后端核心代码实现

1. 数据模型定义

Card.java（卡牌模型）

```java
package com.poker.model;

public class Card {
    // 花色：0-黑桃，1-红桃，2-方块，3-梅花
    private int suit;
    // 点数：2-14（A=14，K=13，Q=12，J=11，T=10）
    private int rank;

    public Card(int suit, int rank) {
        this.suit = suit;
        this.rank = rank;
    }

    // Getters and Setters
    public int getSuit() { return suit; }
    public int getRank() { return rank; }

    // 卡牌字符串表示（如"黑桃A"）
    public String toString() {
        String[] suits = {"黑桃", "红桃", "方块", "梅花"};
        String[] ranks = {"", "", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J",
"Q", "K", "A"};
        return suits[suit] + ranks[rank];
    }
}
```

Player.java（玩家模型）

```
package com.poker.model;

import java.util.ArrayList;
import java.util.List;

public class Player {
```

```java
    private String id;
    private String name;
    private int chips;
    private List<Card> holeCards = new ArrayList<>();
    private boolean isAi;
    private boolean isInGame;
    private int currentBet;

    // 构造函数、Getters和Setters
    public Player(String id, String name, int chips, boolean isAi) {
        this.id = id;
        this.name = name;
        this.chips = chips;
        this.isAi = isAi;
        this.isInGame = true;
    }

    // 下注操作
    public void bet(int amount) {
        if (amount <= chips) {
            chips -= amount;
            currentBet += amount;
        }
    }

    // 重置当前回合下注
    public void resetCurrentBet() {
        currentBet = 0;
    }

    // 其他方法...
}
```

2. 游戏核心服务类

GameService.java

```
package com.poker.service;

import com.poker.model.Card;
import com.poker.model.Player;
import org.springframework.stereotype.Service;
```

```java
import java.util.*;

@Service
public class GameService {
    private List<Card> deck = new ArrayList<>();
    private List<Player> players = new ArrayList<>();
    private List<Card> communityCards = new ArrayList<>();
    private int currentDealer;
    private int currentPlayerTurn;
    private int pot = 0;
    private int currentBetAmount = 0;

    // 初始化 deck
    private void initializeDeck() {
        deck.clear();
        for (int suit = 0; suit < 4; suit++) {
            for (int rank = 2; rank <= 14; rank++) {
                deck.add(new Card(suit, rank));
            }
        }
        Collections.shuffle(deck);
    }

    // 开始新游戏
    public void startGame(List<Player> players) {
        this.players = players;
        initializeDeck();
        communityCards.clear();
        pot = 0;
        currentDealer = (currentDealer + 1) % players.size();
        dealHoleCards();
        // 小盲注和大盲注逻辑...
    }

    // 发底牌
    private void dealHoleCards() {
        for (Player player : players) {
            if (player.isInGame()) {
                player.getHoleCards().add(deck.remove(0));
                player.getHoleCards().add(deck.remove(0));
            }
        }
    }

    // 翻牌（发3张公共牌）
    public void flop() {
        burnCard(); // 弃一张牌
```

```
            for (int i = 0; i < 3; i++) {
                communityCards.add(deck.remove(0));
            }
        }

        // 转牌（发第4张公共牌）
        public void turn() {
            burnCard();
            communityCards.add(deck.remove(0));
        }

        // 河牌（发第5张公共牌）
        public void river() {
            burnCard();
            communityCards.add(deck.remove(0));
        }

        // 弃牌
        private void burnCard() {
            deck.remove(0);
        }

        // 处理玩家下注
        public void playerAction(String playerId, String action, int amount) {
            Player player = findPlayerById(playerId);
            switch (action) {
                case "fold":
                    player.setInGame(false);
                    break;
                case "call":
                    int callAmount = currentBetAmount -
player.getCurrentBet();
                    player.bet(callAmount);
                    pot += callAmount;
                    break;
                case "raise":
                    int raiseAmount = amount - player.getCurrentBet();
                    player.bet(amount);
                    pot += amount;
                    currentBetAmount = amount;
                    break;
                // 其他动作...
            }
            nextPlayerTurn();
        }
```

```java
    // 其他辅助方法...
}
```

3. WebSocket配置

WebSocketConfig.java

```java
package com.poker.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.socket.config.annotation.EnableWebSocket;
import
org.springframework.web.socket.config.annotation.WebSocketConfigurer;
import
org.springframework.web.socket.config.annotation.WebSocketHandlerRegist
ry;

@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        registry.addHandler(new GameWebSocketHandler(), "/ws/game")
                .setAllowedOrigins("*");
    }
}
```

GameWebSocketHandler.java

```
package com.poker.config;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.poker.service.GameService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.socket.TextMessage;
import org.springframework.web.socket.WebSocketSession;
import org.springframework.web.socket.handler.TextWebSocketHandler;

import java.util.HashMap;
```

```java
import java.util.Map;

public class GameWebSocketHandler extends TextWebSocketHandler {
    private Map<String, WebSocketSession> sessions = new HashMap<>();
    @Autowired
    private GameService gameService;
    private ObjectMapper objectMapper = new ObjectMapper();

    @Override
    public void afterConnectionEstablished(WebSocketSession session) throws Exception {
        sessions.put(session.getId(), session);
        // 发送游戏状态更新
    }

    @Override
    protected void handleTextMessage(WebSocketSession session, TextMessage message)
throws Exception {
        // 解析客户端消息（如玩家动作）
        Map<String, Object> action =
objectMapper.readValue(message.getPayload(), Map.class);
        String playerId = (String) action.get("playerId");
        String actionType = (String) action.get("action");
        int amount = (int) action.get("amount");

        // 处理游戏逻辑
        gameService.playerAction(playerId, actionType, amount);

        // 广播游戏状态更新
        broadcastGameState();
    }

    private void broadcastGameState() throws Exception {
        // 构建游戏状态数据
        Map<String, Object> gameState = new HashMap<>();
        // ... 填充数据

        String json = objectMapper.writeValueAsString(gameState);
        for (WebSocketSession s : sessions.values()) {
            s.sendMessage(new TextMessage(json));
        }
    }
}
```