用微乎其微，但拥塞控制机制会带来额外的阶段切换开销。

- 在丢包率不是极小的时候，有拥塞控制的性能要比没有用拥塞控制性能更

好。

这是由于丢包过多可能会造成拥塞，在拥塞控制算法的作用下，拥塞窗口大小发生改变，实现对发送速率的控制，实现对拥塞的避免。

- 随着丢包率增大，拥塞控制对于性能的能力降低。

  主要还是因为丢包率太大，长期处于拥塞避免阶段，发送速率比较低，与无拥塞控制发生拥塞时的发送速率相比快不了太多，丢包率越高差距越小。

- 随着丢包率增大，有拥塞控制性能下降幅度小于无拥塞控制

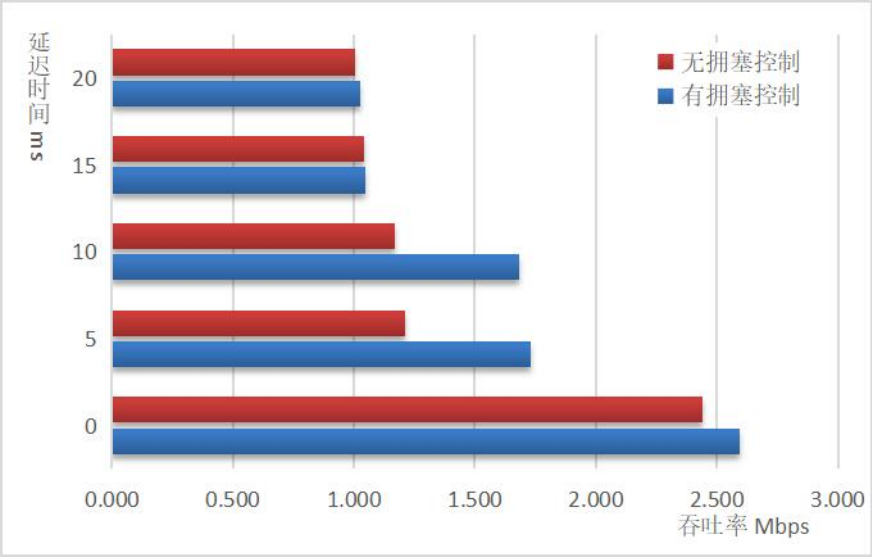
  拥塞控制可以在网络大量丢包时，缩小窗口大小以避免大量的数据重传，减少重传代价

② 丢包率恒为 5%，改变延迟时间（滑动窗口大小=初始 ssthresh=5，初始 cwnd=1）

在不同的延迟时间下运行实验 3-2、3-3 的程序代码进行文件传输，得到如下测试数据：

测试结果 / 延迟时间	文件传输时间（s）		吞吐量（Mbps）		比值
	有拥塞控制	无拥塞控制	有拥塞控制	无拥塞控制	
0	35.18	37.33	2.59	2.44	1.06
5	52.63	75.38	1.73	1.21	1.43
10	54.19	77.93	1.68	1.17	1.44
15	86.92	87.48	1.05	1.04	1.01
20	88.88	90.84	1.03	1.00	1.02

将测试数据绘制成图形结果



通过图表可以看到，对于有无拥塞控制，在相同的丢包率下，改变延迟时间：

- 有拥塞控制性能更好

  这是由于本身在当前丢包率下，拥塞控制算法能发挥出还行的拥塞控制作用。

- 有适当延时的情况下，拥塞控制下的性能提升更多。

  有适当的延时，对于一些小型拥塞不做处理，降低拥塞控制在小型拥塞上的

开销，对于大拥塞，拥塞控制机制缩小窗口大小以避免大量的数据重传，更好的提升了性能。

- **延迟时间较大时，有拥塞控制相较无拥塞控制性能提升微小。**

  延迟时间对于性能的影响远大于拥塞带来的影响，因此有无拥塞控制性能差距不大。

五、传输协议说明

1. rdt3.0 超时重传

在程序中设置了一个最大等待时间，如果在接收到一条消息后的最大等待时间内没有收到下一条消息，就认为是超时。因此在收到消息后会开始计时，如果超过最大等待时间没有收到下一跳正确的消息就会进行重传，重传时重传所有未得到确认的已发送消息，对于实验 3-1，就是重传上一条发送的信息，对于实验 3-2、3-3 是重传序列号属于[lastrecvACK,lastsendSEQ]的消息。

2. 停等机制

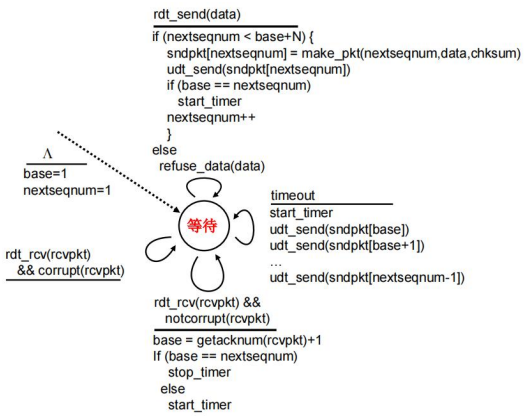
在实验 3-1 中，在数据传递时会进行序列号 Seq 与确认号 Ack 的判断。接收时只接受 Ack 是上一次接收的数据报的序列号，Seq 是上一次发送的数据报序列号加一的信息。

3. 累计确认

在实验 3-2、3-3 中，接收端确认采用累计确认方式。发送端在收到确认信息后会滑动窗口，将序列号小于等于确认号的信息移出窗口，即 ack= x+1 代表对于 序列号为 x+1 之前的所有信息在接收方都已经确认接收完毕，接收方期望发送方发送 x+1 及其后面的消息。

4. 滑动窗口 GBN

在实验 3-2、3-3 中，数据传输时使用滑动窗口 GBN 的方法进行传输。GBN 状态机如下：

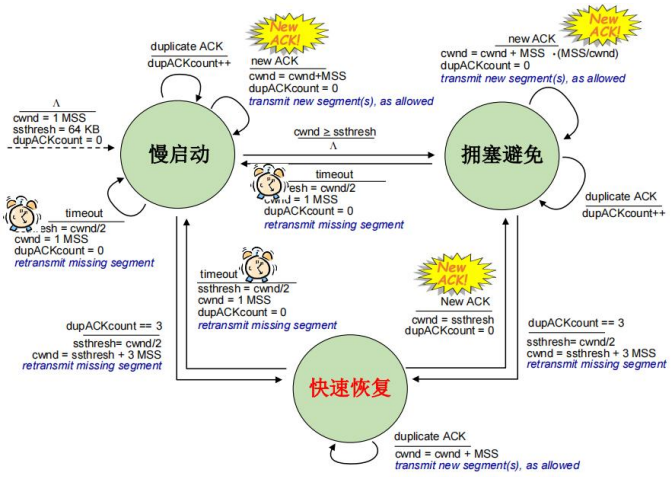


发送端的发送不依靠于收到确认信息，而是采用连续发送的方式，只要滑动窗口没有满，就可以发送，如果窗口已满，就等待收到 ACK 窗口滑动后再次发送。因此发送端接受和发送

的完成需要使用多线程完成，一条线程用于数据发送，另一条线程用于确认信息的接收。滑动窗口由使用动态数组 vector 维护，在发送消息后会将消息加入到动态数组中，更新 lastsendSEQ。每接收到一条消息就更新 lastrecvACK，实现累计确认。每接收或者发送一条消息对于 lastsendSEQ、lastrecvACK 的更新就相当于滑动窗的移动。

5. 拥塞控制——reno 算法

在实验 3-3 中，使用 reno 算法进行拥塞控制，reno 算法的状态图如下：



reno 算法是一种基于窗口的拥塞控制算法，通过拥塞窗口的增大或减小控制发送速率。在程序中预先设定了 MSS（Maximum Segment Size）和上限 ssthresh，在接受消息的过程中记录并根据算法维护连续接收到的冗余 ACK 报文数 dupACKcount。初始处于慢启动阶段，根据是否接受到冗余 ACK，连续接收到的冗余 ACK 报文数以及是否超时，数据传输会在慢启动、拥塞避免、快速恢复三个阶段进行状态切换，并更新 cwnd 和 ssthresh，实现拥塞控制。

6. 丢包设置

在三次实验中都中编写了丢包函数，在发送数据时随机以设置的丢包率 loss_rate 丢包。

```
bool LossPackage()
{
    return rand() % 100 < loss_rate;
}
```

六、思考与感悟

通过对三次实验编写协议性能进行测试，更深刻的理解到了可靠传输协议中各个部分的作用，把理论应用到实践对于掌握所学真的有很大的帮助，代码能力的提升真的很有必要。