



BigWorld 技术培训

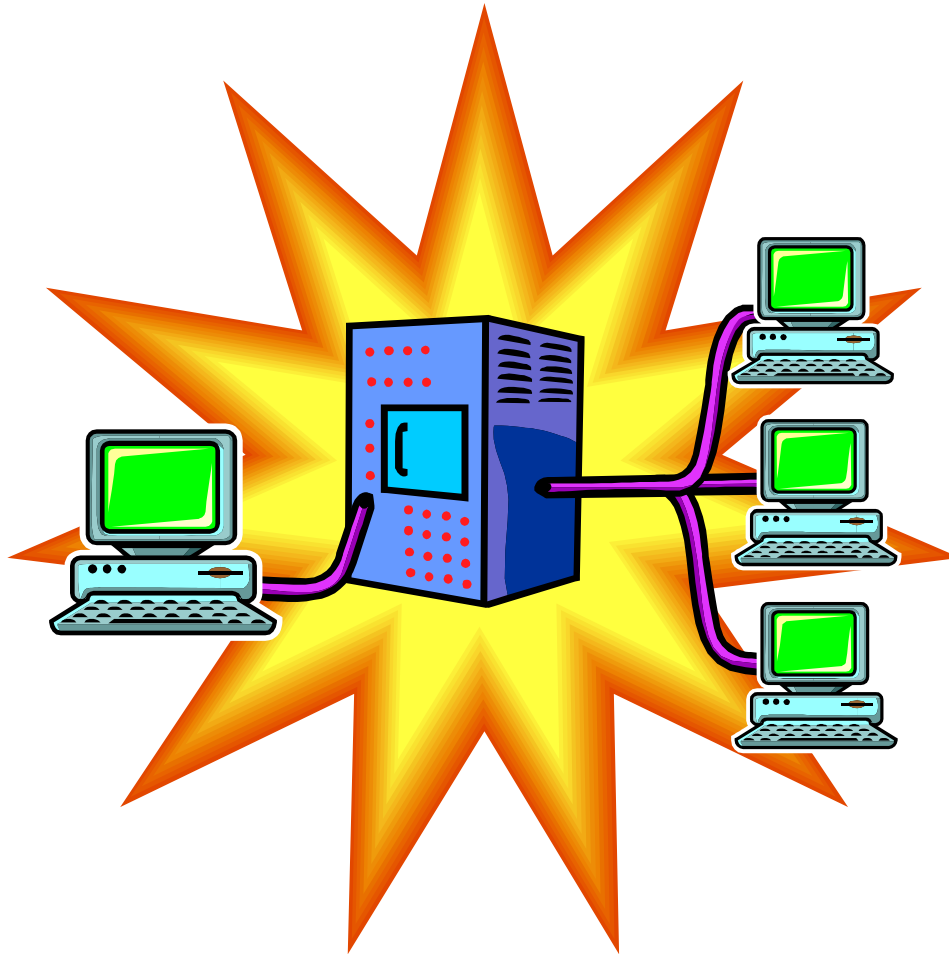
服务器端

概要

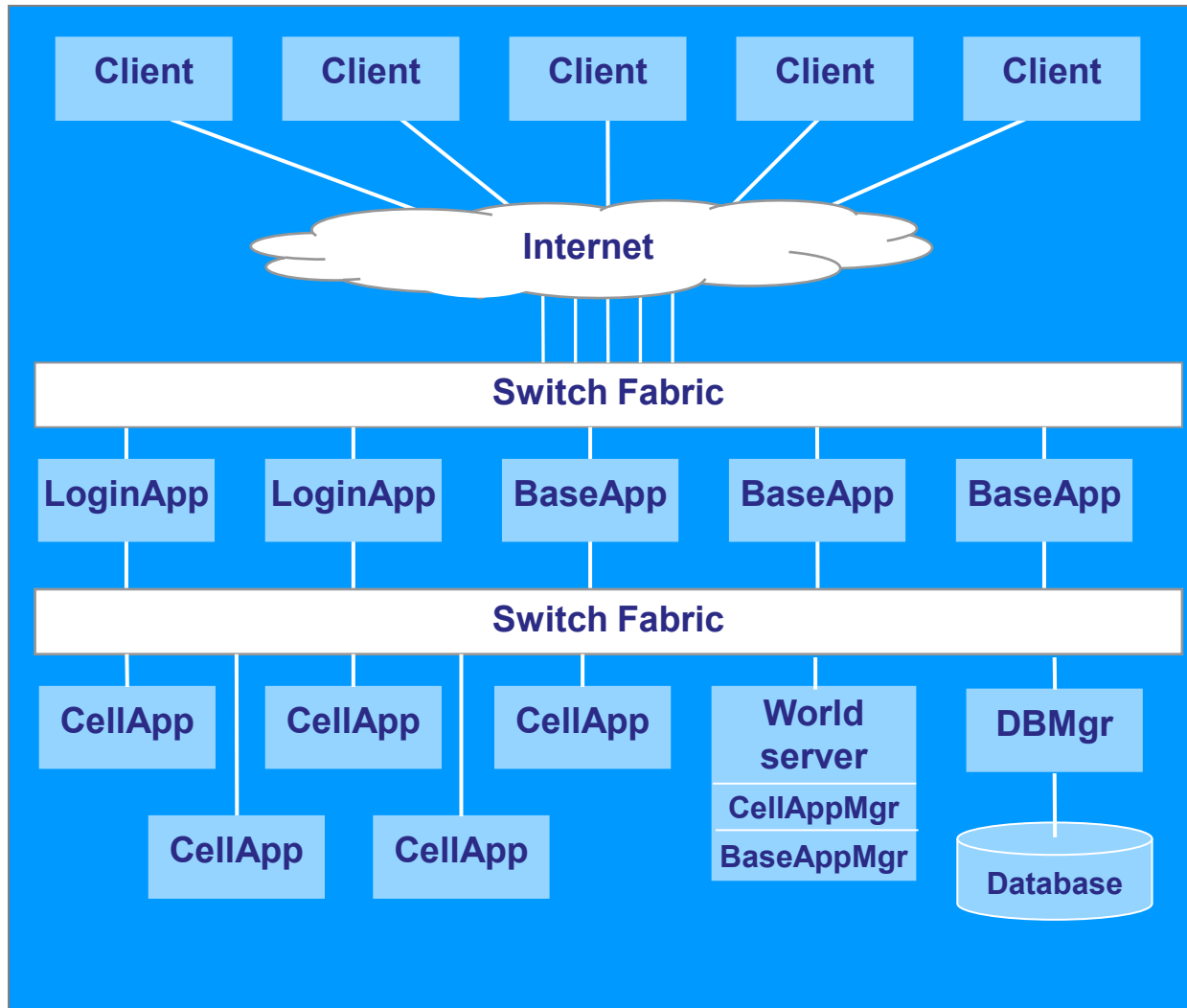
- BigWorld 服务器概览
- 实现一个entity
- Entity通信
- Entity核心部分
- Cell 功能集
- 服务器设置和维护
- 服务器性能分析和压力测试

Session 1

BigWorld 服务器概要



BigWorld 服务器



LoginApp

- 与客户端的第一个连接点
- 固定的端口
- 初始通信时加密
 - 公用密钥对（任意长度的密钥）
 - 基于用户名 / 密码的安全机制
- 使用多个LoginApp来进行负载均衡
 - DNS轮流调度

BaseApp

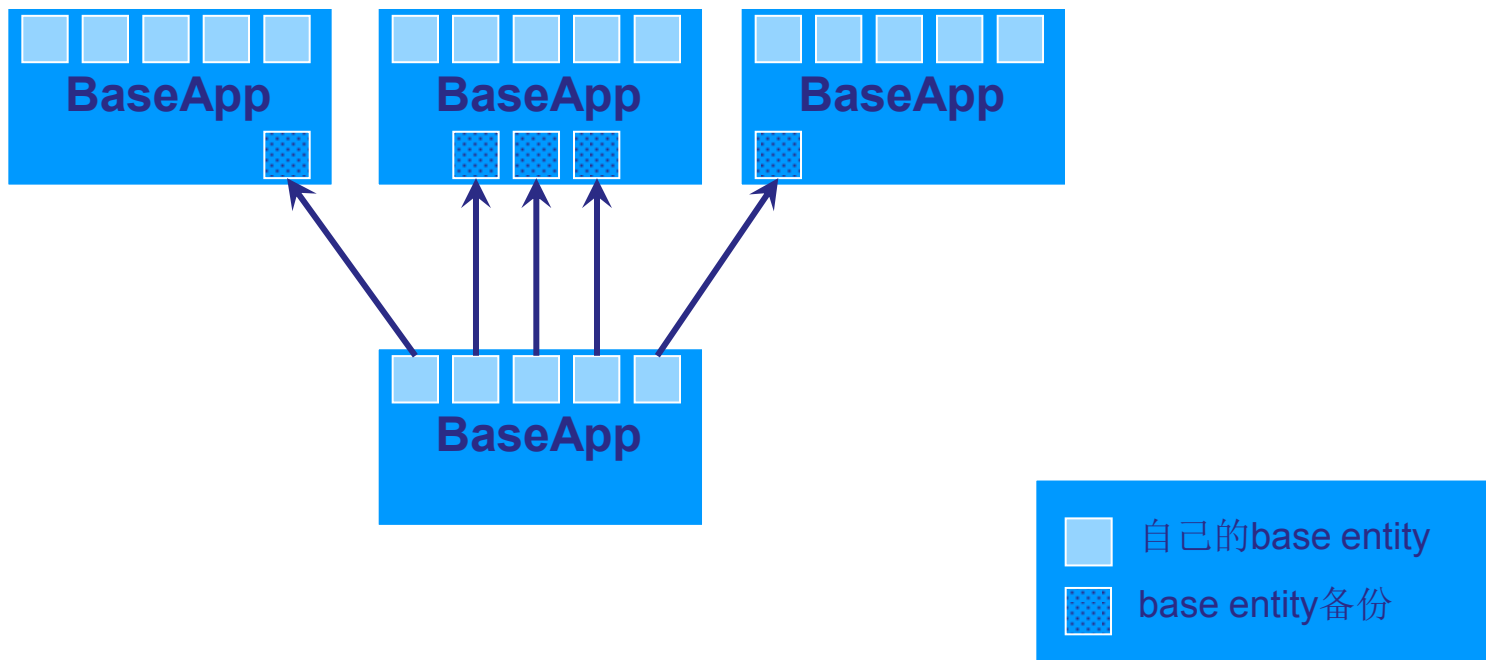
- 与客户端通信的固定点
- 客户端与CellApp通信的中介
- 与客户端连接的负载均衡机制
- 用于处理没有空间属性的entity
 - 拍卖行
 - 公会管理器（Guild Manager）
 - 实例管理器（Instance Manager）
- 为其它的BaseApp容错
- 通常一个CPU / 核上运行一个BaseApp

Base Entity

- 两种base entity
 - Base
 - Proxy
- Base
 - 通常的游戏entity
 - 例如：存储在数据库里的NPC, 拍卖行...
- Proxy
 - 与客户端连接
 - Base的特化

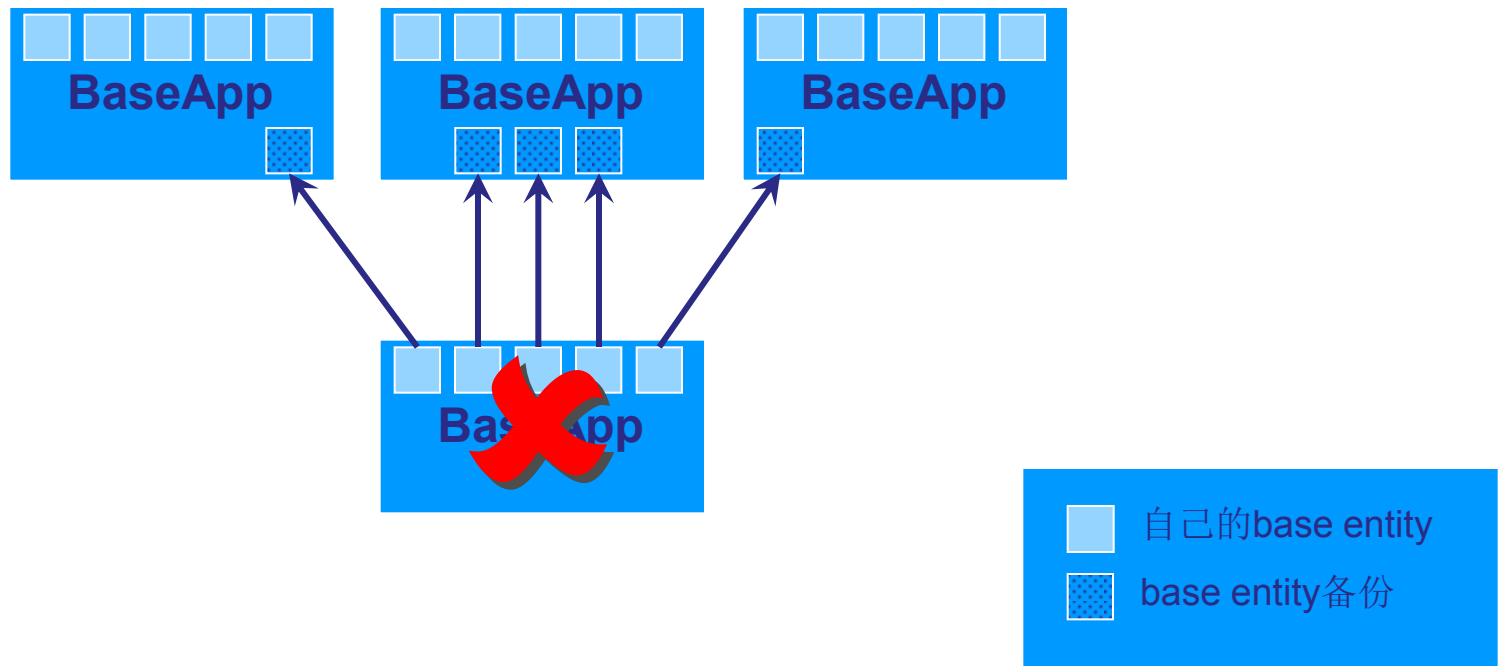
BaseApp 容错

- 备份entity到其它的BaseApp



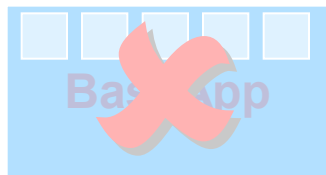
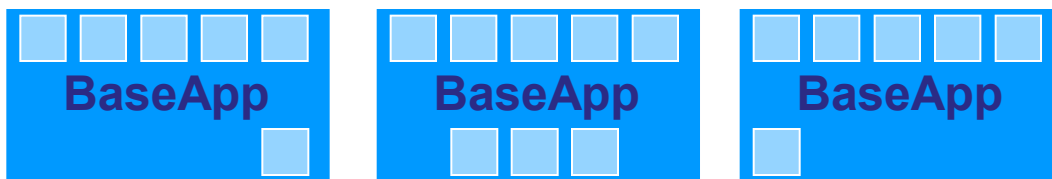
BaseApp 容错



- BaseApp变得不可用



BaseApp容错

- 崩溃的BaseApp上的entity在它们的备份entity上复活



-  自己的base entity
-  base entity备份

BaseApp容错

- 与崩溃的BaseApp连接的客户端会被断开
 - 所有的数据都被保存了
 - 重新建立连接后，它们将继续连接到原来的entity（如果没有超时的话）

BaseApp 管理器 (BaseAppMgr)

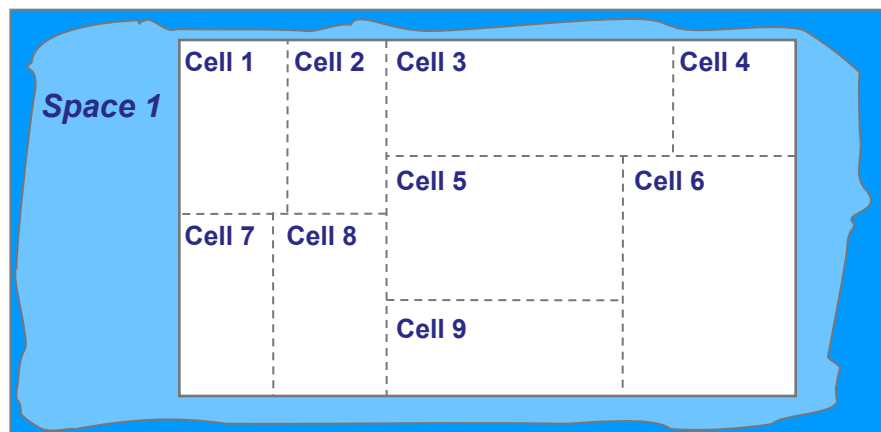
- 负责管理BaseApp间的负载平衡
- 监视所有的BaseApp以实现各个BaseApp之间的容错
- 主要用于玩家登录和创建entity
- 一个服务器群组只需要一个BaseAppMgr实例
- 当崩溃时用Reviver来复活
- 当两个BaseApp崩溃时关闭整个服务器集群
 - 可以自定义，但是不安全

CellApp

- 空间数据的处理
 - 处理玩家所交互的游戏空间（space）
- 处理在space内的entity
- 处理space内的一个区域（cell）
 - 一个CellApp在一个space上只会有一个cell
- 一个CellApp有可能处理多个space
- 通常一个CPU/核上运行一个CellApp

Cells 和 Space

- space通过cell来实现平衡负载
- 每个space至少含有一个cell
- 每个cell处理space的一个区域
- cell的边界根据cell的负载而移动
- cell不影响客户端的游戏体验



CellApp的负载

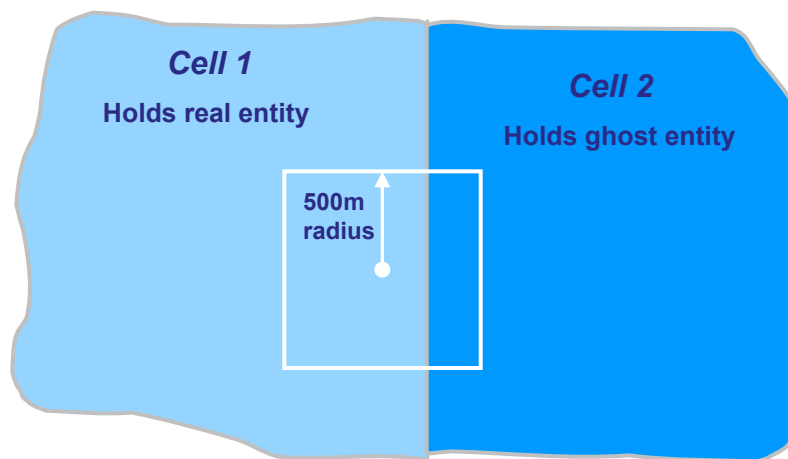
- 所管理**entity**的总数
- **entity**通信的频率
 - 显式的：方法调用
 - 隐式的：属性的更新
 - **entity**的密集度
- **entity**脚本
- **entity**的数据大小

Entity 和 Cell

- 每个space至少有一个entity
 - 初始space例外
- CellApp上每个玩家entity都有一个Witness对象
 - Witness跟踪周围的entity
- entity的兴趣范围（AOI）缺省是500m
 - 可以自定义的，依赖于很多因素

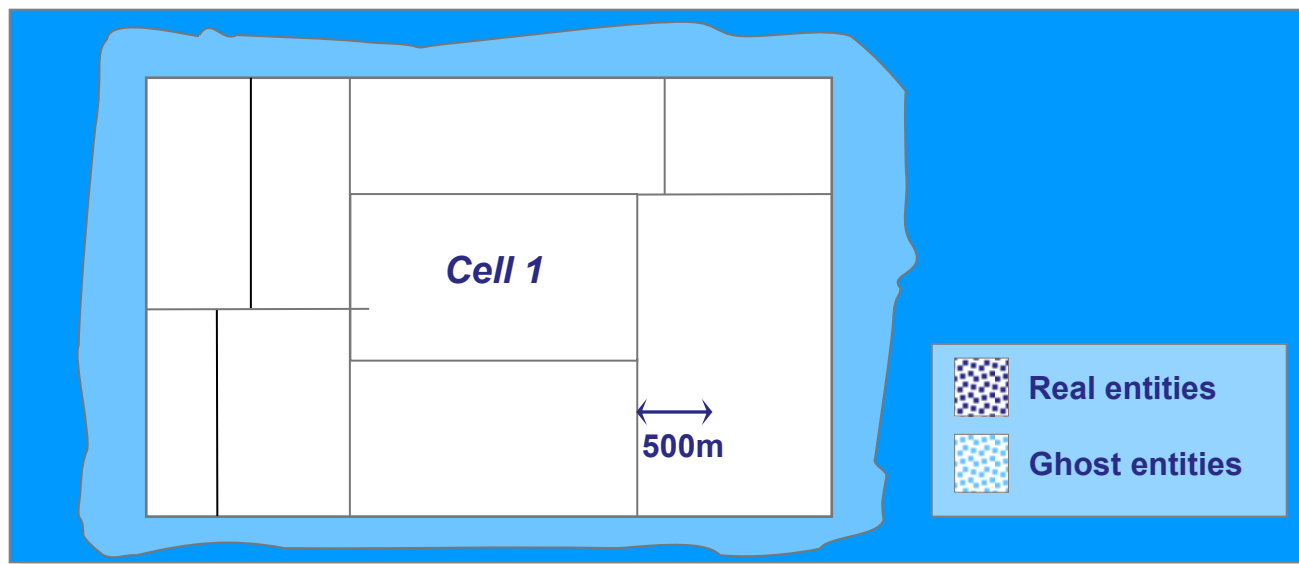
Entity 和 Cell

- **Entity**可以无缝穿越**cell**边界
 - 客户端不知道**cell**的存在
- 每个**cell**维护着一个列表，存放着在其边界外沿的**entity**
 - Ghost entity
 - 500m（和AoI同值）



Entity: Real 和 Ghost

- real entity是权威的entity
- 一个ghost entity是其邻近的cell中与之相对应的real entity中部分数据的拷贝



Ghost entity

- 解决跨越cell边界的entity的交互问题
- 方法调用
 - 转发给其real entity
- 属性
 - 一个属性可以是real only的：
 - 也就是说，将永远不会存在于ghost上
 - 如果一个属性对于客户端是可见的，那么该属性必须在ghost上
 - 当前的武器
 - 装备类型
 - 名字
 - Ghost属性是只读的
 - 要更改属性值只能通过方法调用来更新其对应的real entity

entity的数据更新

- 客户端实现LoD以加速渲染
- CellApp实现LoD以减少：
 - 带宽开销
 - 每个entity使用的CPU
- LoD在CellApp上的作用类似于在客户端的作用
 - 细节程度和它与玩家entity之间的距离相关
- 客户端entity方法可以实现LoD
- entity属性实现LoD可以避免和客户端之间不必要的通信
 - 当前的武器（在距离很远时不可见）

CellApp 管理器 (CellAppMgr)

- **CellAppMgr**知道：
 - 所有的**CellApp**（它们的负载）
 - 所有的**cell**边界
 - 所有**space**
- 管理**CellApp**的负载均衡
 - 告诉**CellApp**它们的**cell**边界应该在哪里
- 把新建的**entity**加入到正确的**cell**上
- 一个服务器集群只有一个**CellAppMgr**实例
- 当崩溃时用**Reviver**复活
 - 服务器能够继续运行（不再进行负载均衡）

Database 管理器 (DBMgr)

- 负责管理 **entity** 的持久化存储
- 负责数据库与其它服务器进程间 **entity** 信息的通信
- 支持的数据库类型：
 - XML（用于快速的原型）
 - MySQL（用于运营）
 - ... 自行定制（提供 **DBMgr** 的全代码）
- 可与收费系统集成
- 独立的机器

entity 备份

- 存档 (archive)
 - 在BaseApp间轮流调度处理
 - BaseApp向CellApp要求数据
 - 再传给DBMgr

Reviver

- 恢复崩溃的服务器进程
- 不是必要的但是在运营时非常有用
- 休眠进程
- 能收到进程崩溃的消息
- 重新启动进程并终止**reviver**进程
 - 可以被定制为继续工作
- 主要用于复活管理器

BigWorld 机器守护程序 (bwmachined)

- 用于监视服务器进程的守护程序
- 每个服务器机器上有一个**bwmahcined**
- 启动/停止服务器进程
- 通知服务器集群各个进程的存活状态
 - 例如：当有进程崩溃时通知**Reviver**
- 监视机器的使用状态
 - **CPU / 内存 / 带宽**

进程间通信

- Mercury

- 基于UDP的远程过程调用（RPC）协议
- 可靠的通信

- 一些用语：

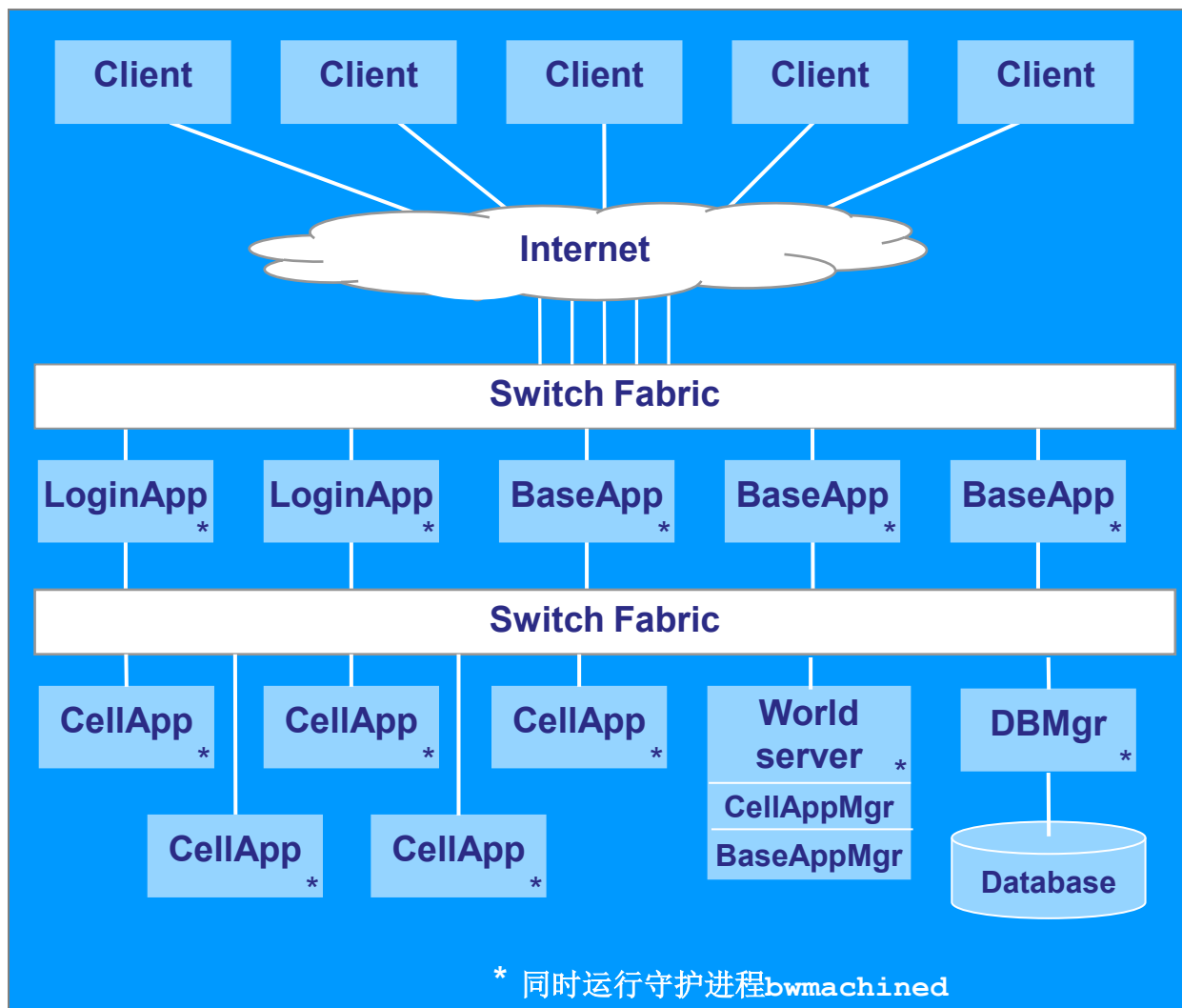
- Bundle

- 一组将要被发送的消息

- Channel

- 2个进程间持续的通信流
- 例如：Client和Proxy间的channel

BigWorld 服务器系统



常用操作

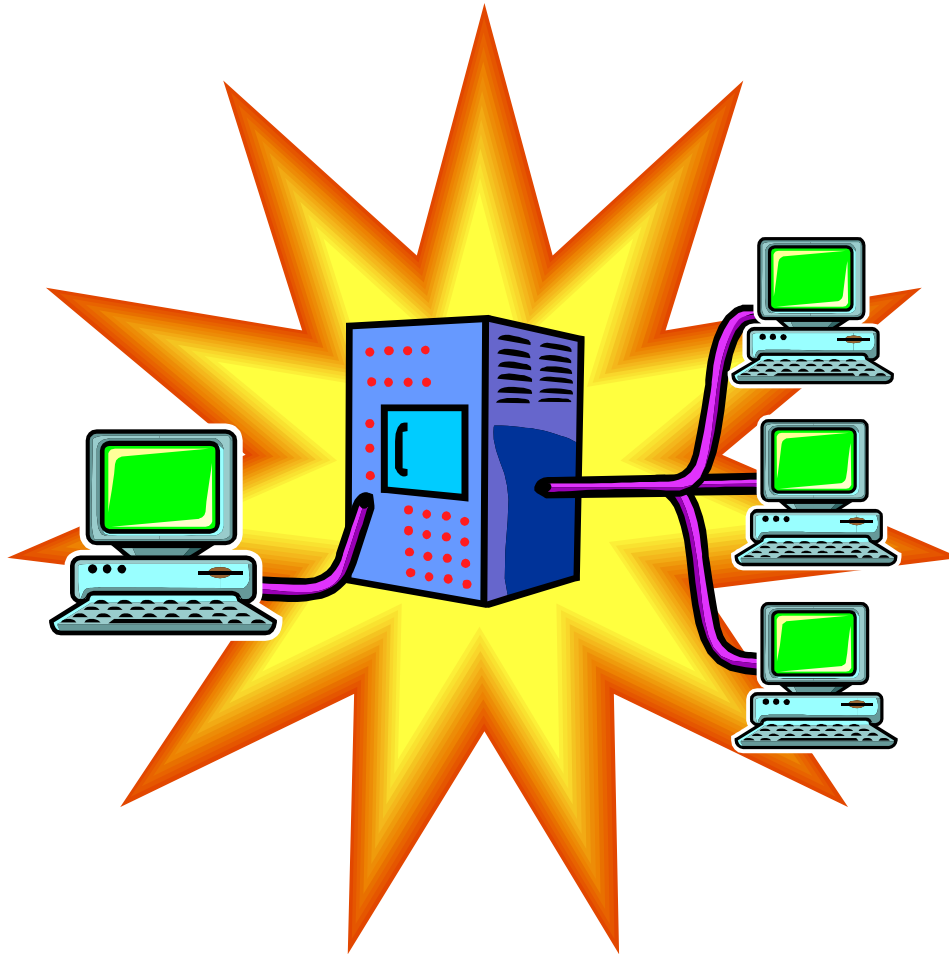
- 每个BaseApp有两个CellApp与之对应
 - 应游戏的不同而不同
 - 尽早并经常进行性能分析
- 为下列进程使用独立的物理服务器：
 - DBMgr
 - 服务端工具

登录过程

- 客户端发送登录请求
 - 主机名/端口是已知的
- LoginApp收到登录请求
 - 解密请求
- LoginApp转发登录消息到DBMgr
- DBMgr验证用户名/密码
 - 查询数据库
- 把有效请求转发到BaseAppMgr
- BaseAppMgr向负载最小的BaseApp发送创建玩家entity的消息
- BaseApp创建一个新的proxy
 - 可能会创建一个新的cell entity
- Proxy的UDP端口被返回给客户端
 - 途径BaseAppMgr、DBMgr和LoginApp

Session 2

实现一个entity



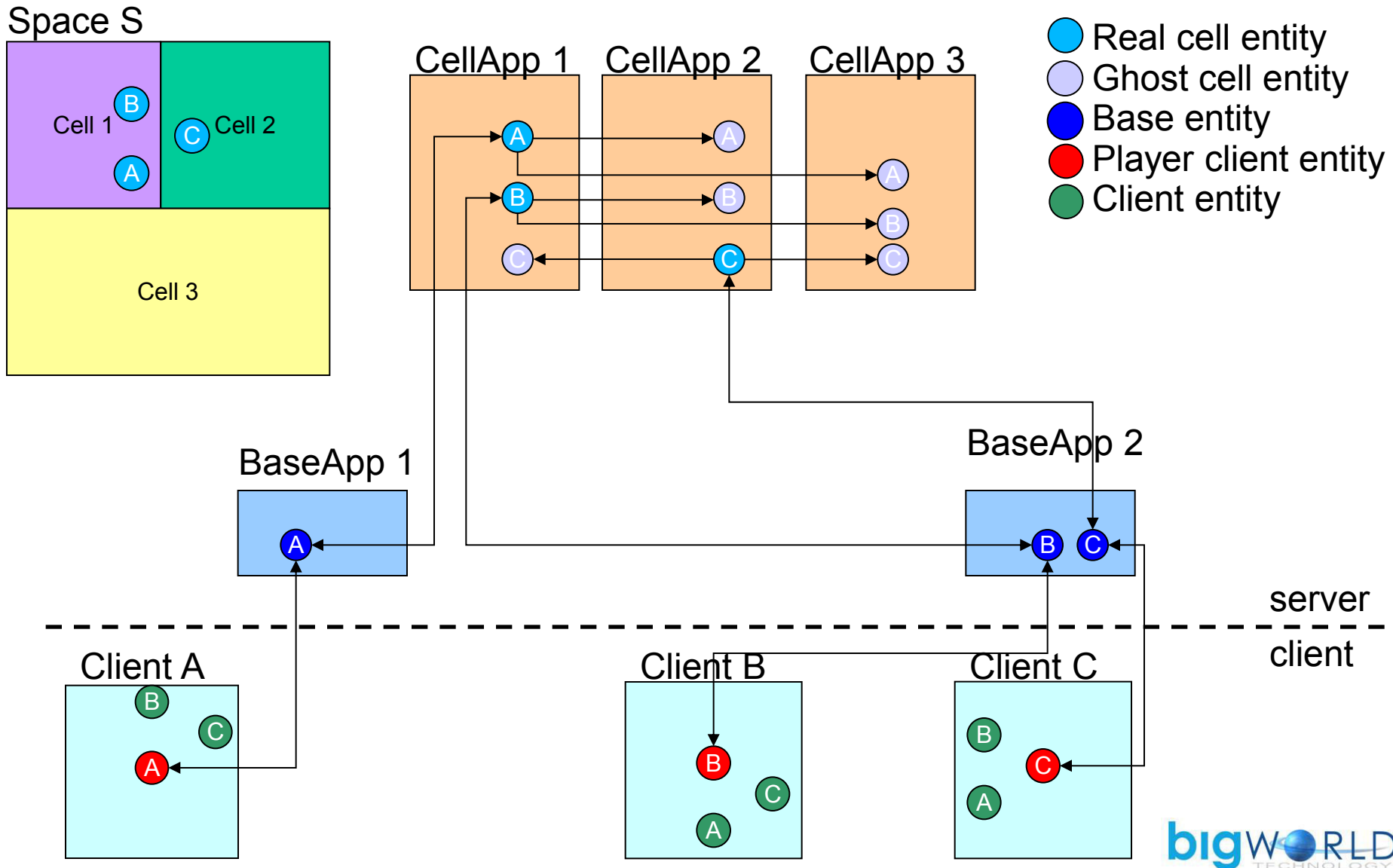
Entity实现文件



Entity的实现

- 每个**entity**必须：
 - 在`entities.xml`文件里列出
 - 必须有一个`<Entity_name>.def`文件
- 每个**entity**可以：
 - 有最多3个部分的实现（**client/cell/base**）
 - 重用`common`路径下的共享脚本
- 客户端/服务端的定义文件必须匹配

分布式entity



简单的角色entity

```
<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 继承

- **Entity**定义文件支持继承

- `<res>/scripts/entity_defs/interfaces`

- 两种继承机制

- `<Parent>`

- 继承所有的东西
 - 属性 / 方法
 - **Volatile** 属性定义
 - **LoD** 级别
 - 简单级别的继承

- `<Implements>`

- 继承属性和方法
 - 多级别的继承

Player Entity

```
<root>
  <Implements>
    <Interface> SimpleCharacter </Interface>
  </Implements>

  ...

</root>
```

Entity 属性

- 类型
 - 像大部分语言一样
 - 为网络传输/数据库存储而标准化
- 缺省值
 - 由类型决定
 - 可以在定义文件里覆盖
- 分布标志
- Detail Level
- volatile 信息
- 持久化

Entity定义中的数据类型

■ 简单类型

- INT8 / UINT8
- FLOAT32 / FLOAT64
- STRING / UNICODE_STRING
- VECTOR3
- ...

■ 序列类型

- ARRAY
- TUPLE

Entity定义类型

```
<root>
  <Properties>
    <name>
      <Type>    STRING    </Type>
    </name>

    <armorColours>
      <Type>    TUPLE
        <of>    UINT8 </of>
        <size>  4      </size>
      </Type>
    </armorColours>
  </Properties>

  ...

</root>
```

Entity定义数据类型

■ 复杂类型

▣ FIXED_DICT

- 和Dictionary类似的对象
- 固定的键集

▣ PYTHON

- 比FIXED_DICT低效
- 快速原型
- 安全问题
（由客户端发送的python对象）
- 使用Python的pickle模块

Entity定义数据类型

```
<root>
  <Properties>

    <CharacterInfo>
      <Type>  FIXED_DICT
      <Properties>
        <name>
          <Type>  STRING </Type>
        </name>

        <class>
          <Type>  UINT8  </Type>
        </class>
      </Properties>
    </Type>
  </CharacterInfo>

</Properties>

...

</root>
```

类型别名

- 可重用的自定义类型

- `scripts/entity_defs/alias.xml`

```
<root>
  <CHARACTER_INFO> FIXED_DICT
    <name> <Type> STRING </Type>
    </name>
    <class> <Type> UINT8 </Type>
    </class>
  </CHARACTER_INFO>
</root>
```

```
<root>
  <Properties>
    <CharacterInfo>
      <Type> CHARACTER_INFO </Type>
    </CharacterInfo>
  </Properties>
  ...
</root>
```

Entity 属性的分布

```
<root>
  <Properties>

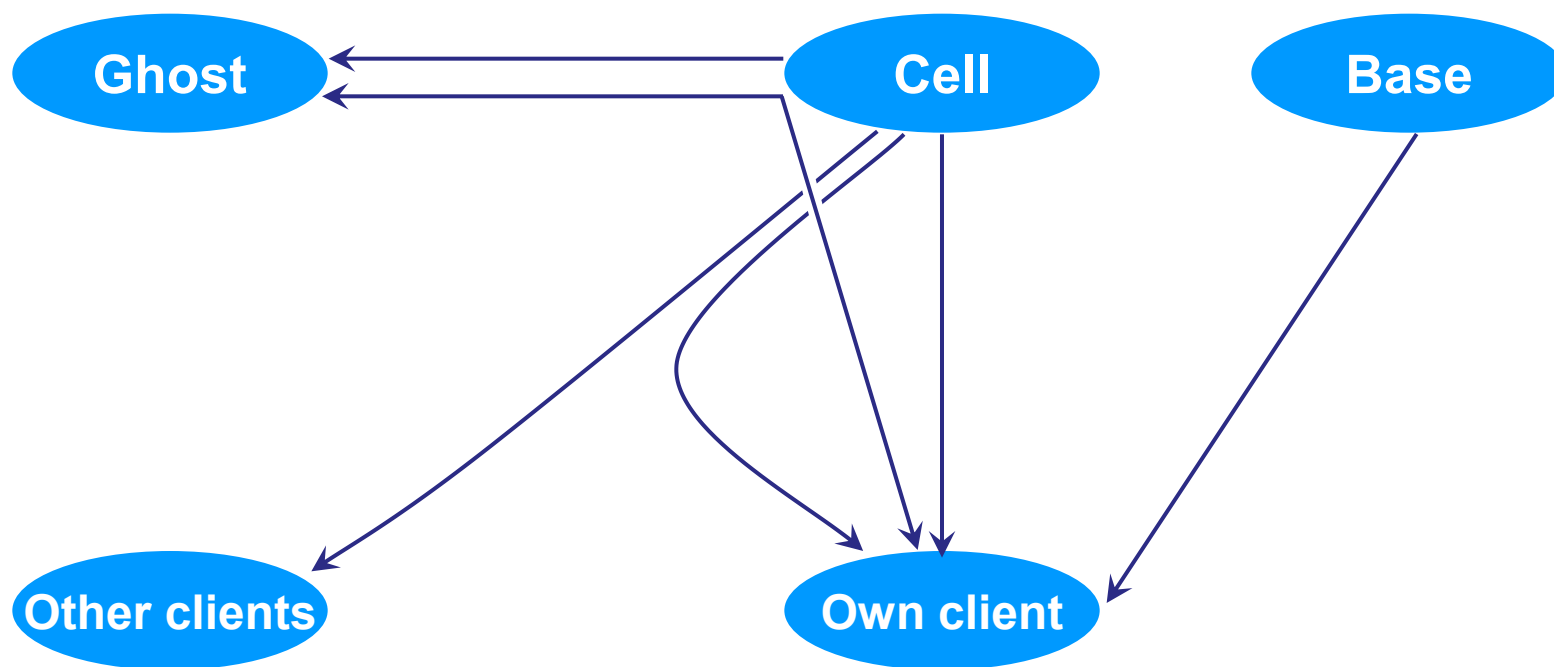
    <name>
      <Type>    STRING      </Type>
      <Flags>   ??  </Flags>

    </name>

  </Properties>

  ...
</root>
```

Entity属性的发布



属性发布 – BASE

- 属于Base
 - 只有Base可以访问
-
- 例如:
 - 聊天室成员列表
 - 角色背包里的物品

BASE 属性

- **BASE**属性的修改不会被发布。
- 把它们定义在**.def**文件里意味着它们会被定期备份到其它
BaseApp和数据库里。



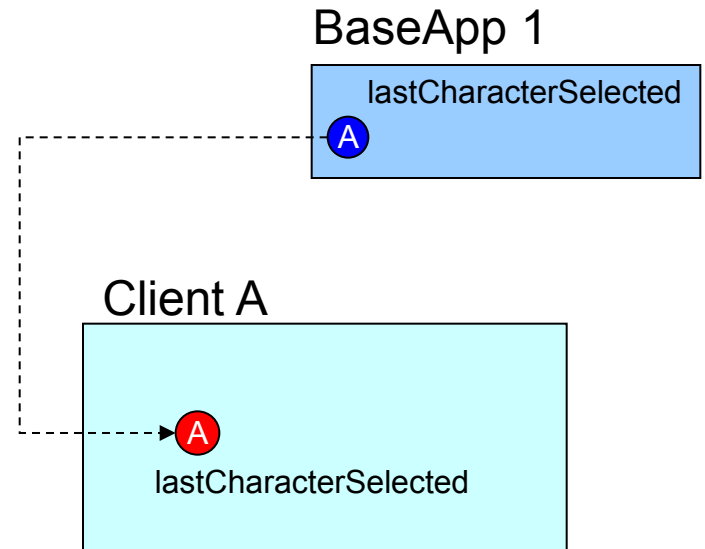
friendList

属性发布 – BASE_AND_CLIENT

- 属于Base
 - Base和自己的客户端可以访问
-
- 只在client entity创建的时候才同步
 - 后续数据变化只能通过显式函数调用进行发布
 - 示例：
 - 同 BASE
 - 很少用到

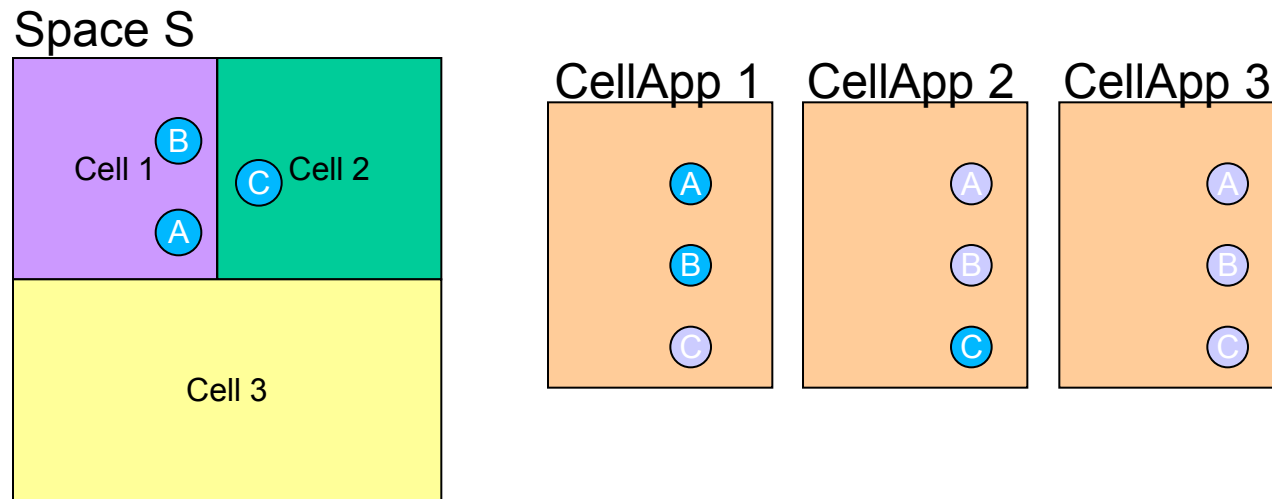
BASE_AND_CLIENT 属性

- 属性只在**client entity**创建时向客户端发布一次。
- 客户端之后不会再收到更新。



Cell Entity 分布的示例

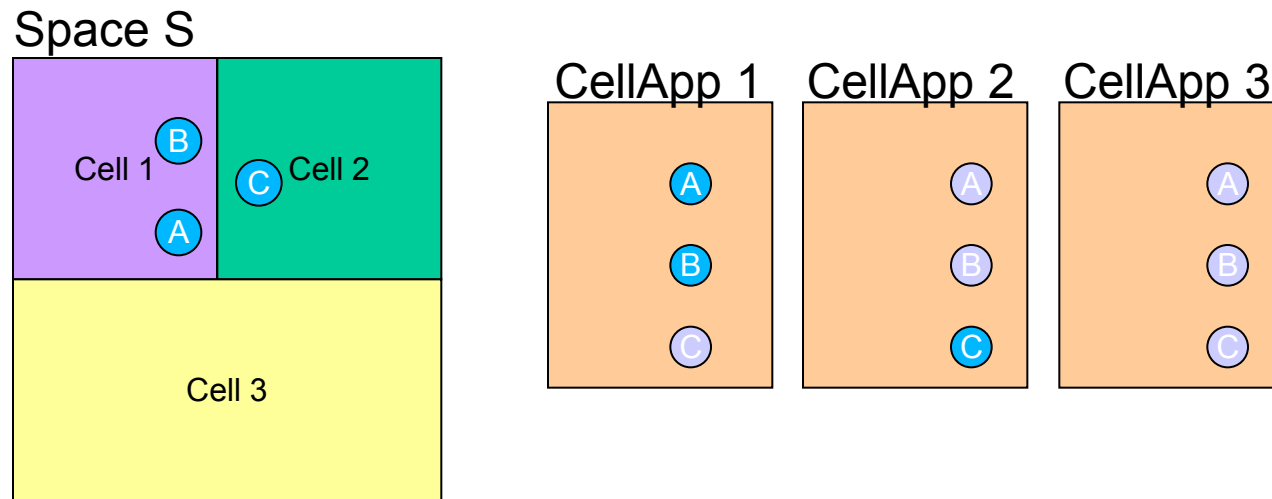
- CellApp 1、CellApp 2和CellApp 3各自都有一个在 Space S 中的cell
- Space S有3个entity A、B 和 C。



从CellApp 1来看

Space S中位于CellApp 1上的cell:

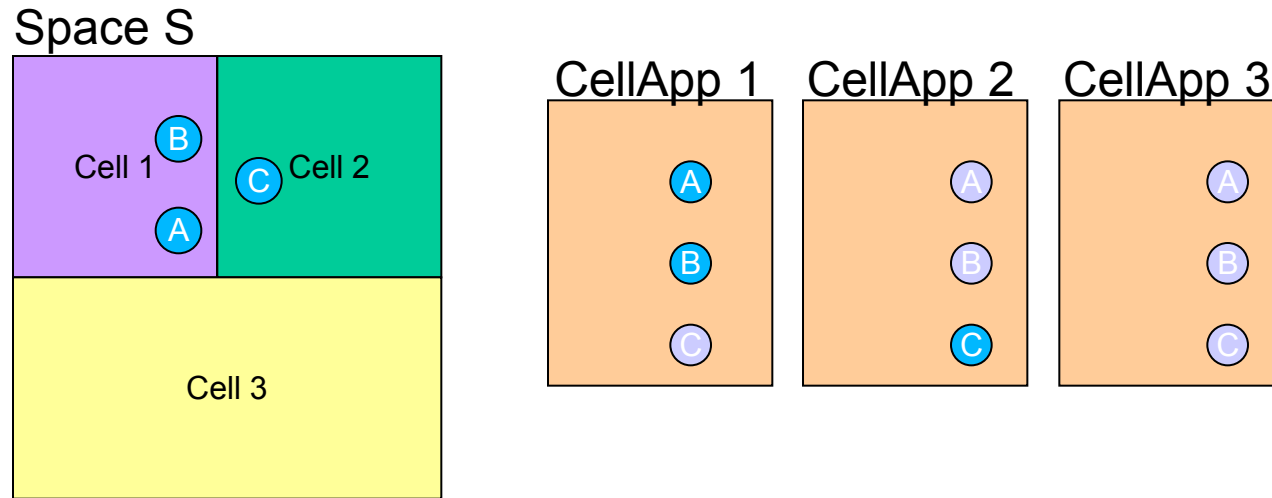
- A和B是real entity。
- C 是一个从CellApp 2上ghost而来的ghost entity。



从CellApp 2来看

Space S中位于CellApp 2上的cell:

- C 是一个real entity。
- A和B是从CellApp 1上ghost而来的ghost entity。

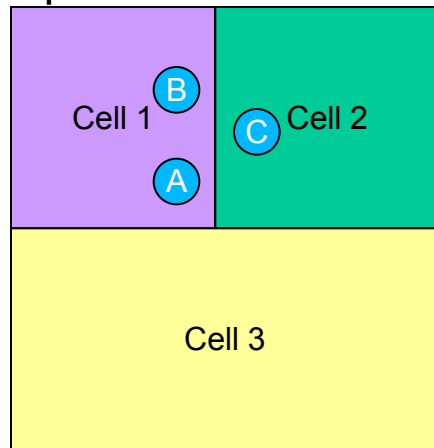


从CellApp 3来看

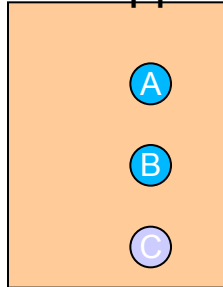
Space S中位于CellApp 3上的cell :

- A和B是从CellApp 1上ghost而来的ghost entity
- C 是一个从CellApp 2上ghost而来的ghost entity。

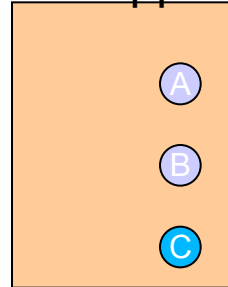
Space S



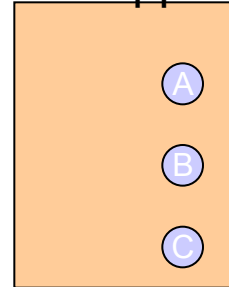
CellApp 1



CellApp 2



CellApp 3

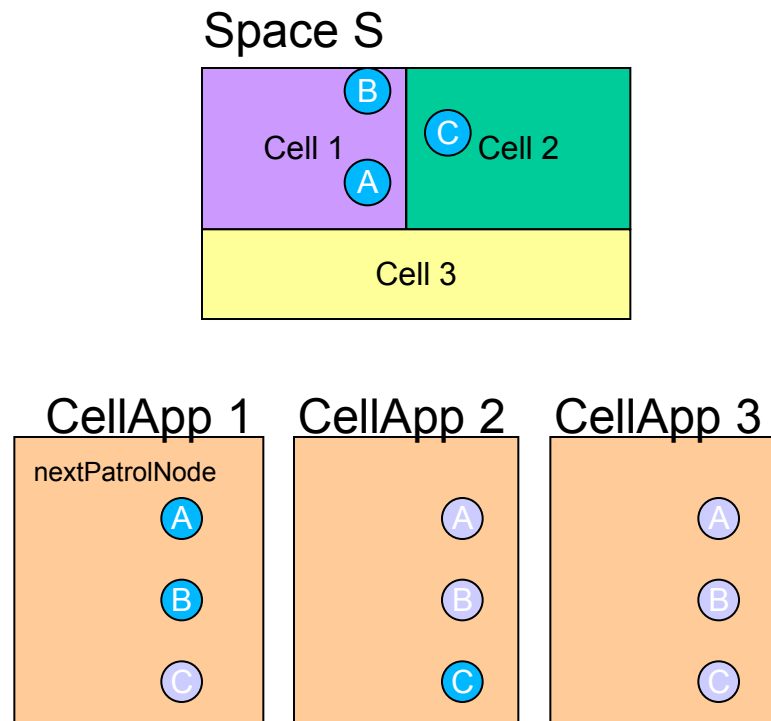


属性发布 – CELL_PRIVATE

- 属于real entity
 - 只有real entity能访问
-
- 例如:
 - NPC AI的‘想法’
 - 玩家entity中和游戏模式相关但是不应该为其他玩家所见的属性

CELL_PRIVATE 示例

- 这些属性属于real cell entity。
- 这些属性不会从real entity发布出去。
- 在.def文件里定义它们就意味着在cell entity从一个cell移动到另一个cell时这些属性也会随之移动到新的cell上。另外，这些属性会被定期备份到base entity上。
- A的属性nextPatrolNode不会被发布到CellApp 2和CellApp 3中的ghost entity A上。



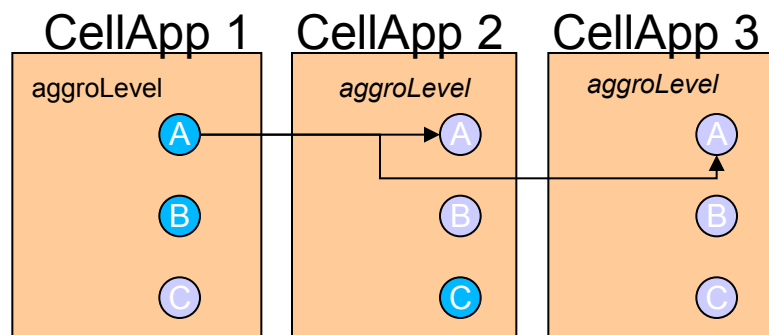
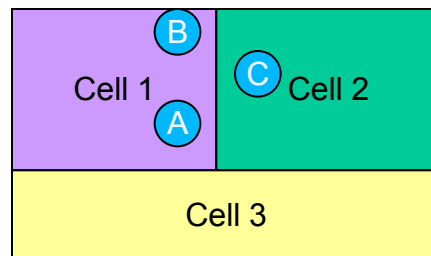
属性发布 – CELL_PUBLIC

- 属于real entity
 - 它所属的real entity和其对应的ghost entity上都可以访问
-
- 例如：
 - 生物的暴力级别（可以被其它生物看到但是不应被客户端看到）
 - 一个NPC的组名

CELL_PUBLIC 示例

- 这些属性属于real cell entity。
- 属性值的改变会被发布到ghost entity上。在ghost entity上这些属性是只读的。
- A的“aggroLevel”属性会被发布到CellApp 2和CellApp 3中的 entity A上。

Space S



属性发布 – CELL_PUBLIC_AND_OWN

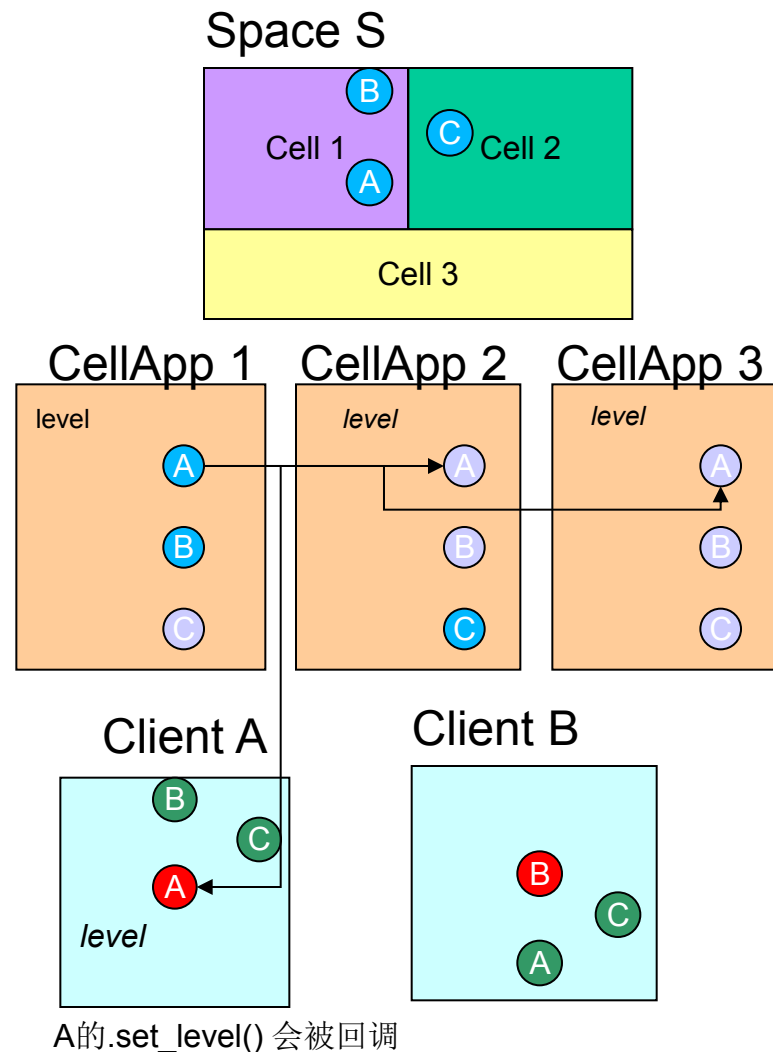
- 属于real entity
 - 它所属的Real entity、和它对应的ghost entity和自己的客户端都可以访问。
-

■ 例如：

- NPC触发的Player debuff
 - 客户端的Player用这个属性来在界面上显示
 - CellApp上的NPC在决定攻击时会用到这个属性

CELL_PUBLIC_AND_OWN 示例

- 这些属性属于real cell entity。
- 属性值的改变会被发布到其对应的ghost entity上。在ghost entity上这些属性是只读的。
- 属性值的改变也会被发布到与其对应的客户端的entity上。在属性发生改变时会调用脚本中的回调函数。
- A的属性“level”被发布到CellApp 2和CellApp 3中的ghost entity A上，同时也会发布到entity A所对应的客户端的entity A上。
- B所对应的客户端上没有这个“level”属性，也不会收到这个属性值的更新。



属性发布 – ALL_CLIENTS

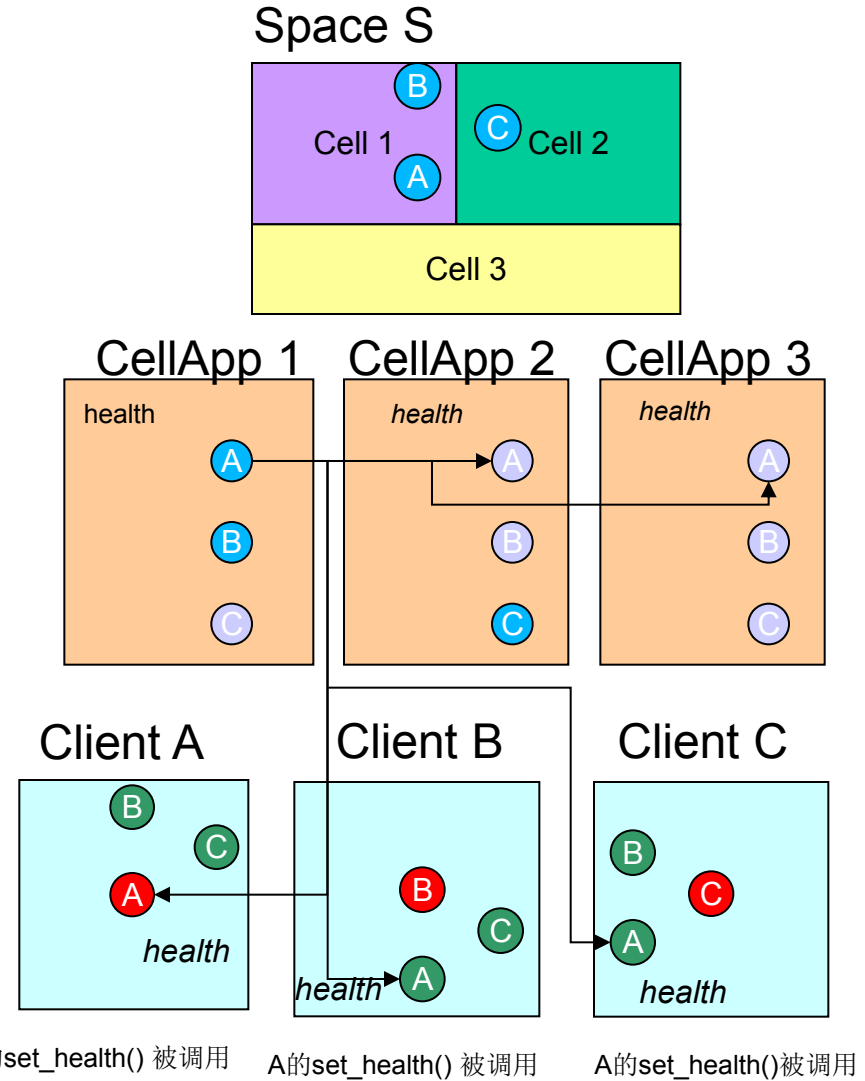
- 属于real entity
 - 它所属的real entity、与其对应的ghost entity和自己的客户端，以及其它的客户端都可以访问
-

- 例如：
 - 玩家姓名
 - 玩家/生物的HP

cell属性值的改变会触发client上的 `set_<property_name>()`

ALL_CLIENTS 示例

- 这些属性属于real cell entity。
- 属性值的改变会被发布到其对应的ghost entity上。在ghost entity上这些属性是只读的。
- 属性值的改变也会被发布到其对应的客户端的entity上。在属性发生改变时会调用脚本中的回调函数。
- 如果其它玩家的AoI范围内有这个属性所属的entity，那么这个属性值的改变也会被发布这些玩家客户端的该entity上。
- A的属性“health”会被发布到Cellapp 2和CellApp 3的ghost entity A上。
- A的属性“health”也会被发布到客户端A,B,C的entity A上。而且A.set_health() 回调函数会被调用。



属性发布 – OWN_CLIENT

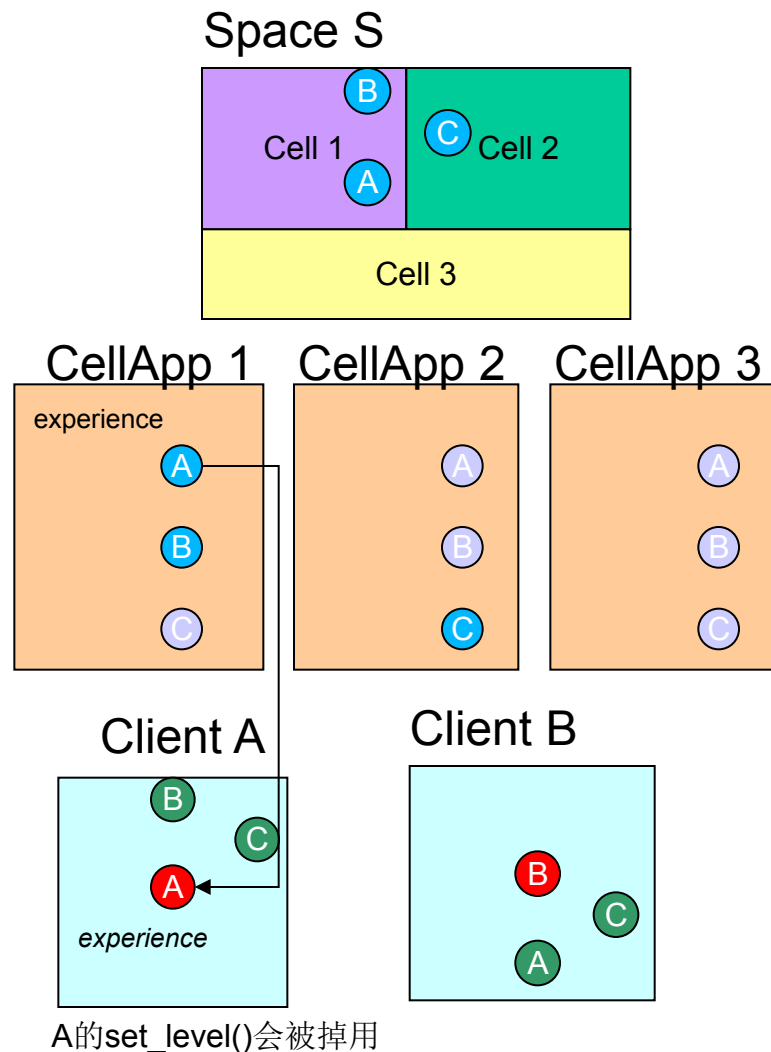
- 属于real entity
 - 它所属的Real entity和自己的客户端可以访问
-

- 例如:
 - 玩家的角色类别
 - 玩家的XP

cell属性更新会触发client上的 `set_<property_name>()`

OWN_CLIENT 示例

- 这些属性属于real cell entity。
- 属性值的改变会被发布到其对应的客户端的entity上。在属性发生改变时会调用脚本中的回调函数。
- Entity A的属性“experience”会被发布到玩家A的客户端的entity A上。



属性发布 – OTHER_CLIENTS

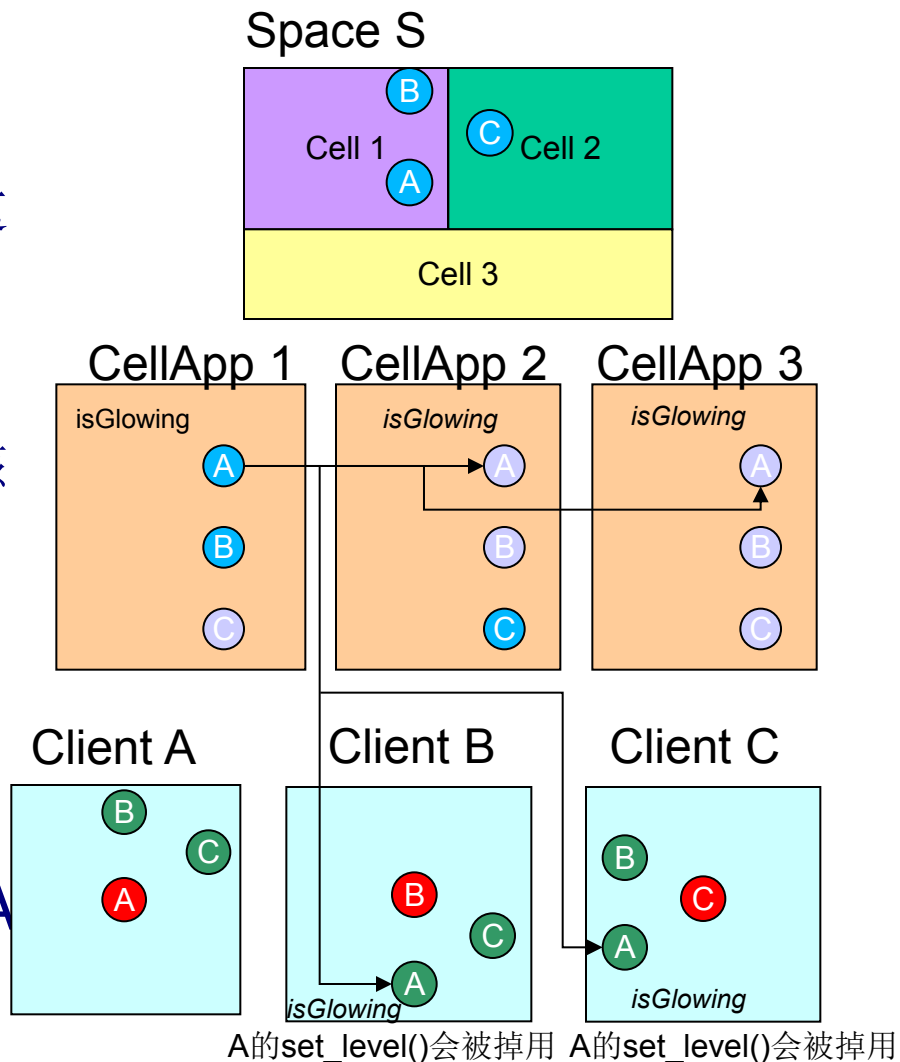
- 属于real entity
 - 它所属的Real entity、与其对应的ghost entity以及其它的客户端可以访问
-

- 例如：
 - 动态世界物品的状态（如：门, 按钮, 战利品）
 - 粒子系统特效的类型
 - 坐在座位上的玩家

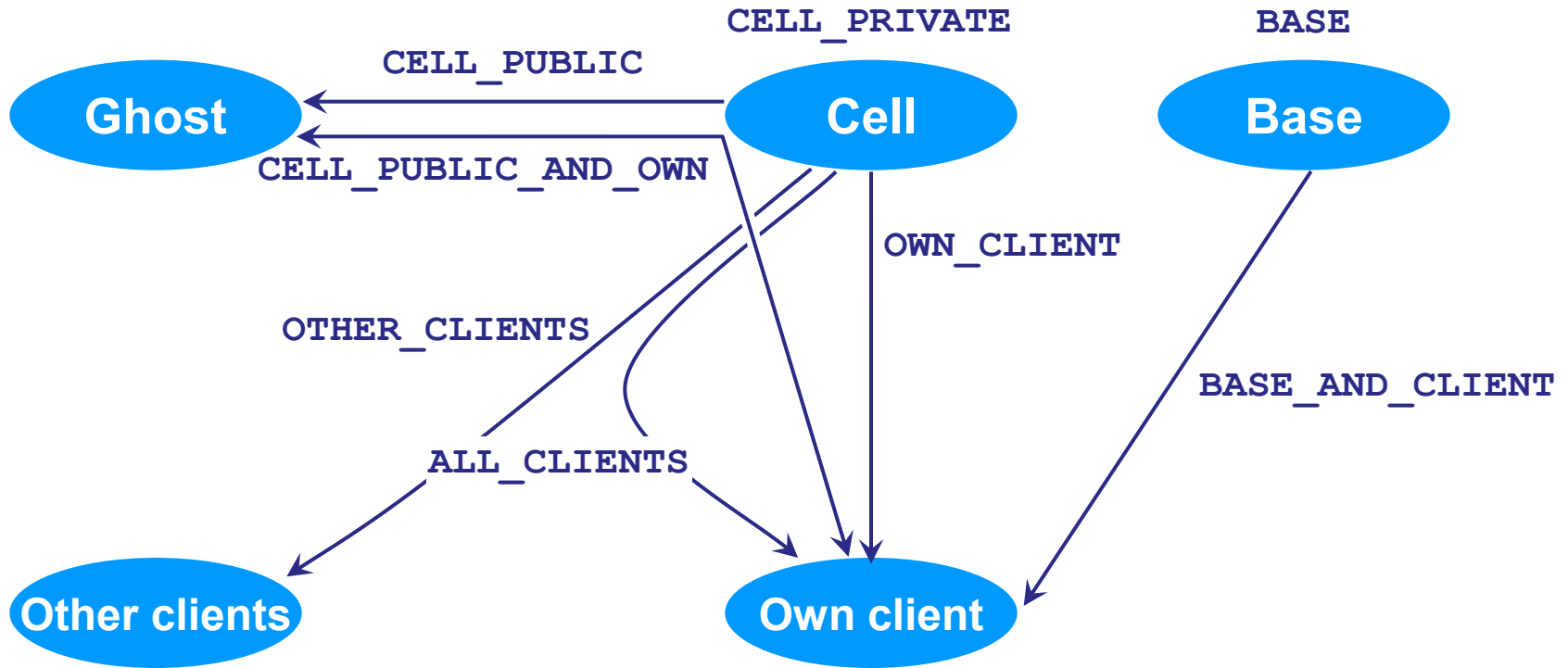
cell属性更新会触发client上的 `set_<property_name>()`

OTHER_CLIENTS 示例

- 这些属性属于real cell entity。
- 属性值的改变会被发布到其对应的ghost entity上。在ghost entity上这些属性是只读的。
- 如果其它玩家的AoI范围内有这个属性所属的entity，那么这个属性值的改变也会被发布位于这些玩家客户端上的该entity上。
- A的属性“isGlowing”被发布到CellApp 2和CellApp 3的ghost entity A上。
- A的属性“isGlowing”被发布到客户端B和客户端C的entity A上。



Entity属性发布



Entity属性发布

```
<root>
  <Properties>

    <name>
      <Type>    STRING    </Type>
      <Flags>   ALL_CLIENTS </Flags>

    </name>

  </Properties>

  ...
</root>
```

属性 Detail Level

- 影响向客户端的属性更新
- 典型地用于可以被看见的属性
- 带宽节省机制
- 可以在需要时使用，但并不必须
- 用<DetailLevel>指定
- 使用<LodLevels>来创建别名

属性 Detail Level

```
<root>

  <LoDLevels>
    <level> 20 <label> NEAR </label> </level>
    <level> 100 <label> MEDIUM </label> </level>
    <level> 250 <label> FAR </label> </level>
  </LoDLevels>

  <Properties>

    <name>
      <Type> STRING </Type>
      <Flags> ALL_CLIENTS </Flags>
      <DetailLevel> NEAR </DetailLevel>
    </name>

  </Properties>

  ...
</root>
```

volatile属性

- 优化的协议
- 仅仅对最近的值感兴趣
- 位置 (x,y,z)
- Yaw, Pitch, Roll

Entity持久化

- 一些entity以及它们的属性需要在服务器重启后仍然保持
- 每个属性独立定义
- entity被写入数据库
- 写入数据库后会自动创建self.databaseID

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

  ...
</root>
```

Entity 属性

■ Cell

- Entity数据被频繁访问
- 当跨越cell boundary时数据会被复制（到新的cell）
- 数据备份到base
- 数据改变时通知客户端：
 - 属性的改变
 - 当一个entity进入玩家的AoI时

■ Base

- 更复杂/访问较少
- 不会自动发布到客户端

Entity 属性

■ Client

- 可访问server属性的一个子集
- 属性值从Cell上发布而来
- Cell属性改变会触发set_<property>()
- 例如:

```
def set_health( self, oldHealth ):  
    if self.health == 0 and oldHealth > 0:  
        self.doDeath()
```

```
def setNested_inventory( path, oldValue ):  
    print "Inventory slot %d changed" % (path[-1],)
```

```
def setSlice_inventory( path, oldValues ):  
    print "%d added. %d removed" % \  
        (path[-1][1] - path[-1][0], len( oldValues ))
```


Entity方法

- 分别定义在
 - Client / Cell / Base
- 必须定义参数
- Base / Cell方法可以被暴露给Client端使用
- Client方法可以指定一个最大的可调用距离
- 必须在entity定义文件里定义以被远程调用

Entity 方法

```
<root>
  <Properties>
    ...
  </Properties>

  <ClientMethods>
    ...
  </ClientMethods>

  <BaseMethods>
    <addToFriendsList>
      <!-- Entity ID -->
      <Arg> INT32 </Arg>

      <!-- Expose to client -->
      <Exposed />
    </addToFriendsList>
  </BaseMethods>

  <CellMethods>
    ...
  </CellMethods>
</root>
```

暴露 Server方法

- 不是所有的**server**方法都被暴露的
- 使用<Exposed />显式暴露
- 暴露的**Cell**方法
 - 自动接收调用方的**EntityID**
 - 通常要检查是否
`self.id == callerID`
- 暴露的**Base**方法
 - 只有自己的**client**可以调用

Entity方法

■ Client方法LoD

- 帮助减少客户端带宽的使用
- 在近距离产生视觉效果
- 当广播**client**消息时较为有用

```
<root>
  ...

  <ClientMethods>
    ...
    <smile>
      <DetailDistance> 30 </DetailDistance>
    </smile>
    ...
  </ClientMethods>

  ...
</root>
```

Entity实现

- **Entity**根据需要在cell/base/client分布平台的一个或多个上。
- 如果**entity**不需要存在于某个部件上，那么在该部件上也不需要该**entity**的python脚本

Entity存在示例

Base	Cell	Client
出生点		
聊天室	召唤的wildlife *	
Player entity		
Server AI/NPC's		

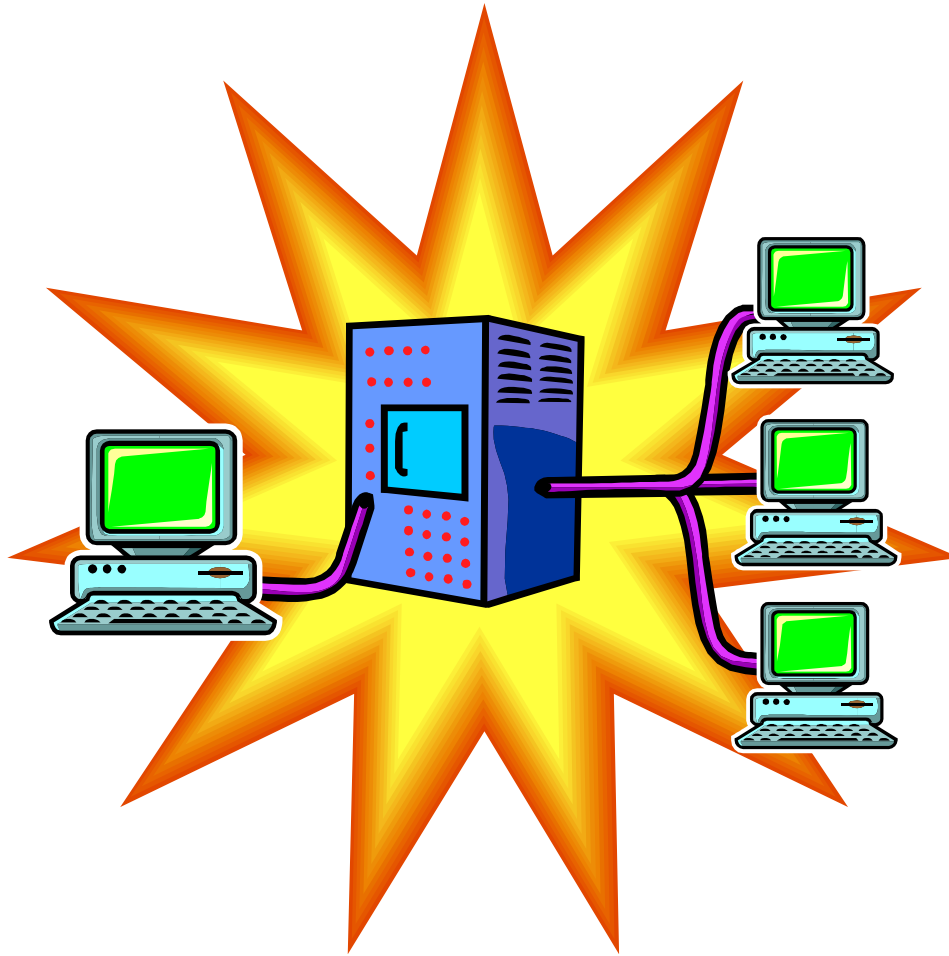
* 没有base部分的entity是不参与容错的

脚本开发的指导

- 尽可能的把负载放到BaseApp
- 尽可能减少持久化entity属性
- 避免过多调用writeToDB()
- 尽量避免嵌套的数据类型
 - 如：多维数组
- 如果脚本的执行时间超过1个game tick，会对服务器效率产生负面影响

Session 3

Entity 通信



Mailbox

- 指向远程进程的**entity**的引用
 - 如：一个**entity**的**Cell**部分
- 使得我们可以进行远程函数调用
 - 如：mb 是一个**cell entity** 的 mailbox
mb.someMethod(a, b)
会在**real cell entity**所在的进程调用someMethod()
- **Entity**内部通信
 - 如：从**cell**部分到**base**部分
- **Entity**之间的通信
 - 如：entity A的**cell**部分到entity B的**base**部分

Mailbox

- 不同的类型
 - Base
 - Cell
 - Client
 - 单步跳转
 - 多步跳转
 - 通过base到cell
- 一些BigWorld方法可能只接收一些特定的类型的mailbox
 - 请参考Python API 文档以获得更多细节

Mailbox

- Entity有mailbox成员变量
 - Client entity: `self.cell, self.base` （用于玩家）
 - Base entity: `self.cell`
 - Proxy entity: `self.cell, self.client`
 - Cell entity:
 - `self.base`
 - `self.ownClient`
 - `self.allClients`
 - `self.otherClients`

Mailbox

- 当一个**entity**对象被传到一个有 **MAILBOX** 参数的**server**方法时，会自动地创建**mailbox**
- 例如：
 - **Cell**方法**talkToMe()** 有一个**MAILBOX**参数
 - 在一个**cell**上，**entityA**调用：
`entityB.talkToMe(self)`
 - **Entity A**的**mailbox**被传到**Entity B**
`def talkToMe(self, mailbox):`
`mailbox.sendMsg("hello")`
 - **Entity A**的**sendMsg()**被调用（以 `"hello"`为参数）

储存Mailbox

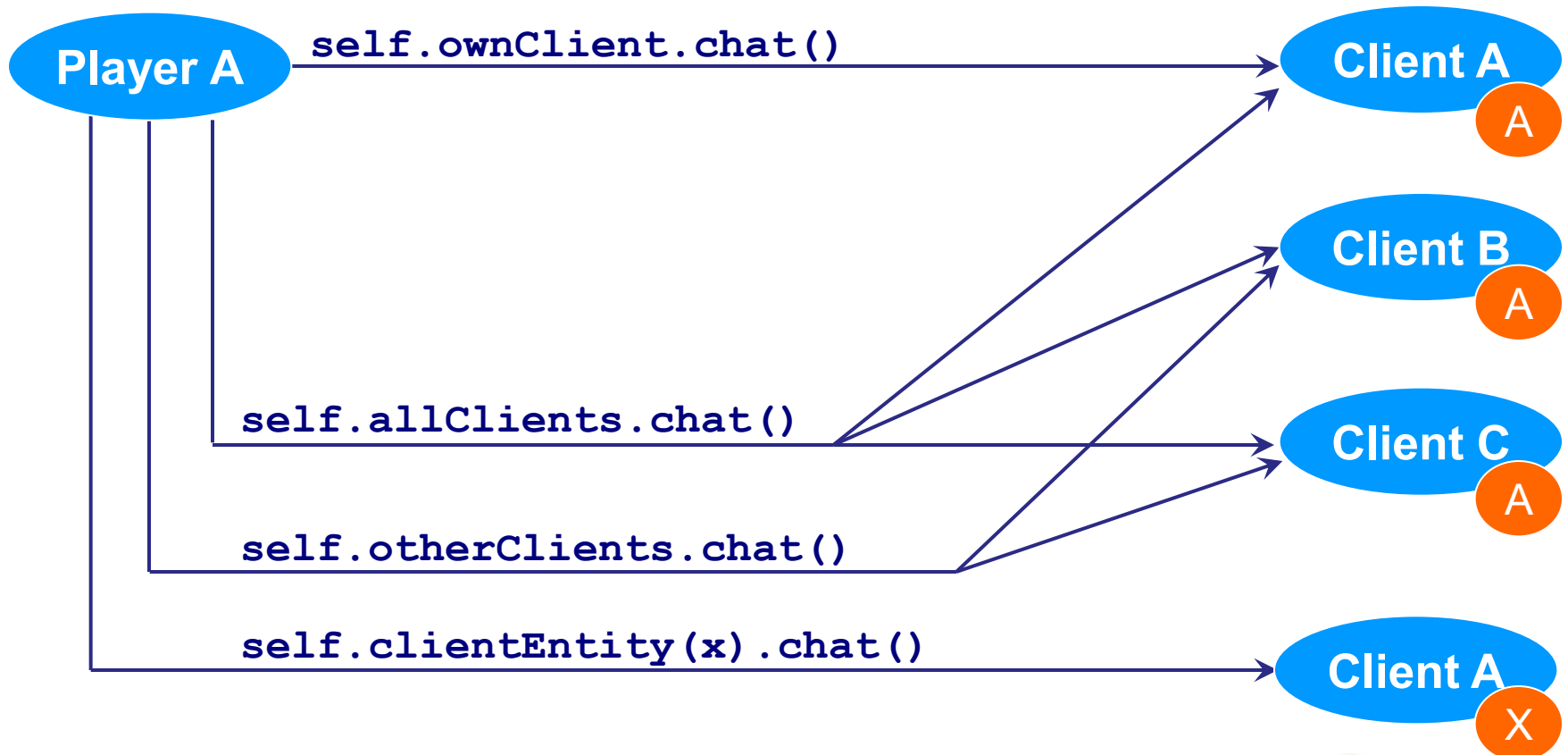
- **Base**的**mailbox**在**entity**的生命周期内保持有效
 - **Base entity**不会在不同**BaseApp**之间移动
 - 可以用于**entity**间的长期通信
 - 如果储存一个**mailbox**，必须实现一个消息通知机制
- **Cell mailbox**只在很短的时间内有效
 - **Cell entity**会在不同**CellApp**之间移动
 - 不要保存**Cell MailBox** 作为属性
 - 立即使用，随后立即释放

储存Mailbox

- 不能把mailbox传递给client，也不能从client接收mailbox
 - 不能信任client
 - 使用entity ID
- Mailbox不能被保存到数据库中
 - 当server重启时IP地址会被改变

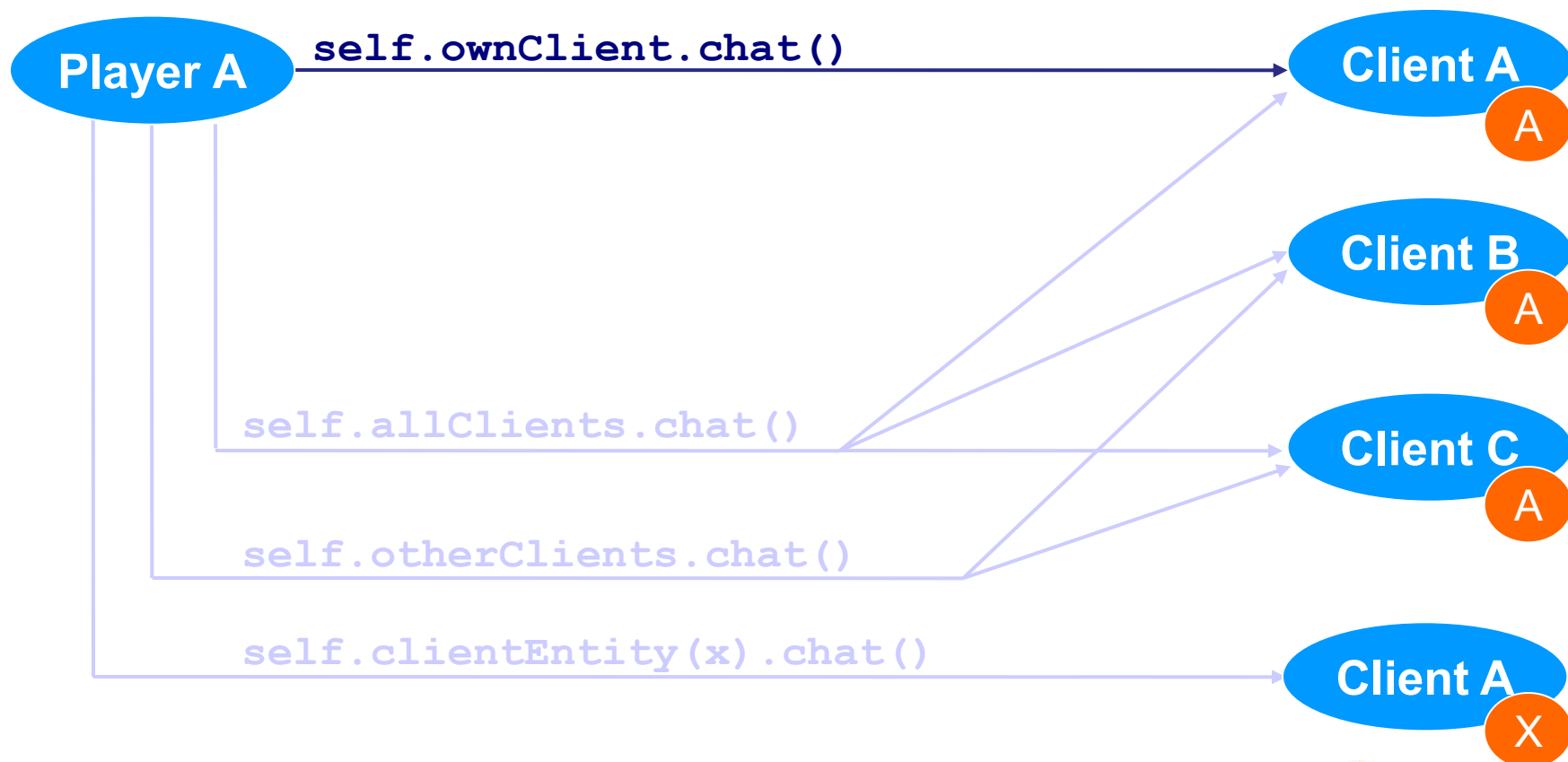
Cell到Client的通信

- **self**是player A
- Player在BaseApp上必须是一个 Proxy
- 这些MailBox不能被传递



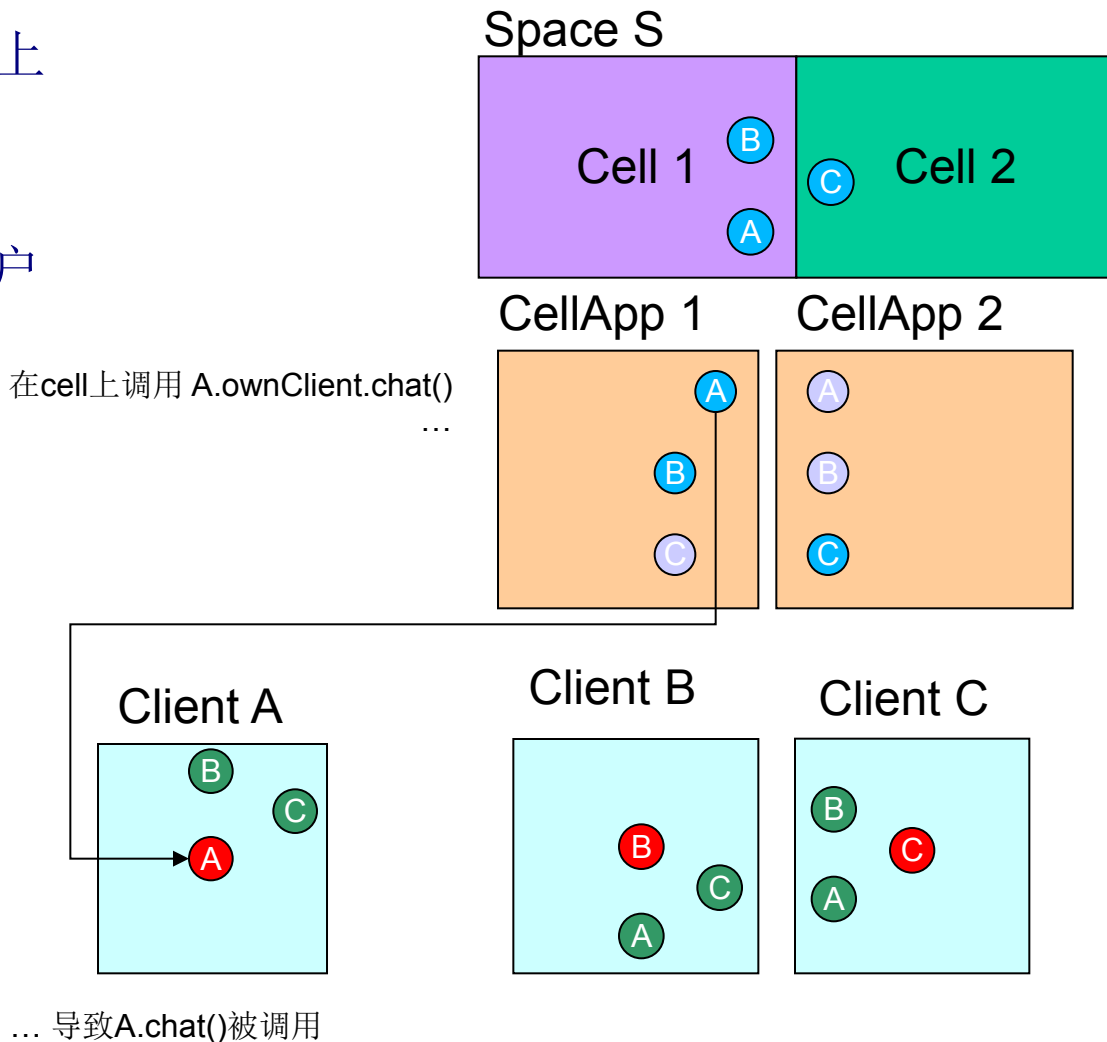
Cell到Client的通信

- **self**是**player A**
- **Player**在**BaseApp**上必须是一个 **Proxy**
- 这些**MailBox**不能被传递



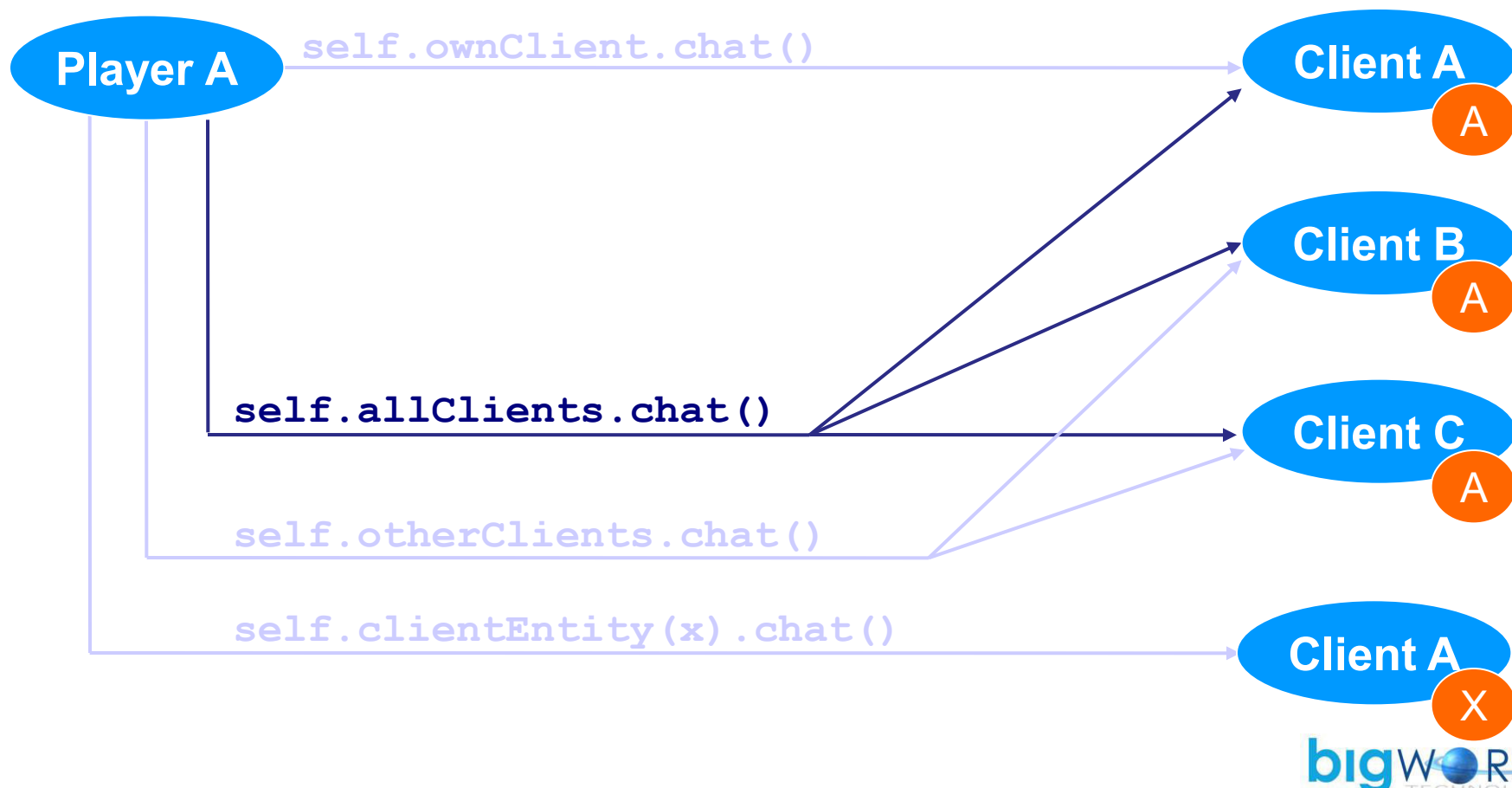
Entity.ownClient 方法调用示例

- `self.ownClient.chat()` 实际上使得chat函数在A客户端的entity A上被调用。
- 其它的客户端不会意识到A客户端上有A.chat()被调用。



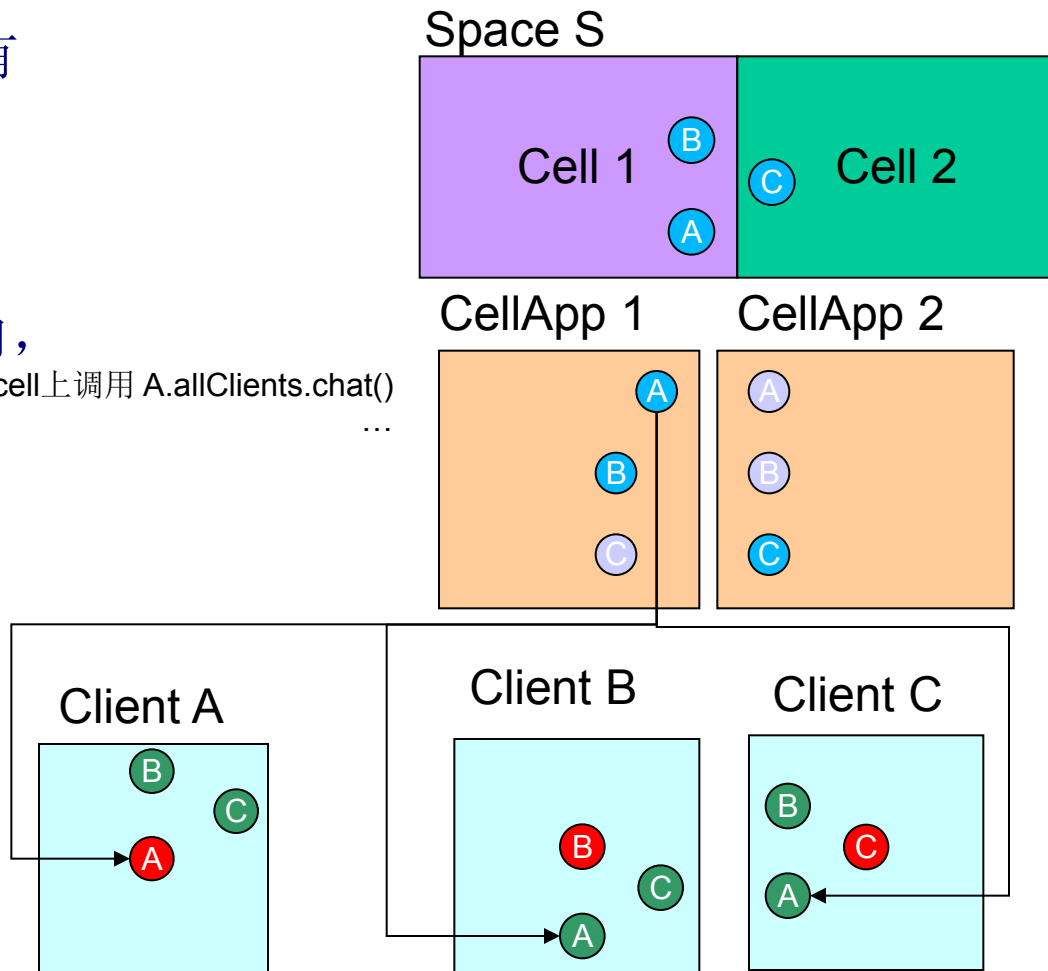
Cell到Client的通信

- **self**是player A
- Player在BaseApp上必须是一个 Proxy
- 这些**MailBox**不能被传递



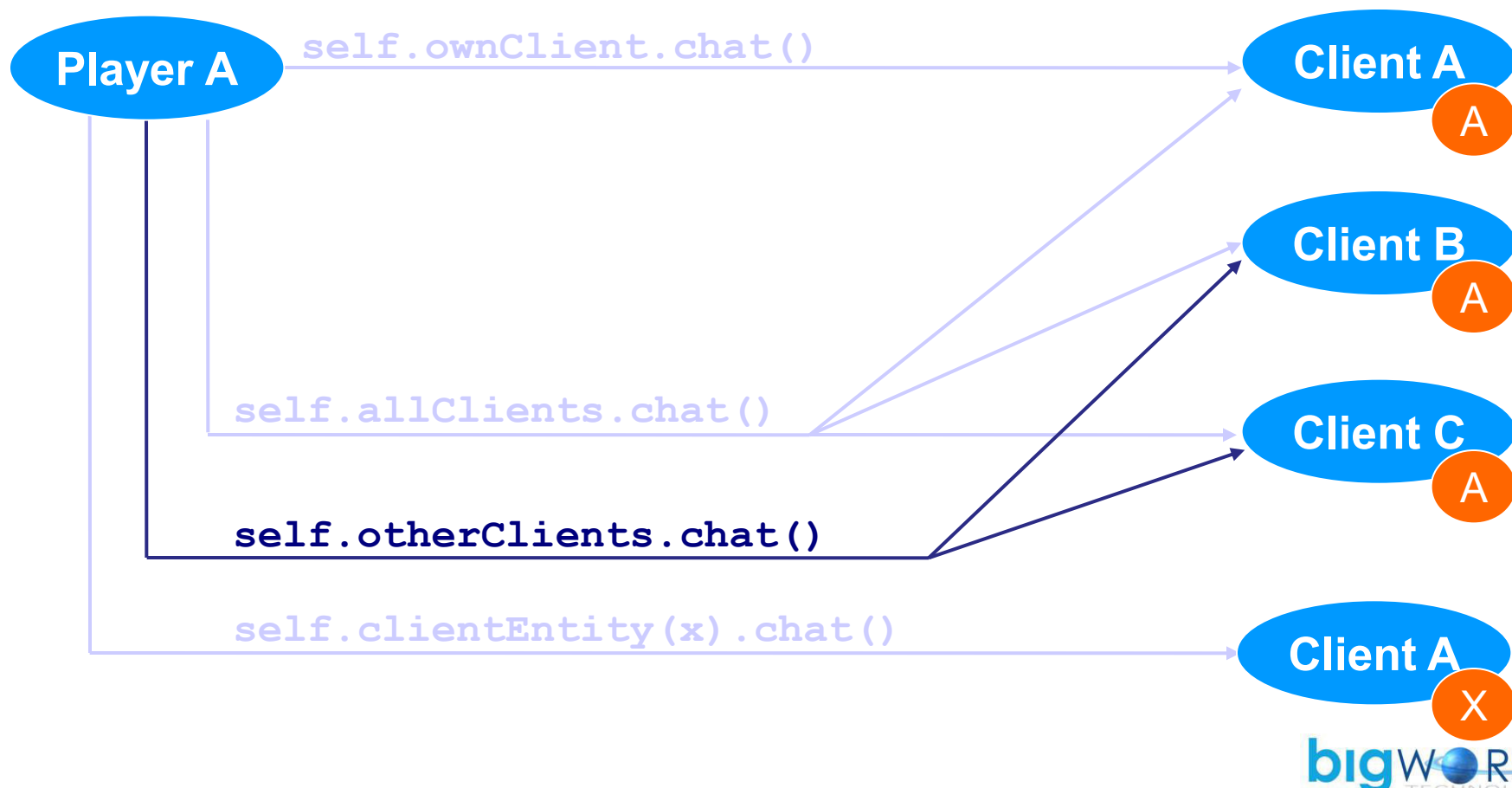
Entity.allClients 方法调用示例

- `self.allClients.chat()` 会在所有可以看到**A**的玩家客户端上调用 **entity A**的`chat()`函数。
- 如果一个玩家和**A**处于同一个 **space**，并且**A**处于其**AoI**范围内，那么在这个玩家的客户端就能调用 `entity A.allClients.chat()` ...
A。



Cell到Client的通信

- **self**是player A
- Player在BaseApp上必须是一个 proxy
- 这些**MailBox**不能被传递



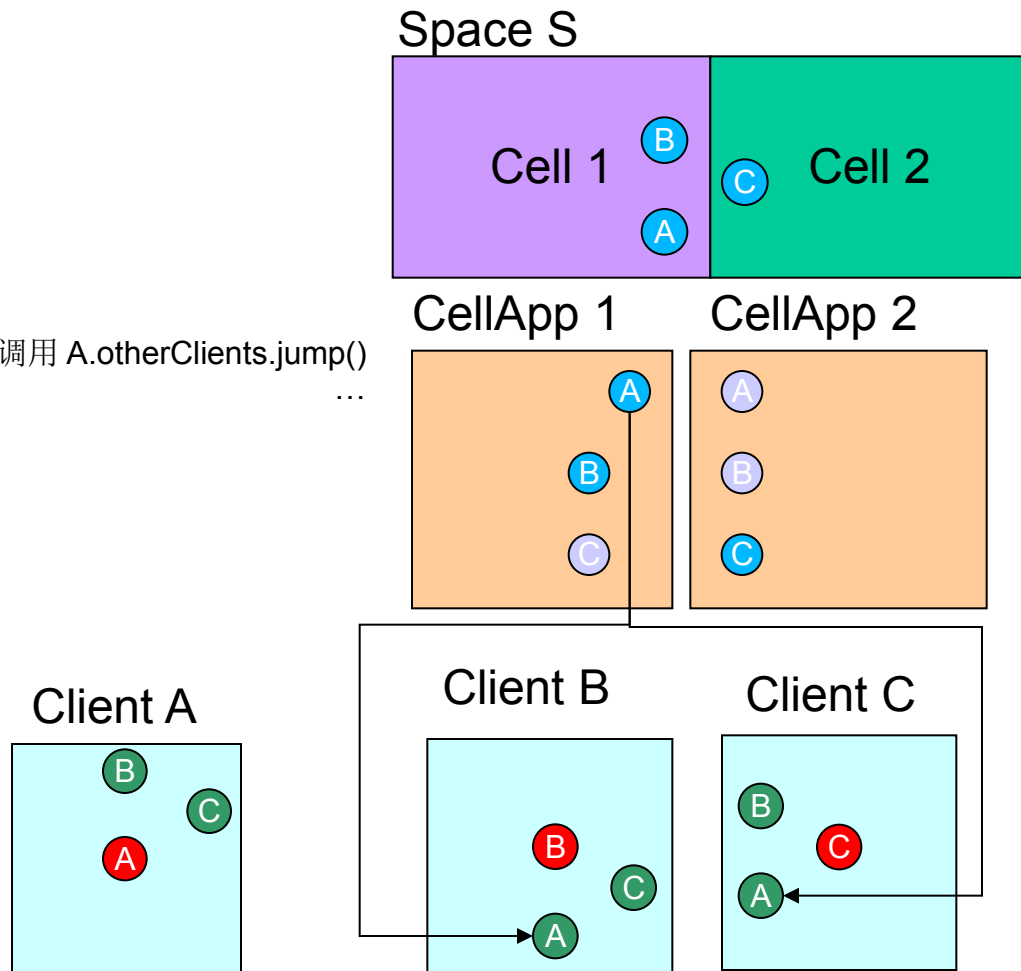
Entity.otherClients 方法调用示例

- `self.otherClients.chat()` 会在除了A本身所在的客户端以外所有可以看到A的玩家客户端上调用entity A的`chat()`函数。

在cell上调用 `A.otherClients.jump()`

- 如果一个玩家和A处于同一个space，并且A处于其AoI范围内，那么这个玩家的客户端就能看到A。

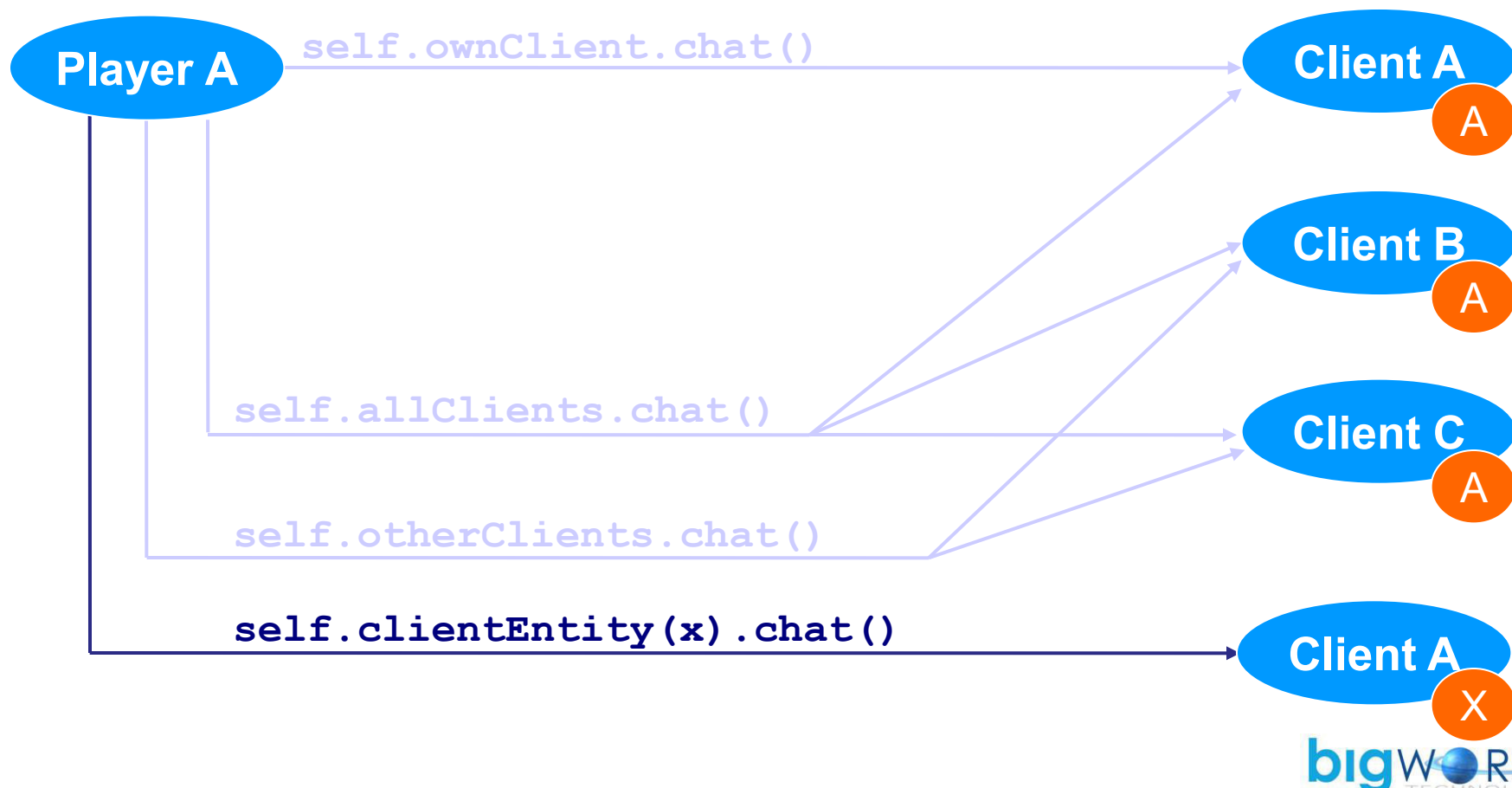
- 通常用于由玩家客户端触发的、立即可以见到效果的动作，并且这些动作需要被广播到其它玩家的客户端。例如：跳跃。



... 导致Clients B和C上的 `A.jump()` 被调用

Cell到Client的通信

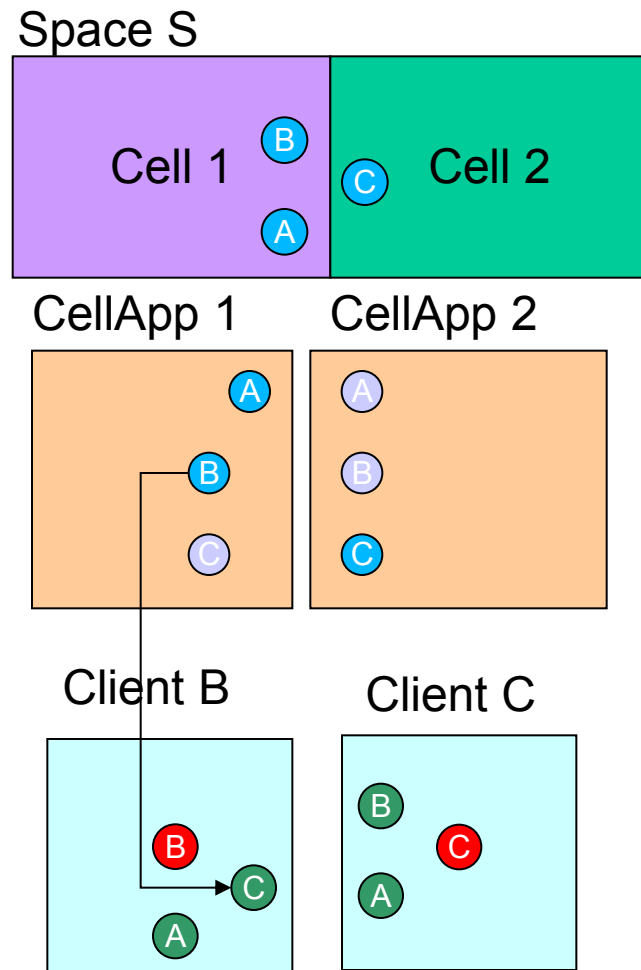
- **self**是player A
- Player在BaseApp上必须是一个 proxy
- 这些**MailBox**不能被传递



Entity.clientEntity(id) 方法调用示例

- 在特定客户端的特定entity上进行远程的客户端方法调用。

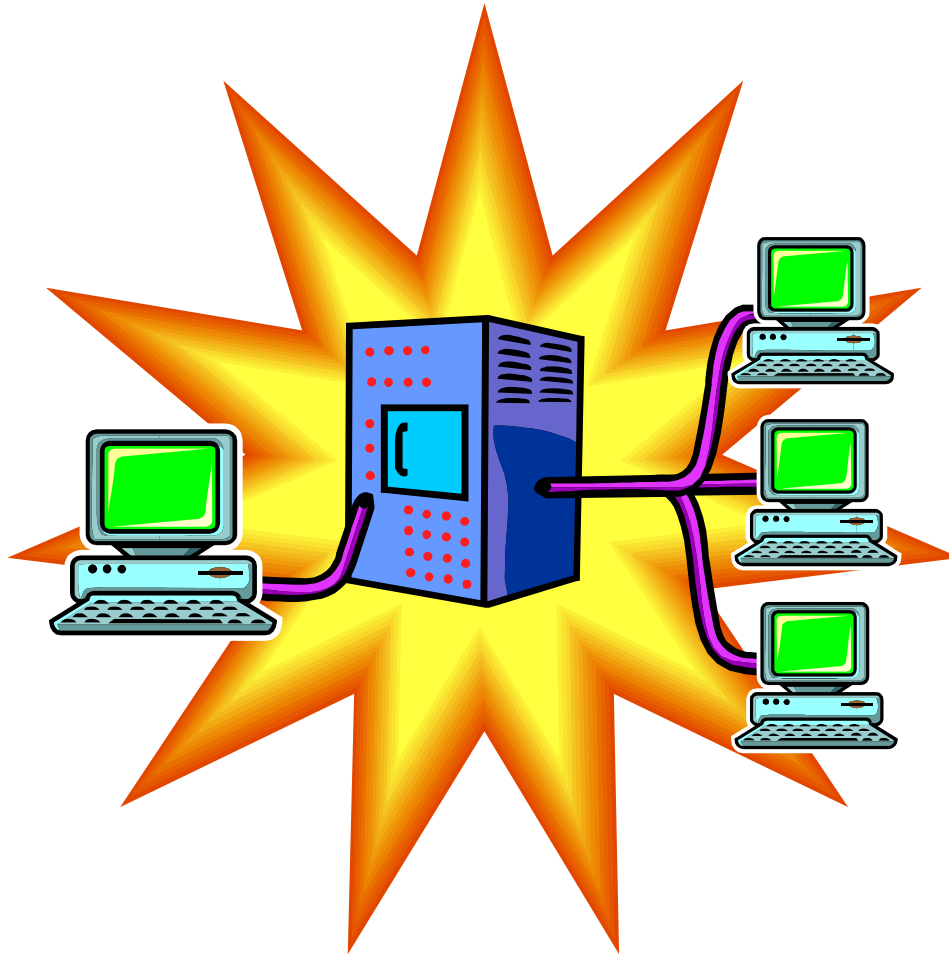
在cell上调用 B.clientEntity(C.id).wave()
...



... 导致客户端B上的 C.wave() 被调用。

Session 4

核心Entity部件



Base Entity类型

■ Base

- 在Python脚本中，继承于**BigWorld.Base**
- 存放大型/复杂的数据
 - 减少当cell entity跨越cell边界时的系统负担
- 可以用来接收方法调用的固定mailbox

■ Proxy

- 在Python脚本中，继承于**BigWorld.Proxy**
 - BigWorld.Proxy内部是从BigWorld.Base继承而来的
- 它是与client的通信点
- Client可以根据需要和一个proxy建立或断开连接

Base Entity属性

- 从BigWorld.Base继承的entity属性

属性	描述
id	唯一的entity ID, cell、base和client共用一个id
databaseID	Entity在数据库里的持久化ID。在非持久化时为零。64位
cell	如果有对应cell entity存在, 表示指向该cell entity的MailBox
cellData	类似于Dictionary的对象, 在cell entity不存在时它会包含entitycell部分的属性

Base Proxy属性

- **BigWorld.Proxy**继承自**BigWorld.Base**，它是所有具有proxy的base entity的父类
- 附加属性：

属性	描述
client	用于与位于对应的客户端进程上该entity进行通信的MailBox
clientAddr	对应客户端机器的地址和端口
bandwidthPerSecond	每秒发送给客户端的信息量 *
wards	存放所有被对应客户端控制的entity的ID 这影响到client和cell之间的信息是怎样传递和处理的。只读

* 很少使用，限制volatile属性的更新

Base Entity方法

方法	描述
<code>addTimer(initOffset [, repeatOffset, userData])</code>	<ul style="list-style-type: none">■ 增添一个时钟（ offsets的单位为秒 ）并返回它的ID■ Entity必须实现方法onTimer(self, timerID, userData)
<code>delTimer(timerID)</code>	<ul style="list-style-type: none">■ 删除指定时钟
<code>createCellEntity([cellMailBox])</code> *	<ul style="list-style-type: none">■ 在mailbox指向的cell上创建entity■ 在base上第一次创建某个entity时，可以用这个函数在cell上创建其cell entity■ 如果不传递cell MailBox，则使用Base.cellData[spaceID]
<code>createInNewSpace ()</code> *	<ul style="list-style-type: none">■ 在一个新的space创建一个entity的cell部分（也会创建新的cell对其进行管理）■ 可以用于创建一个entity来控制一个新的space（如，任务管理器）
<code>destroyCellEntity ()</code>	<ul style="list-style-type: none">■ 删除cell entity，保留base部分■ 如果要在space间移动，可以在CellApp上用“teleport”而不是销毁并重新创建该cell entity■ 此时base上onLoseCell会被调用，Base.cellData属性会被赋以cell entity的属性
<code>destroy ()</code>	<ul style="list-style-type: none">■ 销毁entity的base部分■ Cell Entity必须已经先被销毁掉■ 可以用来把entity从游戏中去除■ 常常在onLoseCell回调函数里使用

*从Base.cellData中取得属性并传送给Cell entity，而且Base.cellData变得不可访问

Cell Entity的属性

■ 一些在BigWorld.Entity定义的属性

属性	描述
id	唯一的entity id, cell、base和client共用一个id
spaceID	Entity所在的BigWorld space
vehicle	Entity当前的vehicle, 如果没有则为None
position	Entity在世界坐标系中的位置
roll	Entity的朝向
pitch	
yaw	
direction	Entity的朝向, 由roll, pitch, yaw来组合表示
volatileInfo	决定每个volatile属性的更新频率, 缺省情况下使用在.def文件里定义的值
topSpeed	Entity的最大速度。用于物理检查

Cell Entity方法

方法	描述
<code>destroySpace()</code>	<ul style="list-style-type: none">删除space里的所有entity,从而删除space
<code>destroy()</code>	<ul style="list-style-type: none">删除entity的Cell部分从space里删除entity
<code>entitiesInRange(range [, entityType, position])</code>	<ul style="list-style-type: none">搜索指定范围内的所有entity可以搜索到AoI范围以外的entity,但是无法找到cell以外的entity球型测试
<code>isReal()</code>	<ul style="list-style-type: none">返回此entity是real还是ghost的
<code>setAoIRadius(radius [, hysteresis])</code>	<ul style="list-style-type: none">改变AoI半径,缺省为500m必须小于ghost距离,缺省为500m
<code>teleport(nearbyEntityMRef, position, direction)</code>	<ul style="list-style-type: none">在同一个space内改变entity的位置把entity放入另一个space - 和nearbyEntityMRef指向的entity所在的space相同

- 所有entity属性和方法都可以在Python API文档内查到:

- BaseApp: bigworld/doc/api_python/python_baseapp.chm
- CellApp: bigworld/doc/api_python/python_cellapp.chm
- client: bigworld/doc/api_python/python_client.chm

典型的Entity生存周期

- **Base**部分先被创建
 - 从数据库，**chunk**或者代码里创建
 - **Base entity**可以没有**cell**部分-**cellData**属性
 - **Base entity**在其**cell**部分存在时不能被销毁
 - **Base entity**通常在**onLoseCell()**回调函数里自行销毁
- **Cell**部分由**base**部分来创建
 - **Cell-only**的**entity**可以用脚本来创建
- **Client**部分通常在**entity**进入到玩家的**AoI**时被创建
 - 应该使用**enterWorld()/leaveWorld()**回调函数而不是**__init__()**函数

Entity创建

- **Entity**在**Cell**上的实例化会在下一次网络更新时发布到合适的**Client**
- 推荐的创建方法:
 - **Base Entity:**
`BigWorld.createBaseAnywhere()`
 - 或者:
`createBaseLocally()`
`createBaseRemotely()`
`createBase...FromDB()`

Entity创建

- 推荐的创建方法:

- **Cell Entity:**

- `createCellEntity()`

- `createInNewSpace()`

- 在从数据库读取**cell entity**属性后，可以在创建**cell entity**之前对其进行修改

- 参看 **Base API** 文档: `BigWorld.Base.cellData`

- 仅存在于**CellApp**上的**Entity**:

- `createEntity()`

- 在**Cell**上调用

- 不能被容错

Entity销毁

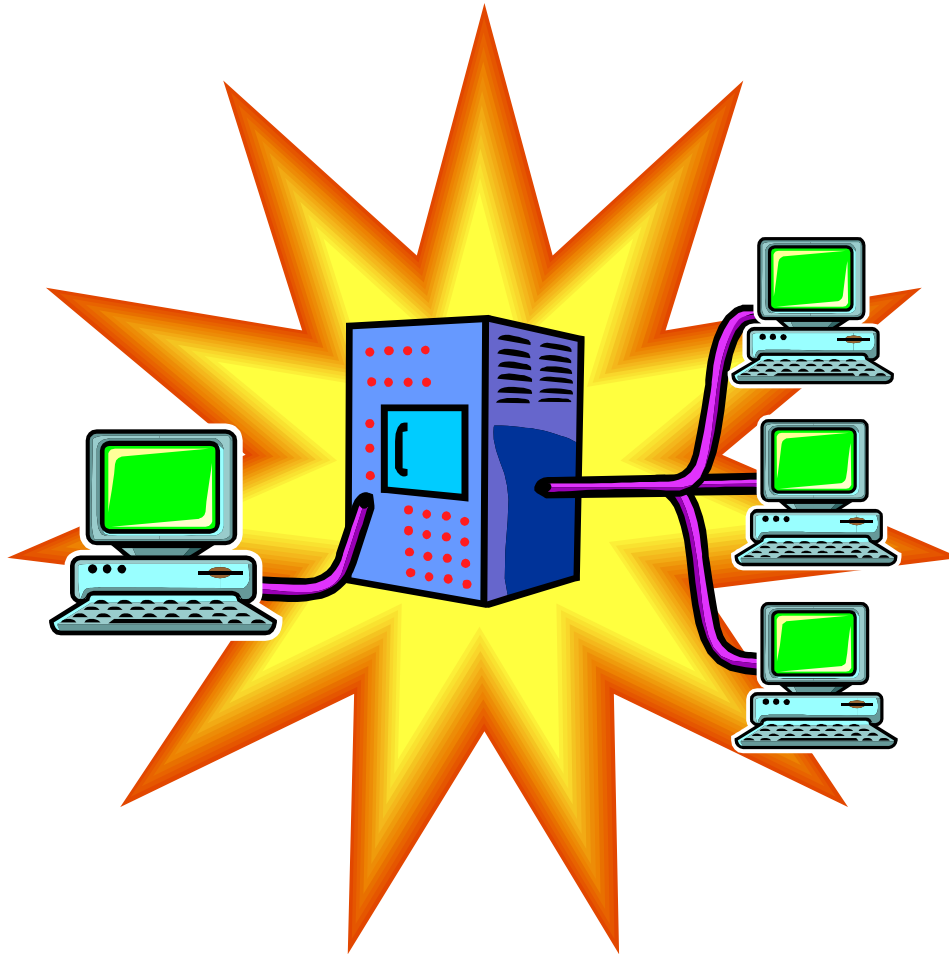
- **Cell entity**作为游戏逻辑的一部分被销毁
- **Base entity**在其**cell**部分还存在之前不能被销毁
- 销毁**cell**部分:
 - **Cell**上: `Entity.destroy()`
 - **Base**上: `Base.destroyCellEntity()`
 - 当**cell**部分销毁时`Base.onLoseCell()`会被调用
- 销毁**base**部分:
 - `Base.destroy()`
 - 如果有持久化属性, 会调用`writeToDB()`

容错

- Cell的属性被备份到base
- Base的属性被备份到另一个BaseApp
- 持久化属性备份到数据库
 - 存档（**archiving**）：持续地轮流进行
- 容错/灾难恢复
 - 灾难 = 同时有多个服务器进程失败

Session 5

Cell功能集



Controller和Entity Extra

- 使用C++来扩展cell entity
- 在对性能要求较高时使用
- 通常会同时使用
- **Controller**
 - 实现那些需要在多个tick中进行后台处理的功能
 - 在结束时回调python脚本
- **EntityExtra**
 - 实现对内部状态/派生状态（derived state）的访问接口
 - 通常用来查询/修改Controller的参数
- 示例： `bigworld/src/examples/cellapp_extension`

Controller 概览

- 用于实现复杂的逻辑
- 使用C/C++来提高性能（相对于script）
- 当entity跨越cell边界时Controller也会被复制
- 对每个entity上的Controller数量没有限制
- 每个实例返回一个Controller ID
 - 删除: `Entity.cancel(id)`
- 能够调用它们的entity脚本中的回调函数

EntityExtra概览

- 从概念上来说，是对**entity**类的扩展
- 同一个类型的**EntityExtra**在一个**entity**上只能有一个
- 作为普通的**entity**方法/属性暴露给Python
- 只在C++或python上需要访问时才会创建
 - 节省数据存储的开销
- 当**entity**跨越cell时不会随之移动
 - 节省数据迁移时的开销
- 创建我们自己的**EntityExtra**:
 - 继承于**EntityExtra**, 位于
`src/server/cellapp/entity_extra.hpp`

Entity cell部分的功能集

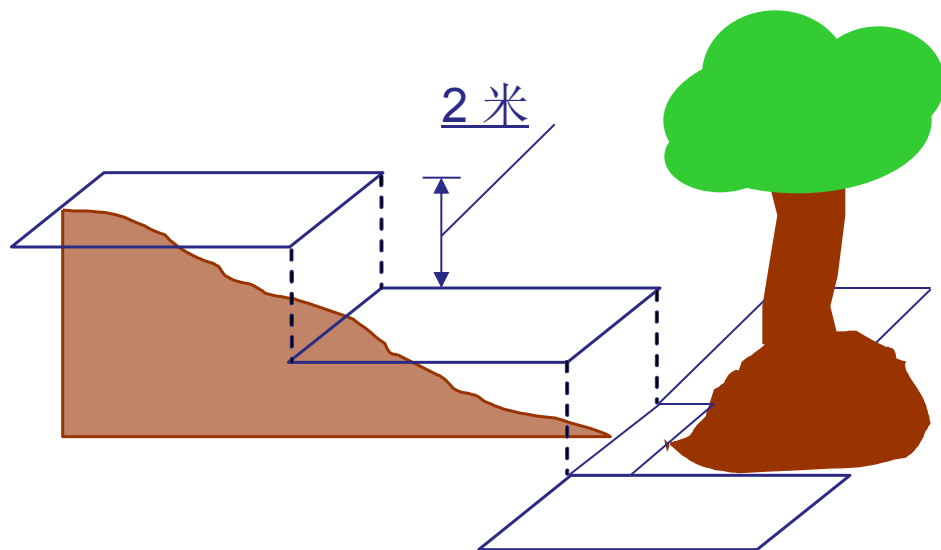
- 在entity的cell部分有许多与空间有关的功能
 - 寻路系统
 - Proximity（陷阱）
 - Vision
 - Rotator
- 这些功能都是用Controller和EntityExtra实现的

Entity导航系统(Navigation)

- 寻路系统提供了许多用于entity移动和寻路的函数
- Navigation Controller使用由NavPoly组成的NavMesh来进行寻路，它们是由NavGen从静态碰撞场景里预先产生的

NavMesh 和 NavPoly

- NavMesh是一个由一些叫做NavPoly的水平多边形组成的图
- Entity在NavPoly的表面移动，随后向上或向下跳跃到下一个NavPoly
- Client端会把entity向下不超过2米的距离使得entity在场景中处于正确的位置



Entity Navigation 方法

- 在直线上移动
 - `moveToPoint()`
 - `moveToEntity()`
- 寻路（使用**NavMesh**）
 - `navigate()`
 - `navigateStep()`
 - `navigateFollow()`
- 通常的
 - `canNavigateTo()`
 - `getStopPoint()`

Entity Proximity

- `ProximityController`实现一个高度无限并与轴平行的立方柱形陷阱
- 应该在陷阱通知函数中进行Y轴的检查
- 一个**entity**可以有很多个**Proximity**陷阱
- 增加一个**Proximity**陷阱:
`Entity.addProximity()`

Entity Vision

- 服务器端的entity也需要视觉
 - 当你接近野猪时，它们会发起攻击
- 两个部分
 - 视野(Vision)
 - 可见性(Visibility)
- 有Vision的entity能（也只能）看见有Visibility的entity

Entity Vision

■ 方法

- ▣ `addVision()`
- ▣ `addScanVision()`
- ▣ `setVisionRange()`
- ▣ `entitiesInView()`

■ 属性

- ▣ `seeingHeight`
- ▣ `visibleHeight`
- ▣ `canBeSeen`
- ▣ `shouldDropVision`

Vehicle 概览

- 任何entity都可以被用作vehicle
- Vehicle是任何可以移动其它entity的entity
- 例如：移动平台，车，马，战舰
 - 移动平台
 - 角色entity在client端应该有脚本根据其位置自动上下vehicle，并且在vehicle上还可以自由移动
 - 车
 - 角色entity应该选取vehicle、进入vehicle，然后移交移动控制来移动它，这些都是在client端进行的
 - 基于client端的动画，角色可以在vehicle上面或者内部进行驾驶，也可以隐身在vehicle上
- vehicle是递归的
 - 角色可能骑着一匹马站在一个位于海中战舰上的移动平台上

Vehicle方法

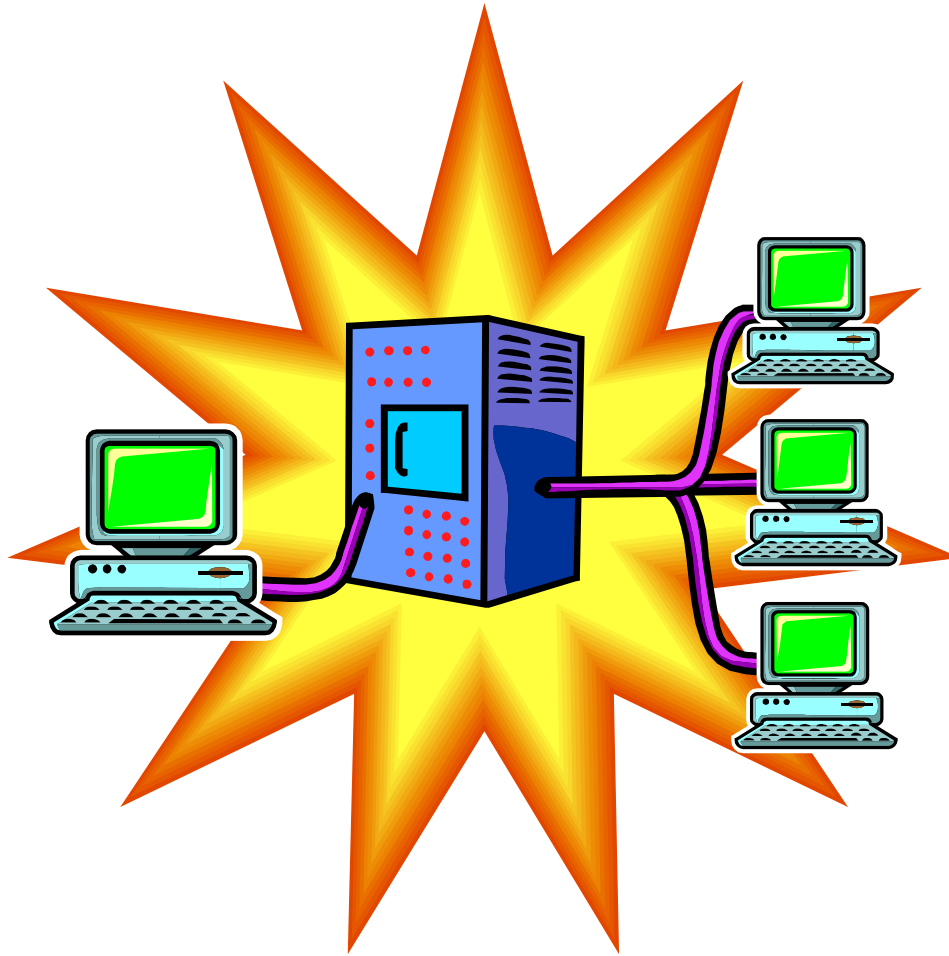
- Passenger EntityExtra为cell entity提供了访问vehicle的接口
 - `self.boardVehicle(vehicleEntityID)`
 - 把entity和vehicle entity关联起来
 - `self.alightVehicle()`
 - 解除entity和当前vehicle entity的关联
 - 在和另一个vehicle entity建立关联之前必须被调用
- 在一个vehicle上仅仅意味着把这两个entity关联起来进行位置和方向的变换
 - 因此，动画，对用户输入的响应，速度等都应该在client上进行
- 角色entity的移动消息包含了它所位于的最里面的vehicle ID及其相对于该vehicle的位置和方向

控制其它的entity

- 包括2个部分：
 - 客户端发送位置更新到新的entity:
`BigWorld.controlEntity()`
 - 服务器接受entity的位置更新: `Entity.controlledBy`
 - 设置成控制该entity的玩家的base mailbox
- 这个entity不能超过控制玩家的AoI范围之外（Proxy entity）
 - 因此，基本上仅适合于玩家所在的vehicle
- 或者，可以从一个玩家转移控制到另一个玩家（(两者都应该有Proxy base部分)）
 - `Proxy.giveClientTo()`
 - `Entity.controlledBy` 会自动地设置给新的玩家
 - 分裂型的 - 销毁并重建AoI，重新加载space

Session 6

BigWorld服务器设置



Server配置

■ **bw.xml** – Server配置文件

- 指定许多server运行时的参数
- 在server资源路径下
- 完整的文档见[Server Operations Guide](#)

■ Personality脚本

- 实现全局的回调函数
- 用BigWorld Python接口处理系统级的消息事件
 - 例如：启动、恢复、关闭
- 缺省情况下CellApp和BaseApp脚本是分离的
- Personality脚本名在**bw.xml**文件里指定。缺省是BWPersonality

Server Personality脚本

- **CellApp Personality**脚本可以在 **onCellAppReady**时设定游戏
 - 见 [示例](#)
 - 用import BigWorld来使用BigWorld提供的函数
 - **BigWorld.addSpaceGeometryMapping(1, None, "spaces/main")**
 - 1 是缺省space, 只有当**bw.xml**里 useDefaultSpace 打开时才有效
 - **None**是可选的、用于改变几何映射的几何变换矩阵
 - 目录路径是一个描述chunk的BigWorld **space.settings**文件的路径
- **BaseApp Personality**脚本可以在 **onBaseAppReady**时设置游戏
 - 如果要创建全局base的话, 可以在这个时候创建
 - 如果不是用缺省space的话应该在这里创建新的space
- 以上两个脚本都必须都必须执行清理工作:
 - 在**onBaseAppShuttingDown** 或 **onCellAppShuttingDown**被调用时
 - **BaseApps**同时还在接近结束的时候接收到**onBaseAppShutDown**消息
- 游戏的状态有可能是从灾难恢复的, 所以注意不要覆盖游戏的状态, 也不要重复地创建全局entity
- **Personality**脚本可以根据需要执行其它的任务
 - 是放置全局游戏脚本的地方, 但不要把所有东西都放在里面
 - 采用模块化的方式, 对每个逻辑单元用分开的脚本文件

加载entity

■ Entity以下列方式加载：

- WorldEditor – 在WorldEditor放置entity，它们在游戏中可以用脚本 `BigWorld.fetchEntitiesFromChunk`来加载
- Personality 脚本 – 典型地用于创建单体的entity
- 数据库 – entity可以被设置成自动地从上次游戏的状态加载
- RunScript/PythonConsole – 在一个正在执行的服务器进程中执行脚本来加载entity

■ BigWorld.fetchEntitiesFromChunk

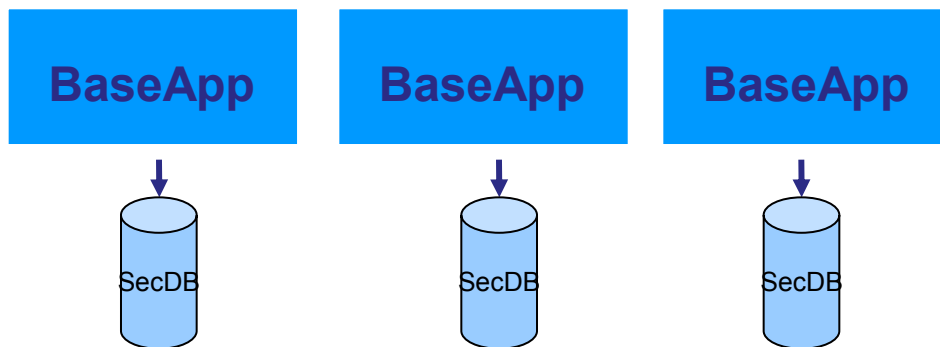
- 用在BaseApp上，用于加载那些用WorldEditor编辑后存储到的chunk文件里的entity
- 当entity的base部分被创建以后，base脚本可以创建其cell部分

二级数据库（Secondary Database）

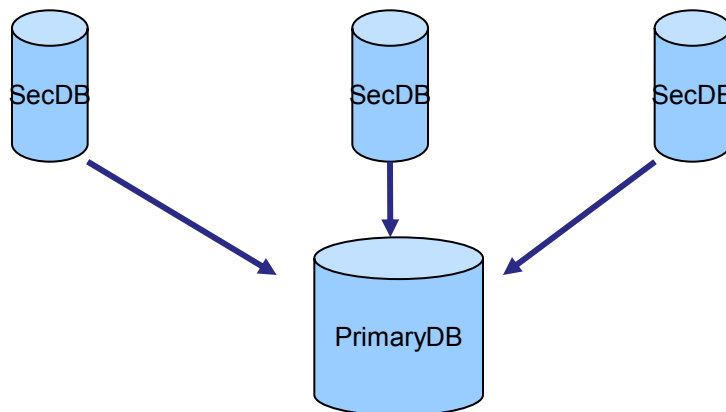
- 为伸缩性而提供的功能
- 数据库通常是最早遇到的瓶颈
- 大部分数据库操作都是写操作

二级数据库

- BaseApp 的写数据库的请求会发送到本地的二级数据库



- 二级数据库在服务器关闭（或下一次启动时）同步到主数据库。



使用Server

- WebConsole

- 基于web的监视界面

- ClusterControl – server进程的管理；可用于查看进程、用户和机器
 - LogViewer – 可用于查看和搜索log输出
 - StatGrapher – 查看由StatLogger搜集的进程和机器数据的统计图
 - PythonConsole – 连接正在运行的进程中的python解释器
 - Commands – 执行预定义的脚本

- ControlCluster (CLI)

- 多功能的工具，可用于开启/关闭服务器、集群查询及察看watcher

- SpaceViewer

- 用于监视一个正在运行的服务器群组中的space、cell和它们中的entity
 - 跨平台（Win32 和 Linux）

- 请参考[Server Operations Guide](#)以获得更多细节

与服务器交互

- 通过RunScript来改变内容
- 可以用control_cluster.py在运行时刻通过ssh来执行Python命令
 - `control_cluster.py pyconsole [process] [--entity {cell|base}:{id}]`
- 用BigWorld Python接口进行交互
(例如: 在BaseApp上)

```
>>> g=BigWorld.createBase( "Guard", position = (2, 3, 5) )
      |-----> Guard.Guard实例      |-----> Entity名字      |-----> 缺省属性值
      |-----> <address>
```

```
>>> g.id
```

```
1234
```

```
■ 在CellApp上:
```

```
■ >>> g = BigWorld.entities[1234]
```

```
■ >>> g.position
```

```
■ (2.000000, 3.000000, 5.000000)
```

- 注意y是在BigWorld里的垂直高度

```
■ dir(g)
```

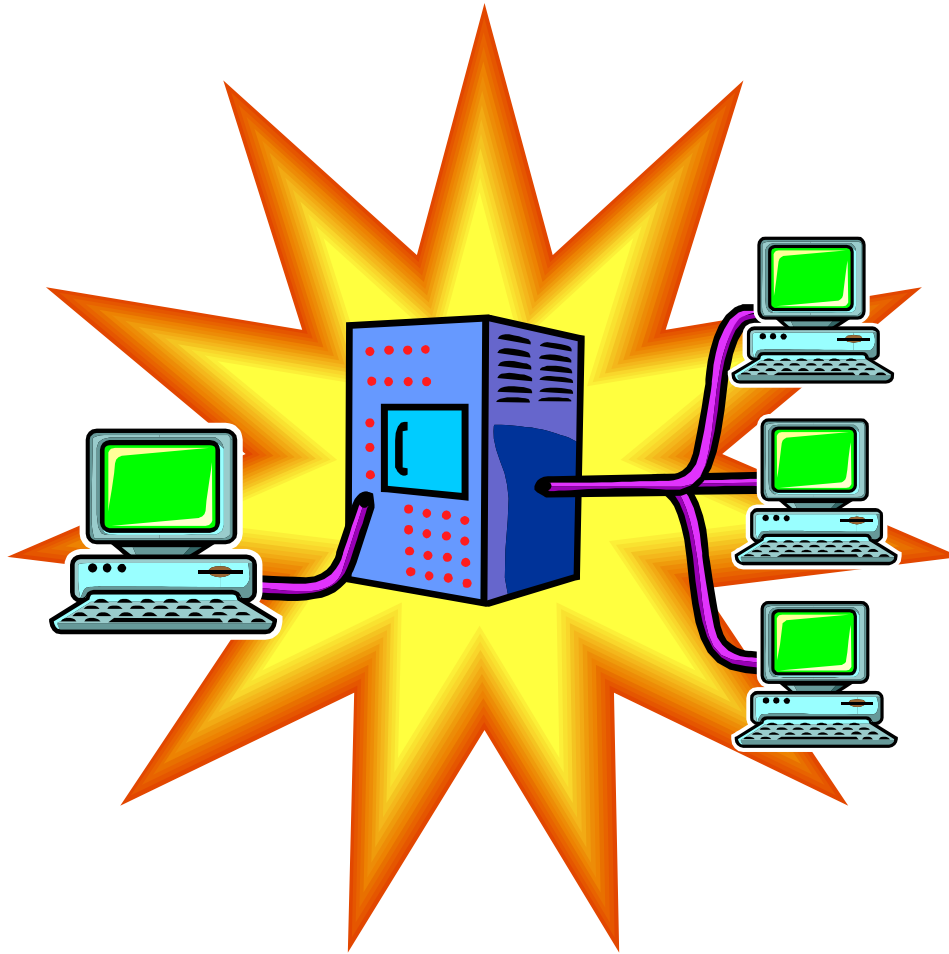
- 很多内建的属性、方法。其中一部分是EntityExtras提供的
- 在entity定义文件中特定于entity的属性和方法

```
■ g.destroy()
```

- 在BaseApp上调用**base.onLoseCell()**，使得entity的base部分能够销毁自己

Session 7

BigWorld 性能分析和压力测试



用机器人做压力测试

- 模拟大量的玩家
- 强烈建议在大规模玩家测试前进行压力测试
- 不会加载地形
- 不支持寻路

Bot 脚本

- 每个类型的entity在**res/scripts/bot**下面都需要一个Python脚本
 - Bot脚本应该实现entity的client部分
 - 但是因为很多client里用到的UI和3D对象在Bots中不存在，所以简单地复制client脚本通常不可行
 - 对大多数entity类型，只需实现一个空的class
 - 对Account和Player entity，需要编写脚本来进行登录以及对玩家进行模拟
 - 编写A.I.程序来模拟一个玩家工作量可能会很大

添加bot

- 运行Bots进程，并使用WebConsole来添加bot
- 或者用
bigworld/tools/server/bot_op.py 来自
动地在 一组Bots进程间发布bot

Profiling 工具

- **ControlCluster** 有很多CLI命令可以用来对一个正在运行的服务器群组的各个方面进行性能分析。
- **StatGrapher** 可以显示每个server进程的负载。
- 应该尽早地进行性能分析，确保内部带宽和外部带宽不会被昂贵的方法调用所占据。
 - 同时建议进行网络硬件性能分析，这样可以判断出什么时候网络饱和了。
- 使用性能分析得到的数据来定位需要优化的部分。

Profiling命令

- **eventprofile** – 诊断出消耗最大的方法调用和状态更新。
- **mercurypprofile** – 诊断出占用带宽最大的mercury级别的消息。
- **pyprofile** – 诊断出消耗cpu时间最多的Python函数调用。
- **cprofile** – 诊断出CellApp引擎中消耗cpu时间最多的C++函数调用。

更多参考

- [Server Overview](#) 有关于概念的详细介绍
- [Server Programming Guide](#) 是一个C++编程和Python脚本的参考
- [Server Operations Guide](#) 是一个执行server软件操作的参考

结束语

- 客户端的培训基于对服务器概念的理解
- 谢谢参与

