GEW项目大纲

1. 环境及架构
2. IDE：VS2013 Community
3. 相比6.0的好处？
4. 代码提示（Intellisense）
5. 面向对象对程序员的友好性？
6. 不应该刻意背代码。
7. 语法检查（Intellisense）
8. 为什么不应该人为检查语法？人类的模式识别优势天生不适合
9. 能花钱的地方尽量不要花时间。
10. 高度集成
11. 多语言：C++，C#，VB，Python，Node.js…
12. 多模版：控制台，桌面，网站，移动客户端…
13. 插件丰富：Git
14. 版本比较
15. 时间版本
16. 6.0
17. 2005
18. 2008
19. 2010
20. 2012
21. 2013
22. 2015
23. 功能版本
24. Ultimate
25. Premium
26. Professional
27. Test Professional
28. Express
29. Community
30. 获取方式
31. Community
32. Express
33. DreamSpark
34. 演示：计时调试
35. 解决方案（sln）与项目（\*proj）
36. 为什么要分解决方案和项目？
37. 解决方案和项目的关系？
38. 语言：C#
39. 名称：See Sharp（五线谱中升半调符号 -> #）
40. 分类：强类型、静态语言（也可作动态语言）、纯面向对象
41. 语法：与C、C++、Java相似
42. 运行方式：

|编译到IL，IL经即时编译成本机代码

|编译到IL，同时继续编译到本机代码（Ngen、.NET Native）

1. .NET Framework：
2. 理念：与Java相反
3. 实现方式：与Java相似
4. 库：.NET Framework 类库，任何dll文件
5. 常用的项目类型
6. Windows Desktop
7. Windows Store（Win8 & WinPhone）
8. ASP.NET
9. Unity3D
10. Xamarin（iOS、OSX、Android）
11. 与C++的比较
12. 界面库：WPF
13. 语言：XAML
14. 演示：响应式布局、动画特效
15. 项目开发流程
16. 模块流程
17. 先做客户端，再做服务器端
18. 为什么先做客户端再做服务器端？
19. |为什么不同时做客户端和服务器端？
20. ||为什么能够不同时做客户端和服务器端？Mock
21. ||为什么应该不同时做客户端和服务器端？人类不擅长多任务
22. |为什么不先做服务器端而是先做客户端？现有的Mock Server
23. 一个解决方案，两个项目
24. 为什么需要用独立项目？
25. 时刻获取
26. 调用非.NET代码实现

//时钟周期数

[DllImport("kernel32.dll")]

private extern static bool QueryPerformanceFrequency(ref long x);

//时钟频率

[DllImport("kernel32.dll")]

private extern static bool QueryPerformanceCounter(ref long x);

1. ref/out

标识引用关键字。Swap函数的实现。值类型的值传递、值类型的引用传递、引用类型的值传递、引用类型的引用传递。

1. C#中的指针

可行，但不建议。

1. 日志记录

JSON

1. 核心功能及技术
2. 响应式布局
3. 输入控件校验（绝不能相信用户）
4. 客户端及服务器整合
5. TCP/UDP多协议并行
6. 客户端多连接并行
7. 路由跳数记录（调用traceroute）
8. 实时信息图表显示
9. 指定内容or指定长度
10. Hostname解析
11. 问题及解决方案
12. 图表实时更新无效

每次发送和接收消息后自动更新图表，所有代码均正确无误，但界面不会一直卡死（所有网络及计算代码均在后台线程），直到所有消息发送完毕后才会显示结果。

1. Accept内存泄漏

.NET中有三种Accept方式，Accept、BeginAccept和AcceptAsync：

Accept：阻塞，顺序执行；BeginAccept：非阻塞，执行回调函数；AcceptAsync：较为复杂，有兴趣自己搜。

while(true)

{

mySocket.BeginAccept(callback, stateObj);

}

然后，妥妥的内存泄露。

1. int recv(…)

一次recv的数据不一定是一次send的数据：

|由于客户机和服务器均为一次send对一次recv，故不存在多次发送一次接收的问题；

|但一次发送的数据可能无法一次到达，故一次recv的内容可能只是一次send的内容的一部分，并且下次recv时收到的可能是上次未收完的剩余数据，导致时间记录错乱。

测试临界值：TCP（1368），UDP（1472）。

解决方案：

|根据发送数据长度while循环（需要增加额外逻辑）

|限制用户发送大数据（本项目采用）

1. UDP丢包

UDP不保证可靠性，可能在中途丢包，若直接放弃该数据点，接着后面的测试，则万一一定时间后该数据包返回，则引起计时错乱。

|使用额外的应用层协议，比如增加序号（需要增加额外逻辑）

|等待一个足够长的时间后继续进行（需要增加额外耗时）

|切换到另一个端口后继续进行（需要增加额外逻辑）

|放弃后面的测试数据点（本项目采用）

1. 测试目标与计划
2. 近场通信（时序-延时图，延时-频率图）
3. localhost
4. LAN-Direct
5. WLAN-Direct
6. LAN-Router-LAN
7. WLAN-Router-LAN（2.4GHz）
8. WLAN-Router-LAN（5GHz）
9. WLAN-Router-WLAN（2.4GHz）
10. WLAN-Router-WLAN（5GHz）
11. 远端通信【自动探测】（跳数-平均延时图）

分段随机IP地址，尝试在TCP端口7建立连接，若连接成功，则启动对目标主机的回音壁测试，同时启动traceroute程序获取到目标主机的距离。已获取延时与距离的关系。

1. 压力测试
2. 进程限速【电脑管家】（限速-延时图\*3）

单独限制下行速率、单独限制上行速率、同时等量限制上下行速率。

1. 消息长度（长度-延时图\*2）

指定网络环境，采用不同的消息长度，测试所需延时（TCP + UDP）。

1. 附加测试
2. DNS解析时间测试（不同级别的域名）
3. TCP连接建立时间测试（不同距离）
4. UDP丢包率测试（不同距离）
5. 数据结果及分析