三层网络通讯

优化后的通讯协议设计

1. 通信结构概述

通信架构采用三层设计，按照数据变化频率和重要性划分：

1. **静态层 (StaticInfo)**
   - 变更频率：低（初始化或重大变更时）
   - 通信端口：普通UDP端口
   - 数据类型：用户基本信息（ID、名称、性别、角色等）
2. **上下文层 (ContextualInfo)**
   - 变更频率：中（每1-2秒）
   - 通信端口：普通UDP端口
   - 数据类型：场景信息、进度、状态等
3. **动态层 (DynamicInfo)**
   - 变更频率：高（每100ms左右）
   - 通信端口：实时UDP端口
   - 数据类型：位置、旋转等实时变化的信息

2. 消息基本结构

所有消息共享以下基本结构：

```
{
  "priority": 1,        // 优先级：1=低频(Static), 2=中频(Contextual), 3=高频(Dynamic)
  "type": "消息类型",    // StaticInfo/ContextualInfo/DynamicInfo
  "data": [             // 数据列表，包含多个客户端的信息
    {
      "id": "客户端ID",
      // 根据消息类型包含不同字段
      // ...
      "message": {      // 可选，包含向客户端发送的指令
        "command": "指令名称",
        "parameters": "指令参数"  // 使用字符串参数简化结构
      }
    }
```

```
  ]
}
```

3. 各层数据结构详细说明

3.1 静态层 (StaticInfo)

包含用户基本信息，变化频率低。

数据结构:

```
{
  "priority": 1,
  "type": "StaticInfo",
  "data": [
    {
      "id": "client123",
      "name": "玩家1",
      "sex": 0,
      "role": "visitor",
      "story": "时空之钥",
      "levels": [1, 2, 3],
      "message": {  // 可选
        "command": "ready",
        "parameters": ""
      }
    }
  ]
}
```

3.2 上下文层 (ContextualInfo)

包含场景上下文信息，变化频率适中。

数据结构:

```
{
  "priority": 2,
  "type": "ContextualInfo",
  "data": [
    {
      "id": "client123",
      "state": "ready",
      "groupId": "group1",
```

```
      "level": 1,
      "timeline": 45.5,
      "timelineRate": 1.0,
      "timelineDuration": 120.0,
      "message": {  // 可选
        "command": "running",
        "parameters": ""
      }
    }
  ]
}
```

3.3 动态层 (DynamicInfo)

包含用户实时位置和旋转信息，变化频率高。

数据结构:

```
{
  "priority": 3,
  "type": "DynamicInfo",
  "data": [
    {
      "id": "client123",
      "pos": [10.0, 1.5, 5.0],
      "rot": [0.0, 45.0, 0.0],
      "leftHandPos": [9.5, 1.2, 5.2],
      "leftHandRot": [10.0, 30.0, 5.0],
      "rightHandPos": [9.5, 1.2, 5.2],
      "rightHandRot": [10.0, 30.0, 5.0],
      "message": {  // 可选
        "command": "abort",
        "parameters": ""
      }
    }
  ]
}
```

4. 客户端发送的消息结构

客户端发送的消息结构简化，只包含单个客户端的信息：

静态信息:

```
{
  "priority": 1,
  "type": "StaticInfo",
  "data": {
    "id": "client123",
    "name": "玩家1",
    "sex": 0,
    "role": "visitor",
    "story": "时空之钥",
    "levels": [1, 2, 3]
  }
}
```

上下文信息:

```
{
  "priority": 2,
  "type": "ContextualInfo",
  "data": {
    "id": "client123",
    "state": "ready",
    "groupId": "group1",
    "level": 1,
    "timeline": 45.5,
    "timelineRate": 1.0,
    "timelineDuration": 120.0
  }
}
```

动态信息:

```
{
  "priority": 3,
  "type": "DynamicInfo",
  "data": {
    "id": "client123",
    "pos": [10.0, 1.5, 5.0],
    "rot": [0.0, 45.0, 0.0],
    "leftHandPos": [9.5, 1.2, 5.2],
    "leftHandRot": [10.0, 30.0, 5.0],
    "rightHandPos": [9.5, 1.2, 5.2],
    "rightHandRot": [10.0, 30.0, 5.0]
  }
}
```

5. 指令系统设计

5.1 指令类型

指令用于服务器向客户端发送控制命令，触发状态变更或场景操作。

主要指令类型:

1. **状态控制指令**:
   - `idle`：将客户端设置为闲置状态
   - `ready`：将客户端设置为准备状态
   - `running`：将客户端设置为运行状态
   - `reconnect`：将客户端设置为重连状态
   - `abort`：将客户端设置为中止状态

5.2 指令参数格式

为简化结构，指令参数使用字符串格式。不同指令的参数格式如下:

1. **基本状态指令** (idle/ready/running/abort):
2. **重连指令** (reconnect):
   - 参数格式: `"client123:1:45.5"` （客户端ID:场景索引:时间线位置）
3. **场景切换指令** (reconnect):
   - 参数格式: `"client123:2:0.0"` （客户端ID:场景索引）

5.3 指令示例

在静态层中负责初始化:

```
{
  "priority": 1,
  "type": "StaticInfo",
  "data": [
    {
      "id": "client123",
      "name": "玩家1",
      "sex": 0,
      "role": "visitor",
      "story": "时空之钥",
      "levels": [1, 2, 3],
      "message": { //可选
        "command": "",
        "parameters": ""
```

```
      }
    }
  ]
}
```

在上下文层中发送指令:

```
{
  "priority": 2,
  "type": "ContextualInfo",
  "data": [
    {
      "id": "client123",
      "state": "ready",
      "groupId": "group1",
      "level": 1,
      "timeline": 0.0,
      "timelineRate": 1.0,
      "timelineDuration": 120.0,
      "message": {
        "command": "ready/running/reconnect/abort/idle",
        "parameters": ""
      }
    }
  ]
}
```

在动态层中发送实时数据:

```
{
  "priority": 3,
  "type": "DynamicInfo",
  "data": [
    {
      "id": "client123",
      "pos": [10.0, 1.5, 5.0],
      "rot": [0.0, 45.0, 0.0],
      "leftHandPos": [9.5, 1.2, 5.2],
      "leftHandRot": [10.0, 30.0, 5.0],
      "rightHandPos": [9.5, 1.2, 5.2],
      "rightHandRot": [10.0, 30.0, 5.0],
      "message": {
        "command": "",
        "parameters": ""
      }
```

```
    }
  ]
}
```

向所有客户端发送指令:

```
{
  "priority": 2,
  "type": "ContextualInfo",
  "data": [
    {
      "id": "client123",
      "state": "ready",
      "groupId": "group1",
      "level": 0,
      "timeline": 0.0,
      "timelineRate": 1.0,
      "timelineDuration": 0.0,
      "message": {
        "command": "ready",
        "parameters": ""
      }
    },
    {
      "id": "client456",
      "state": "idle",
      "groupId": "group1",
      "level": 0,
      "timeline": 0.0,
      "timelineRate": 1.0,
      "timelineDuration": 0.0,
      "message": {
        "command": "ready",
        "parameters": ""
      }
    }
  ]
}
```

重连指令示例:

```
{
  "priority": 2,
  "type": "ContextualInfo",
  "data": [
```

```
    {
      "id": "client123",
      "state": "idle",
      "groupId": "group1",
      "level": 0,
      "timeline": 0.0,
      "timelineRate": 1.0,
      "timelineDuration": 60.0,
      "message": {
        "command": "reconnect",
        "parameters": "client123:2:45.5"
      }
    }
  ]
}
```

6. 通信流程示例

6.1 基本流程

1. **初始状态**:
   - 客户端启动并连接服务器
   - 客户端默认处于Idle状态，不发送信息
2. **准备阶段**:
   - 服务器通过ContextualInfo层向客户端发送ready指令
   - 客户端接收指令，切换到ready状态，开始向服务器发送信息
   - 客户端此时state为ready，服务器的command此时应该为空
3. **运行阶段**:
   - 所有客户端均准备就绪后，服务器发送running指令
   - 客户端接收指令，加载并开始游戏
   - 客户端持续发送位置和状态信息
   - 客户端此时state为running，服务器的command此时应该为空
4. **结束阶段**:
   - 场景结束后
   - 客户端此时state中为leave
   - 服务器需要将command置为idle
   - 客户端返回空闲状态，停止发送信息
   - 服务器接收不到数据后开始删除这条数据
5. **重连/插入阶段**:
   - 默认idle状态，重连和插入调用同一个指令

- 客户端接收指令，加载并运行指定场景
- 进入后客户端state会变成running，服务器command为空

6. **强制退出阶段**:
   - 客户端state可以为任意状态，idle/ready/running/reconnect
   - 服务器发送command为abort
   - 客户端开始退出
   - 客户端此时state中为leave
   - 服务器需要将command置为idle
   - 客户端返回空闲状态，停止发送信息
   - 服务器接收不到数据后开始删除这条数据

6.2 完整通信示例

1. 服务器发送静态信息:

```
{
  "priority": 1,
  "type": "StaticInfo",
  "data": [
    {
      "id": "client123",
      "name": "玩家1",
      "sex": 0,
      "role": "visitor",
      "story": "故事1",
      "levels": [1, 2, 3]
    },
    {
      "id": "client456",
      "name": "玩家2",
      "sex": 1,
      "role": "visitor",
      "story": "故事2",
      "levels": [1, 2, 3]
    }
  ]
}
```

2. 服务器发送ready指令:

```
{
  "priority": 2,
```

```
  "type": "ContextualInfo",
  "data": [
    {
      "id": "client123",
      "state": "idle",
      "groupId": "group1",
      "level": 0,
      "timeline": 0.0,
      "timelineRate": 1.0,
      "timelineDuration": 0.0,
      "message": {
        "command": "ready",
        "parameters": ""
      }
    },
    {
      "id": "client456",
      "state": "idle",
      "groupId": "group1",
      "level": 0,
      "timeline": 0.0,
      "timelineRate": 1.0,
      "timelineDuration": 0.0,
      "message": {
        "command": "ready",
        "parameters": ""
      }
    }
  ]
}
```

3. 客户端发送状态更新:

```
{
  "priority": 2,
  "type": "ContextualInfo",
  "data": {
    "id": "client123",
    "state": "ready",
    "groupId": "group1",
    "level": 0,
    "timeline": 0.0,
    "timelineRate": 1.0,
    "timelineDuration": 0.0
```

```
  }
}
```

4. 服务器广播动态信息:

```
{
  "priority": 3,
  "type": "DynamicInfo",
  "data": [
    {
      "id": "client123",
      "pos": [10.0, 1.5, 5.0],
      "rot": [0.0, 45.0, 0.0],
      "leftHandPos": [9.5, 1.2, 5.2],
      "leftHandRot": [10.0, 30.0, 5.0],
      "rightHandPos": [9.5, 1.2, 5.2],
      "rightHandRot": [10.0, 30.0, 5.0]
    },
    {
      "id": "client456",
      "pos": [15.0, 1.5, 8.0],
      "rot": [0.0, 90.0, 0.0],
      "leftHandPos": [9.5, 1.2, 5.2],
      "leftHandRot": [10.0, 30.0, 5.0],
      "rightHandPos": [9.5, 1.2, 5.2],
      "rightHandRot": [10.0, 30.0, 5.0]
    }
  ]
}
```

5. 服务器发送running指令:

```
{
  "priority": 2,
  "type": "ContextualInfo",
  "data": [
    {
      "id": "client123",
      "state": "ready",
      "groupId": "group1",
      "level": 1,
      "timeline": 0.0,
      "timelineRate": 1.0,
      "timelineDuration": 60.0,
      "message": {
```

```
        "command": "running",
        "parameters": ""
      }
    },
    {
      "id": "client456",
      "state": "ready",
      "groupId": "group1",
      "level": 1,
      "timeline": 0.0,
      "timelineRate": 1.0,
      "timelineDuration": 60.0,
      "message": {
        "command": "running",
        "parameters": ""
      }
    }
  ]
}
```

6. 客户端更新状态:

```
{
  "priority": 2,
  "type": "ContextualInfo",
  "data": {
    "id": "client123",
    "state": "running",
    "groupId": "group1",
    "level": 1,
    "timeline": 0.0,
    "timelineRate": 1.0,
    "timelineDuration": 60.0
  }
}
```

7. 结束时发送idle指令:

```
{
  "priority": 2,
  "type": "ContextualInfo",
  "data": [
    {
      "id": "client123",
      "state": "running",
```

```
      "groupId": "group1",
      "level": 1,
      "timeline": 60.0,
      "timelineRate": 1.0,
      "timelineDuration": 60.0,
      "message": {
        "command": "idle",
        "parameters": ""
      }
    },
    {
      "id": "client456",
      "state": "running",
      "groupId": "group1",
      "level": 1,
      "timeline": 60.0,
      "timelineRate": 1.0,
      "timelineDuration": 60.0,
      "message": {
        "command": "idle",
        "parameters": ""
      }
    }
  ]
}
```


7. 状态机设计与处理

客户端内部维护一个状态机，根据接收到的指令进行状态转换。

7.1 状态定义

- **Idle**: 空闲状态，不发送数据
- **Ready**: 准备状态，开始发送数据
- **Running**: 运行状态，场景运行中
- **Reconnect**: 重连状态，插入特定场景和时间点
- **Abort**: 中止状态，中断当前操作

7.2 状态转换逻辑

状态转换逻辑如下:

当前为Idle状态时:

- 收到Ready指令 → Ready状态，开始发送信息
- 收到Reconnect指令 → Reconnect状态，插入指定场景
- 收到Abort指令 → Abort状态

当前为Ready状态时:

- 收到Running指令 → Running状态，加载场景
- 收到Reconnect指令 → Reconnect状态，插入指定场景
- 收到Abort指令 → Abort状态

当前为Running状态时:

- 收到Abort指令 → Abort状态，离开场景

当前为Reconnect状态时:

- 收到Running指令 → Running状态

8. 通信优化建议

1. **批量处理**:
   - 服务器可以批量收集客户端信息，定期广播
   - 静态信息和上下文信息可以在同一个端口以不同频率发送
2. **差异化更新**:
   - 只发送有变化的数据，减少网络流量
   - 动态信息可以只包含位置和旋转变化大的客户端
3. **指令优先**:
   - 包含指令的消息可以提高优先级立即发送
   - 重要状态变更可以通过多个通道同时发送以确保到达
4. **错误处理**:
   - 添加消息序列号，处理网络丢包情况
   - 实现心跳机制，及时检测连接状态