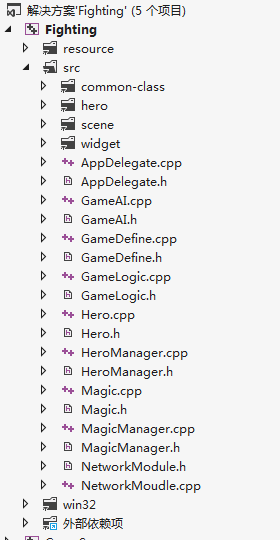
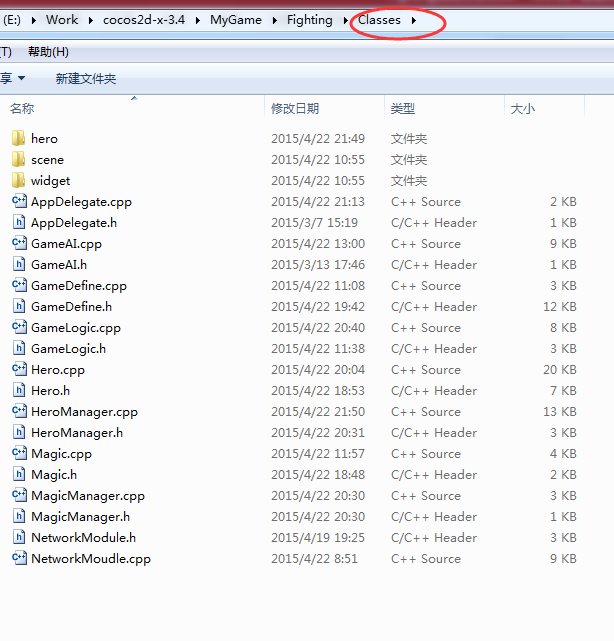
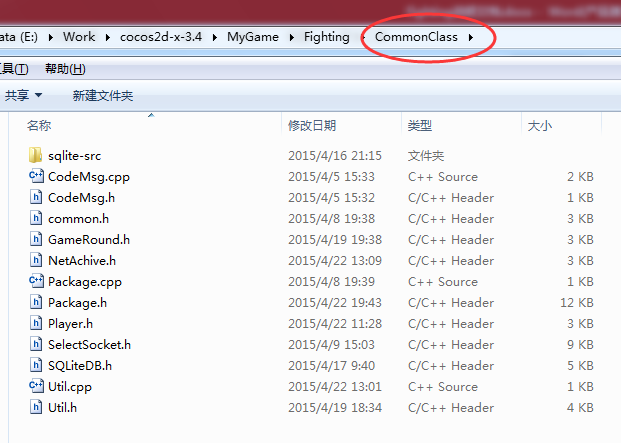
# Fighting工程结构



# Fighting的目录结构







Classes里面存放着Fighting工程需要的代码文件

GameServer里面存放着服务器的工程和文件

CommonClass里面存放着客户端和服务器通用的代码文件

# 游戏运行过程：

# 游戏启动后程序首先进入：AppDelegate::applicationDidFinishLaunching()，在这个函数内部调用

auto scene = StartScene::createScene();

director->runWithScene(scene);

即运行开始场景StartScene

1. 在开始场景的初始化函数StartScene::init中加载了启动画面背景之后，调用一个计时器this->scheduleOnce(schedule\_selector(StartScene::jumpToMainScene),1);，在1秒后跳入游戏选择场景ChooseGameScene。
2. 在游戏选择场景的初始化函数ChooseGameScene::init中，加载背景和菜单，然后根据用户选择的不同进入不同的游戏：选择单机则进入SingleGameSelectHeroScene场景，选择网络游戏则进入LoginGameScene登陆界面
3. 在登陆界面点击登陆后，首先连接服务器，连接成功后，将用户id和密码发送到服务器上去验证，验证成功后返回成功，否则返回失败；如果登陆成功则可以进入CreateOrJoinScene场景，否则停留在原场景。
4. 在CreateOrJoinScene场景，用户选择是创建游戏还是加入游戏，如果是创建游戏，那么进入CreateGameScene；如果是加入游戏，那么进入JoinGameScene。
5. 在CreateGameScene中，玩家选择一个英雄，然后输入游戏的名字，接着点击创建游戏，然后一直等待，直到有另外一个玩家加入该游戏，然后进入主场景MainScene进行游戏。
6. 在JoinGameScene中，玩家选择一局游戏，然后选择自己的英雄（玩家的英雄不能和对方的英雄相同），点击加入游戏，然后进入主场景进行游戏。
7. 在游戏主场景中进行游戏，如果对方失败了或者对方中途退出，那么自己胜利，否则自己失败；接着进入ResultScene场景，显示游戏结果。

# Fighting的设计思想

1. 涉及到图像处理的类有：场景类、Hero类及其子类（英雄有对应的图像需要处理）、Magic类（技能也有对应的图像需要处理），其他的类均不涉及图像显示处理。写代码的时候不要把处理图像的代码到处写。
2. GameLogic相当于游戏的控制器和包装器还有主逻辑，所有类之间的交互基本上都要通过GameLogic类进行处理（即GameLogic相当于一个中间层），这样就能把原来类之间的紧耦合关系进行解耦，所有的类基本只和GameLogic交互。这样各个模块就能专心的处理自己的事情，然后提供借口给其他模块调用。
   1. 例子1：NetworkMoudle专注于处理和服务器之间的通信；在登陆场景中需要调用网络模块并进行相应的逻辑判断，此时我们不应该将这部分代码直接写在登陆场景中，而是在GameLogic中处理，在GameLogic中应该有一个Login函数（如果没有，就自己添加），参数是id和密码，返回值是true或false，在登陆场景中得到用户的用户和密码之后，我们直接传给GameLogic的Login函数，然后在Login函数中判断合法性、处理网络调用还有返回结果等等，然后登陆场景跟Login函数的返回值来进行判断登陆是否成功！——这样我们就能将游戏逻辑与界面分离，游戏逻辑的处理与具体的界面无关。
   2. 例子2：HeroManager模块专注于处理管理英雄和控制英雄的行为；但是在主场景中需要控制英雄的行为，这些东西我们也是透过GameLogic来调用HeroManager，这样无论HeroManager怎么变化，只要GameLogic提供的接口不变，主场景的代码就都不需要改变。
   3. 例子3：在联网模式下，HeroManager中需要调用网络模块进行通信，也是通过GameLogic而不是直接调用，这样无论网络模块如何变化，只要GameLogic的接口不变，HeroManager就不需要改变。
   4. 注释：目前代码中还没有完全实现这一步，还有很多紧耦合的代码！
3. 与业务处理（业务处理就是登陆、注册、创建游戏、加入游戏、离开游戏等等）相关的代码只能放在GameLogic中，其他的模块无论是HeroManager（管理英雄）还是NetworkMoudle（处理通信）都只提供功能性的代码而不涉及逻辑处理，这样游戏的逻辑就清晰明了
4. 其他

# 代码风格

1. 函数以Fun为例子，变量以Var为例子
2. 数据类型缩写：
   1. 整形：n或i
   2. 字符串数组：sz
   3. 字符串指针：psz或pstr
   4. string（stl中的string）类型的字符串：str
   5. 指针：p
   6. 字符：c
   7. 布尔类型：b
3. 全局变量，开头加G，然后是下划线，接着是数据类型，最后是变量名。例如：G\_nVar、G\_iVar、G\_szVar、G\_pszVar、G\_strVar、G\_pVar、G\_cVar、G\_bVar等等
4. 全局函数，开头是G，加下划线，然后是函数名，例如：G\_Fun
5. 类属性（变量），开头加m，然后是下划线，然后是变量类型，最后是变量名，例如：m\_nVar，m\_iVar，m\_szVar，m\_pstrVar，m\_strVar，m\_pVar，m\_cVar，m\_bVar
6. 类函数：由头字母大写的若干个单词组成，例如：GetName，SetName，PutValue，GetValue等等
7. 函数内的变量：最好以类型开头，接着是变量名，例如：iVar、nVar等等，但是不做强制要求，但是不要写一些别人看不懂的变量，例如x、v、y等等意义不明确的变量；除了for迭代，不允许使用i、j，k等变量。

# Fighting的代码

1. 宏定义和一些全局的定义
   1. 文件：GameDefine.h GameDefine.cpp
   2. \_USE\_PC\_：使用PC主机的宏标志，因为cocos2dx可以移植到手机上，所以定义这个宏，在定义了这个宏的情况下，玩家可以使用键盘控制英雄。
   3. \_TEST\_：测试宏。主要用于测试。
   4. CONFIG\_FILE：配置文件的路径
   5. BACKGROUND\_PATH：启动画面的背景图片路径
   6. BANDIT\_NAME~ WOODY\_NAME：英雄的名字
   7. 枚举tagDirecton：
      1. directon\_stay：原地不动
      2. directon\_right：向右
      3. directon\_up：向上
      4. directon\_left：向左
      5. directon\_down：向下
   8. 枚举HERO\_ID：英雄ID
   9. 枚举：SCENE\_ID，场景ID
   10. 枚举：HERO\_STATE，英雄状态
       1. INVALID\_STATE：无效状态
       2. STATIC\_STATE：静止状态
       3. RUNNING\_STATE：移动状态
       4. ATTACK\_STATE：攻击状态
       5. MAGICATTACK\_STATE：魔法攻击状态
       6. DEATH\_STATE：死亡状态
   11. 结构体AI\_Probability——AI中使用的各种概率：
       1. static\_probability：静止状态的概率
       2. move\_probability：来回移动的概率
       3. attack\_probability：物理攻击的概率
       4. magic\_probability：魔法攻击的概率
   12. GlobalValue类：一些全局可以访问的配置信息以及访问函数
   13. 宏DEFINE\_ANIMATION\_WITH\_CALLBACK，用于创建带回调函数的动画，动画播放完之后会调用该回调函数，不建议使用这个宏！
   14. 宏DEFINE\_ANIMATION\_WITHOUT\_CALLBACK，用于创建不带回调函数的动画，不建议使用这个宏！
   15. 宏DEFINE\_MOVE\_MAGIC，创建一个魔法动画，不建议使用这个宏！
2. 英雄基类Hero
   1. 相关文件：Hero.h Hero.cpp
   2. 增加英雄的时候需要实现一个Hero的子类，并实现相关的虚函数！
   3. 类函数：
      1. InitHero：
         1. 根据英雄的ID，英雄的站位（在左边还是右边），英雄的面部朝向，是否使用AI来初始化英雄
      2. Run ：
         1. 根据方向来移动英雄，并播放相应的移动动画
      3. RunEnd：
         1. 英雄移动结束后调用的函数（可以将该函数调整为虚函数，以方便以后的子类进行自己的控制）
      4. StopAnimation
         1. 停止英雄所有的动作，并根据英雄的面部朝向显示静态的英雄图像
      5. GetCurrentDirection
         1. 返回英雄当前的移动方向
      6. GetPreDirection
         1. 返回英雄上一个移动的方向
      7. GetFaceDirection
         1. 返回英雄的面部朝向
      8. GetHeroDirection
         1. 返回英雄的站位（是屏幕的左边还是右边），这个函数不常用，考虑下可以删除
      9. SetFaceDirection
         1. 设置英雄的面部朝向
      10. SetBloodValue
          1. 设置英雄的血量
      11. GetBloodValue
          1. 返回英雄的血量
      12. SetMagicValue
          1. 设置英雄的魔法量
      13. GetMagicValue
          1. 返回英雄的魔法量
      14. SetMagicDistance
          1. 设置魔法的有效距离（即技能可以放多远）
      15. GetMagicDistance
          1. 获取魔法的有效距离
      16. UseAI
          1. 开启/关闭AI
      17. GetHeroID
          1. 获取英雄的ID
      18. GetHeroName
          1. 获取英雄名字
      19. GetHeroState
          1. 获取英雄的状态
      20. SetHeroState
          1. 设置英雄的状态
      21. Die
          1. 英雄死亡的处理函数
      22. Attack
          1. 英雄进行物理攻击的处理函数
      23. AttackEnd
          1. 攻击结束后的处理函数
      24. MagicAttack
          1. 英雄进行魔法攻击的处理函数
      25. MagicAttackEnd
          1. 魔法攻击结束后的处理函数
      26. Parry
          1. 英雄被攻击时的格挡函数
      27. GetSequense
          1. 英雄移动、攻击、魔法攻击都是通过播放动画来实现的，该函数就是获取当前的运动序列
      28. CreateHeadButton
          1. 创建英雄的大头像
      29. CreateBloodView
          1. 创建血条
      30. CreateMagicView
          1. 创建魔法条
      31. ====子类需要实现的虚函数 begin（具体如何实现可以参照Bandit类）====
      32. SetAllAnimationFile
          1. 设置英雄所需的所有资源文件：包括，头像文件、移动动画的文件、攻击动画的文件等等
      33. SetFacetoLeftStaticAnimation
          1. 创建英雄面向左边静止时的动画
      34. SetFacetoRightStaticAnimation
          1. 创建英雄面向右边静止时的动画
      35. SetFacetoLeftMoveAnimation
          1. 创建英雄面向左边移动时的动画
      36. SetFacetoRightMoveAnimation
          1. 创建英雄面向右边移动时的动画
      37. SetFacetoLeftAttackAnimation
          1. 创建英雄面向左边攻击时的动画
      38. SetFacetoRightAttackAnimation
          1. 创建英雄面向右边攻击时的动画
      39. SetFacetoLeftMagicAttackAnimation
          1. 创建英雄面向左边魔法攻击时的动画
      40. SetFacetoRightMagicAttackAnimation
          1. 创建英雄面向右边魔法攻击时的动画
      41. SetFacetoLeftParryAnimation
          1. 创建英雄面向左边格挡时的动画
      42. SetFacetoRightParryAnimation
          1. 创建英雄面向右边格挡时的动画
      43. SetFacetoLeftBeHitAnimation
          1. 创建英雄面向左边被击中时的动画
      44. SetFacetoRightBeHitAnimation
          1. 创建英雄面向右边被击中时的动画
      45. SetFacetoLeftGetUpAnimation
          1. 创建英雄面向左边从地上站起来的动画（被击倒后应该有这样的动作）
      46. SetFacetoRightGetUpAnimation
          1. 创建英雄面向右边从地上站起来的动画
      47. SetFacetoLeftDieAnimation
          1. 创建英雄面向左边死亡时的动画
      48. SetFacetoRightDieAnimation
          1. 创建英雄面向右边死亡时的动画
      49. =======子类需要实现的虚函数 end==========
   4. 类属性：
      1. m\_pSprite，英雄对应的精灵
      2. m\_pHeadButton，英雄头像按钮
      3. m\_pLifebloodProgress，血条
      4. m\_pMagicProgress，魔法条
      5. m\_pSequence，英雄的动作序列
      6. m\_nPreDirection，英雄前一个运动的方向
      7. m\_nCurrentDirection，英雄当前的运动的方向
      8. m\_nFaceTo，英雄的面部朝向
      9. m\_strHeroName，英雄名字
      10. m\_HeroID，英雄ID
      11. m\_fBlood，英雄的血量
      12. m\_bUseAI，是否具有AI
      13. m\_fMagic，英雄的魔法量
      14. m\_nMagicDistance，有效的魔法施放距离
      15. m\_State，英雄的状态
      16. ====下面的属性需要在子类中直接设置 begin=========
      17. m\_strHeroBigHeadFileName，大头像的文件
      18. m\_strHeroSmallHeadFileName，小头像的文件
      19. m\_strFacetoLeftStaticFileName，面向左边静止的图像的文件
      20. m\_strFacetoRightStaticFileName，面向右边静止的图像的文件
      21. m\_strFacetoLeftMoveFileName，面向左边移动的动画的文件
      22. m\_strFacetoRightMoveFileName，面向右边移动的动画的文件
      23. m\_strFacetoLeftAttackFileName，面向左边攻击的动画的文件
      24. m\_strFacetoRightAttackFileName，面向右边攻击的动画的文件
      25. m\_strFacetoLeftMagicAttackFileName，面向左边魔法攻击的动画的文件
      26. m\_strFacetoRightMagicAttackFileName，面向右边魔法攻击的动画的文件
      27. m\_strFacetoLeftParryFileName，面向左边格挡的动画的文件
      28. m\_strFacetoRightParryFileName，面向右边格挡的动画的文件
      29. m\_strFacetoLeftBeHitFileName，面向左边被攻击的动画的文件
      30. m\_strFacetoRightBeHitFileName，面向右边被攻击的动画的文件
      31. m\_strFacetoLeftGetUpFileName，面向左边被击倒后站起来的动画的文件
      32. m\_strFacetoRightGetUpFileName，面向右边被击倒后站起来的动画的文件
      33. m\_strFacetoLeftDieFileName，面向左边死亡的动画的文件
      34. m\_strFacetoRightDieFileName，面向右边死亡的动画的文件
      35. ====下面的属性需要在子类中直接设置 end==========
3. 游戏逻辑类GameLogic（控制整个游戏的逻辑）
   1. 相关文件GameLogic.h GameLogic.cpp
   2. 类函数：
      1. StartGame (MainScene\* s)，根据主场景指针初始化游戏逻辑
      2. SuspendGame，暂停游戏（未实现）
      3. StopGame，停止游戏
      4. OpponentStopGame，游戏进行中对方忽然离开游戏的处理函数
      5. Clean，游戏逻辑清理
      6. GetMainScene，获取主场景指针
      7. ConnectToServer，连接到服务器（联网模式下使用）
      8. GetNetworkModule，获取网络模块（联网模式下使用）
      9. IsGameRunning，判断/设置游戏是否在进行
      10. JumpToScene，场景之间的跳转
      11. GetPlayer，获取玩家
      12. LoadConfig，加载配置文件
      13. ExitApp,退出程序（目前没有用到）
      14. SetSuspendState，设置游戏的暂停状态（在游戏的进行中，有三种暂停状态：未暂停，对方暂停，自己暂停）
      15. GetSuspendState，获取游戏的暂停状态
      16. GetGameType，获取游戏类型（单机游戏，创建的游戏，加入的游戏）
      17. SetGameType，设置游戏类型
   3. 属性：
      1. m\_pMainScene，主场景，游戏主要在主场景进行，所以要一个指向主场景的指针
      2. m\_pGameLogicInstance，游戏逻辑类是个单件类，这是它唯一的对象
      3. m\_bInit，用于判断逻辑是否已经初始化完毕
      4. m\_pNetwork，网络模块的指针
      5. m\_bIsGameRunning，游戏是否在进行
      6. m\_Player，玩家信息类
      7. m\_nSuspendState，暂停状态
4. 游戏AI类：GameAI
   1. 相关文件：GameAI.h GameAI.cpp
   2. 类函数：
      1. MainProcess，AI的主要处理过程
      2. Run，具有AI的英雄自动来回移动
      3. Action，AI的行为
      4. CheckAndAdjustPosition，检测边界
      5. Follow，跟随对方移动
   3. 类属性：
      1. m\_pHero，AI对象所属的英雄
5. 魔法类Magic
   1. 相关文件：Magic.h Magic.cpp
   2. 类函数：
      1. PlayMagic，放魔法
      2. IsValid，该魔法是否还有效
      3. BelongTo，该魔法属于哪个英雄
      4. PlayOver，魔法施放完成调用的函数
   3. 类属性：
      1. m\_pMagic，魔法的动作序列
      2. m\_bIsValid，魔法是否有效
      3. m\_Direction，魔法的方向
      4. m\_nHeroID，魔法所属的英雄的ID
6. 英雄信息HeroInfoMap
   1. 相关文件：HeroManager.h HeroManager.cpp
   2. 作用：存放英雄的信息，与Hero不同，Hero里面的信息必须要创建一个Hero对象之后才能访问，如果不创建一个Hero对象那么就得不到一个英雄的ID和名字，这个结构体的作用就是专门存放英雄的ID和名字，这样得到一个英雄的ID和名称（例如在玩家在选择英雄的时候，我们并不需要创建每一个英雄，而只需要从这个结构体数组中获取信息即可）就可以不用创建一个Hero对象。该结构体有一个全局的数组G\_HeroInfo
   3. 类函数：无
   4. 类属性：英雄ID、英雄名字
7. 英雄管理器HeroManager
   1. 相关文件：HeroManager.h HeroManager.cpp
   2. 类函数：
      1. GetHeroIDByName，静态，根据英雄的名字获取英雄ID（与英雄是否已经创建无关，只和游戏定义的英雄有关）
      2. GetHeroNameByID，静态，根据英雄的ID获取英雄的名字（同上）
      3. GetHeroDefineCount，静态，获取游戏创建的英雄的个数（同上）
      4. Create，静态工厂类，专门用于创建，新增英雄的时候，需要在这个函数里添加自己的实现
      5. InitHero，根据玩家的选择初始化英雄
      6. SendHeroStateToServer，发送玩家的英雄状态给服务器
      7. UpdateHeroOnce，从服务器上获取对手的英雄人物的状态然后更新
      8. GetHero，根据英雄ID获取英雄对象的指针
      9. SetCurrentHero，设置当前玩家选择的英雄的ID，以方便以后创建
      10. SetCurrentEnemy，设置对手选择的英雄的ID,以方便以后创建
      11. GetCurrentHero，获取当前玩家选择的英雄的对象的指针
      12. GetCurrentEnemy，获取对手选择的英雄的对象的指针
      13. HeroMove，根据英雄ID和方向控制英雄的移动
      14. HeroMagicAttack，英雄施放技能
      15. HeroAttack，英雄攻击（物理攻击）
      16. GetHeroList，获取英雄对象的指针的列表（一般该列表只存在两个对象，即玩家选择的英雄和对手选择的英雄）
      17. Clean
   3. 类属性：
      1. m\_HeroVector，英雄对象指针的列表
      2. m\_pCurrentHero，玩家当前的英雄
      3. m\_pCurrentEnemy，对方选择的英雄
      4. m\_CurrentHeroID，当前选择的英雄的ID（该ID不依赖于英雄人物是否创建，通常游戏进行中的那个英雄人物是根据这个ID来创建的）
      5. m\_CurrentEnemyID，当前对手选择的英雄的ID
      6. m\_pGameAI，游戏AI
8. 魔法管理器MagicManager
   1. 相关文件：MagicManager.h MagicManager.cpp
   2. 类函数：
      1. MagicCollisionDetection，技能与英雄人物之间的碰撞检测
      2. InsertMagic，将每一个英雄的任何一个已经施放的、但是还未失效的技能（或者叫魔法）加入到列表中，以方便碰撞检测
      3. UpdateMagic，更新魔法的位置信息（由计时器定时调用）
      4. Clean，清理函数
   3. 类属性：
      1. m\_MagicVector，技能列表
9. 网络模块类NetworkMoudle
   1. 相关文件：NetworkMoudle.h NetworkMoudle.cpp
   2. 类函数：
      1. Connect，连接服务器
      2. ReadPackageHead，读取一个包头部
      3. ReadControlPackBody，读取控制包数据部分
      4. ReadDataPackBody，读取数据包数据部分
      5. ReadControlPack，读取控制包（包括包头部）
      6. SendControlPack，发送数据包
      7. SendDataPack，发送数据包
      8. ReadDataPack，读取数据包（包括包头部）
      9. Login（登陆）
      10. Logout（注销）
      11. Leave（断开连接）
      12. CreateGame（创建一局游戏）
      13. JoinGame（加入一局游戏）
      14. RequestAllGameInfo（请求所有处于等待状态的游戏的信息）
      15. GetGameRound（获取游戏信息列表）
      16. GetFirstGame（获取可用的第一局游戏）
   3. 类属性：
      1. m\_nServerPort，服务器端口号
      2. m\_strIP，服务器IP
      3. m\_PlatformInitSocket，平台相关的套接字初始化对象，用于初始化套接字，只需定义这个对象几颗，不用对它进行其他处理
      4. m\_GameRoundList，游戏信息列表
10. 套接字基类（服务器端）SelectSocket
    1. 相关文件：SelectSocket.h
    2. 类函数：
       1. CreateServer，创建服务器
       2. ServerLoop，服务器的主循环
       3. Close，关闭套接字
       4. HandleAccpet，虚函数，处理客户端接入
       5. HandleRead，虚函数，处理客户端发送来的数据
       6. HandleClose，虚函数，处理客户端关闭
    3. 类属性：
       1. m\_readSet，套接字的读集合
       2. m\_serverSocketfd，服务器套接字
       3. m\_Addr，地址
       4. m\_nMaxSocket，用于记录当前最大的套接字的值，在Linux下的select模式需要这个属性
11. 套接字基类（客户端）SelectSocketClient
    1. 相关文件：SelectSocket.h
    2. 类函数：
       1. ConnectToServer，连接到服务器
       2. Read，没有设置超时的读取（会阻塞）
       3. TryRead，设置类超时的读取（达到一定时间后会超时返回，而不会一直阻塞下去）
       4. Write，向服务器发送数据
       5. Close，关闭套接字
    3. 类属性：
       1. m\_serverSocketfd，套接字
       2. m\_Addr，地址
       3. m\_nMaxSocket，最大的套接字的数量（select模式下会用到）
12. 游戏类（表示一局游戏）GameRound
    1. 相关文件：GameRound.h GameRound.cpp
    2. 作用：只用于服务器端，并不用于客户端
    3. 类函数：主要是一些访问器函数，可以自行查看
    4. 类属性：
       1. m\_pCreatePlayer，创建游戏的玩家
       2. m\_pJoinPlayer，加入游戏的玩家
       3. m\_nGameState，游戏状态
       4. m\_strGameName，游戏的名称
       5. m\_nGameIndex，游戏的索引，唯一标识游戏
13. 游戏信息类GameRoundInfo
    1. 相关文件：GameRound.h GameRound.cpp
    2. 作用：用于服务器与客户端之间的信息传输，因为玩家加入游戏的时候需要从服务器中获取游戏信息。因此该类的主要是用于信息交换。主要用在客户端
    3. 类属性：
       1. m\_nGameState，游戏状态
       2. m\_strGameName，游戏名字
       3. m\_nGameIndex，游戏索引
       4. m\_strCreateUserID，创建者ID
       5. m\_nCreateHeroID，创建者的英雄的ID
14. 控制包类型（代码）PackageCode和控制包类型对应的信息CodeMsg
    1. 相关文件：CodeMsg.h
    2. 作用：PackageCode代表控制包的类型（如：注册、登陆，注销及其的回应码等等），CodeMsg则包含了PackageCode和PackageCode代表的意义，有一个全局的CodeMsg数组，可以根据需要往里面添加相应的代码和对应的信息。
15. 玩家类Player
    1. 相关文件：Player.h Player.cpp
    2. 说明：玩家这一概念只在联网模式下才有！单机模式下不需要玩家！
    3. 类函数：一些访问器函数，可以自行查看
    4. 类属性：
       1. m\_strUserID，玩家ID
       2. m\_strPassword，密码
       3. m\_strUserName，用户名
       4. m\_nHeroID，玩家选择的英雄的ID
       5. m\_nUserState，玩家状态（目前没有用到）
       6. m\_bIsCreator，是否是游戏的创建者
       7. m\_bLogin，是否已经登陆（可以用玩家状态来取代）
16. 工具类Util
    1. 相关文件：Util.h Util.cpp
    2. 类函数（都是静态函数）：
       1. Log，日志函数，记录日志，会在程序运行的目录下生成一个以当天日期为名字的日志文件
       2. Spilit，切割函数，用于字符串切割，以指定的字符切割一串字符串，结果放在一个vector中
       3. GetPort，从一个已经初始化的sockaddr\_in中获取端口
       4. GetIPString，从一个已经初始化的sockaddr\_in中获取IP地址
       5. ConvertToStr，将double、int转换为字符串
       6. GetRandomNum(a,b)，获取一个[a,b)之间的整形随机数
       7. GetProbability，获取一个0~1的浮点数
17. 网络序列包类NetAchive
    1. 相关文件NetAchive.h
    2. 类函数：
       1. NetAchive(char\* buf,int len,int mode,char cSpilitChar)，设置序列化/反序列化的缓冲区、缓冲区长度，模式（SAVE表示序列化，LOAD表示反序列化），cSpilitChar分隔符：默认是|，可以自己设置。
       2. Save\_string：保存（即序列化）一串字符串到缓冲区中
       3. Load\_string：从缓冲区中加载（即反序列化）一串字符串
       4. Load\_string(int len):从缓冲区中加载指定长度的一串字符串而不是根据分隔符来分割出一串字符串。
       5. Save\_int：保存一个整形到缓冲区中
       6. Load\_int：从缓冲区中加载一个整形数据
       7. Save\_bool：保存一个布尔型数据到缓冲区中
       8. Load\_bool：从缓冲区中加载一个布尔型数据
       9. Save\_float：保存一个浮点型数据到缓冲区中
       10. Load\_float：从缓冲区中加载一个浮点型数据
       11. GetBufferLength：获取缓冲区长度
       12. Reset：重置
       13. Clean：清空
    3. 类属性：
       1. m\_pBuffer：缓冲区指针
       2. m\_nBufferLen：缓冲区长度
       3. m\_nPosition：缓冲区的下标
18. 包头部类PackageHead
    1. 相关文件：Package.h Package.cpp
    2. 类函数：
       1. Clean，清理
       2. SetVersion，设置版本
       3. GetVersion，获取版本
       4. SetPackageLength，设置包长度（不包括头部）
       5. GetPackageLength，获取包长度（不包括头部）
       6. SetPackageType，设置包类型（控制包或者数据包）
       7. GetPackageType，获取包类型（控制包或者数据包）
       8. GetHeadBuffer，将头部数据转换成可以在网络上传输的字符串
       9. Analyze，将一串字符串解析为包头部数据
    3. 类属性：
       1. m\_szVersion，版本
       2. m\_szRequestCode，请求码（目前没用）
       3. m\_szReturnCode，返回码（目前没用）
       4. m\_szPackageType，包类型（有两种：控制包、数据包）
       5. m\_szPackageLength，包的长度（不包括头部）
       6. m\_szReserved1，保留字1
       7. m\_szReserved2，保留字2
       8. m\_szReserved3，保留字3
19. 控制包ControlPackage——这是一个很杂乱的包，里面存放着各种需要在客户端与服务器间传输的数据
    1. 相关文件：Package.h Package.cpp
    2. 类函数：都是比较简单的访问器函数，可自行查看
    3. 类属性：
       1. m\_PackageHead，包头部
       2. m\_nPackageCode，控制码，服务器和客户端以此来识别需要进行哪一种操作
       3. m\_strUserID，用户ID
       4. m\_strUserPassword，用户密码
       5. m\_strUserName，用户名
       6. m\_nUserHeroID，用户使用的英雄ID
       7. m\_bUserLogin，用户是否已经登陆
       8. m\_bIsCreator，用户是否为游戏创建者
       9. m\_strGameName，游戏名称
       10. m\_nGameRoundIndex，游戏的索引
       11. m\_nGameState，游戏的状态（等待状态、进行中等状态）
       12. m\_nGameRoundWaitCount，处于等待状态的游戏的数量
       13. m\_nOtherInfoLen，其他信息长度
       14. m\_strOtherInfo，其他信息
       15. m\_strPackageBuffer，缓冲区
20. 数据包DataPackage——游戏进行中，客户端之间交互的数据
    1. 相关的文件：Package.h Package.cpp
    2. 类函数：
       1. Clean，清空
       2. SetHeroID，设置英雄ID
       3. GetHeroID，获取英雄ID
       4. SetBlood，设置血量
       5. GetBlood，获取血量
       6. SetMagic，设置魔法量
       7. GetMagic，获取魔法量
       8. SetPosition，设置英雄位置坐标
       9. GetPositionX，获取英雄横坐标
       10. GetPositionY，获取英雄纵坐标
       11. SetState，设置英雄状态
       12. GetState，获取英雄状态
       13. SetFaceTo，设置英雄面部朝向
       14. GetFaceTo，获取英雄面部朝向
       15. GetPackageBuffer，将数据包的数据转换为可以在网络上传输的数据
       16. Analyze，将一串字符串（通常是从网络上读取的）转换成数据包的数据
       17. GetPackageBufferLength，获取数据包的数据的长度（即用于在网络上传输的字符串的长度）
       18. Print，打印
    3. 类属性：
       1. m\_PackageHead，包头部
       2. m\_nHeroID，英雄ID
       3. m\_nHeroState，英雄状态
       4. m\_nFaceTo，英雄面部朝向
       5. m\_nPositionX，英雄横坐标
       6. m\_nPositionY，英雄纵坐标
       7. m\_nBloodValue，英雄的血量
       8. m\_nMagicValue，英雄的魔法值
       9. m\_strPackageBuffer，缓冲区

# GameServer代码

1. 游戏服务器类FightingServer
   1. 相关文件：FightingServer.h
   2. 作用：主要是处理客户端的接入、管客户端关系，发送数据、接收数据等与网络直接相关的事情
   3. 类函数：
      1. HandleAccpet，处理客户端的接入
      2. HandleRead，处理客户端的数据,读取客户端发送过来的数据包（服务器不处理数据包，直接转发）或控制包，如果是控制包，则调用服务器逻辑类进行处理
      3. HandleClose，处理客户端关闭
      4. ReadPackageHead，读取包头部，判断是哪一种类型的包，如果是控制包就进行处理，如果是数据包就进行转发
      5. ReadControlPack，读取控制包的数据放到一个buffer中
      6. ReadDataPackBuffer，读取数据包的数据到一个buffer中
      7. SendDataPackBuffer，转发数据包（因为服务器端并不处理数据包）
      8. SendControlPack，发送控制包
   4. 类属性：
      1. m\_pServerLogic
2. 服务器逻辑类ServerLogic
   1. 相关文件：ServerLogic.h ServerLogic.cpp
   2. 作用：游戏服务器的各种逻辑处理
   3. 类函数：
      1. NewPlayer，玩家接入服务器的时候，创建一个新玩家，并添加到玩家列表中
      2. Leave，玩家离开服务器的时候，应该调用该函数
      3. Analyze，分析控制包，提取出控制包的类型，并调用相应的处理函数
      4. Login，玩家登陆
      5. Logout，玩家注销
      6. RequestGameInfo，请求所有处于等待状态的游戏的信息
      7. CreateGameRound，玩家创建一局游戏
      8. JoinGameRound，玩家加入一局游戏
      9. GetGameRound，根据游戏索引查找一局游戏
      10. GetPlayer，根据套接字查找一个玩家
      11. TransmitDataPackBuffer，转发数据包
      12. Register，玩家注册
      13. IsLogin，判断玩家是否已经登陆
      14. IsUserIDValid，判断用户ID（玩家的用户信息）是否有效
      15. IsUserValid，判断用户ID和密码（玩家的用户信息）是否有效
      16. Respond，对客户端进行回应
      17. GetAllGameInfo，获取服务器中所有处于等待状态的游戏的信息
      18. DumpGameInfo，打印游戏信息
      19. \*\*\*\*\*待添加\*\*\*\*\*\*
   4. 类属性：
      1. m\_pFightingServer，游戏服务器指针
      2. m\_GameRoundList，游戏列表
      3. m\_nGameRoundCount，游戏的数量
      4. m\_PlayerList，玩家列表
3. SQLite数据库类——数据库连接类：SQLiteConnector
   1. 相关文件SQLiteDB.h
   2. 作用：连接数据库
   3. 类函数：
      1. Connect
      2. Close
      3. ErrorMsg
      4. GetHandler
      5. IsConnect
4. SQLite数据库类——SQL语句执行类： SQLiteExcer
   1. 相关文件SQLiteDB.h
   2. 作用：执行SQL语句，如果SQL语句有返回结果，那么将结果放到结果集类中
   3. 类函数：
      1. ExecSQL，执行SQL语句
      2. GetResult，获取结果列表
      3. Next，用于遍历结果集
      4. MoveToFist，将游标移动到结果集的开头
      5. GetRowCount，获取结果集中行的个数
      6. Help，帮助函数
   4. 类属性：
      1. m\_nRowCount，语句执行后，返回结果的行数
      2. m\_ResultList，语句执行后，结果列表（存放执行的结果）
      3. m\_nRowIndex，结果集游标
5. SQLite数据库类——结果集类：SQLiteRow
   1. 相关文件SQLiteDB.h
   2. 作用： 存放SQLiteExcer执行的结果
   3. 类函数：
      1. GetFieldCount，获取列的数量
      2. GetValue，根据列的索引所以对应的值
      3. GetFileldName，获取列名
      4. Add，私有函数，一般由SQLiteExcer调用
   4. 类属性：
      1. m\_nFieldCount，列数量
      2. m\_nCurrentFieldIndex，当前列的索引
      3. m\_NameValueMap，数据存放的得放：列名和数据值结构数组
6. 等等