BigWorld 技术培训

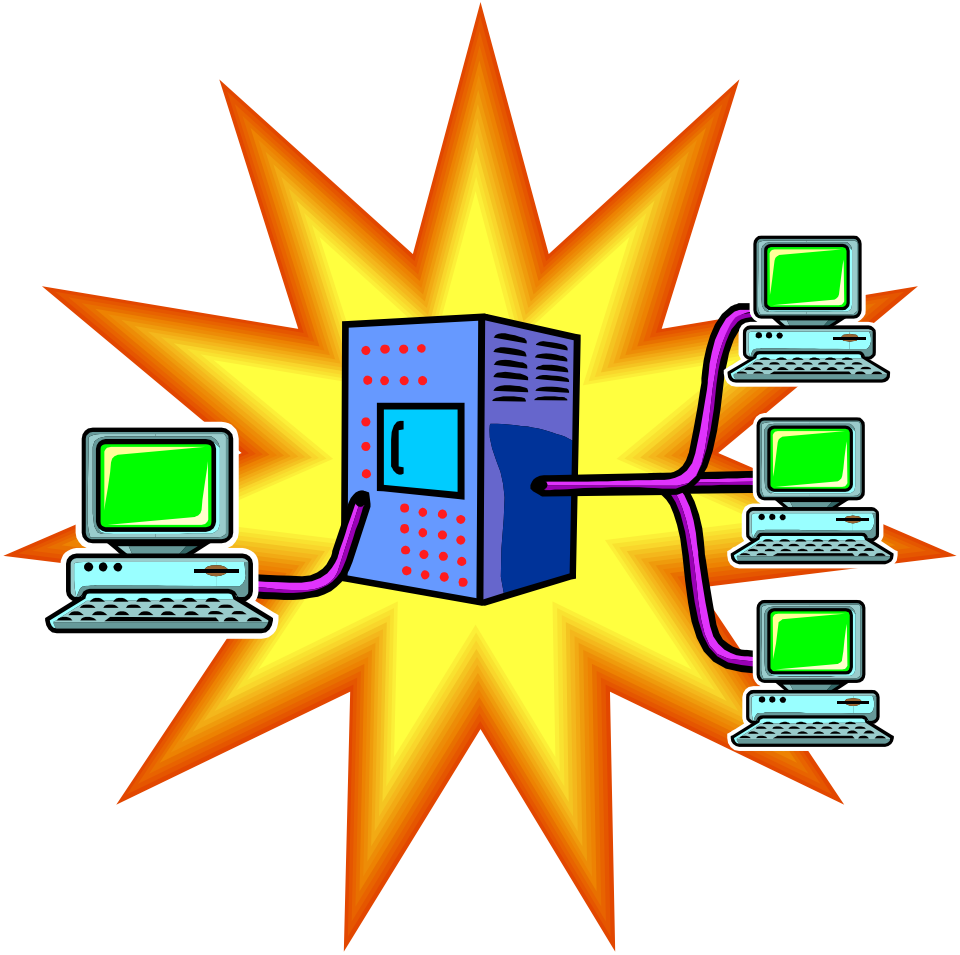
基本概念

概要

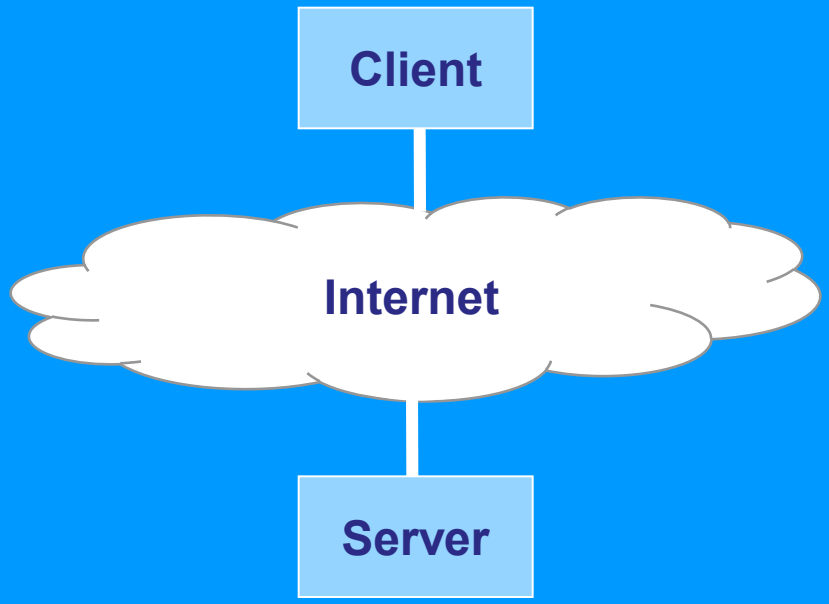
- 系统结构概览
- Entity
- Space
- 游戏资源
- Python API 概览

Session 1

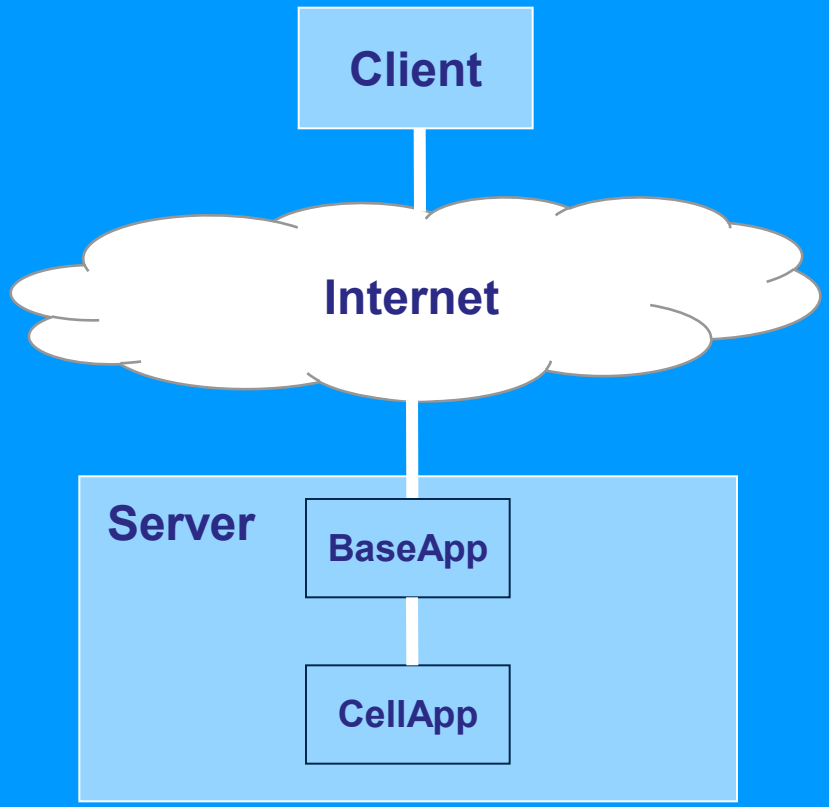
系统结构概要 / Entity



系统结构概要



系统结构概要



系统结构概要

- Client
- BaseApp
  - Client端的固定通信接口
  - 与CellApp间的通信中介
- CellApp
  - 世界的处理
  - 基于世界里的entity的数量的负载平衡机制

Entity

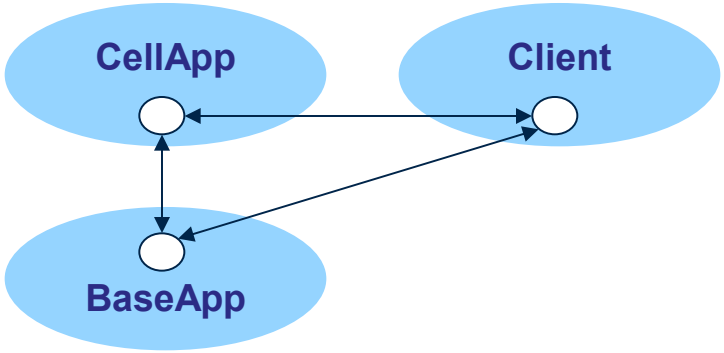
- *Entity是一个游戏对象*

  "一个可以与之交户的东西"

- 示例:
  - Player
  - NPC
  - 门
  - 凳子

Entity

- 拥有整个游戏世界唯一的ID
- 用Python实现
- 发布的对象
- 最多3个的发布点
  - Client
  - BaseApp
  - CellApp



Client Entity

- 为玩家实现可视化
  - 玩家周围的粒子特效
  - 加载和变换模型
  - 模型的动画
- 存贮了需要与可视的entity交互的数据
  - 玩家的背包里的道具
  - 当前装备的武器
  - Player当前的目标

Cell Entity

- 类似于client entity但是更集中于交互
- 存储了其它的周围的entity需要知道我们的数据
  - NPC HP
  - 当前装备的武器
    - 用于决定攻击产生的伤害

Base Entity

- 没有空间的概念
  - 账户
  - 工会管理系统
  - 拍卖行
- 长期的entity的属性
  - 一个玩家的朋友列表
- 用于数据的长期存储以防服务器重启动

Entity 属性

- 表明entity在当前部件的状态
  - 左手的武器
  - 朋友列表
- 可以存在于多个部件
  - 例如: 玩家HP (Client 和 Cell)

Entity属性

- 当被修改时自动被发布
  - 由属性发布标记控制
  - 客户端回调 `set_<property_name>()`
- 发布的工作与AoI配合
  - 仅仅发送周围entity的属性更新
  - 更新的优先度因距离的不同而不同

Entity 方法

- 用于与一个entity交互
  - Entity对自己执行操作
  - Entity之间交互
  - 在部件间执行操作
- 方法是与一个部件相关联的
  - 例如:
    - Base 实现 `addToFriendsList( newFriend )`
    - Cell 实现 `notifyNewFriend( newFriend )`
    - Client 实现 `receiveFriendNotifaction( player,newFriend )`

Entity 方法

- 在其它entity上调用的方法是异步的
  - 没有返回值 (必须要用回调函数)
  - 只有entity是在同一个部件上时才是同步的
    - 不要依赖于这样的假设, 这样会在后期造成扩容的问题.

Entity 方法

- Cell方法只在real entity上被调用
  - Server的培训会有更多详细
- 从cell上，client方法可以被在一个entity的AoI范围内的所有client上调用
  - 例如:
    - Player对世界叫喊 “I am not an animal!”
    - 所有在该player附近的player们收到该文字并显示

Entity定义

- Entity必须有一个定义文件
  - 通常是 'entity def' 或 '.def'
  - <Entity_name>.def
- 象C++头文件或Java接口一样
- XML 文件
- 每个entity类型必须存在于entities.xml

Entity 定义

- 定义:
  - 属性: 它们存在于哪个部件以及怎样被发布
  - 方法: 在每个部件上什么方法被实现了
- 可以从其它的entity定义继承
- 要成功连接服务器，client端的定义文件必须和服务器端的定义文件匹配

Entity 定义

- 示例:

```
<root>
   <Properties>
      <name>
         <Type>         STRING        </Type>
         <Flags>        ALL_CLIENTS   </Flags>
         <Persistent>   true          </Persistent>
      </name>
   </Properties>

   <ClientMethods>
   </ClientMethods>

   <BaseMethods>
   </BaseMethods>

   <CellMethods>
      <setName>
         <Exposed/>
      </setName>
   </CellMethods>
</root>
```

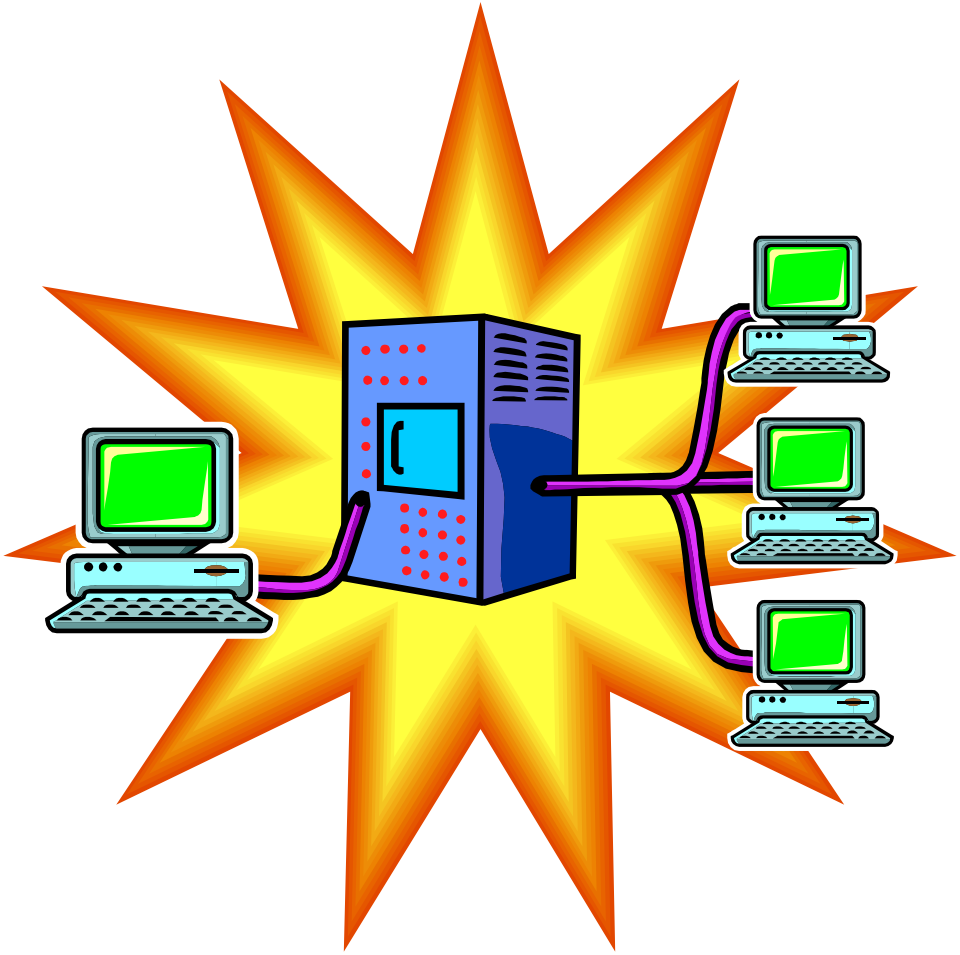
Entity 实现

- 大小写是很重要的
  - 使用的entity名字必须一致
  - .def必须匹配.py实现的类

Entity的文件


<res>
scripts
entities.xml
列出了所有的entity
base
<entity>.py
BaseApp脚本
cell
<entity>.py
CellApp脚本
client
<entity>.py
Client脚本
common
<entity>.py
公用的script
Cell,base,client 相互之间共用的实现函数
server_common
<entity>.py
公用的script
Cell,base 相互之间共用的实现函数
entity_defs
<entity>.def
定义文件
定义entity的属性和方法 (XML)
实现属性和方法 (Python)

Session 2
Space

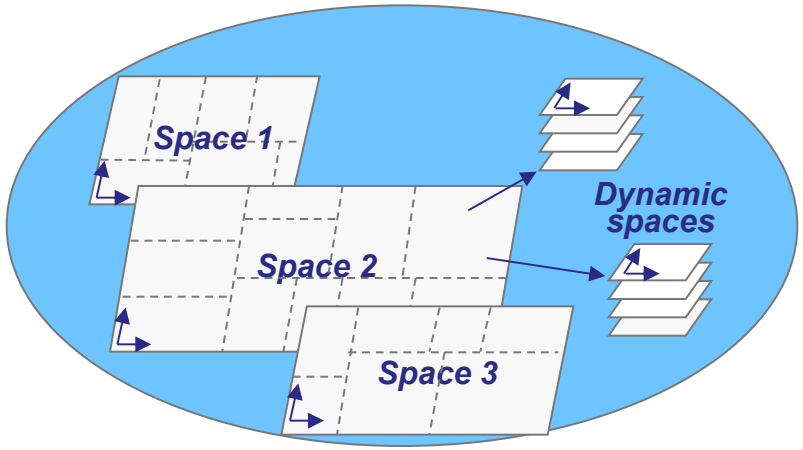


Space

- 一个space是一个player可以看见和交互的世界
- 由下列组成:
  - 地形
  - 静态模型 (房子, 喷泉, ...)
  - NPCs (出生点)
  - NPC 寻路信息
  - 空间的元数据 (User Data Object – 后有细节)
- Space数据可以在多个space间共享

Space

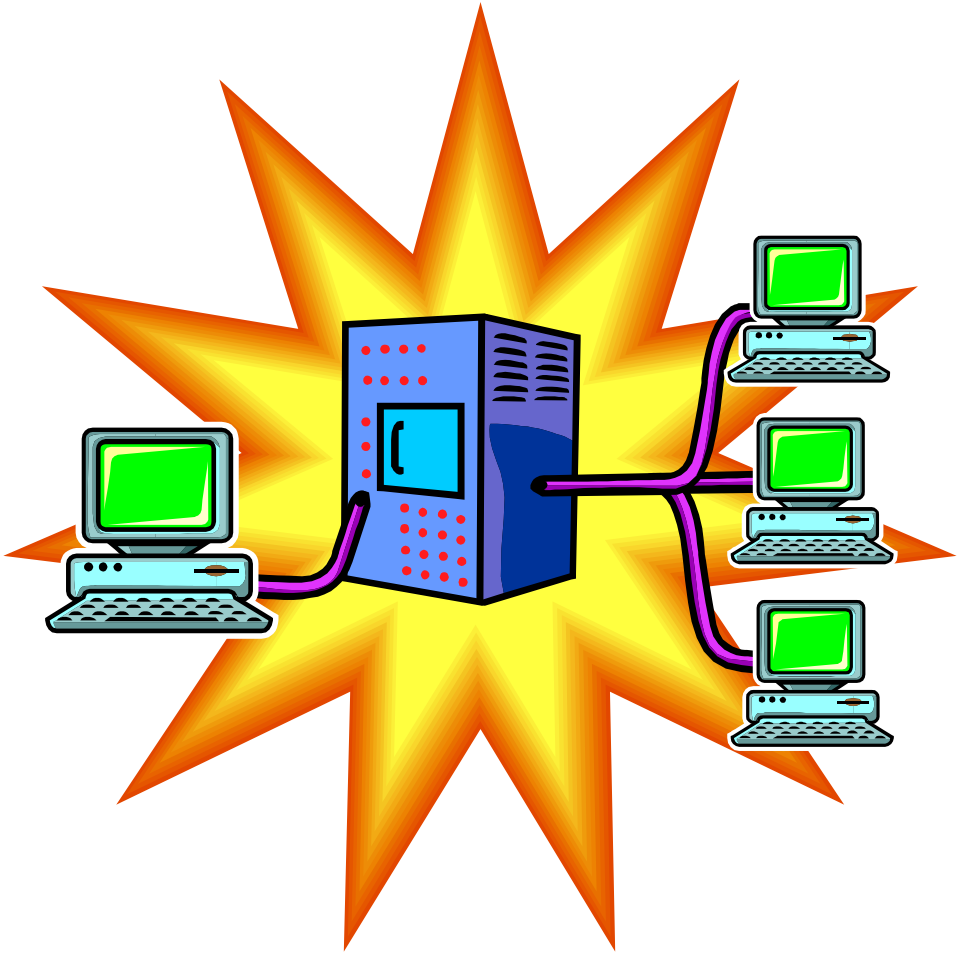
- 不同类型的space应用：
  - 每个server多个独立的大型space
    - 所有玩家存在于的主要的游戏世界
  - 相互隔离的玩家的副本space
    - 地下城副本
    - 玩家自己的房间



Space 空间信息 - Chunks

- 分割成很多chunk单元
  - 100m x 100m 一个单元, x-z 轴对齐
  - 加载的单元
- Chunk根据需要实时加载和卸载
  - Client端: 加载chunk范围为稍微远于far plane的距离
  - Server端: 加载范围+800m的管理的space的边界
  - 卸载比加载稍微缓和以避免频繁地反复加载最近看到的chunk

Session 3
游戏资源



游戏资源

- 每个游戏需要一些资源以使得看起来和操作起来不同
  - 声音
  - 模型
  - 动画
  - 贴图
  - 代码 / 脚本
  - Space 数据

游戏资源

- BigWorld通过定义res目录来访问这些数据
- 当在游戏里访问文件时资源路径是最上层的路径
- 每个游戏项目可以用多个res目录
  - <install_dir>/fantasydemo/res
  - <install_dir>/bigworld/res

Entity的文件


<res>
scripts
entities.xml
列出了所有的entity
base
<entity>.py
BaseApp脚本
cell
<entity>.py
CellApp脚本
client
<entity>.py
Client脚本
common
<entity>.py
公用的script
Cell,base,client 相互之间共用的实现函数
server_common
<entity>.py
公用的script
Cell,base 相互之间共用的实现函数
entity_defs
<entity>.def
定义文件
定义entity的属性和方法 (XML)
实现属性和方法 (Python)

游戏资源

- 多个游戏资源路径会相互覆盖
  - res 路径的顺序是重要的
  - 例如:
    - /tmp/bw/fantasydemo/res
    - /tmp/bw/bigworld/res
    - 都有一个server/test.xml文件
    - BigWorld使用:
      /tmp/bw/fantasydemo/res/server/test.xml

游戏资源

- 资源管理器(ResMgr)
  - 处理繁琐的层级的资源的库
  - 可以读取不同的res目录类型
    - 标准目录
    - 打包的数据 (zip文件)
  - 以C++实现的库
  - 以Python module封装了C++的功能

游戏资源

- 用ResMgr访问文件会返回一个DataSection对象
- DataSection
  - 数据的句柄
  - 数据可以是一个简单的信息
    - 如: 一个字符窜“hello world”
  - 也可以是其它的DataSection的层级树
    - 如: 一个XML文件,其每个子节点都是一个DataSection

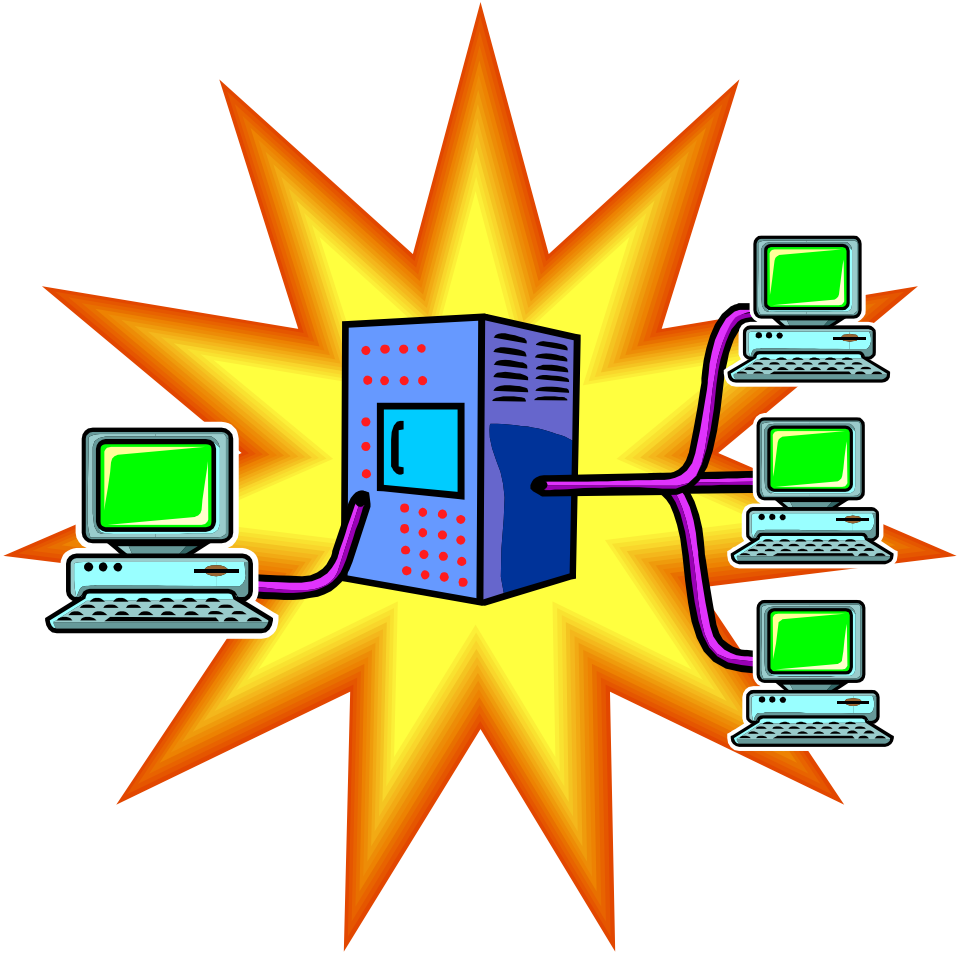
异步的加载资源

- 在游戏运行的同时加载资源时，用不阻塞主线程的方式是很重要的。
- 在客户端，如果同步的加载资源会造成帧率降低同时也会阻塞网络的处理
- 在服务器端，如果同步地加载资源会造成网络阻塞并且会导致处理超时
- 客户端和BaseApp都各自有一个加载资源的线程
- 在游戏初始化时同步加载资源是可以的
- 同样的，在游戏初始话后再import python模块是不推荐的

异步的加载资源

- 在客户端，实现`Entity.prerequisites()`和/或用`BigWorld.loadResourceListBG()`.
- 在BaseApp，用`BigWorld.fetchEntitiesFromChunks()`，`BigWorld.fetchFromChunks()`，或`BigWorld.fetchDataSection()`。 在加载结束时会有回调函数被调用。
- 如果在主线程加载资源，系统会有警告输出

Session 4
BigWorld Python API



bigWORLD™ TECHNOLOGY

Python API概览

- Client / Server通过Python模块来暴露核心部分的功能
- 在不同的分布平台上的相同名字的模块实现不同的功能

Client	Server	
	BaseApp	CellApp
BigWorld	BigWorld	BigWorld
GUI		
Math	Math	Math
Pixie		
ResMgr	ResMgr	ResMgr
<BWPersonality>	<BWPersonality>	<BWPersonality>

Python API 概览

- 简单的client端的示例:

```
import BigWorld

…

class Avatar( BigWorld.Entity ):

    …

    deathWarp = BigWorld.Model( “models/fx/deathWarp.model” )

    self.addModel( deathWarp )
```

Python API – Client模块

- BigWorld
  - 世界相关的客户端接口
  - 许多子模块来处理:
    - 动画
    - Entity交互
    - 移动物理
    - ..等等

Python API – Client模块

- <BWPersonality>
  - 不是import
  - Personality名字定义在游戏配置文件里
  - 包括事件的回调函数:
    - 接口 (改变窗口大小)
    - 输入 (键盘 / 鼠标)
    - 游戏时间改变

Python API – Client模块

- GUI
  - 用于创建游戏界面接口 (菜单..等等)
  - 处理从personality脚本传来的输入事件并导向相应的子组件
- Math
  - 用C++实现的以下部分的接口:
    - Vector (2/3/4)
    - Matricies
    - … 以及对之的设置

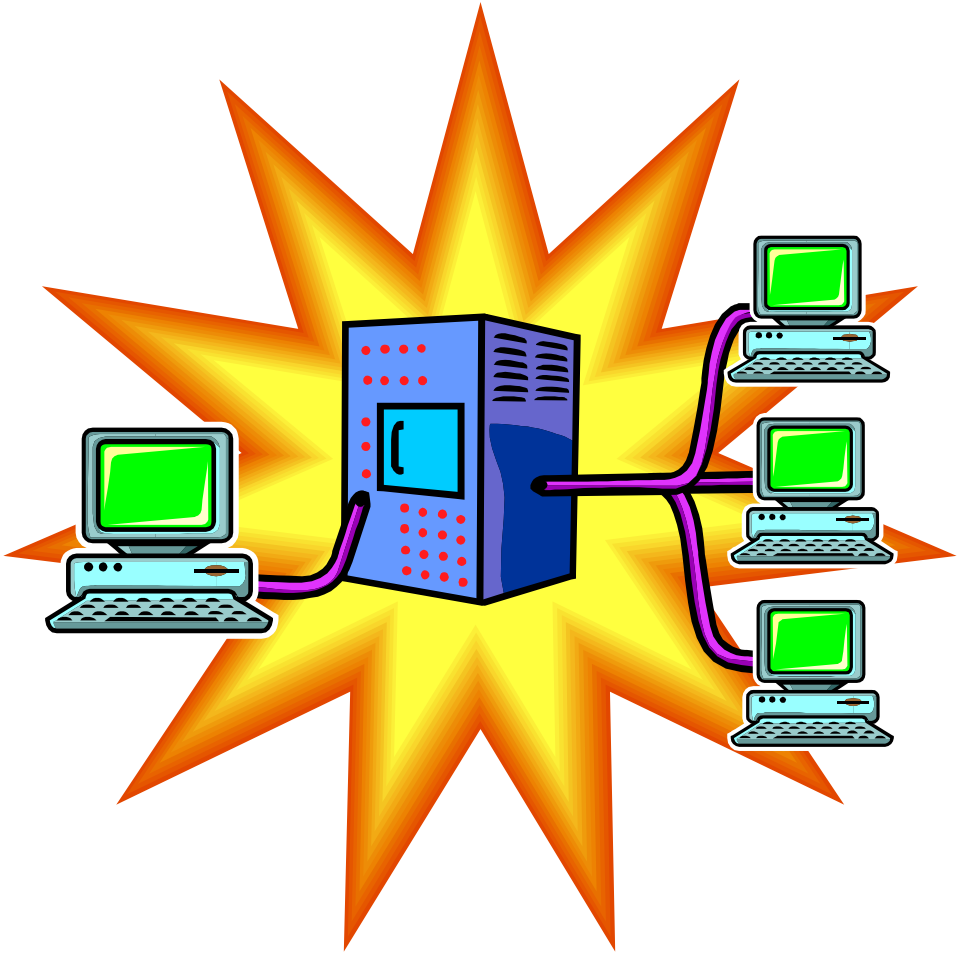
Python API – Client模块

- Pixie
  - BigWorld粒子系统的接口
  - 用于创建短期的粒子特效
    - 一个发动机的火花
    - 玩家施法的闪电
- ResMgr
  - 访问游戏资源的接口
  - 读 / 写 / 删除 / 判定是否存在

Python API – Server模块

- Math / ResMgr
  - 同client端相同
- <BWPersonality>
  - 服务器处理/ 事件通知
- BigWorld
  - CellApp
    - 基于Entity / World的服务器端的交互
  - BaseApp
    - 游戏范围的entity的创建
    - 玩家账号管理/交互

Session 5
怎样有效的工作



有效的工作

- 研发一个游戏需要许多人共同工作
- 使用版本控制
  - SVN / Perforce… 等
- 美术
  - 用bwlockd来管理世界的编辑

有效的工作

- Windows用户mount资源到server机器
  - 每个开发者 (甚至美术人员) 应该在Linux server上有他们自己的账户以方便测试
  - 把你的游戏资源从Windows上共享
  - Mount目录到你的Linux用户帐号下
  - 启动一个server来测试你的修改
- 在整个办公室内运行一个最新版本的server最为测试点
  - 对于美术特别有用

有效的工作

- Entity脚本开发
  - 用client Python console (CAPS-P)
  - 连接一个Python console到一个正在运行的 server (server培训更多讲解)
- 要有扩容(scaling)计划!
  - 考虑你的游戏预计的容量
  - 预先计划怎样实现之
  - 当你实现milestone时同时测试scaling