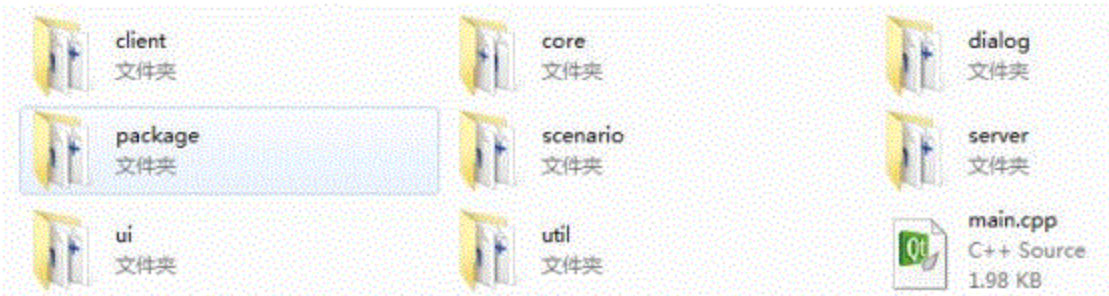
太阳神三国杀是用QT4及C++编写的,QT这个类库提供给我们很多好用的结构类型,如QList, QMap, QSet 等等。它对于图形界面的排版有一套自己的架构，不仅灵活而且可以根据窗口的变化自动适应新的排版。对于有 C++基础的同学来说，QT 其实只是一个类库，直接拿来用就可以了，最多就是多了信号和槽 （Signal & Slot）的概念。QT 通过 connect 函数把信号和槽、信号和信号连接起来，形成一个事件触发的网络，来区分对不同控件执行的不同动作，并且进行相应的处理。作为预备知识，同学们可以找一些 QT 编程的书看一下，相信很快就可以掌握。

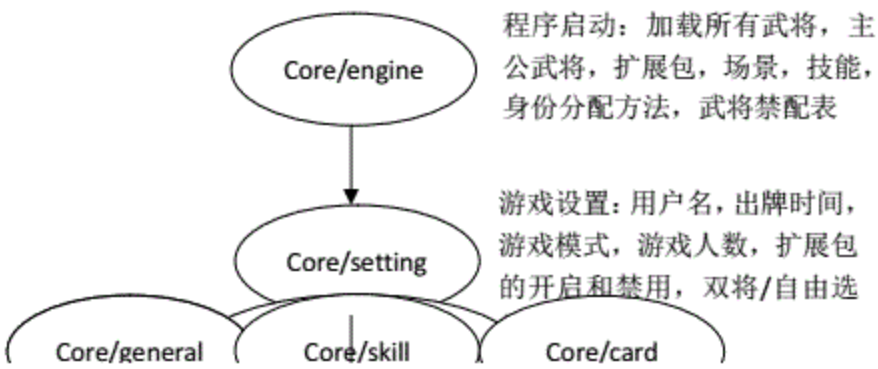
下面我们开始第一篇：太阳神三国杀的代码结构

首先，我们需要获得开发环境和源代码，这个神上已经提供了教程，详情请看：
http://qsanguosha.com/forum.php?mod=viewthread&tid=755&extra=page%3D1(Window平台)
http://qsanguosha.com/forum.php?mod=viewthread&tid=1760&extra=page%3D1(Linux平台)
http://qsanguosha.com/forum.php?mod=viewthread&tid=5397&extra=page%3D1(MacOS平台)

完成教程上的所有步骤后，我们可以看一下程序的源代码，包含下图几个文件夹



Client:客户端相关代码，没太弄清楚，但是不影响 DIY 武将开发
Core:核心类，包括引擎、卡牌、技能、角色、设置、玩家部分的代码，我们首先要读懂的就是这部分，为了实现自己的功能可稍作修改，但不宜改变太多
Dialog:游戏界面相关代码，如果只是 DIY 武将的话，这部分可以忽略
Package:顾名思义，各个包的代码，所有的武将技能，卡牌效果都在这里，也是我们主要编写的部分
Scenario:场景模式，玩过神杀的朋友们可能知道官渡之战、樊城之战这些有时间触发的玩法。场景模式就是用来做这个的，目前我也没太细读，算是下一步学习的内容吧
Server:游戏房间，服务器相关代码，所有事件（判定、拼点、翻面、伤害、回复、卡牌使用）都在其中的 room 类里执行
UI: 应该是一些空间的美化和贴图什么的，如果只是 DIY 武将的话，这部分可以忽略
Util:游戏日志、socket 网络套接字部分的代码，比较底层，可以忽略
Main.cpp:这个不用解释了吧。。。主函数代码，程序入口
所以，如果想加入 DIY 武将，我们需要读懂 Core、Server 和 Package 部分的代码。总的来说，我们在 DIY 过程中需要了解的几个类和结构体的关系如下：



下面我们开始第二篇：太阳神三国杀的核心类的研读—general 类

最基本的类要数武将的 general 类和卡牌的 card 类了，今天先说说武将类
General 类定义如下：

```
class General : public QObject
{
    Q_OBJECT
    Q_ENUMS(Gender);
    Q_PROPERTY(QString kingdom READ getKingdom CONSTANT)
    Q_PROPERTY(int maxhp READ getMaxHp CONSTANT)
    Q_PROPERTY(bool male READ isMale STORED false CONSTANT)
    Q_PROPERTY(bool female READ isFemale STORED false CONSTANT)
    Q_PROPERTY(Gender gender READ getGender CONSTANT)
    Q_PROPERTY(bool lord READ isLord CONSTANT)
    Q_PROPERTY(bool hidden READ isHidden CONSTANT)
public:
    explicit General(Package *package, const QString &name, const QString &kingdom, int
max_hp = 4, bool male = true, bool hidden = false);
    // property getters/setters
    int getMaxHp() const;
    QString getKingdom() const;
    bool isMale() const;
    bool isFemale() const;
    bool isNeuter() const;
    bool isLord() const;
    bool isHidden() const;
    enum Gender {Male, Female, Neuter};
    Gender getGender() const;
    void setGender(Gender gender);
    void addSkill(Skill* skill);
    void addSkill(const QString &skill_name);
    bool hasSkill(const QString &skill_name) const;
    QList<const Skill *> getVisibleSkillList() const;
    QSet<const Skill *> getVisibleSkills() const;
    QSet<const TriggerSkill *> getTriggerSkills() const;
    QString getPixmapPath(const QString &category) const;
    QString getPackage() const;
    QString getSkillDescription() const;
    static QSize BigIconSize;
    static QSize SmallIconSize;
    static QSize TinyIconSize;
public slots:
    void lastWord() const;
private:
```

```
    QString kingdom;
    int max_hp;
    //Gender gender;
    bool lord;
    QSet<QString> skill_set;
    QSet<QString> extra_set;
//Joshua
public:
    mutable Gender gender;
private:
    bool hidden;
public:
    mutable bool genderChanged; //Joshua
};
```

首先看第一段

```
    Q_OBJECT
    Q_ENUMS(Gender);
    Q_PROPERTY(QString kingdom READ getKingdom CONSTANT)
    Q_PROPERTY(int maxhp READ getMaxHp CONSTANT)
    Q_PROPERTY(bool male READ isMale STORED false CONSTANT)
    Q_PROPERTY(bool female READ isFemale STORED false CONSTANT)
    Q_PROPERTY(Gender gender READ getGender CONSTANT)
    Q_PROPERTY(bool lord READ isLord CONSTANT)
    Q_PROPERTY(bool hidden READ isHidden CONSTANT)
```

在第一篇中我说过，QT 较之普通 C++最大的特色就是信号和槽的消息传送机制。如果需要用到信号和槽，在类的一开始需要加上宏 Q_OBJECT。

第二行是对于性别的枚举，Gender 的具体定义为 enum Gender {Male, Female, Neuter}; 男女不必多说，第三个中性是神上最近加上的，可能是一些新功能扩展的需要，目前没发现与之对应的技能。

再下面就是武将一些属性的声明了，包括国籍，血上限，是否为男/女性，性别，是否为主公，是否被隐藏。相信除了隐藏其他都不难理解，在选将过程中，有些将是不能选的，比如一些未完成的武将，以及虎牢关吕布等等，这就需要我们把这个将隐藏掉。

----------小贴士----------

宏 Q_PROPERTY 语法如下：

Q_PROPERTY(type name READ getFunction [WRITE setFunction][RESET resetFunction][DESIGNABLE bool][SCRIPTABLE bool][STORED bool])

宏 Q_PROPERTY 说明如下：

type name 是属性的类型及名字，它可以是一个 QVariant 支持的类型或者在类中已经定义的枚举类型。枚举类型必须使用 Q_ENUMS 宏来进行注册。

READ getFunction 表示用于读取属性的函数是 getFunction。

WRITE setFunction 表示用于写（或设置）属性的函数是 setFunction。

RESET resetFunction 表示用函数 resetFunction 设置属性到缺省状态（这个缺省状态可能和初始状态不同）。这个函数必须返回 void 并且不带有参数。

DESIGNABLE bool 声明这个属性是否适合被一个图形用户界面设计工具修改。Bool 缺省为 TRUE，说明这个属性可写，否则，FALSE 说明不能被图形用户界面设计工具修改。

SCRIPTABLE bool 声明这个属性是否适合被一个脚本引擎访问。Bool 缺省为 TRUE，说明可以被访问。

STORED bool 声明这个属性的值是否必须作为一个存储的对象状态而被记住。STORED 只对可写的属性有意义。缺省是 TRUE。

下面看 public 函数

```
explicit General(Package *package, const QString &name, const QString &kingdom, int max_hp = 4, bool male = true, bool hidden = false);
// property getters/setters
int getMaxHp() const;
QString getKingdom() const;
bool isMale() const;
bool isFemale() const;
bool isNeuter() const;
bool isLord() const;
bool isHidden() const;
enum Gender {Male, Female, Neuter};
Gender getGender() const;
void setGender(Gender gender);
void addSkill(Skill* skill);
void addSkill(const QString &skill_name);
bool hasSkill(const QString &skill_name) const;
QList<const Skill *> getVisibleSkillList() const;
QSet<const Skill *> getVisibleSkills() const;
QSet<const TriggerSkill *> getTriggerSkills() const;
QString getPixmapPath(const QString &category) const;
QString getPackage() const;
QString getSkillDescription() const;
```

第一条是构造函数，使用 explicit 是为了阻止不应该允许的经过转换构造函数进行的隐式转换的发生，参数分别为扩展包名，武将名，国籍，血上限，是否为男性，是否隐藏。

下面的几个函数，从 int getMaxHp() 到 void setGender(Gender gender)不用说了，都是一行代码就能实现的功能。

addSkill(Skill* skill)和 addSkill(const QString &skill_name)都可以给一个武将添加技能，有所不同的是前者适用于第一次添加，后者适用于以后的添加。举个例子，程序的编译是有顺序的，而且各个包的编译应该是按照时间先后完成的，我们知道火包的庞德有马术（标包技能），山包的姜维有观星（标包技能），山包的刘禅有激将（标包技能），山包的孙策有英姿（标包技能）和英魂（林包技能）。因此，在给老诸葛添加观星技能时应该使用前者，给姜维觉醒添加观星时应该使用后者。

hasSkill(const QString &skill_name) 很简单，用来看武将有没有某技能。

getVisibleSkillList()，getVisibleSkills()这两个函数返回可见技能的 QList 和 QSet，除了返回值类型，其他都一样。目前我不太了解神上为什么会这么做，但感觉应该是 QList 和 QSet 各有处理上的优点，这样可能比较方便。基本上所有技能都是可见的，不过左慈的化身和老诸葛的空城例外。化身这个技能其实是由 huashen，huashen-begin，huashen-end 共同实现的，但是游戏时我们只看到一个化身按钮，这是因为 huashen-begin 和 huashen-end 不是 visible 的。同理，空城是由 kongcheng 和 kongcheng-effect 共同实现的，后者也不是 visible 的。

getTriggerSkills()返回触发技能的 QSet，触发技能，顾名思义是由某个事件触发后才能发动的，简单点说就是不能主动发动的。典型的触发技能有：魏国一干卖血流，集智，枭姬，全军突击等等。非触发技能有：武圣，倾国等卡牌转化技能，仁德，驱虎，强袭等主动发动的技能。

getPixmapPath(const QString &category) 获取武将图片路径，没什么说的。

getPackage()，getSkillDescription()用来获取武将所处的扩展包及其技能描述，也没什么说的。

然后是一个槽 Slot

```
void lastWord() const;
```

槽也是一个函数，只不过只有与其相连的信号被发出后才能执行。这个槽应该是和玩家死亡的事件相连的，它用来播放武将死亡时的最后一句话。……“这就是桃园么。。。”

最后看一下 general 类的成员变量（注释 Joshua 以下的是我修改过的）

```
private:
    QString kingdom;
    int max_hp;
    //Gender gender;
    //bool hidden;
    bool lord;
    QSet<QString> skill_set;
    QSet<QString> extra_set;
//Joshua
public:
    mutable Gender gender;
private:
    bool hidden;
public:
    mutable bool genderChanged;
```

QString kingdom, int max_hp, Gender gender, bool hidden, bool lord 相信大家已经了解了，无需多言。

skill_set 就通过 addSkill(Skill* skill)添加的武将技能，extra_set 是通过 addSkill(const QString &skill_name) 添加的武将技能。

至于注释 Joshua 下的部分，我们所添加的武将在原始代码中都是 const General*类型的，因此如果我们不加上 mutable 关键字并把变量放在 public 下，是无法对 const 的数据类型进行修改的。本来我想将原始代码中的 const 全部去掉，但工程实在浩大，并且可能出

其他问题，只好上网搜，没想到出来个 mutable，功能和 const 正好相反。学 C 语言的时候想必大家就觉得 const 这个东西很恶心了，每次对 const 变量修改都会报错。但如果我们想修改 const 类中的成员变量呢？我们可以将想要修改的成员变量加上 mutable 关键字，这样就可以对 const 的类中的成员变量进行修改了。

mutable bool genderChanged 这个变量完全是为了修复吴健这个武将的 bug 而加上的。举个例子，八人局中，吴健的【杀】对孙权造成了伤害，并发动了“阉割”技能，是孙权变为女性。这局游戏结束后，我们没有关掉游戏，而是开始玩新的一局，此时正好又有孙权，你会发现他是女的。。。这是因为第一局游戏结束时我们没有把被阉割掉的角色改回男性。于是我加上了这个变量，当角色被阉割时 genderChanged 置 true，游戏结束时遍历所有武将，如果 genderChanged 为 true，就把他变回男性。

其实神上在倚天包里有一个叫陆伯言的角色可以每回合男女互换，不过他的性别转换机制是用 transfigure 来换成一个新武将。也就是说，陆伯言是两个武将，一男一女，只不过每回合互换罢了。对于可以改自己性别的武将，这样做没什么问题。但是对于吴健这个吴健这个可以改变人性别的武将来说，我不可能对所有男性武将都加一个女性版本。因此如果想实现阉割这个技能，并且保证游戏正常运行，我必须对 general 这个核心类做一些修改。

第三篇：太阳神三国杀的核心类的研读—card 类

这篇继续介绍太阳神三国杀的核心类

Card 类定义如下：

```
class Card : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString suit READ getSuitString CONSTANT)
    Q_PROPERTY(bool red READ isRed STORED false CONSTANT)
    Q_PROPERTY(bool black READ isBlack STORED false CONSTANT)
    Q_PROPERTY(int id READ getId CONSTANT)
    Q_PROPERTY(int number READ getNumber WRITE setNumber)
    Q_PROPERTY(QString number_string READ getNumberString CONSTANT)
    Q_PROPERTY(QString type READ getType CONSTANT)
    Q_PROPERTY(QString pixmap_path READ getPixmapPath)
    Q_PROPERTY(bool target_fixed READ targetFixed)
    Q_PROPERTY(bool once READ isOnce CONSTANT)
    Q_PROPERTY(bool mute READ isMute CONSTANT)
    Q_PROPERTY(bool equipped READ isEquipped)
    Q_ENUMS(Suit)
    Q_ENUMS(CardType)
public:
    // enumeration type
    enum Suit {Spade, Club, Heart, Diamond, NoSuit};
    static const Suit AllSuits[4];
    // card types
    enum CardType{
        Skill,
        Basic,
        Trick,
        Equip,
    };
    // constructor
    Card(Suit suit, int number, bool target_fixed = false);
    // property getters/setters
    QString getSuitString() const;
    bool isRed() const;
    bool isBlack() const;
    int getId() const;
    void setId(int id);
    int getEffectiveId() const;
    QString getEffectIdString() const;
    int getNumber() const;
    void setNumber(int number);
```

```
    QString getNumberString() const;
    Suit getSuit() const;
    void setSuit(Suit suit);
    bool sameColorWith(const Card *other) const;
    bool isEquipped() const;
    QString getPixmapPath() const;
    QString getIconPath() const;
    QString getPackage() const;
    QIcon getSuitIcon() const;
    QString getFullName(bool include_suit = false) const;
    QString getLogName() const;
    QString getName() const;
    QString getSkillName() const;
    void setSkillName(const QString &skill_name);
    QString getDescription() const;
    QString getEffectPath() const;
    bool isVirtualCard() const;
    virtual bool match(const QString &pattern) const;
    void addSubcard(int card_id);
    void addSubcard(const Card *card);
    QList<int> getSubcards() const;
    void clearSubcards();
    QString subcardString() const;
    void addSubcards(const QList<CardItem *> &card_items);
    int subcardsLength() const;
    virtual QString getType() const = 0;
    virtual QString getSubtype() const = 0;
    virtual CardType getTypeId() const = 0;
    virtual QString toString() const;
    virtual QString getEffectPath(bool is_male) const;
    bool isNDTrick() const;
    // card target selection
    virtual bool targetFixed() const;
    virtual bool targetsFeasible(const QList<const Player *> &targets, const Player *Self) const;
    virtual bool targetFilter(const QList<const Player *> &targets, const Player *to_select, const
Player *Self) const;
    virtual bool isAvailable(const Player *player) const;
    virtual const Card *validate(const CardUseStruct *card_use) const;
    virtual const Card *validateInResposing(ServerPlayer *user, bool *continuable) const;
    bool isOnce() const;
    bool isMute() const;
    bool willThrow() const;
    virtual void onUse(Room *room, const CardUseStruct &card_use) const;
```

```
    virtual void use(Room *room, ServerPlayer *source,   const QList<ServerPlayer *> &targets)
const;
    virtual void onEffect(const CardEffectStruct &effect) const;
    virtual bool isCancelable(const CardEffectStruct &effect) const;
    virtual void onMove(const CardMoveStruct &move) const;
    // static functions
    static bool CompareByColor(const Card *a, const Card *b);
    static bool CompareBySuitNumber(const Card *a, const Card *b);
    static bool CompareByType(const Card *a, const Card *b);
    static const Card *Parse(const QString &str);
    static Card * Clone(const Card *card);
    static QString Suit2String(Suit suit);
    static QString Number2String(int number);
    static QStringList IdsToStrings(const QList<int> &ids);
    static QList<int> StringsToIds(const QStringList &strings);
protected:
    QList<int> subcards;
    bool target_fixed;
    bool once;
    QString skill_name;
    bool mute;
    bool will_throw;
private:
    Suit suit;
    int number;
    int id;
};
```

很庞大是吧，不过大家不要被吓到，且听我细细道来，先看属性定义

```
    Q_OBJECT
    Q_PROPERTY(QString suit READ getSuitString CONSTANT)
    Q_PROPERTY(bool red READ isRed STORED false CONSTANT)
    Q_PROPERTY(bool black READ isBlack STORED false CONSTANT)
    Q_PROPERTY(int id READ getId CONSTANT)
    Q_PROPERTY(int number READ getNumber WRITE setNumber)
    Q_PROPERTY(QString number_string READ getNumberString CONSTANT)
    Q_PROPERTY(QString type READ getType CONSTANT)
    Q_PROPERTY(QString pixmap_path READ getPixmapPath)
    Q_PROPERTY(bool target_fixed READ targetFixed)
    Q_PROPERTY(bool once READ isOnce CONSTANT)
    Q_PROPERTY(bool mute READ isMute CONSTANT)
    Q_PROPERTY(bool equipped READ isEquipped)
    Q_ENUMS(Suit)
    Q_ENUMS(CardType)
```

从上到下分别是花色、是否为红、是否为黑、卡牌 ID、点数、点数字符串（主要因为 JQKA）、卡牌类型、图片路径，这些不用解释。下面几个属性乍一看不太清楚是干什么的，我们需要在代码中晚些具体分析。

Suit：花色，enum Suit {Spade, Club, Heart, Diamond, NoSuit}，其中 NoSuit 为无色，比如夏侯渊的神速，凌统的旋风打出的杀。

CardType：卡牌类型，enum CardType{Skill, Basic, Trick, Equip}，第一个是技能牌，后三个是基本牌、锦囊牌和装备牌。何为技能牌？技能牌就是武将发动武将技时使用的牌，比如貂蝉离间的弃牌、孙权制衡换的牌、华佗青囊用的牌、荀彧驱虎拼点的牌等等。

target_fixed：默认为 false。对于技能卡牌来说，target_fixed 为 true 的技能有：制衡、苦肉、鬼才、鬼道、乱武、急袭、陷阵、心战、神愤、极略、殉志（倚天姜伯约）、义舍（倚天张公祺）。其他卡牌有闪、桃、无中生有、天灾卡牌和屎。表示不需要手动选择目标，直接使用即可。（至于急袭算一个例外，请看下图）

AI-太阳神上<moligaloo@163.com> 14:47:55
表示不需要手动选择目标
AI-Joshua1989(276364504) 14:48:04
原来如此
AI-太阳神上<moligaloo@163.com> 14:48:18
这些你不需要指定目标的
AI-太阳神上<moligaloo@163.com> 14:48:24
直接用就行了
AI-Joshua1989(276364504) 14:48:16
急袭不需要么？
AI-太阳神上<moligaloo@163.com> 14:48:44
急袭的发动方式不大一样
AI-Joshua1989(276364504) 14:48:47
哦
AI-太阳神上<moligaloo@163.com> 14:49:09
点急袭，出现急袭的牌，选牌，再动态给出可能的目标
AI-太阳神上<moligaloo@163.com> 14:49:39
因为急袭所用的牌不在手牌区和装备区

once：默认为 false。once 为 true 的技能有：制衡、结姻、反间、离间、青囊、黄天、缔盟、驱虎、强袭、天义、挑衅、举荐、眩惑、陷阵、明策、甘露、心战、攻心、神愤、无前、绝汲（倚天张儁乂）、樵采（倚天夏厚娟）、归汉（倚天蔡昭姬）、乐学（倚天姜伯约）。不难发现，once 为 true 的技能都是可以主动发动并且每回合只能发动一次的。

mute：默认为 false。mute 为 true 的技能有结姻、神速、蛊惑、放逐、英魂、驱虎、挑衅、乐学（倚天姜伯约）。这些技能都需要分情况说台词。比如结姻，香香双将为副将且主将为男性时可以和自己及其他男性结姻，因此在不同情况会说不同的台词。神速分为神速 1 和神速 2；蛊惑分将牌扣出和将牌使用两个阶段；放逐分为翻过去和翻回来；英魂分为摸 1 弃 X 和摸 X 弃 1；驱虎成功后会有额外台词……不过 DIY 的时候如果没有对配音有比较高的要求，其实 mute 这一项可以忽略。

AI-太阳神上<moligaloo@163.com> 14:39:04

没看出来这些技能都需要分情况说台词的么？

AI-William915<william915@gmail.com> 14:39:18

一语道破天机啊。

AI-太阳神上<moligaloo@163.com> 14:40:15

挑衅也是分情况的，身上有八卦说的会跟其他不一样的

AI-Joshua1989(276364504) 14:41:05

结姻是看是不是搞基？

AI-太阳神上<moligaloo@163.com> 14:41:23

是的

AI-太阳神上<moligaloo@163.com> 14:41:44

搞基还分3种情况呢

AI-太阳神上<moligaloo@163.com> 14:42:00

攻、受和自慰

AI-Joshua1989(276364504) 14:42:12

哇靠。。。我从来没注意过

will_throw：默认为 true。will_throw 为 false 的技能有：仁德、鬼才、鬼道、好施、驱虎、天义、制霸、直谏、眩惑、陷阵、明策、义舍（倚天张公祺）。从效果上来看，就是这些技能牌使用后不会立即进入弃牌堆。

下面看下基本的属性读写函数（从 QString getSuitString()到 QString getEffectPath()），由于比较长并且易懂我就不粘了，大家可以看上面。大部分函数可以通过函数名一眼看出来是什么作用，我拣几个重点的说下：

getEffectiveId()：所有实体卡牌都有一个对应的 ID，如果 ID=-1，我们就把它视为 VirtualCard，即虚拟卡牌。如果是实体卡牌，直接获取其 ID；若是虚拟卡牌，获取其第一个子卡牌。虚拟卡牌与实体卡牌是相对的，如神速和旋风的杀，离间的决斗，这些牌是无法奸雄到的。

getFullName()：获取卡牌的全名，也就是花色+点数+卡牌名称。

再看卡牌的进一步细化

```
bool isVirtualCard() const;
virtual bool match(const QString &pattern) const;
void addSubcard(int card_id);
void addSubcard(const Card *card);
QList<int> getSubcards() const;
void clearSubcards();
QString subcardString() const;
void addSubcards(const QList<CardItem *> &card_items);
int subcardsLength() const;

virtual QString getType() const = 0;
virtual QString getSubtype() const = 0;
```

```
virtual CardType getTypeId() const = 0;
virtual QString toString() const;
virtual QString getEffectPath(bool is_male) const;
bool isNDTrick() const;
```

isVirtualCard()：判断卡牌 ID 是否为-1，是的话就是虚拟卡牌

match(const QString &pattern)：一个虚函数，用来看卡牌的名称、类型、子类型是否符合输入参数。

addSubcard(int card_id)和 addSubcard(const Card *card)都是为卡牌添加子卡牌类型。所谓子卡牌，就是拥有卡牌转换技能的武将使用该技能时用的牌。比如甄姬使用了一张黑桃 A 的闪电当作闪，那么主卡牌是一张【闪】的类 Jink（继承于 card 类），花色点数为黑桃 A，但是子卡牌要加上这张黑桃 A 的 ID。

需要用到 addSubcard 的武将技能有：鬼才、倾国、龙胆、奇袭、国色、流离、无双、离间、青囊、急救、鬼道、黄天、天香、蛊惑、断粮、好施、酒池、肉林、驱虎、双雄、连环、火攻、看破、天义、巧变、急袭、制霸、直谏、眩惑、戒酒、明策、武神、绝汲（倚天张傩义）。还有不要忘了丈八的两张牌当杀。

我们将这些技能再细分一下。首先是卡牌转化类：倾国、龙胆、奇袭、国色、急救、蛊惑、断粮、酒池、双雄、连环、火攻、看破、急袭、戒酒、武神、丈八技能、无双和酒池的实现方法是将两张【闪】当作一张【闪】的 subcard。第二类是拼点类：驱虎、天义、制霸、绝汲。第三类是使用一张牌发动效果的技能：流离、离间、青囊、黄天、天香、巧变、直谏、眩惑、明策。第四类是改判技能：鬼才、鬼道。总体来说，这些技能的实现方法是相似的，具体的我们留在技能编写那一课再说。

getSubcards()，clearSubcards()，addSubcards(const QList<CardItem *> &card_items)，subcardsLength()这几个函数看看名字就明白了吧。

getType()：获取卡牌类型，仅考虑卡牌不考虑武将技能的话，分为：基本牌（basic）、锦囊牌（trick）、装备牌（equip）。

getSubtype()：获取卡牌子类型，分为：

- 攻击牌（attack_card：雷火普通杀）
- 防御牌（defense_card：闪）
- 回复牌（recover_card：桃）
- 全局效果（global_effect：五谷桃园）
- AOE（aoe 南蛮万箭）
- 单体锦囊（single_target_trick 无中无懈顺拆火攻决斗借刀）
- 延时锦囊（delayed_trick 乐兵天灾）
- 武器牌（weapon）、防具牌（armor）
- 进攻马（offensive_horse）、防御马（defensive_horse）
- 辅助牌（buff_card 酒）
- 攻击范围扩展牌（demage_spread 铁锁）
- 恶心牌（disgusting_card 屎）

getTypeId()：获取卡牌类型在枚举变量里的值

getEffectPath(bool is_male)：获取使用该牌是播放的音效文件

isNDTrick()：判断卡牌是否为非延时锦囊

下面是卡牌目标选择时用到的一些函数，这些函数大多是虚函数，因为具体卡牌要继承 card 类，根据具体情况实现自己的功能。

```
virtual bool targetFixed() const;
virtual bool targetsFeasible(const QList<const Player *> &targets, const Player *Self) const;
virtual bool targetFilter(const QList<const Player *> &targets, const Player *to_select, const Player *Self) const;
virtual bool isAvailable(const Player *player) const;
virtual const Card *validate(const CardUseStruct *card_use) const;
virtual const Card *validateInResposing(ServerPlayer *user, bool *continuable) const;
bool isOnce() const;
bool isMute() const;
bool willThrow() const;
virtual void onUse(Room *room, const CardUseStruct &card_use) const;
virtual void use(Room *room, ServerPlayer *source,  const QList<ServerPlayer *> &targets) const;
virtual void onEffect(const CardEffectStruct &effect) const;
virtual bool isCancelable(const CardEffectStruct &effect) const;
virtual void onMove(const CardMoveStruct &move) const;
```

targetFixed()、isOnce()、isMute()、willThrow()参照前面讲过的。

targetsFeasible(const QList<const Player *> &targets, const Player *Self)：用于判断选择的目标是否合理，如借刀/离间的目标是否为两人、结姻选择的男性角色是否受伤等。

targetFilter(const QList<const Player *> &targets, const Player *to_select, const Player *Self)：用于暗掉不合理的目标，比如使用杀的时候会暗掉攻击范围外的玩家、离间时会暗掉女性角色。

isAvailable(const Player *player)：用于暗掉出牌阶段不能使用的卡牌，如没人杀时的闪、没有锦囊使用时的无懈、满血时的桃、场上没人装武器时的借刀等。

validate(const CardUseStruct *card_use)、validateInResposing(ServerPlayer *user, bool *continuable)：这两个函数只在于吉的蛊惑里用到了，没有什么普适性，暂且不提了（其实我也不会……）。

onUse(Room *room, const CardUseStruct &card_use)：在使用之前的一些预判，只有极少数情况会需要重载这个，如极略、急袭、陷阵、铁锁重铸。

use(Room *room, ServerPlayer *source, const QList<ServerPlayer *> &targets)：指定好了目标，确定开始使用

onEffect(const CardEffectStruct &effect)：单张卡牌对某个人的直接效果

上面三个函数是不是有点晕？我也纠结了很久，在官方 AI 群里讨论很久才弄明白了。

onUse 是使用前做的一些预处理

use 是指定目标后从 room 里调用 playCardEffect

onEffect 是卡牌使用后造成的效果

下面我给一个最直截了当的例子，就是铁锁连环的使用：在打出铁锁连环时，最先响应的 onUse 函数，在 onUse 函数里首先判断这张铁索连环指定了几个目标。如果没有指定目标就视为重铸，弃掉这张铁锁再摸一张牌。如果指定了 1~2 个目标，调用 use 函数。在 use 函数中，首先要弃掉这张铁锁，然后向 room 申请执行使用铁锁的效果，体现在代码里就是 playCardEffect 这个函数。最后指定 onEffect 函数，将选定的目标重置或横置。

isCancelable(const CardEffectStruct &effect)：卡牌效果是否可取消，如放桃园时满血角色的补血效果被取消。极少情况使用。

onMove(const CardMoveStruct &move)：卡牌被移动时发动的效果，如倚天剑从装备去失去，屎从手牌进入弃牌堆。极少情况使用。

终于只剩下一些辅助功能函数了

```
static bool CompareByColor(const Card *a, const Card *b);
static bool CompareBySuitNumber(const Card *a, const Card *b);
static bool CompareByType(const Card *a, const Card *b);
static const Card *Parse(const QString &str);
static Card * Clone(const Card *card);
static QString Suit2String(Suit suit);
static QString Number2String(int number);
static QStringList IdsToStrings(const QList<int> &ids);
static QList<int> StringsToIds(const QStringList &strings);
```

前三个是根据颜色、花色+点数、类型对卡牌进行比较。剩下的函数主要是进行字符串的操作，比较底层，与 DIY 关系不大。

最后看一下 card 类的成员变量

```
protected:
    QList<int> subcards;
    bool target_fixed;
    bool once;
    QString skill_name;
    bool mute;
    bool will_throw;
private:
    Suit suit;
    int number;
    int id;
```

subcards：子卡牌的 QList，之所以是 QList 是因为有些技能是多张牌当作一张牌的，如乱击、龙魂、丈八，无双和酒池也是用了这一机制

skill_name：当转化系的技能发动时，该技能的名字就会是打出这张牌的 skill_name。如倾国使用了黑桃 A 的闪电，代码中就会声明一个闪牌的类，将这张黑桃 A 写进 subcards 里，skill_name 赋值为 qingguo，花色和点数都不变。这样我们就实现了倾国的技能。

target_fixed、once、mute、will_throw 之前说过了。

suit、number、id 不用说了吧……

第四篇：太阳神三国杀的核心类的研读—card 类相关

上一课我们详细介绍了 card 类，卡牌作为三国杀最重要的一个部分，光有 card 类是远远不够的。首先，上讲我们提到了武将的某些技能需要使用卡牌来发动，这些技能有：制衡、仁德、结姻、突袭、反间、苦肉、离间、青囊、鬼才、流离、激将、鬼道、雷击、黄天、神速、天香、蛊惑、英魂、好施、缔盟、乱武、放逐、驱虎、节命、强袭、天义、巧变、挑衅、直谏、制霸、急袭、举荐、明策、甘露、陷阵、眩惑、心战、攻心、业炎、神愤、无前、狂风、大雾、极略。

以上这些技能都是通过 SkillCard 类来实现的，而 SkillCard 是 card 的子类，其定义如下：

```
class SkillCard: public Card{
    Q_OBJECT
public:
    SkillCard();
    void setUserString(const QString &user_string);
    virtual QString getSubtype() const;
    virtual QString getType() const;
    virtual CardType getTypeId() const;
    virtual QString toString() const;
protected:
    QString user_string;
};
```

除了多一个 user_string 来记录技能的使用者，其他函数在 card 类里都有，不过仍然是虚函数。这是因为上面说道的技能还有各自的类，继承于 SkillCard，需要具体问题具体分析。

SkillCard 类有一个比较特殊子类叫 DummyCard，主要用于弃牌和交换牌时使用。如弃牌阶段、主杀忠、行殇、缔盟。此外，无双、肉林的两张【闪】是放到一个 DummyCard 里作为 subCard，再把这个 DummyCard 赋值给 Jink 的。还有贯石斧弃牌用的也是 DummyCard。这个用的不多，可以忽略。

```
class DummyCard: public SkillCard{
    Q_OBJECT
public:
    DummyCard();
    virtual QString getSubtype() const;
    virtual QString getType() const;
    virtual QString toString() const;
};
```

然后是三个关于 card 的结构体：

```
struct CardEffectStruct{
    CardEffectStruct();
    const Card *card;
    ServerPlayer *from;
    ServerPlayer *to;
```

```
    bool multiple;
};
```

卡牌效果结构体：

成员变量分别表示卡牌指针、卡牌使用者、卡牌使用目标、卡牌目标是否唯一（如桃园五谷南蛮万箭借刀还有连环对两个人使用的情况）。

```
struct CardUseStruct{
    CardUseStruct();
    bool isValid() const;
    void parse(const QString &str, Room *room);
    const Card *card;
    ServerPlayer *from;
    QList<ServerPlayer *> to;
};
```

卡牌使用结构体：

isValid()：判断卡牌是否合法，其实就是看卡牌指针是否为空。

parse(const QString &str, Room *room)：通过分析 QString 内容给 CardUseStruct 赋值。

成员变量分别表示卡牌指针、卡牌使用者、卡牌使用目标。

```
struct CardMoveStruct{
    int card_id;
    Player::Place from_place, to_place;
    ServerPlayer *from, *to;
    bool open;
    QString toString() const;
};
```

卡牌移动结构体：

成员变量分别表示卡牌 ID、被移动卡牌来源位置、被移动卡牌去向位置、被移动卡牌原持有者、被移动卡牌新持有者。最后一个 open 表示卡牌移动时是否被公开，如仁德、突袭、归心不公开，穿上装备、使用延时锦囊、直谏、甘露、再起是公开的。

第五篇：太阳神三国杀的核心类的研读—skill 类

看完武将和卡牌，理所当然该看技能类了
首先看一下 skill 类的定义：

```
class Skill : public QObject
{
    Q_OBJECT
    Q_ENUMS(Frequency);

public:
    enum Frequency{
        Frequent,
        NotFrequent,
        Compulsory,
        Limited,
        Wake
    };

    explicit Skill(const QString &name, Frequency frequent = NotFrequent);
    bool isLordSkill() const;
    QString getDescription() const;
    QString getText() const;
    bool isVisible() const;

    virtual QString getDefaultChoice(ServerPlayer *player) const;
    virtual int getEffectIndex(ServerPlayer *player, const Card *card) const;
    virtual QDialog *getDialog() const;

    void initMediaSource();
    void playEffect(int index = -1) const;
    void setFlag(ServerPlayer *player) const;
    void unsetFlag(ServerPlayer *player) const;
    Frequency getFrequency() const;
    QStringList getSources() const;

protected:
    Frequency frequency;
    QString default_choice;

private:
    bool lord_skill;
    QStringList sources;
};
```

enum Frequency{Frequent, NotFrequent, Compulsory, Limited, Wake}：这个代表技能的发动频率。分别是频繁使用技、非频繁使用技、锁定技、限定技、觉醒技。一般来说，我们默认技能为非频繁使用技。那么什么叫频繁使用技呢？就是平常我们觉得理所应当使用甚至认为是锁定技的技能，比如英姿、集智、枭姬、连营等等。如果你细心的话，在玩神杀时会发现这个技能是通过一个复选框控制发动的，复选框默认是被勾上的，如果你勾掉复选框，这些技能就不会发动了。非平凡使用计就是剩下的技能，一些需要主动使用，如仁德、制衡、离间，一些需要询问你是否发动，如刚烈、反馈、奸雄。至于剩下的几个相信大家就很清楚了。

isLordSkill()：判断是否为主公技。

getDescription()：获取技能描述。

getText()：限定技、锁定技、觉醒技前面会有[XX 技]的字样，是通过这个函数得到的

isVisible()：判断技能是否可见，这个在 general 类的讲解里的 getVisibleSkillList()有说过，属于比较例外的情况，可以忽略。

getDefaultChoice(ServerPlayer *player)：获取默认选择。这个应该是在有出牌时限的情况下，对于一些需要选择选项发动的技能，如果玩家没有选择，那么系统会自动返回默认选项。如董卓的崩坏（上限>=血量+2 就掉上限，否则掉血）和凌统的旋风（默认不发动）。

getEffectIndex(ServerPlayer *player, const Card *card)、initMediaSource()、playEffect(int index = -1)、getSources()：与声效播放等有关的函数，可以忽略。

getDialog()：仅针对于吉的蛊惑和左慈的化身，可忽略。

setFlag(ServerPlayer *player)、unsetFlag(ServerPlayer *player)：增加、减少标记。

getFrequency()：获取技能发动频率。

下面看一下成员变量：

```
protected:
    Frequency frequency;
    QString default_choice;

private:
    bool lord_skill;
    QStringList sources;
```

分别是技能发动频率、默认选择、是否为主公技，最后一个与技能发动音效有关可忽略。

Skill 类的直系子类有四种：ViewAsSkill、TriggerSkill、ProhibitSkill、DistanceSkill。在和神上交流的时候他有说过有五类技能，分别为视作技、触发技、距离技、禁止技和系统耦合技。前四种就是这四个子类了，最后一种系统耦合技直接写在了其他事件里，例如完杀是直接写在 Dying 事件里的。系统耦合技比较特殊，暂且不提。在所有技能中，视作技和触发技占绝大部分（80%以上）。此外，视作技和触发技还有进一步细分的子类，下面我们具体分析。

ViewAsSkill：视作技，如乱击、双雄、倾国、天香、武神，可以将某（些）牌当作其他牌用的技能，这个其他牌可以是基本牌、锦囊牌，也可以是技能牌（如离间）。

视作技有一些子类型，下面将会一一介绍，属于视作技（非其子类型）的技能有：丈八转化杀、贯石斧弃牌、仁德、制衡、结姻、神速、好施、强袭、乱击、举荐、陷阵、中

业炎（继承于大业炎）、大业炎、龙魂、称象（倚天曹植）、归汉（倚天蔡昭姬）、乐学（倚天姜伯约）、义舍（倚天张公祺）、抬榇（倚天 SP 庞德）。

这些技能的共同点是发动机能的时候使用的卡牌个数为 2 或者数目不定，不能准确用其子类 ZeroCardViewAsSkill 和 OneCardViewAsSkill 准确归类。其中需要注意的是陷阵这个技能，你可能会说“陷阵不是拼点么，拼点就是用一张牌，为什么不是 OneCardViewAsSkill 呢？”。其实陷阵分为两个阶段，第一个阶段是拼点阶段，确实是用一张牌。但是如果拼点成功会有第二个阶段，就是出过一张杀后，可以发动陷阵再出杀，此时发动陷阵不需要牌，因此发动陷阵时需要的手牌数量不定。

```
class ViewAsSkill:public Skill{
    Q_OBJECT
public:
    ViewAsSkill(const QString &name);
    virtual bool viewFilter(const QList<CardItem *> &selected, const CardItem *to_select) const
= 0;
    virtual const Card *viewAs(const QList<CardItem *> &cards) const = 0;
    bool isAvailable() const;
    virtual bool isEnabledAtPlay(const Player *player) const;
    virtual bool isEnabledAtResponse(const Player *player, const QString &pattern) const;
};
```

viewFilter(const QList<CardItem *> &selected, const CardItem *to_select)：用于发动技能时过滤掉不能使用的卡牌，如倾国会过滤掉装备牌和红色手牌、青囊会过滤掉黑装备黑手牌。

viewAs(const QList<CardItem *> &cards)：用于在发动技能时将牌视作转化后的卡牌，如倾国会视作闪、离间会视作离间技能牌。

isAvailable()：用于判断技能是否可以使用，如返回 false，技能按钮会被禁用，如回合外的青囊、无人濒死时的急救。

isEnabledAtPlay(const Player *player)：判断技能是否可以在出牌阶段无其他事件激励的条件下是否可以发动。举个例子：刘备在无连弩、没出杀的情况下，可以使用仁德，否则仁德按钮是被禁用的。在没有判定的时候，司马的鬼才是被禁用的。

isEnabledAtResponse(const Player *player, const QString &pattern)：一些技能只有在特定条件下才能被激活，比如被杀时的倾国和龙胆、使用锦囊时的看破。这个条件由参数 pattern 给出，当条件符合时，这些技能会被激活。

ZeroCardViewAsSkill：继承于 ViewAsSkill，从名字就可以看出来，零张牌可以当作技能。是不是有点晕？其实换种方式来说就是不需要牌就可以发动的技能。

这些技能有：突袭、激将、反间、苦肉、雷击、放逐、英魂、缔盟、乱武、急袭、挑衅、酒诗、甘露、激将、伪帝、攻心、业炎、神愤、无前、狂风、大雾、极略、连理出杀（倚天夏侯娟）、连理（倚天夏侯娟）、樵采（倚天夏侯娟）、殉志（倚天姜伯约）、义舍要牌（倚天张公祺）。

可能你会问道：反间不是用了牌了么？缔盟不是要弃两人牌数之差的牌么？怎么叫没用牌呢？你需要注意，ZeroCardViewAsSkill 是发动阶段不需要使用牌的技能，你反间的时候用牌了么，指定缔盟的两人时用牌了么？

```
class ZeroCardViewAsSkill: public ViewAsSkill{
    Q_OBJECT
```

```
public:
    ZeroCardViewAsSkill(const QString &name);
    virtual bool viewFilter(const QList<CardItem *> &selected, const CardItem *to_select)
const;
    virtual const Card *viewAs(const QList<CardItem *> &cards) const;
    virtual const Card *viewAs() const = 0;
};
```

成员函数和 ViewAsSkill 差不多，只是重载了一个 viewAs()，用来直接返回技能牌。

OneCardViewAsSkill：继承于 ViewAsSkill，表示使用一张牌可以发动的技能。

这些技能有：鬼才改判、倾国、武圣、龙胆、奇袭、国色、流离、离间、青囊、急救、鬼道改判、黄天给牌、天香、蛊惑、断粮、酒池、驱虎、双雄、连环、火计、看破、天义、巧变、制霸拼点、直谏、眩惑、明策、绝汲（倚天张儁乂）。可以看出大部分是转化、拼点、改判类技能，少数是给一张牌或弃一张牌发动。

```
class OneCardViewAsSkill: public ViewAsSkill{
    Q_OBJECT
public:
    OneCardViewAsSkill(const QString &name);
    virtual bool viewFilter(const QList<CardItem *> &selected, const CardItem *to_select)
const;
    virtual const Card *viewAs(const QList<CardItem *> &cards) const;
    virtual bool viewFilter(const CardItem *to_select) const = 0;
    virtual const Card *viewAs(CardItem *card_item) const = 0;
};
```

成员函数和 ViewAsSkill 差不多，只是重载了一个 viewAs(CardItem *card_item)，用来直接返回技能牌或转化牌。

FilterSkill：继承于 OneCardViewAsSkill，用于“视为”锁定技，只有四个：红颜、戒酒、武神和僵尸模式僵尸的感染（装备牌视为铁锁）

```
class FilterSkill: public OneCardViewAsSkill{
    Q_OBJECT
public:
    FilterSkill(const QString &name);
};
```

TriggerSkill：就是触发技，通过某一事件触发才能使用的技能，如刚烈、反馈、铁骑等。触发技有很多子类，单纯的触发技（非其子类）的技能有：寒冰箭弃牌、猴子偷桃（欢乐包）、护驾、天妒、鬼才、裸衣攻击加成、洛神、激将、空城效果、集智、救援、克己、流离、枭姬、无双、鬼道、雷击、狂骨、不屈、不屈回血弃不屈牌、红颜判定、天香、行殇、颂威、祸首、巨象、烈刃、肉林、暴虐、双雄、猛进、涅槃、八阵、悲歌、断肠、屯田判定、激昂、制霸拒绝拼点、固政、享乐、毅重、落英、酒诗回合外受伤翻面、无言、恩怨、挥泪、旋风、破军、陷阵、智迟、智迟标记清除、补益、鸡肋、啖酪、庸肆、武魂增加标记、武魂、琴音、狂暴、无谋、狂风火系加成、大雾非雷系无效、忍戒、连破（回合内杀人获得额外回合）、极略、绝境、绝汲获得对方拼点牌（倚天张儁乂）、连理出闪

（倚天夏侯娟）、连理标记清除（倚天夏侯娟）、五灵（倚天晋宣帝）、胡笳（倚天蔡昭姬）、神君（倚天陆伯言）、纵火（倚天陆伯言）、烧营（倚天陆伯言）、毒士（倚天贾文和）、死战（倚天古之恶来）、神力（倚天古之恶来）、争功（倚天邓士载）、惜粮（倚天张公祺）、神威（倚天倚天剑）、倚天（倚天倚天剑）。

是不是很吓人？正如太阳神上所说，神杀是一个基于事件触发的架构。而张春华的伤逝技能基于条件触发（虽然我认为可以用很多事件触发来实现，之前也有很多补丁实现了张春华，说明张春华在技术层面是完全可以实现的），这是神上在神杀中剔除张春华的原因之一。

```
class TriggerSkill:public Skill{
    Q_OBJECT
public:
    TriggerSkill(const QString &name);
    const ViewAsSkill *getViewAsSkill() const;
    QList<TriggerEvent> getTriggerEvents() const;
    virtual int getPriority() const;
    virtual bool triggerable(const ServerPlayer *target) const;
    virtual bool trigger(TriggerEvent event, ServerPlayer *player, QVariant &data) const = 0;
protected:
    const ViewAsSkill *view_as_skill;
    QList<TriggerEvent> events;
};
```

getViewAsSkill()：一些技能是由某个事件出发后可以发动，然后使用卡牌发动技能，因此需要把触发技和视作技进行连接（这个操作在构造函数中完成），这种技能有：鬼才、激将、流离、鬼道、雷击、天香、双雄、陷阵、狂风、大雾、极略。

getTriggerEvents()：获取触发技能的事件，详见神上的教学贴：

http://qsanguosha.com/forum.php?mod=viewthread&tid=1398&extra=page%3D1

getPriority()：获取时间的优先级，详见神上的教学贴：

http://qsanguosha.com/forum.php?mod=viewthread&tid=2969&extra=page%3D1

triggerable(const ServerPlayer *target)：判断技能能否被触发，比如出现了判定但司马空城，那么这个技能就无法触发。

trigger(TriggerEvent event, ServerPlayer *player, QVariant &data)：技能发动的主体代码，用一个 swich 分支语句分别对所有的 event 进行响应。

view_as_skill：与触发技相联的视作技。

event：可以触发该技能的所有时间。

MasochismSkill：继承于 TriggerSkill。Masochism 是受虐狂的意思，所以这个叫做卖血技。属于卖血技的技能有：奸雄、遗计、刚烈、反馈、放逐、节命、新生、归心、称象（倚天曹冲）、同心（倚天夏侯娟）、偷渡（倚天邓士载）。

```
class MasochismSkill: public TriggerSkill{
    Q_OBJECT
public:
    MasochismSkill(const QString &name);
    virtual int getPriority() const;
    virtual bool trigger(TriggerEvent event, ServerPlayer *player, QVariant &data) const;
```

```
    virtual void onDamaged(ServerPlayer *target, const DamageStruct &damage) const = 0;
};
```

成员函数比 TriggerSkill 多了一个 onDamaged(ServerPlayer *target, const DamageStruct &damage)，卖血技的主体代码都在这里。

PhaseChangeSkill：继承于 TriggerSkill，阶段转换时发动的技能。这种技能有：仁德、突袭、观星、克己跳弃牌、闭月、神速、据守、再起、英魂、好施给牌、崩坏、天义、巧变、凿险、魂姿、志继、固政拿牌、放权、若愚、化身（回合开始阶段）、化身（回合结束阶段）、鸡肋标记清除、修罗、单骑、涉猎、无前、七星、拜印、极略标记清除（完杀和制衡）、连破、聪慧（倚天曹冲）、早夭（倚天曹冲）、归心 2（倚天魏武帝）、绝汲（倚天张儁乂）、围堰（倚天陆抗）、克构（倚天陆抗）、连理（倚天夏侯娟）、五灵（倚天晋宣帝）、共谋（倚天钟士季）、共谋换牌（倚天钟士季）、殉志（倚天姜伯约）、洞察（倚天贾文和）。

不难发现，这些技能大多都是回合开始阶段和结束阶段发动的，或者是一些需要标记的技能，如天义拼点成功后会有天义标记，虽然在游戏时并没有任何图片及文字提示。

```
class PhaseChangeSkill: public TriggerSkill{
    Q_OBJECT
public:
    PhaseChangeSkill(const QString &name);
    virtual bool trigger(TriggerEvent event, ServerPlayer *player, QVariant &data) const;
    virtual bool onPhaseChange(ServerPlayer *target) const =0;
};
```

成员函数比 TriggerSkill 多了一个 onPhaseChange(ServerPlayer *target)，阶段转换技的主体代码都在这里。

DrawCardsSkill：继承于 TriggerSkill，有改变摸牌阶段摸牌数目的技能。只作用于裸衣、英姿、好施、神威（SP 神吕布第二形态）。神赵云的绝境虽然也是这类技能，但是代码中用 TriggerSkill 实现了，应该是因为绝境多摸牌的个数不一定吧。

```
class DrawCardsSkill: public TriggerSkill{
    Q_OBJECT
public:
    DrawCardsSkill(const QString &name);
    virtual bool trigger(TriggerEvent event, ServerPlayer *player, QVariant &data) const;
    virtual int getDrawNum(ServerPlayer *player, int n) const = 0;
};
```

成员函数比 TriggerSkill 多了一个 getDrawNum(ServerPlayer *player, int n)，摸牌技的主体代码都在这里。

SlashBuffSkill：继承于 TriggerSkill，杀的辅助技能，目前只有铁骑和烈弓。

```
class SlashBuffSkill: public TriggerSkill{
    Q_OBJECT
public:
    SlashBuffSkill(const QString &name);
    virtual bool trigger(TriggerEvent event, ServerPlayer *player, QVariant &data) const;
    virtual bool buff(const SlashEffectStruct &effect) const = 0;
```

```
};
```

成员函数比 TriggerSkill 多了一个 buff(const SlashEffectStruct &effect)，辅杀技的主体代码都在这里。

GameStartSkill：继承于 TriggerSkill，游戏开始就可使用的技能，比如张角主群雄角色获得黄天技能、化身游戏开始摸两个武将牌、游戏开始摸七张七星牌、游戏开始时被连理的男性角色可以使用连理杀（倚天夏侯娟）、张公祺在场时其他角色获得义舍要牌技能。

```
class GameStartSkill: public TriggerSkill{
    Q_OBJECT
public:
    GameStartSkill(const QString &name);
    virtual bool trigger(TriggerEvent event, ServerPlayer *player, QVariant &data) const;
    virtual void onGameStart(ServerPlayer *player) const = 0;
};
```

成员函数比 TriggerSkill 多了一个 onGameStart(ServerPlayer *player)，游戏开始技的主体代码都在这里。

ProhibitSkill：禁止技，表示拥有该类技能的武将不可成为某些牌的目标，例如空城、谦逊、帷幕。这里需要注意一下，不能成为目标和卡牌对你无效是两个不同的概念，一个是不能使用，一个是能使用但是没用。

```
class ProhibitSkill: public Skill{
    Q_OBJECT
public:
    ProhibitSkill(const QString &name);
    virtual bool isProhibited(const Player *from, const Player *to, const Card *card) const = 0;
};
```

isProhibited(const Player *from, const Player *to, const Card *card)：这个函数里过滤掉了被禁用的牌。

DistanceSkill：距离技，拥有此类技能的武将在计算距离的时候需要进行修正。这类技能有：装备马的技能、马术、屯田、义从、单骑、飞影

```
class DistanceSkill: public Skill{
    Q_OBJECT
public:
    DistanceSkill(const QString &name);
    virtual int getCorrect(const Player *from, const Player *to) const = 0;
};
```

getCorrect(const Player *from, const Player *to)：用来修正距离。

WeaponSkill、ArmorSkill：继承于 TriggerSkill，武器和防具的技能。

MarkAssignSkill：继承于 GameStartSkill，用于在游戏开始给武将标记，例如乱武、涅槃、业炎，用来标记限定技是否使用过。还有就是神吕布游戏开始有两个狂暴标记。

到现在为止，所有的 Skill 类相关我们就讲解完了，但是之前说过还有一种系统耦合技不是通过 Skill 类来实现的。我筛选了一下，系统耦合技有这些：咆哮、奇才、完杀、天义拼点成功出两杀、增加手牌上限技能（血裔绝境神威）、神戟（SP 神吕布第二形态）、杨修的鸡肋、方天画戟三杀技能、诸葛连弩无限出杀技能。

最后提醒大家一下，并不是所有的技能都是由一个类来实现的，技能越复杂，需要的类就越多，比如鬼才需要一个单牌视作技类和一个触发技类、裸衣需要一个触发技和一个摸牌技等等。

第六篇：太阳神三国杀的核心类的研读—player 类

讲完了武将、卡牌、技能就该说下游戏的主体—玩家了。player 类很长，但是没有什么难理解的部分。首先看下 player 类的定义：

```
class Player : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString screenname READ screenName WRITE setScreenName)
    Q_PROPERTY(int hp READ getHp WRITE setHp)
    Q_PROPERTY(int maxhp READ getMaxHP WRITE setMaxHP)
    Q_PROPERTY(QString kingdom READ getKingdom WRITE setKingdom)
    Q_PROPERTY(bool wounded READ isWounded STORED false)
    Q_PROPERTY(QString role READ getRole WRITE setRole)
    Q_PROPERTY(QString general READ getGeneralName WRITE setGeneralName)
    Q_PROPERTY(QString general2 READ getGeneral2Name WRITE setGeneral2Name)
    Q_PROPERTY(QString state READ getState WRITE setState)
    Q_PROPERTY(int handcard_num READ getHandcardNum)
    Q_PROPERTY(int seat READ getSeat WRITE setSeat)
    Q_PROPERTY(QString phase READ getPhaseString WRITE setPhaseString)
    Q_PROPERTY(bool faceup READ faceUp WRITE setFaceUp)
    Q_PROPERTY(bool alive READ isAlive WRITE setAlive)
    Q_PROPERTY(QString flags READ getFlags WRITE setFlags)
    Q_PROPERTY(bool chained READ isChained WRITE setChained)
    Q_PROPERTY(bool owner READ isOwner WRITE setOwner)
    Q_PROPERTY(int atk READ getAttackRange)
    Q_PROPERTY(General::Gender gender READ getGender)
    Q_PROPERTY(bool kongcheng READ isKongcheng)
    Q_PROPERTY(bool nude READ isNude)
    Q_PROPERTY(bool all_nude READ isAllNude)
    Q_PROPERTY(bool caocao READ isCaoCao)
    Q_ENUMS(Phase)
    Q_ENUMS(Place)
    Q_ENUMS(Role)
public:
    enum Phase {Start, Judge, Draw, Play, Discard, Finish, NotActive};
    enum Place {Hand, Equip, Judging, Special, DiscardedPile, DrawPile};
    enum Role {Lord, Loyalist, Rebel, Renegade};

    explicit Player(QObject *parent);

    void setScreenName(const QString &screen_name);
    QString screenName() const;

    // property setters/getters
    int getHp() const;
```

```
void setHp(int hp);
int getMaxHP() const;
void setMaxHP(int max_hp);
int getLostHp() const;
bool isWounded() const;
General::Gender getGender() const;

bool isOwner() const;
void setOwner(bool owner);

int getMaxCards() const;

QString getKingdom() const;
void setKingdom(const QString &kingdom);
QString getKingdomIcon() const;
QString getKingdomFrame() const;

void setRole(const QString &role);
QString getRole() const;
Role getRoleEnum() const;

void setGeneral(const General *general);
void setGeneralName(const QString &general_name);
QString getGeneralName() const;

void setGeneral2Name(const QString &general_name);
QString getGeneral2Name() const;
const General *getGeneral2() const;

void setState(const QString &state);
QString getState() const;

int getSeat() const;
void setSeat(int seat);
QString getPhaseString() const;
void setPhaseString(const QString &phase_str);
Phase getPhase() const;
void setPhase(Phase phase);

int getAttackRange() const;
bool inMyAttackRange(const Player *other) const;

bool isAlive() const;
bool isDead() const;
```

```
void setAlive(bool alive);

QString getFlags() const;
virtual void setFlags(const QString &flag);
bool hasFlag(const QString &flag) const;
void clearFlags();

bool faceUp() const;
void setFaceUp(bool face_up);

virtual int aliveCount() const = 0;
void setFixedDistance(const Player *player, int distance);
int distanceTo(const Player *other) const;
const General *getAvatarGeneral() const;
const General *getGeneral() const;

bool isLord() const;

void acquireSkill(const QString &skill_name);
void loseSkill(const QString &skill_name);
void loseAllSkills();
bool hasSkill(const QString &skill_name) const;
bool hasInnateSkill(const QString &skill_name) const;
bool hasLordSkill(const QString &skill_name) const;
virtual QString getGameMode() const = 0;

void setEquip(const EquipCard *card);
void removeEquip(const EquipCard *equip);
bool hasEquip(const Card *card) const;
bool hasEquip() const;

QList<const Card *> getJudgingArea() const;
void addDelayedTrick(const Card *trick);
void removeDelayedTrick(const Card *trick);
QList<const DelayedTrick *> delayedTricks() const;
bool containsTrick(const QString &trick_name) const;
const DelayedTrick *topDelayedTrick() const;

virtual int getHandcardNum() const = 0;
virtual void removeCard(const Card *card, Place place) = 0;
virtual void addCard(const Card *card, Place place) = 0;

const Weapon *getWeapon() const;
const Armor *getArmor() const;
```

```
const Horse *getDefensiveHorse() const;
const Horse *getOffensiveHorse() const;
QList<const Card *> getEquips() const;
const EquipCard *getEquip(int index) const;

bool hasWeapon(const QString &weapon_name) const;
bool hasArmorEffect(const QString &armor_name) const;

bool isKongcheng() const;
bool isNude() const;
bool isAllNude() const;

void addMark(const QString &mark);
void removeMark(const QString &mark);
virtual void setMark(const QString &mark, int value);
int getMark(const QString &mark) const;

void setChained(bool chained);
bool isChained() const;

bool canSlash(const Player *other, bool distance_limit = true) const;
int getCardCount(bool include_equip) const;

QList<int> getPile(const QString &pile_name) const;
QString getPileName(int card_id) const;

void addHistory(const QString &name, int times = 1);
void clearHistory();
bool hasUsed(const QString &card_class) const;
int usedTimes(const QString &card_class) const;
int getSlashCount() const;

QSet<const TriggerSkill *> getTriggerSkills() const;
QSet<const Skill *> getVisibleSkills() const;
QList<const Skill *> getVisibleSkillList() const;
QSet<QString> getAcquiredSkills() const;

virtual bool isProhibited(const Player *to, const Card *card) const;
bool canSlashWithoutCrossbow() const;
virtual bool isLastHandCard(const Card *card) const = 0;

void jilei(const QString &type);
bool isJilei(const Card *card) const;
bool isCaoCao() const;
```

```
    void copyFrom(Player* p);
    QList<const Player *> getSiblings() const;

    QVariantMap tag;

protected:
    QMap<QString, int> marks;
    QMap<QString, QList<int> > piles;
    QSet<QString> acquired_skills;
    QSet<QString> flags;
    QHash<QString, int> history;

private:
    QString screen_name;
    bool owner;
    const General *general, *general2;
    int hp, max_hp;
    QString kingdom;
    QString role;
    QString state;
    int seat;
    bool alive;

    Phase phase;
    const Weapon *weapon;
    const Armor *armor;
    const Horse *defensive_horse, *offensive_horse;
    bool face_up;
    bool chained;
    QList<const Card *> judging_area;
    QList<const DelayedTrick *> delayed_tricks;
    QHash<const Player *, int> fixed_distance;

    QSet<Card::CardType> jilei_set;

signals:
    void general_changed();
    void general2_changed();
    void role_changed(const QString &new_role);
    void state_changed();
    void kingdom_changed();
    void phase_changed();
    void owner_changed(bool owner);
};
```

老套路，还是先看这个类有哪些属性：

```
Q_OBJECT
Q_PROPERTY(QString screenname READ screenName WRITE setScreenName)
Q_PROPERTY(int hp READ getHp WRITE setHp)
Q_PROPERTY(int maxhp READ getMaxHP WRITE setMaxHP)
Q_PROPERTY(QString kingdom READ getKingdom WRITE setKingdom)
Q_PROPERTY(bool wounded READ isWounded STORED false)
Q_PROPERTY(QString role READ getRole WRITE setRole)
Q_PROPERTY(QString general READ getGeneralName WRITE setGeneralName)
Q_PROPERTY(QString general2 READ getGeneral2Name WRITE setGeneral2Name)
Q_PROPERTY(QString state READ getState WRITE setState)
Q_PROPERTY(int handcard_num READ getHandcardNum)
Q_PROPERTY(int seat READ getSeat WRITE setSeat)
Q_PROPERTY(QString phase READ getPhaseString WRITE setPhaseString)
Q_PROPERTY(bool faceup READ faceUp WRITE setFaceUp)
Q_PROPERTY(bool alive READ isAlive WRITE setAlive)
Q_PROPERTY(QString flags READ getFlags WRITE setFlags)
Q_PROPERTY(bool chained READ isChained WRITE setChained)
Q_PROPERTY(bool owner READ isOwner WRITE setOwner)
Q_PROPERTY(int atk READ getAttackRange)
Q_PROPERTY(General::Gender gender READ getGender)
Q_PROPERTY(bool kongcheng READ isKongcheng)
Q_PROPERTY(bool nude READ isNude)
Q_PROPERTY(bool all_nude READ isAllNude)
Q_PROPERTY(bool caocao READ isCaoCao)
Q_ENUMS(Phase)
Q_ENUMS(Place)
Q_ENUMS(Role)
```

这个类的属性虽然多，但都不难理解，从上至下分别为：玩家名称、血量、血上限、国籍、是否受伤、角色、主将、副将、状态（在线掉线）、手牌数、座次、游戏阶段、是否被翻面、是否活着、标志（不是标记，一些技能需要，如双雄、无前、放权等）、是否横置、是否拥有者（玩家 true，AIfalse）、攻击范围、性别、是否空城（无手牌）、是否裸（无手牌无装备）、是否全裸（无手牌装备且判定区为空）、是否为曹操（曹操、魏武帝、神曹操）。

最后三个是枚举变量：

Phase {Start, Judge, Draw, Play, Discard, Finish, NotActive}：游戏阶段，分为开始阶段、判定阶段、摸牌阶段、出牌阶段、弃牌阶段、结束阶段和回合外。

Place {Hand, Equip, Judging, Special, DiscardedPile, DrawPile}：卡牌位置，分为手牌区、装备区、判定区、特殊区（判定牌、再起亮牌、蛊惑扣牌）、弃牌堆、摸牌堆。

Role {Lord, Loyalist, Rebel, Renegade}：游戏角色，主忠反内。

关于属性 Get/Set 函数（从 setScreenName(const QString &screen_name)到 setFaceUp(bool face_up)），再加上 isKongcheng()、isNude()、isAllNude()和 setChained(bool chained)、isChained()。我只捡几个重要的说：

getMaxCards()：获取手牌上限。

setGeneralName(const QString &general_name)、setGeneral2Name(const QString &general_name)：设置主将、副将名字。

inMyAttackRange(const Player *other)：判断另一个玩家是否在该玩家攻击范围内。

下面是一些游戏过程中需要用到的功能函数：

```
void acquireSkill(const QString &skill_name);
void loseSkill(const QString &skill_name);
void loseAllSkills();
bool hasSkill(const QString &skill_name) const;
bool hasInnateSkill(const QString &skill_name) const;
bool hasLordSkill(const QString &skill_name) const;
virtual QString getGameMode() const = 0;

void setEquip(const EquipCard *card);
void removeEquip(const EquipCard *equip);
bool hasEquip(const Card *card) const;
bool hasEquip() const;

QList<const Card *> getJudgingArea() const;
void addDelayedTrick(const Card *trick);
void removeDelayedTrick(const Card *trick);
QList<const DelayedTrick *> delayedTricks() const;
bool containsTrick(const QString &trick_name) const;
const DelayedTrick *topDelayedTrick() const;

virtual int getHandcardNum() const = 0;
virtual void removeCard(const Card *card, Place place) = 0;
virtual void addCard(const Card *card, Place place) = 0;

const Weapon *getWeapon() const;
const Armor *getArmor() const;
const Horse *getDefensiveHorse() const;
const Horse *getOffensiveHorse() const;
QList<const Card *> getEquips() const;
const EquipCard *getEquip(int index) const;

bool hasWeapon(const QString &weapon_name) const;
bool hasArmorEffect(const QString &armor_name) const;

void addMark(const QString &mark);
```

```
void removeMark(const QString &mark);
virtual void setMark(const QString &mark, int value);
int getMark(const QString &mark) const;

bool canSlash(const Player *other, bool distance_limit = true) const;
int getCardCount(bool include_equip) const;

QList<int> getPile(const QString &pile_name) const;
QString getPileName(int card_id) const;

void addHistory(const QString &name, int times = 1);
void clearHistory();
bool hasUsed(const QString &card_class) const;
int usedTimes(const QString &card_class) const;
int getSlashCount() const;

QSet<const TriggerSkill *> getTriggerSkills() const;
QSet<const Skill *> getVisibleSkills() const;
QList<const Skill *> getVisibleSkillList() const;
QSet<QString> getAcquiredSkills() const;

virtual bool isProhibited(const Player *to, const Card *card) const;
bool canSlashWithoutCrossbow() const;
virtual bool isLastHandCard(const Card *card) const = 0;

void jilei(const QString &type);
bool isJilei(const Card *card) const;
bool isCaoCao() const;
void copyFrom(Player* p);
QList<const Player *> getSiblings() const;
```

距离方面：

aliveCount()：返回存活的玩家数量，用于计算距离。

setFixedDistance(const Player *player, int distance)：锁定与某玩家的距离，用于高顺戒酒拼点成功。

distanceTo(const Player *other)：计算与另一玩家的距离。

getGeneral()：获取主将。

技能方面：

acquireSkill(const QString &skill_name)：用于游戏过程中获取技能，觉醒技都是这样获得的，如志继获得观星、魂姿获得英魂英姿、若愚获得激将、单骑获得马术、无前获得无双、拜印获得极略、使用极略获得完杀等等。

loseSkill(const QString &skill_name)：用于游戏过程中失去技能，如无前使用完毕后失去无双。

hasLordSkill(const QString &skill_name)：用于判断该武将是否有某主公技。

装备区方面：

setEquip(const EquipCard *card)、removeEquip(const EquipCard *equip)、hasEquip(const Card *card)：用来穿装备、卸装备、查看是否有某装备。

getWeapon()、getArmor()、getDefensiveHorse()、getOffensiveHorse()、getEquips()：获取装备区武器、防具、防御马、进攻马、所有装备。

hasWeapon(const QString &weapon_name)：判断是否有某武器。

hasArmorEffect(const QString &armor_name)：判断是否可以发动防具技能。

判定区方面：

getJudgingArea()：获取判定区所有牌。

addDelayedTrick(const Card *trick)、removeDelayedTrick(const Card *trick)、containsTrick(const QString &trick_name)、topDelayedTrick()：贴上、去掉、查看判定区是否有、返回最近被贴上的延时锦囊。

标记物方面：

addMark(const QString &mark)、removeMark(const QString &mark)、setMark(const QString &mark, int value)、getMark(const QString &mark)：添加、去除、设置、获取某标记。

getPile(const QString &pile_name)：获取特殊牌堆的牌，如七星、屯田、不屈、米（倚天张公祺）这些在游戏外的牌。

技能使用方面：

hasUsed(const QString &card_class)：是否用过某牌，主要用于每回合只能发动一次的技能，判断技能是否发动过。

usedTimes(const QString &card_class)：用于回合内可多次使用但有次数限制的技能，如制霸孙权的制衡、神司马的极略、倚天张公祺的义舍。

卡牌使用方面：

isProhibited(const Player *to, const Card *card)：禁止技就是通过它实现的。

canSlashWithoutCrossbow()：无连弩时是否可以出杀，用于咆哮、天义拼点成功、还有关于使用方片 A 的连弩做杀是的情况（第三种情况真的很难想到啊）。

isLastHandCard(const Card *card)：是否是最后一张手牌，用于方天画戟三杀的发动。

剩下的函数不太重要了，isCaocao()用于行殇的一个音效还有倚天包倚天剑与曹操的联动技。getSiblings()用于获得场上所有玩家。

再往下是 player 类的成员变量：

从上往下分别为：标记物 mark、特殊牌堆 pile、游戏中获得技能 acquire_skill、标志 flag、历史 history（使用某牌的记录），然后是我们开始说的那一堆属性，再往下是玩家所处游戏阶段 phase、装备、是否正面朝上 face_up、是否被横置 chained、判定区 judge_area、延时锦囊 delayed_tricks、与其他玩家的距离 fixed_distance，最后一个用于实现鸡肋 jilei_set。

最后就是信号了，用于系统一些事件、操作的连接，我们可以不管这个。

第七篇：太阳神三国杀的核心类的研读—serverplayer 类

上一讲我们介绍了 player 类，熟悉了玩家对象自身需要实现的一些功能，但是玩家之间如何与服务器相联、相互作用仍然没有提到。今天我们就介绍一下 player 类的子类 serverplayer 类，它实现了更多的基本功能。

先来看下 serverplayer 的定义：

```
class ServerPlayer : public Player
{
    Q_OBJECT
    Q_PROPERTY(QString ip READ getIp)
public:
    explicit ServerPlayer(Room *room);

    void setSocket(ClientSocket *socket);
    void invoke(const char *method, const QString &arg = ".");
    QString reportHeader() const;
    void sendProperty(const char *property_name, const Player *player = NULL) const;
    void unicast(const QString &message) const;
    void drawCard(const Card *card);
    Room *getRoom() const;
    void playCardEffect(const Card *card);
    int getRandomHandCardId() const;
    const Card *getRandomHandCard() const;
    void obtainCard(const Card *card);
    void throwAllEquips();
    void throwAllHandCards();
    void throwAllCards();
    void bury();
    void throwAllMarks();
    void clearPrivatePiles();
    void drawCards(int n, bool set_emotion = true);
    bool askForSkillInvoke(const QString &skill_name, const QVariant &data = QVariant());
    QList<int> forceToDiscard(int discard_num, bool include_equip);
    QList<int> handCards() const;
    QList<const Card *> getHandcards() const;
    QList<const Card *> getCards(const QString &flags) const;
    DummyCard *wholeHandCards() const;
    bool hasNullification() const;
    void kick();
    bool pindian(ServerPlayer *target, const QString &reason, const Card *card1 = NULL);
    bool pindian(ServerPlayer *target, const QString &reason, const Card *card1,
PindianStruct& pindian_struct);//Joshua
    void turnOver();
    void play();
```

```
QList<Player::Phase> &getPhases();
void skip(Player::Phase phase);

void gainMark(const QString &mark, int n = 1);
void loseMark(const QString &mark, int n = 1);
void loseAllMarks(const QString &mark_name);

void setAI(AI *ai);
AI *getAI() const;
AI *getSmartAI() const;

virtual int aliveCount() const;
virtual int getHandcardNum() const;
virtual void removeCard(const Card *card, Place place);
virtual void addCard(const Card *card, Place place);
virtual bool isLastHandCard(const Card *card) const;

void addVictim(ServerPlayer *victim);
QList<ServerPlayer *> getVictims() const;

void startRecord();
void saveRecord(const QString &filename);

void setNext(ServerPlayer *next);
ServerPlayer *getNext() const;
ServerPlayer *getNextAlive() const;

// 3v3 methods
void addToSelected(const QString &general);
QStringList getSelected() const;
QString findReasonable(const QStringList &generals);
void clearSelected();

int getGeneralMaxHP() const;
virtual QString getGameMode() const;

QString getIp() const;
void introduceTo(ServerPlayer *player);
void marshal(ServerPlayer *player) const;

void addToPile(const QString &pile_name, int card_id, bool open = true);

void copyFrom(ServerPlayer* sp);
```

```
private:
    ClientSocket *socket;
    QList<const Card *> handcards;
    Room *room;
    AI *ai;
    AI *trust_ai;
    QList<ServerPlayer *> victims;
    Recorder *recorder;
    QList<Phase> phases;
    ServerPlayer *next;
    QStringList selected; // 3v3 mode use only

private slots:
    void getMessage(char *message);
    void castMessage(const QString &message);

signals:
    void disconnected();
    void request_got(const QString &request);
    void message_cast(const QString &message) const;
};
```

首先看属性，就一个 IP 地址，没什么说的。

下面看一下 serverplayer 需要实现的一些功能函数（从 setSocket 到 play）。由于我们主要讲 DIY，所以与底层有关的如游戏记录、服务器连接、AI 实现的部分我们跳过，并且 3V3 方面的也暂且不提：

invoke(const char *method, const QString &arg = ".")：只有少数技能调用了这个函数，它们是：激将、急袭、固政、心战、化身、鸡肋、七星、连理（倚天夏侯娟）、义舍要牌（倚天张公祺）。这些技能都是需要弹出一个牌框并从中选牌或是让别人替自己出杀（化身和鸡肋比较特殊）。其实这个函数就是用来调用 unicast 函数记录游戏过程的，故略。

getRoom()：获取 serverplayer 所在的 room，room 类我们下一讲会介绍，游戏中的各种事件都是在 room 中实现的，因此很多技能的实现需要用到这个函数。

playCardEffect(const Card *card)：与播放游戏音效有关。

getRandomHandCard()、getRandomHandCardId()：随机获取一张手牌及其 ID，例如周瑜的反间就用到了这个函数。

drawCard(const Card *card)：将某牌收入手牌，在 handcards 中加入所得的牌。

obtainCard(const Card *card)：将某牌收入手牌，将该牌放入玩家手牌区。

throwAllEquips()：弃掉所有装备牌。

throwAllHandCards()：弃掉所有手牌。

throwAllCards()：弃掉所有牌，包括装备牌、手牌以及判定区里的牌。如庞统涅槃。

bury()：玩家死亡时的操作，弃掉所有牌、标记以及私有牌堆。

throwAllMarks()：弃掉所有标记。

clearPrivatePiles()：清除私有牌堆。

drawCards(int n, bool set_emotion = true)：从摸牌堆拿 n 张牌。

askForSkillInvoke(const QString &skill_name, const QVariant &data = QVariant())：询问是否发动机能，所有触发技都会用到。

forceToDiscard(int discard_num, bool include_equip)：强制弃牌。

handCards()：返回所有手牌的 ID。

getHandcards()、getCards(const QString &flags)：获取手牌、根据选择获取该玩家的牌（手牌、装备牌、判定区的牌）。

wholeHandCards()：将所有手牌放入一张 dummy card 的 subcards 内，并返回该卡牌，用于同时处理不定量的多张牌，如行殇、缔盟和共谋（倚天钟士季）。

hasNullification()：判断玩家是否有无懈可击，这里把看破、蛊惑、龙魂都加进来了，还有神关羽武神把红桃无懈当作杀，所以这几个技能也属于部分系统耦合技。如果 DIY 的时候有技能和无懈可击有关，需要记得在这里加上。否则如果回合外有别人使用锦囊，而你又没有真的无懈或者以上的技能，系统是不会提示你是否打出无懈可击的。

pindian(ServerPlayer *target, const QString &reason, const Card *card1 = NULL)、pindian(ServerPlayer *target, const QString &reason, const Card *card1, PindianStruct& pindian_struct)：这两个函数都是用来拼点，前者是原始代码中的，后者多一个 pindian_struct 的输出参数，可以获得拼点的牌。这两个函数返回值都是拼点发起人是否拼点成功。

turnOver()：翻面。

play()：回合开始阶段执行。技能方面主要用于争功（倚天邓士载）的实现。

与游戏阶段有关的函数：

getPhases()：获取所有游戏阶段。

skip(Player::Phase phase)：跳过某阶段。如乐不思蜀、兵粮寸断、克己、神速。

与标记物有关的函数：

gainMark(const QString &mark, int n = 1)：获取某标记。

loseMark(const QString &mark, int n = 1)：失去某标记。

loseAllMarks(const QString &mark_name)：失去所有某种标记。

几个虚函数，但是我不太清楚这几个函数为什么要声明为虚函数：

aliveCount()：计算场上存活玩家数。

getHandcardNum()：获取手牌数。

removeCard(const Card *card, Place place)：从某处移除某牌。

addCard(const Card *card, Place place)：添加某牌至某处。

isLastHandCard(const Card *card)：判断是否为最后一张手牌，用于方天画戟三杀的实现。

杀死玩家时用到的函数：

addVictim(ServerPlayer *victim)：杀死任意玩家时，通过此函数将死者与凶手关联，进一步进行奖惩。

getVictims()：获取该玩家杀死的所有玩家。

与下家有关的函数：

setNext(ServerPlayer *next)：设置某玩家为该玩家的下家。

getNext()：获取该玩家的下家，下家死亡也可以返回。

getNextAlive()：获取该玩家活着的下家，下家若死亡则继续向下找。

剩下几个函数，不分类了：

findReasonable(const QStringList &generals)：查看双将组合是否禁配。

getGeneralMaxHP()：获取玩家血上限。

addToPile(const QString &pile_name, int card_id, bool open = true)：将某牌加入私有牌堆，如屯田、不屈、七星、惜粮（倚天张公祺）。

最后我们看一下 serverplayer 的成员变量：

```
ClientSocket *socket;
QList<const Card *> handcards;
Room *room;
AI *ai;
AI *trust_ai;
QList<ServerPlayer *> victims;
Recorder *recorder;
QList<Phase> phases;
ServerPlayer *next;
```

第一个是套接字，与服务器连接相关。

handcards：改玩家的手牌。

room：该玩家所在房间。

victims：该玩家杀死的玩家。

phases：该玩家的游戏阶段。

next：该玩家的下家。

第八篇：太阳神三国杀的核心类的研读—有关 room 类的常用结构体

很高兴的告诉大家，room 类将是我们真正开始 DIY 前需要讲解的最后一个类。不过在此之前，我们需要先对与 room 类相关的几个结构体有所了解。和第四讲一样，这些结构体在 server 文件夹下的 structs.h 文件里。

首先是伤害结构体：

```
struct DamageStruct{
    DamageStruct();
    enum Nature{
        Normal, // normal slash, duel and most damage caused by skill
        Fire,   // fire slash, fire attack and few damage skill (Yeyan, etc)
        Thunder // lightning, thunder slash, and few damage skill (Leiji, etc)
    };
    ServerPlayer *from;
    ServerPlayer *to;
    const Card *card;
    int damage;
    Nature nature;
    bool chain;
};
```

它的成员变量分别为伤害来源、伤害对象、造成伤害的卡牌、伤害点数、伤害属性（普通、雷、火），最后一个 chain 可能是用于判断是否是铁锁传递来的伤害。

然后是使用杀的结构体：

```
struct SlashEffectStruct{
    SlashEffectStruct();
    const Slash *slash;
    const Card *jink;
    ServerPlayer *from;
    ServerPlayer *to;
    bool drank;
    DamageStruct::Nature nature;
};
```

它的成员变量分别为使用的杀、抵消这张杀用的闪、杀的使用者、杀的目标、是否喝酒、杀的属性。说实话我没太弄明白为什么在 SlashEffectStruct 还要加 nature，可能是因为朱雀羽扇。

然后是濒死结构体：

```
struct DyingStruct{
    DyingStruct();
    ServerPlayer *who; // who is ask for help
    DamageStruct *damage; // if it is NULL that means the dying is caused by losing hp
};
```

它的成员变量分别为濒死的玩家、造成此濒死状态的伤害（如果为空表示是体力流失造成的濒死）。

然后是回复结构体：

```
struct RecoverStruct{
    RecoverStruct();
    int recover;
    ServerPlayer *who;
    const Card *card;
};
```

它的成员变量分别为回复点数、回复作用对象、产生回复效果的卡牌。

然后是拼点结构体：

```
struct PindianStruct{
    PindianStruct();
    bool isSuccess() const;
    ServerPlayer *from;
    ServerPlayer *to;
    const Card *from_card;
    const Card *to_card;
    QString reason;
};
```

它的成员变量分别为拼点发起者、拼点响应者、拼点发起者的拼点牌、拼点响应者的拼点牌、拼点原因。还有一个成员函数用来判断拼点发起者是否拼点胜利。

最后是判定结构体：

```
struct JudgeStruct{
    JudgeStruct();
    bool isGood(const Card *card = NULL) const;
    bool isBad() const;
    ServerPlayer *who;
    const Card *card;
    QRegExp pattern;
    bool good;
    QString reason;
};
```

它的成员变量分别为接受判定的玩家、判定牌、判定规则、判定结果是否有利、判定原因。两个成员函数用来判断判定结果是否有利。

到此为止，structs.h 文件中的结构体已经全部讲解完，下一讲我们讲解 room 类。

第九篇：太阳神三国杀的核心类的研读—room 类的研读

OK，这是我们开始 DIY 前的最后一课，话不多说，我们开始吧。

先看一下 room 类的定义，又是一个很长的类。

```
class Room : public QThread{
    Q_OBJECT
public:
    friend class RoomThread;
    friend class RoomThread3v3;
    friend class RoomThread1v1;

    typedef void (Room::*Callback)(ServerPlayer *, const QString &);

    explicit Room(QObject *parent, const QString &mode);
    QString createLuaState();
    ServerPlayer *addSocket(ClientSocket *socket);
    bool isFull() const;
    bool isFinished() const;
    int getLack() const;
    QString getMode() const;
    const Scenario *getScenario() const;
    RoomThread *getThread() const;
    void playSkillEffect(const QString &skill_name, int index = -1);
    ServerPlayer *getCurrent() const;
    void setCurrent(ServerPlayer *current);
    int alivePlayerCount() const;
    QList<ServerPlayer *> getOtherPlayers(ServerPlayer *except) const;
    QList<ServerPlayer *> getPlayers() const;       //Joshua
    QList<ServerPlayer *> getAllPlayers() const;
    QList<ServerPlayer *> getAlivePlayers() const;
    void output(const QString &message);
    void enterDying(ServerPlayer *player, DamageStruct *reason);
    void killPlayer(ServerPlayer *victim, DamageStruct *reason = NULL);
    void revivePlayer(ServerPlayer *player);
    QStringList aliveRoles(ServerPlayer *except = NULL) const;
    void gameOver(const QString &winner);
    void slashEffect(const SlashEffectStruct &effect);
    void slashResult(const SlashEffectStruct &effect, const Card *jink);
    void attachSkillToPlayer(ServerPlayer *player, const QString &skill_name);
    void detachSkillFromPlayer(ServerPlayer *player, const QString &skill_name);
    bool obtainable(const Card *card, ServerPlayer *player);
    void setPlayerFlag(ServerPlayer *player, const QString &flag);
    void setPlayerProperty(ServerPlayer *player, const char *property_name, const QVariant
&value);
    void setPlayerMark(ServerPlayer *player, const QString &mark, int value);
```

```
    void useCard(const CardUseStruct &card_use, bool add_history = true);
    void damage(const DamageStruct &data);
    void sendDamageLog(const DamageStruct &data);
    void loseHp(ServerPlayer *victim, int lose = 1);
    void loseMaxHp(ServerPlayer *victim, int lose = 1);
    void applyDamage(ServerPlayer *victim, const DamageStruct &damage);
    void recover(ServerPlayer *player, const RecoverStruct &recover, bool set_emotion = false);
    void playCardEffect(const QString &card_name, bool is_male);
    bool cardEffect(const Card *card, ServerPlayer *from, ServerPlayer *to);
    bool cardEffect(const CardEffectStruct &effect);
    void judge(JudgeStruct &judge_struct);
    void sendJudgeResult(const JudgeStar judge);
    QList<int> getNCards(int n, bool update_pile_number = true);
    ServerPlayer *getLord() const;
    void doGuanxing(ServerPlayer *zhuge, const QList<int> &cards, bool up_only);
    void doGongxin(ServerPlayer *shenlumeng, ServerPlayer *target);
    int drawCard();
    const Card *peek();
    void fillAG(const QList<int> &card_ids, ServerPlayer *who = NULL);
    void takeAG(ServerPlayer *player, int card_id);
    void provide(const Card *card);
    QList<ServerPlayer *> getLieges(const QString &kingdom, ServerPlayer *lord) const;
    void sendLog(const LogMessage &log);
    void showCard(ServerPlayer *player, int card_id, ServerPlayer *only_viewer = NULL);
    void showAllCards(ServerPlayer *player, ServerPlayer *to = NULL);
    void getResult(const QString &reply_func, ServerPlayer *reply_player, bool move_focus =
true);
    void acquireSkill(ServerPlayer *player, const Skill *skill, bool open = true);
    void acquireSkill(ServerPlayer *player, const QString &skill_name, bool open = true);
    void adjustSeats();
    void swapPile();
    int getCardFromPile(const QString &card_name);
    ServerPlayer *findPlayer(const QString &general_name, bool include_dead = false) const;
    ServerPlayer *findPlayerBySkillName(const QString &skill_name, bool include_dead = false)
const;
    void installEquip(ServerPlayer *player, const QString &equip_name);
    void resetAI(ServerPlayer *player);
    void transfigure(ServerPlayer *player, const QString &new_general, bool full_state, bool
invoke_start = true);
    void swapSeat(ServerPlayer *a, ServerPlayer *b);
    lua_State *getLuaState() const;
    void setFixedDistance(Player *from, const Player *to, int distance);
    void reverseFor3v3(const Card *card, ServerPlayer *player, QList<ServerPlayer *> &list);
    bool hasWelfare(const ServerPlayer *player) const;
```

```
    ServerPlayer *getFront(ServerPlayer *a, ServerPlayer *b) const;
    void signup(ServerPlayer *player, const QString &screen_name, const QString &avatar, bool
is_robot);

    void reconnect(ServerPlayer *player, ClientSocket *socket);
    void marshal(ServerPlayer *player);

    bool isVirtual();
    void setVirtual();
    void copyFrom(Room* rRoom);
    Room* duplicate();

    const ProhibitSkill *isProhibited(const Player *from, const Player *to, const Card *card)
const;

    void setTag(const QString &key, const QVariant &value);
    QVariant getTag(const QString &key) const;
    void removeTag(const QString &key);

    void setEmotion(ServerPlayer *target, const QString &emotion);

    Player::Place getCardPlace(int card_id) const;
    ServerPlayer *getCardOwner(int card_id) const;
    void setCardMapping(int card_id, ServerPlayer *owner, Player::Place place);

    void drawCards(ServerPlayer *player, int n);
    void obtainCard(ServerPlayer *target, const Card *card);
    void obtainCard(ServerPlayer *target, int card_id);

    void throwCard(const Card *card);
    void throwCard(int card_id);
    void moveCardTo(const Card *card, ServerPlayer *to, Player::Place place, bool open = true);
    void doMove(const CardMoveStruct &move, const QSet<ServerPlayer *> &scope);

    // interactive methods
    void activate(ServerPlayer *player, CardUseStruct &card_use);
    Card::Suit askForSuit(ServerPlayer *player);
    QString askForKingdom(ServerPlayer *player);
    bool askForSkillInvoke(ServerPlayer *player, const QString &skill_name, const QVariant
&data = QVariant());
    QString askForChoice(ServerPlayer *player, const QString &skill_name, const QString
&choices);
    bool askForDiscard(ServerPlayer *target, const QString &reason, int discard_num, bool
optional = false, bool include_equip = false);
```

```
    const Card *askForExchange(ServerPlayer *player, const QString &reason, int discard_num);
    bool askForNullification(const TrickCard *trick, ServerPlayer *from, ServerPlayer *to, bool
positive);
    bool isCanceled(const CardEffectStruct &effect);
    int askForCardChosen(ServerPlayer *player, ServerPlayer *who, const QString &flags, const
QString &reason);
    const Card *askForCard(ServerPlayer *player, const QString &pattern, const QString
&prompt, const QVariant &data = QVariant());
    bool askForUseCard(ServerPlayer *player, const QString &pattern, const QString &prompt);
    int askForAG(ServerPlayer *player, const QList<int> &card_ids, bool refusable, const QString
&reason);
    const Card *askForCardShow(ServerPlayer *player, ServerPlayer *requestor, const QString
&reason);
    bool askForYiji(ServerPlayer *guojia, QList<int> &cards);
    const Card *askForPindian(ServerPlayer *player, ServerPlayer *from, ServerPlayer *to,
const QString &reason);
    ServerPlayer *askForPlayerChosen(ServerPlayer *player, const QList<ServerPlayer *>
&targets, const QString &reason);
    QString askForGeneral(ServerPlayer *player, const QStringList &generals, QString
default_choice = QString());
    void askForGeneralAsync(ServerPlayer *player);
    const Card *askForSinglePeach(ServerPlayer *player, ServerPlayer *dying);

    void speakCommand(ServerPlayer *player, const QString &arg);
    void trustCommand(ServerPlayer *player, const QString &arg);
    void kickCommand(ServerPlayer *player, const QString &arg);
    void surrenderCommand(ServerPlayer *player, const QString &);
    void commonCommand(ServerPlayer *player, const QString &arg);
    void addRobotCommand(ServerPlayer *player, const QString &arg);
    void fillRobotsCommand(ServerPlayer *player, const QString &arg);
    void chooseCommand(ServerPlayer *player, const QString &general_name);
    void choose2Command(ServerPlayer *player, const QString &general_name);
    void broadcastProperty(ServerPlayer *player, const char *property_name, const QString
&value = QString());
    void broadcastInvoke(const char *method, const QString &arg = ".", ServerPlayer *except =
NULL);
    void startTest(const QString &to_test);

    //Joshua
    void addCastrateGenerals(const General* general);
    void resetCastrateGenerals();

protected:
    virtual void run();
```

```
private:
    QString mode;
    QList<ServerPlayer*> players, alive_players;
    ServerPlayer *owner;
    int player_count;
    ServerPlayer *current;
    ServerPlayer *reply_player;
    QList<int> pile1, pile2;
    QList<int> table_cards;
    QList<int> *draw_pile, *discard_pile;
    bool game_started;
    bool game_finished;
    int signup_count;
    lua_State *L;
    QList<AI *> ais;

    RoomThread *thread;
    RoomThread3v3 *thread_3v3;
    RoomThread1v1 *thread_1v1;
    QSemaphore *sem;
    QString result;
    QString reply_func;

    QHash<QString, Callback> callbacks;

    QMap<int, Player::Place> place_map;
    QMap<int, ServerPlayer*> owner_map;

    const Card *provided;

    QVariantMap tag;
    const Scenario *scenario;

    bool _virtual;

    static QString generatePlayerName();
    void prepareForStart();
    void assignGeneralsForPlayers(const QList<ServerPlayer *> &to_assign);
    void chooseGenerals();
    AI *cloneAI(ServerPlayer *player);
    void broadcast(const QString &message, ServerPlayer *except = NULL);
    void initCallbacks();
    void arrangeCommand(ServerPlayer *player, const QString &arg);
```

```
    void takeGeneralCommand(ServerPlayer *player, const QString &arg);
    QString askForOrder(ServerPlayer *player);
    void selectOrderCommand(ServerPlayer *player, const QString &arg);
    QString askForRole(ServerPlayer *player, const QStringList &roles, const QString &scheme);
    void selectRoleCommand(ServerPlayer *player, const QString &arg);

    void makeCheat(const QString &cheat_str);
    void makeDamage(const QStringList &texts);
    void makeKilling(const QStringList &texts);
    void makeReviving(const QStringList &texts);

    //Joshua
    QList<const General*> castrateGenerals;
private slots:
    void reportDisconnection();
    void processRequest(const QString &request);
    void assignRoles();
    void startGame();

signals:
    void room_message(const QString &msg);
    void game_start();
    void game_over(const QString &winner);
};
```

这么长也不用怕，我们只需要了解与游戏过程有关的部分即可。

playSkillEffect(const QString &skill_name, int index = -1)：播放技能台词。

getCurrent()：返回被激活的玩家（回合内）。

setCurrent(ServerPlayer *current)：激活某玩家。

alivePlayerCount()：计算活着的玩家。

getOtherPlayers(ServerPlayer *except)：获取其他玩家。

getAllPlayers()：以当前玩家为起点获取所有玩家。

getAlivePlayers()：获取活着的玩家。

enterDying(ServerPlayer *player, DamageStruct *reason)：进入濒死状态。

killPlayer(ServerPlayer *victim, DamageStruct *reason = NULL)：杀死玩家。

revivePlayer(ServerPlayer *player)：复活某玩家。

aliveRoles(ServerPlayer *except = NULL)：获取活着的游戏角色。

gameOver(const QString &winner)：游戏结束。

slashEffect(const SlashEffectStruct &effect)：杀的效果。函数中包含 SlashEffect、SlashEffected 两个事件的触发。它们一个是造成杀的效果，如朱雀羽扇使无属性杀变火杀、纵火、雌雄双股剑、青釭剑、激昂摸牌；另一个是受到杀的效果，如仁王盾使黑杀无效、藤甲使普通杀无效、享乐令杀者弃基本牌、激昂摸牌。

slashResult(const SlashEffectStruct &effect, const Card *jink)：杀的结果。函数中包含 SlashHit、SlashMissed 两个事件的触发。前者是杀命中，如麒麟弓、寒冰剑，结果是造成 1 点杀的伤害、若此杀的杀伤力被酒加强，则造成 2 点伤害；后者是杀被闪避，如贯石斧、青龙偃月刀、猛进。

attachSkillToPlayer(ServerPlayer *player, const QString &skill_name)：给玩家附加技能，一般是与他人有关或武器的技能，如黄天、制霸、义舍要牌等。

detachSkillFromPlayer(ServerPlayer *player, const QString &skill_name)：同上，只不过是卸掉技能。

obtainable(const Card *card, ServerPlayer *player)：判断某人是否可以获得某牌。虚拟卡牌和自己手牌无法获得。

setPlayerFlag(ServerPlayer *player, const QString &flag)：给玩家添加或去掉某标志。

setPlayerProperty(ServerPlayer *player, const char *property_name, const QVariant &value)：修改某玩家的某属性，多用于觉醒技修改血量上限、sp 武将与原版武将变身时修改国籍。

setPlayerMark(ServerPlayer *player, const QString &mark, int value)：设置玩家的标记物。

useCard(const CardUseStruct &card_use, bool add_history = true)：使用某卡牌。

damage(const DamageStruct &data)：处理伤害。

loseHp(ServerPlayer *victim, int lose = 1)：流失体力。

loseMaxHp(ServerPlayer *victim, int lose = 1)：流逝体力上限。

recover(ServerPlayer *player, const RecoverStruct &recover, bool set_emotion = false)：恢复体力。

playCardEffect(const QString &card_name, bool is_male)：播放卡牌音效。

cardEffect(const Card *card, ServerPlayer *from, ServerPlayer *to)、cardEffect(const CardEffectStruct &effect)：卡牌使用效果。

judge(JudgeStruct &judge_struct)：

getNCards(int n, bool update_pile_number = true)：摸 N 张牌。

getLord()：获得该次游戏身份为主公的玩家。

doGuanxing(ServerPlayer *zhuge, const QList<int> &cards, bool up_only)：执行观星及相似技能，如心战。

doGongxin(ServerPlayer *shenlumeng, ServerPlayer *target)：执行攻心。

drawCard()：抓一张牌。

fillAG(const QList<int> &card_ids, ServerPlayer *who = NULL)：填充所有合理卡牌到 AG。AG 应该是 AmazingGrace 的缩写，神上把五谷丰登翻译为 AmazingGrace。但类似五谷效果，可以打开一个窗口，显示多张牌的技能也可以使用 XXXAG 这一系列的函数。比如七星、不屈、急袭、固政等。

takeAG(ServerPlayer *player, int card_id)：从 AG 中选择一张牌。

provide(const Card *card)：由系统提供某牌，即无需自己打出。如八卦判红的闪，激将的杀，护驾的闪。

getLieges(const QString &kingdom, ServerPlayer *lord)：获得臣民。用于返回某主公麾下的武将。

showCard(ServerPlayer *player, int card_id, ServerPlayer *only_viewer = NULL)：展示某牌，如火攻。

showAllCards(ServerPlayer *player, ServerPlayer *to = NULL)：展示所有牌。

acquireSkill(ServerPlayer *player, const Skill *skill, bool open = true)、acquireSkill(ServerPlayer *player, const QString &skill_name, bool open = true)：游戏过程中获得技能，如觉醒技、左慈化身等。

adjustSeats()：调整座次。

swapPile()：洗牌。

getCardFromPile(const QString &card_pattern)：从牌堆获得某牌。

findPlayer(const QString &general_name, bool include_dead = false)、findPlayerBySkillName(const QString &skill_name, bool include_dead = false)：通过武将名、技能名找到符合条件的玩家。

installEquip(ServerPlayer *player, const QString &equip_name)：装上装备。

transfigure(ServerPlayer *player, const QString &new_general, bool full_state, bool invoke_start = true)：更改武将 sp 武将和原武将的变身机制、杀死蔡文姬变素将、倚天陆伯言男女互变都是用的这个函数。

swapSeat(ServerPlayer *a, ServerPlayer *b)：交换座次。倚天蔡昭姬的归汉使用了它。

setFixedDistance(Player *from, const Player *to, int distance)：锁定两玩家的距离，高顺陷阵拼点胜利使用了该函数。

getFront(ServerPlayer *a, ServerPlayer *b)：获得 a、b 两玩家相对于当前玩家靠前的一位。

isProhibited(const Player *from, const Player *to, const Card *card)：判断某玩家对另一玩家使用某牌是否被禁止。

getCardPlace(int card_id)：获取某牌的位置。

getCardOwner(int card_id)：获取某牌的拥有者。

drawCards(ServerPlayer *player, int n)、obtainCard(ServerPlayer *target, const Card *card)、throwCard(const Card *card)、moveCardTo(const Card *card, ServerPlayer *to, Player::Place place, bool open = true)：摸牌、获得牌、弃牌、移动牌。

下面是一些提示用的函数：

askForSuit(ServerPlayer *player)：周瑜反间提示选择花色。

askForKingdom(ServerPlayer *player)：神将游戏开始提示选择国籍。

askForSkillInvoke(ServerPlayer *player, const QString &skill_name, const QVariant &data = QVariant())：提示技能触发，选择是否发动。

askForChoice(ServerPlayer *player, const QString &skill_name, const QString &choices)：当技能可以有不同选项时，提示玩家选择，如旋风、琴音、明策等。

askForDiscard(ServerPlayer *target, const QString &reason, int discard_num, bool optional = false, bool include_equip = false)：提示弃牌。

askForExchange(ServerPlayer *player, const QString &reason, int discard_num)：提示交换牌，如七星、共谋（倚天钟士季）。

askForNullification(const TrickCard *trick, ServerPlayer *from, ServerPlayer *to, bool positive)：提示是否打出无懈。

isCanceled(const CardEffectStruct &effect)：返回某卡牌效果是否被取消，也就是锦囊是否被无懈。

askForCardChosen(ServerPlayer *player, ServerPlayer *who, const QString &flags, const QString &reason)：提示选牌，如顺拆、挑衅对方没杀。

askForCard(ServerPlayer *player, const QString &pattern, const QString &prompt, const QVariant &data = QVariant())：提示要求打出某牌，如杀和万箭提示闪、南蛮和挑衅提示杀。

askForUseCard(ServerPlayer *player, const QString &pattern, const QString &prompt)：提示你可以使用某牌，包括技能牌。如突袭、流离。

askForAG(ServerPlayer *player, const QList<int> &card_ids, bool refusable, const QString &reason)：提示显示 AG。

askForCardShow(ServerPlayer *player, ServerPlayer *requestor, const QString &reason)：提示展示手牌，如火攻、补益、乐学（倚天姜伯约）。

askForYiji(ServerPlayer *guojia, QList<int> &cards)：提示使用遗计发牌。

askForPindian(ServerPlayer *player, ServerPlayer *from, ServerPlayer *to, const QString &reason)：提示打出拼点牌。

askForPlayerChosen(ServerPlayer *player, const QList<ServerPlayer *> &targets, const QString &reason)：提示选择玩家。

askForGeneral(ServerPlayer *player, const QStringList &generals, QString default_choice = QString())：提示选择武将，如左慈的化身以及游戏开始的选将。

askForSinglePeach(ServerPlayer *player, ServerPlayer *dying)：玩家濒死时提示出一个桃。

最后我们看下 room 类的成员变量：

```
QString mode;
QList<ServerPlayer*> players, alive_players;
ServerPlayer *owner;
int player_count;
ServerPlayer *current;
ServerPlayer *reply_player;
QList<int> pile1, pile2;
QList<int> table_cards;
QList<int> *draw_pile, *discard_pile;
bool game_started;
bool game_finished;
int signup_count;
lua_State *L;
QList<AI *> ais;
RoomThread *thread;
RoomThread3v3 *thread_3v3;
RoomThread1v1 *thread_1v1;
QSemaphore *sem;
```

```
QString result;
QString reply_func;
QHash<QString, Callback> callbacks;
QMap<int, Player::Place> place_map;
QMap<int, ServerPlayer*> owner_map;
const Card *provided;
```

从上至下分别为：

mode：游戏模式。

players, alive_players：所有玩家、存活玩家。

owner：与 DIY 无关，可忽略。

player_count：玩家数量。

current：当前激活玩家。

reply_player：与 DIY 无关，可忽略。

pile1, pile2：与 DIY 无关，可忽略。

table_cards：摆在桌面上但未进入弃牌堆的牌，如判定牌、孟获再起的牌。

draw_pile, discard_pile：摸牌堆、弃牌堆。

game_started, game_finished：游戏是否开始、结束。

从 signup_count 到 callbacks：与 DIY 无关，可忽略。

place_map：记录了每张牌及其位置。

owner_map：记录每张牌及其拥有者。

剩下的部分成员变量、私有函数以及信息和槽与 DIY 无关，就不讲了。

OK，感谢大家看完了神杀的基础教学，下一讲就开始 DIY 了，相信大家也等了很长时间。先说下 DIY 部分的教学安排：先讲如何添加一个新的扩展包，然后是找几个有代表性的 DIY 武将实现，如果这些差不多了就开始 DIY 卡牌的教学。

第十篇：太阳神三国杀的武将 DIY—在神杀中新建一个扩展包

我们 DIY 武将必须把他放到一个包里，从这点考虑，我们完全可以把他们放到已经有的包里。但是这样会使你的代码和原来的代码混在一起，不便于管理。所以，添加属于自己的扩展包是必要的。

第一步：

在 package 文件夹中建立属于你扩展包的头文件 MyPackage.h，然后加入以下代码：

```
#include "package.h"
#include "card.h"
class MyPackage : public Package{
    Q_OBJECT
public:
    MyPackage();
};
```

以上代码声明了 MyPackage 类，如果这个扩展包中的武将有需要技能牌的技能，还要在下面加上各种技能牌的声明。需要注意的是，扩展包类的命名必须以 Package 结尾。

第二步：

在 package 文件夹中建立属于你扩展包的源文件 MyPackage.cpp，然后加入以下代码：

```
#include "MyPackage.h"
#include "general.h"
#include "skill.h"
#include "standard.h"
#include "client.h"
#include "carditem.h"
#include "engine.h"
#include "maneuvering.h"

MyPackage::MyPackage()
    :Package("My")
{

}

ADD_PACKAGE(My)
```

将来武将技能的代码、技能牌的代码和 MyPackage 类的构造函数的内容（在此添加武将和关联其技能），都要写在这里（ADD_PACKAGE(My)）之上。Package("My")是有讲究的，My 就是 MyPackage 去掉 Package 后的部分，自定义扩展包时必须遵守这个规定。

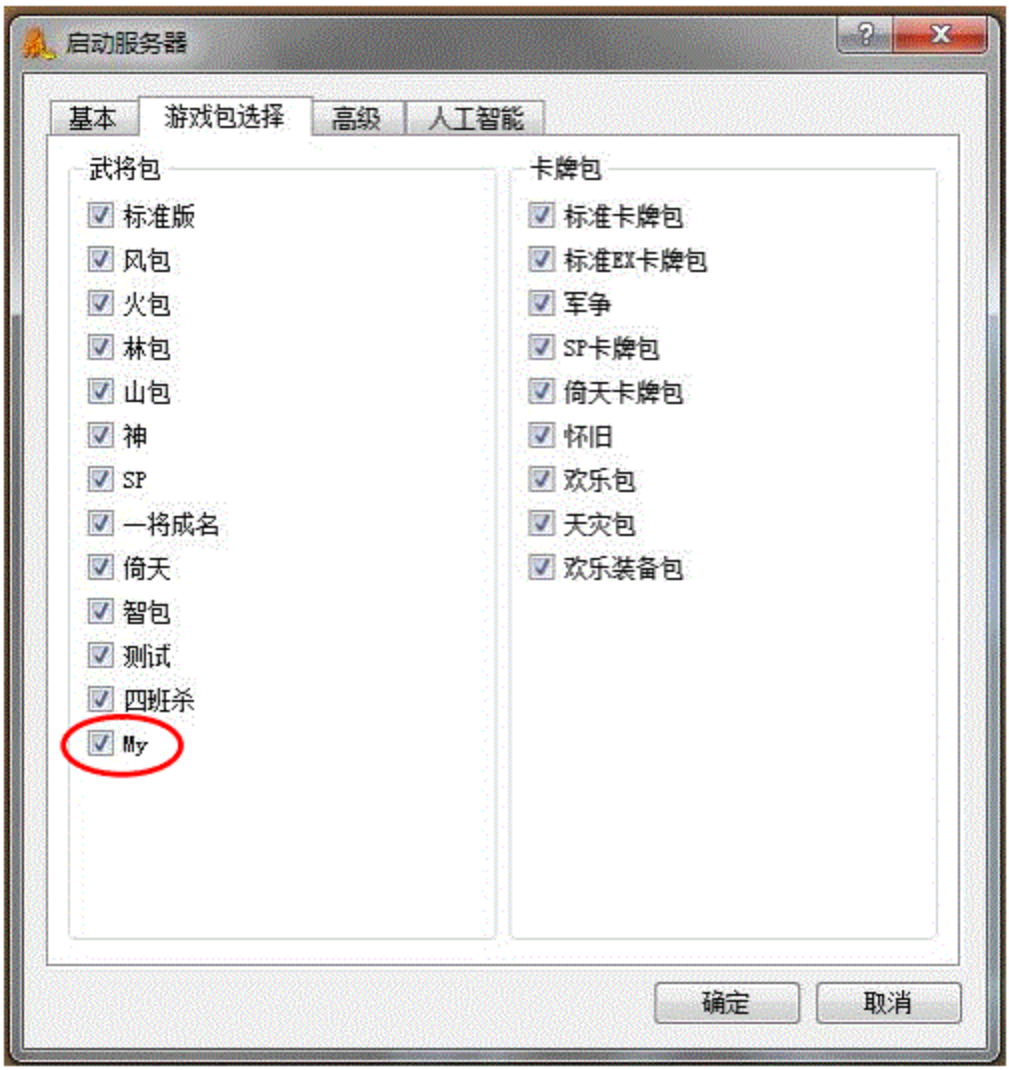
第三步：

在 core 文件夹下找到 engin.cpp，在 Package *NewStandard(); …… Package *NewTest(); 下加上 Package *NewMy();

在 addPackage(NewStandard()); …… addPackage(NewTest());下加上 addPackage(NewMy());

第四步：

编译代码，运行游戏，单机启动，选择游戏包选择，你会看到我们新加的 My 包出现了。



第十一篇：太阳神三国杀的武将 DIY—武将技能的分类及示例

之前有提到过，太阳神三国杀中武将的技能分为视作技、触发技、禁止技、距离技和耦合技。今天我们把这几类技能再细分一下:

视作技:

ViewAsSkill-用不定或多于一张的卡牌来主动发动的技能，如仁德、制衡、结姻等。

ZeroCardViewAsSkill-继承 ViewAsSkill，不需要卡牌来主动发动的技能，如乱武、挑衅、酒诗等。

OneCardViewAsSkill-继承 ViewAsSkill，用一张卡牌来主动发动的技能，如武圣、龙胆、奇袭等。

FilterSkill-继承 OneCardViewAsSkill，完成"锁定视为"的功能，如红颜、戒酒、武神。

触发技：

TriggerSkill-由一个或多个事件触发发动的技能，如护驾、枭姬、天妒等。

MasochismSkill-继承于 TriggerSkill，卖血技，如奸雄、反馈、刚烈等。

PhaseChangeSkill-继承于 TriggerSkill，在游戏阶段变化时触发发动的技能，如观星、突袭、闭月等。

DrawCardSkill-继承于 TriggerSkill，摸牌阶段改变摸牌数目的技能，如英姿、裸衣、绝境等。

SlashBuffSkill-继承于 TriggerSkill，杀辅助技能，如铁骑、烈弓。

GameStartSkill-继承于 TriggerSkill，游戏开始时需要特殊操作的技能，如化身、七星

SPConvertSkill-继承于 GameStartSkill,用来完成原版武将和 SP 武将的变身。

禁止技：

ProhibitSkill-禁止拥有该类技能的武将成为某类牌的目标，如空城、谦逊、帷幕

距离技：

DistanceSkill-用来修正玩家之间的距离，如马术、飞影、屯田等

系统耦合技：

这类技能没有自己的类，而是写在了基本卡牌和玩家类的函数中，一般这类技能与出杀次数、攻击距离、杀的目标个数、手牌上限、使用锦囊的距离限制、完杀等有关。

这里，我们把所有种类的技能都列举完了，在编程实现 DIY 技能之前，我们先考虑好 DIY 的技能是以上的哪种或哪几种的组合就可以了。

先来一个视作技御兽，类似乱击，但是是两张点数大于等于 10 的手牌当南蛮入侵用，这里的重点在 viewFilter 函数里，viewAs 函数相对比较简单：

```
class Yushou:public ViewAsSkill{
public:
    Yushou():ViewAsSkill("yushou"){
    }
    virtual bool viewFilter(const QList<CardItem *> &selected, const CardItem *to_select)
const{
        if(selected.isEmpty())
            return !to_select->isEquipped() && (to_select->getFilteredCard()->getNumber() >
9);
        else if(selected.length() == 1){
            return !to_select->isEquipped() && (to_select->getFilteredCard()->getNumber() >
9);
        }else
```

```
            return false;
        }
        virtual const Card *viewAs(const QList<CardItem *> &cards) const{
            if(cards.length() == 2){
                const Card *first = cards.first()->getCard();
                SavageAssault *sa = new SavageAssault(first->getSuit(), 0);
                sa->addSubcards(cards);
                sa->setSkillName(objectName());
                return sa;
            }else
                return NULL;
        }
};
```

然后是一个触发技\卖血技复仇，每当你受到一次伤害时，你可以指定一名角色，视为对他出了一张无属性无花色的杀。这里需要注意的是对杀的目标的判断，要剔除诸葛的空城。

```
class Fuchou:public MasochismSkill{
public:
    Fuchou():MasochismSkill("fuchou"){
    }
    virtual void onDamaged(ServerPlayer *yuyang, const DamageStruct &damage) const{
        Room *room = yuyang->getRoom();
        QList<ServerPlayer *> targets;
        foreach(ServerPlayer *target, room->getAlivePlayers()){
            if(yuyang->canSlash(target, false) && !(target->hasSkill("kongcheng") &&
target->isKongcheng()))
                targets << target;
        }
        if(targets.empty())
            return;
        if(room->askForSkillInvoke(yuyang, "fuchou"))
        {
            ServerPlayer *target = room->askForPlayerChosen(yuyang, targets,
"fuchou-slash");
            Slash *slash = new Slash(Card::NoSuit, 0);
            slash->setSkillName(objectName());
            CardUseStruct card_use;
            card_use.card = slash;
            card_use.from = yuyang;
            card_use.to << target;
            room->useCard(card_use, false);
        }
    }
};
```

最后来一个禁止技禁武，该角色不能成为杀的目标。

```
class Jinwu: public ProhibitSkill{
public:
    Jinwu():ProhibitSkill("jinwu"){
    }
    virtual bool isProhibited(const Player *, const Player *, const Card *card) const{
        return card->inherits("Slash");
    }
};
```

不知为什么，我用两个真人对战的时候这个禁止技管用，但是 AI 还能对我出杀。后来我查看空城的实现方法，发现 core\player.cpp 中的 canSlash 函数里还要加上，这样一个禁止技就完成了：

```
if(other->hasSkill("jinwu"))
        return false;
```

第十二篇：太阳神三国杀的卡牌 DIY—打造属于自己的神兵利器(武器篇)

这次我们讲解如何在太阳神三国杀里编写自己 DIY 的装备，为了方便管理，我们可以参照第十篇的内容自己添加一个新的装备扩展包叫 EquipDIY，把我们 DIY 的装备都放到这个包里编写。

首先是大部分武器技能的实现都是使用一个叫 WeaponSkill 的类完成的，WeaponSkill 继承了 TriggerSkill，所以算是触发技，也就是说，大部分武器技能的使用都是需要一个特定事件来触发的。比如寒冰剑、麒麟弓是在杀命中时触发，青龙偃月刀、贯石斧是在杀被闪避时触发，朱雀羽扇、雌雄双股剑、青釭剑是在杀指定目标后生效时触发。

但是诸葛连弩、方天画戟的实现耦合在了“杀”的代码里，属于系统耦合技；青釭剑无视防具实际上使用了 Mark，属于触发技和系统耦合技混合实现；丈八蛇矛、贯石斧弃牌属于视作技。因此，这一讲我们针对这几种不同的实现方法给几个例子。

由简到繁，先来一个简单的：

可以把这个武器理解成雷属性版的朱雀羽扇，所以实现它还是很容易的。

在 EquipDIY.h 中声明这个武器的类，注意它是继承 Weapon 类的：

```
class Harp: public Weapon{
    Q_OBJECT
public:
    Q_INVOKABLE Harp(Card::Suit suit = Club, int number = 3);
};
```

然后在 EquipDIY.cpp 中实现镇魂琴的技能 HarpSkill：

```
class HarpSkill: public WeaponSkill{
public:
    HarpSkill():WeaponSkill("harp"){
        events << SlashEffect;
    }
    virtual bool trigger(TriggerEvent , ServerPlayer *player, QVariant &data) const{
        SlashEffectStruct effect = data.value<SlashEffectStruct>();
        if(effect.nature == DamageStruct::Normal){
            if(player->getRoom()->askForSkillInvoke(player, objectName(), data)){
                effect.nature = DamageStruct::Thunder;
                data = QVariant::fromValue(effect);
```

```
            }
        }
        return false;
    }
};
```

最后再完成 EquipDIY.h 中声明的构造函数，在这个函数中设置好武器的攻击范围、名称，挂接好它的 WeaponSkill 即可。

```
Harp::Harp(Suit suit, int number):Weapon(suit, number, 4){
    setObjectName("harp");
    skill = new HarpSkill;
}
```

第二个武器是用视作技实现的：

这个武器的难点就在于如何确定是在决斗过程中，为了区分决斗过程中和非决斗过程，我们需要找到决斗的代码并稍作修改，在 package\standard-cards.cpp 中，红色的部分代表我们需要的修改：在决斗出杀的这个循环开始之前，对决斗双方添加“in-duel”标记，循环结束之后，对决斗双方去除“in-duel”标记。这样，我们只需要查看装备这件武器的玩家是否有“in-duel”标记就可以决定是否可以发动武器技能了。

```
void Duel::onEffect(const CardEffectStruct &effect) const{
    ServerPlayer *first = effect.to;
    ServerPlayer *second = effect.from;
    Room *room = first->getRoom();
    room->setEmotion(first, "duel-a");
    room->setEmotion(second, "duel-b");
    first->gainMark("in-duel");
    second->gainMark("in-duel");
    forever{
        ............
    }
    first->loseMark("in-duel");
    second->loseMark("in-duel");
    DamageStruct damage;
```

```
    damage.card = this;
    damage.from = second;
    damage.to = first;
    room->damage(damage);
}
```

下面我们来完成这个视作技：由于这个技能不能主动发动，只有决斗时响应出杀时才能发动，因此我们要重写 isEnabledAtPlay 函数，使它永远返回 false；isEnabledAtResponse 中，我们要查看是否在决斗过程中、是否响应的牌为杀；viewFilter 将装备牌过滤掉，因为技能描述中只有手牌能当杀使用；viewAs 中完成了一张手牌视为一张杀，新建一张点数与花色和选中的牌相同的杀，并将选中的牌作为此杀的子卡牌即可。

```
class MaceSkill: public OneCardViewAsSkill{
public:
    MaceSkill():OneCardViewAsSkill("mace"){
    }
    virtual bool isEnabledAtPlay(const Player * player) const{
        return false;
    }
    virtual bool isEnabledAtResponse(const Player * player, const QString &pattern) const{
        return pattern == "slash" &&    (player->getMark("in-duel") == 1);
    }
    virtual bool viewFilter(const CardItem *to_select) const{
        return !to_select->isEquipped();
    }
    virtual const Card *viewAs(CardItem *card_item) const{
        const Card *card = card_item->getCard();
        Card *slash = new Slash(card->getSuit(), card->getNumber());
        slash->addSubcard(card->getId());
        slash->setSkillName(objectName());
        return slash;
    }
};
```

最后我们完成构造函数，还是设定武器名称，由于没有触发技，因此和第一个武器有所不同，attach_skill 用来在左下角显示技能按钮来主动发动，像丈八蛇矛一样：

```
Mace::Mace(Suit suit, int number)
    :Weapon(suit, number, 3)
{
    setObjectName("mace");
    attach_skill = true;
}
```

第三个武器该是耦合技实现的了，比如我来个武器叫双刃剑（名字弱爆了），技能是装备后可以额外指定一个杀的目标，那么可以参照方天画戟。

构造函数只用来命名就可以了：

```
DoubleEdgeSword:: DoubleEdgeSword (Suit suit, int number)
    :Weapon(suit, number, 3)
```

```
{
    setObjectName("double_edge_sword");
}
```

在 stardard-cards.cpp 的 slash 的 targetFilter 中加入以下代码即可：

```
if(Self->hasWeapon("double_edge_sword ")){
        slash_target++;
}
```

第十三篇：太阳神三国杀的卡牌 DIY—打造属于自己的神兵利器(防具篇)

比起武器来，防具最大的特点就是技能触发的被动性，不过这并不是说我们 DIY 防具的时候不能有主动发动的技能。下一讲马的 DIY 就有这样的例子，如果要给防具加上主动技能的话，参照那个例子即可。

下面是例子，这件防具有伤害使无效的能力，卸掉时还可以摸牌。如果希望装备在装卸过程中有技能触发，像白银狮子那样，需要在声明的时候重载 onInstall 和 onUninstall 函数：


```
class Dress:public Armor{
    Q_OBJECT
public:
    Q_INVOKABLE Dress(Card::Suit suit = Diamond, int number = 3);
    virtual void onUninstall(ServerPlayer *player) const;
};
Dress::Dress(Suit suit, int number):Armor(suit, number){
    setObjectName("dress");
    skill = new DressSkill;
}
```

失去装备时摸 2 张牌在 onUninstall 里实现

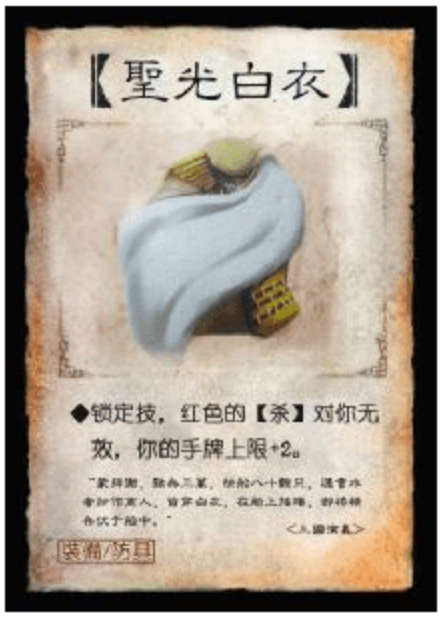
```
void Dress::onUninstall(ServerPlayer *player) const{
    if(player->isAlive() && player->getMark("qinggang") == 0){
        player->drawCards(2);
    }
}
```

使伤害无效就是在受到的伤害即将造成之前 Predamaged，使伤害点数为 0，我们只需要在 trigger 函数中返回 true，之后的结算就不会执行；如果返回 false，在 DamageDone 事件中会完成扣血的操作。

```
class DressSkill: public ArmorSkill{
public:
    DressSkill():ArmorSkill("dress"){
        events << Predamaged;
    }
    virtual bool trigger(TriggerEvent event, ServerPlayer *player, QVariant &data) const{
```

```
        DamageStruct damage = data.value<DamageStruct>();
        if(damage.to->getHp() == 1){
            Room *room = player->getRoom();
            LogMessage log;
            log.type = "#DressProtect";
            log.from = player;
            log.arg = QString::number(damage.damage);
            log.arg2 = "damage";
            room->sendLog(log);
            return true;
        }
        return false;
    }
};
```

第二个例子，带有一点系统耦合技：




```
class HaloCape:public Armor{
    Q_OBJECT
public:
    Q_INVOKABLE HaloCape(Card::Suit suit = Heart, int number = 2);
};
```

结合上一个防具的技能，使 XXX 无效就是在 trigger 里加一个条件语句，如果符合技能触发条件就返回 true 即可。

```
class HaloCapeSkill: public ArmorSkill{
public:
    HaloCapeSkill():ArmorSkill("halo_cape"){
        events << SlashEffected;
    }
    virtual bool trigger(TriggerEvent event, ServerPlayer *player, QVariant &data) const{
        SlashEffectStruct effect = data.value<SlashEffectStruct>();
```

```
            if(effect.slash->isRed()){
                LogMessage log;
                log.type = "#ArmorNullify";
                log.from = player;
                log.arg = objectName();
                log.arg2 = effect.slash->objectName();
                player->getRoom()->sendLog(log);
                return true;
            }else
                return false;
        }
};
HaloCape::HaloCape(Suit suit, int number)
    :Armor(suit, number)
{
    setObjectName("halo_cape");
    skill = new HaloCapeSkill;
}
```

增加手牌上限要在 core\player.cpp 的 getMaxCards 里修改：

```
int Player::getMaxCards() const{
            ..........
    int halo_cape = 0;
    if(getArmor() && getArmor()->inherits("HaloCape"))
        halo_cape = 2;
    return qMax(hp,0) + extra + juejing + xueyi + shenwei + halo_cape;
}
```

第十四篇：太阳神三国杀的卡牌 DIY—打造属于自己的神兵利器(马篇)

普通的+1 马-1 马我们略过不提，先看下+2 马如何实现：

只需要把距离的修正值改成+2 即可（下方代码红色粗体部分）

```
class Elephant: public Horse{
    Q_OBJECT
public:
    Q_INVOKABLE Elephant(Card::Suit suit = Diamond, int number = 13, int correct = +2);
    virtual QString getSubtype() const;
};
Elephant::Elephant(Suit suit, int number, int correct)
    :Horse(suit, number, correct)
{
    setObjectName("elephant");
}
QString Elephant::getSubtype() const{
    return "defensive_horse";
}
```

如果要给马加上技能：

声明类的时候由于马不带技能，因此我们要加一个 triggerSkill 的私有变量，还要重载 onInstall 函数来把 hawk_skill 赋值。

```
class Hawk: public DefensiveHorse{
    Q_OBJECT
public:
    Q_INVOKABLE Hawk(Card::Suit suit = Heart, int number = 1);
    virtual void onInstall(ServerPlayer *player) const;
private:
    TriggerSkill *hawk_skill;
};
```

这个战鹰的功能就是回合开始有一张牌的观星，为了正常触发技能，需要重载 triggerable 函数，如果装备了战鹰即可触发技能；三国杀游戏中，一般武将技能的优先级比装备技能的优先级要高，如果甄姬装备了战鹰，会先洛神后观星，为了改变这个顺序，我们需要提高这个技能的优先级，把优先级设为 2，这是锁定技的优先级，具体请看神上的讲解：http://qsanguosha.com/forum.php?mod=viewthread&tid=2969&extra=page%3D1

```
class HawkSkill:public PhaseChangeSkill{
public:
    HawkSkill():PhaseChangeSkill("hawk"){
    }
    virtual bool triggerable(const ServerPlayer *player) const{
        return player->getDefensiveHorse()&&player->getDefensiveHorse()->inherits("Hawk");
    }
    virtual int getPriority() const{
        return 2;
    }
    virtual bool onPhaseChange(ServerPlayer *player) const{
        if(player->getPhase() == Player::Start &&
            player->askForSkillInvoke(objectName()))
        {
            Room *room = player->getRoom();
            room->playSkillEffect(objectName());
            room->doGuanxing(player, room->getNCards(1, false), false);
        }
        return false;
    }
};
```

构造函数：完成命名、给私有变量 hawk_skill 赋值。

```
Hawk::Hawk(Suit suit, int number)
    :DefensiveHorse(suit, number)
{
    setObjectName("hawk");
    hawk_skill = new HawkSkill;
    hawk_skill->setParent(this);
}
```

OnInstall 函数：在装备战鹰的时候，玩家获得 hawk_skill 技能，参照欢乐包的猴子即可。

```
void Hawk::onInstall(ServerPlayer *player) const{
    player->getRoom()->getThread()->addTriggerSkill(hawk_skill);
}
```

第十五篇：太阳神三国杀的卡牌 DIY—打造属于自己的锦囊（一）

三国杀中锦囊分为全局效果（五谷、桃园）、AOE（南蛮、万箭）、单体锦囊（决斗、顺拆、借刀、火攻、无懈）、延时锦囊（乐、兵、天灾）、伤害传导（铁锁）。除了铁锁的重铸比较特殊的用了 onUse 这个函数外，其他锦囊都是通过重载 onEffect 这个函数来实现的。

首先我们来 DIY 个 AOE，声明的类继承 AOE 类：


```
class Argue:public AOE{
    Q_OBJECT
public:
    Q_INVOKABLE Argue(Card::Suit suit, int number);
    virtual void onEffect(const CardEffectStruct &effect) const;
};
Argue::Argue(Card::Suit suit, int number)
    :AOE(suit, number)
{
    setObjectName("argue");
}
```

我们要达到的效果写在 onEffect 里，函数里面我就不细讲了，相信大家差不多能看懂。

```
void Argue::onEffect(const CardEffectStruct &effect) const{
    Room *room = effect.to->getRoom();
    const Card *trick = room->askForCard(effect.to, "trick", "argue-trick:" +
effect.from->objectName());
    if(trick == NULL){
        if(!effect.to->isNude()){
            int card_id = room->askForCardChosen(effect.from, effect.to, "he", "argue");
            room->throwCard(card_id);
        }
        else{
            DamageStruct damage;
            damage.card = this;
            damage.damage = 1;
```

```
            damage.from = effect.from;
            damage.to = effect.to;
            damage.nature = DamageStruct::Normal;
            room->damage(damage);
        }
    }
}
```

第二个是单体锦囊，继承 SingleTargetTrick 类，唯一需要说明的就是 targetFilter 这个函数，它用来滤除不符合锦囊条件的角色，比如顺的距离限制、火攻目标要有手牌等：


```
class Uninstall:public SingleTargetTrick{
    Q_OBJECT
public:
    Q_INVOKABLE Uninstall(Card::Suit suit, int number);
    virtual bool targetFilter(const QList<const Player *> &targets, const Player *to_select, const
Player *Self) const;
    virtual void onEffect(const CardEffectStruct &effect) const;
};
Uninstall::Uninstall(Suit suit, int number)
    :SingleTargetTrick(suit, number, true)
{
    setObjectName("uninstall");
}
```

在 targetFilter 中，我们只需判断玩家的装备区是否为空即可。

```
bool Uninstall::targetFilter(const QList<const Player *> &targets, const Player *to_select, const
Player *Self) const{
    if(to_select->getEquips().empty())
        return false;
    return targets.isEmpty();
}
void Uninstall::onEffect(const CardEffectStruct &effect) const{
    Room *room = effect.from->getRoom();
```

```
        room->throwCard(effect.to->getWeapon());
        room->throwCard(effect.to->getArmor());
        room->throwCard(effect.to->getDefensiveHorse());
        room->throwCard(effect.to->getOffensiveHorse());
}
```

第三个来个延时锦囊，继承 DelayedTrick 类。因为要在使用的时候记住施毒者，所以需要重载 use 函数：


```
class FillVenom:public DelayedTrick{
    Q_OBJECT
public:
    Q_INVOKABLE FillVenom(Card::Suit suit, int number);
    virtual bool targetFilter(const QList<const Player *> &targets, const Player *to_select, const Player *Self) const;
    virtual void use(Room *room, ServerPlayer *source, const QList<ServerPlayer *> &targets) const;
    virtual void takeEffect(ServerPlayer *target) const;
private:
    mutable ServerPlayer *poisoner;
};
```

judge.pattern 使用正则表达式实现，我们只需要写成"(.*):(失效花色):(.*)"即可

```
FillVenom::FillVenom(Suit suit, int number)
    :DelayedTrick(suit, number)
{
    setObjectName("fill_venom");
    target_fixed = false;
    judge.pattern = QRegExp("(.*):(heart|diamond):(.*)");
    judge.good = true;
    judge.reason = objectName();
}
bool FillVenom::targetFilter(const QList<const Player *> &targets, const Player *to_select, const Player *Self) const
{
    if(!targets.isEmpty())
        return false;
```

```
    if(to_select->containsTrick(objectName()))
        return false;
    if(to_select == Self)
        return false;
    return true;
}
```

在使用锦囊的时候就把使用者记为 poisoner，放在延时锦囊的私有变量里

```
void FillVenom::use(Room *room, ServerPlayer *source, const QList<ServerPlayer *> &targets)
const{
    poisoner = source;
    ServerPlayer *target = targets.value(0, source);
    room->moveCardTo(this, target, Player::Judging, true);
}
void FillVenom::takeEffect(ServerPlayer *target) const{
    DamageStruct damage;
    damage.card = this;
    damage.from = poisoner;
    damage.to = target;
    target->getRoom()->damage(damage);
}
```

第十六篇：太阳神三国杀的卡牌 DIY—打造属于自己的锦囊（二）

上一讲虽然例举了不少锦囊，但是纵观这个三国杀，有一种锦囊是最特殊的，因为只有它不能主动打出去，而是用来响应其他锦囊。没错，这个锦囊就是无懈可击，它在实现方法上也十分复杂，这一讲我们就来 DIY 一个被动的锦囊：

首先，现在 TrickDIY.h 中加入以下代码，这里完全参照无懈可击：

```
class SoloRun:public SingleTargetTrick{
    Q_OBJECT
public:
    Q_INVOKABLE SoloRun(Card::Suit suit, int number);
    virtual void use(Room *room, ServerPlayer *source, const QList<ServerPlayer *> &targets)
const;
    virtual bool isAvailable(const Player *player) const;
};
```

再在 TrickDIY.cpp 中加上：

```
SoloRun::SoloRun(Suit suit, int number)
    :SingleTargetTrick(suit, number, false)
{
    setObjectName("solo_run");
}
void SoloRun::use(Room *room, ServerPlayer *, const QList<ServerPlayer *> &) const{
    room->throwCard(this);
}
bool SoloRun::isAvailable(const Player *) const{
    return false;
}
```

无懈可击是在锦囊即将生效时用来打出响应，因此它在 gamerule 的 CardEffected 事件中，通过 isCanceled 这个函数里的 askForNullification 来完成。相似的，我们需要仿照设个函数写一个 askForSoloRun。在 server\room.cpp 中（在 room.h 中先声明函数原型不用说了吧）：

```
bool Room::askForSoloRun(const DamageStruct damage){
    /*记下造成伤害的卡牌，有些伤害没有卡牌，比如强袭，返回 no_card*/
    QString card_name;
```

```
if(damage.card)
    card_name = damage.card->objectName();
else
    card_name = "no_card";
QList<ServerPlayer *> players = getAllPlayers();
ServerPlayer *from = damage.from;
ServerPlayer *to = damage.to;
```

/*判断每个玩家有没有单刀赵会且受伤者是否在其攻击范围内，ServerPlayer 类和 AI 类的 hasSoloRun 函数也需要加上*/

```
foreach(ServerPlayer *player, players){
    if(!player->hasSoloRun() || !player->inMyAttackRange(to))
        continue;
    trust:
    AI *ai = player->getAI();
    const Card *card = NULL;
    if(ai){
        card = ai->askForSoloRun(damage);
        if(card)
            thread->delay(Config.AIDelay);
    }else{
        QString ask_str;
        ask_str = QString("%1:%2->%3").arg(card_name)
        .arg(from ? from->objectName() : ".")
        .arg(to->objectName());
        player->invoke("askForSoloRun", ask_str);
        getResult("responseCardCommand", player, false);
        if(result.isEmpty())
            goto trust;
        if(result != ".")
            card = Card::Parse(result);
    }
    if(card == NULL)
        continue;
    bool continable = false;
    card = card->validateInResposing(player, &continable);
    if(card){
        CardUseStruct use;
        use.card = card;
        use.from = player;
        useCard(use);
        LogMessage log;
        log.type = "#SoloRunDetails";
        log.from = from;
        log.to << to;
```

```
                log.arg = card_name;
                sendLog(log);
                broadcastInvoke("animate", QString("solo_run:%1:%2")
                                .arg(player->objectName()).arg(from->objectName()));
                QVariant decisionData = QVariant::fromValue(use);
                thread->trigger(ChoiceMade, player, decisionData);
            /*同样是锦囊，也可能被无懈可击，因此返回值还要调用 askForNullification，在这里我 new 了一个 SoloRunFake 卡牌，因为无懈可击找不到我们打出的那张单刀赴会，所以只能用一个只有名字的单刀赴会假牌来代替*/
                return !askForNullification(new SoloRunFake(Card::NoSuit,0,true), player, to,
true);
            }else if(continable)
                goto trust;
        }
        return false;
}
```

在 server\serverPlayer.cpp 中完成 hasSoloRun 函数：

```
bool ServerPlayer::hasSoloRun() const{
        foreach(const Card *card, handcards){
            if(card->objectName() == "solo_run")
                return true;
        }
        return false;
}
```

在 server\ai.cpp 中完成 askForSoloSoloRun 函数，如果不实现这个函数，到了可以响应单刀赴会的时机，并且 AI 手里有单刀赴会的话，程序会卡死在那里：

```
const Card *TrustAI::askForSoloRun(const DamageStruct damage){
        if(self == damage.to){
            QList<const Card *> cards = self->getHandcards();
            foreach(const Card *card, cards){
                if(card->objectName() == "solo_run")
                    return card;
            }
        }
        return NULL;
}
```

在 client\client.cpp 中添加回调函数 askForSoloRun（加在 askForNullification 下方即可），并且完成该函数：

```
    callbacks["askForSoloRun"] = &Client::askForSoloRun;

void Client::askForSoloRun(const QString &ask_str){
        QRegExp rx("(\\w+):(.+)->(.+)");
        QStringList texts = rx.capturedTexts();
        QString card_name = texts.at(1);
```

```
    const Card *damage_card = Sanguosha->findChild<const Card *>(card_name);
    if(damage_card == NULL)
        /*考虑到有些伤害不是卡牌造成的，比如强袭，如果不加这一句会导致程序崩溃，
因此我定义了一个“无卡牌”来表示造成此类伤害的卡牌*/
        damage_card = new NoCard(Card::NoSuit, 0, true);
    QString source_name = texts.at(2);
    ClientPlayer *source = NULL;
    if(source_name != ".")
        source = getPlayer(source_name);
    QString trick_path = damage_card->getPixmapPath();
    QString to = getPlayer(texts.at(3))->getGeneral()->getPixmapPath("big");
    if(source == NULL){
        prompt_doc->setHtml(QString("<img src='%1' /> ==&gt; <img src='%2'
/>").arg(trick_path).arg(to));
    }else{
        QString from = source->getGeneral()->getPixmapPath("big");
        prompt_doc->setHtml(QString("<img src='%1' /> <img src='%2'/> ==&gt; <img src='%3'
/>").arg(trick_path).arg(from).arg(to));
    }
    card_pattern = "solo_run";
    refusable = true;
    use_card = false;
    setStatus(Responsing);
}
```

在 package\standards.h 的最后加上我们用蓝色粗体加亮的那两种卡牌的定义：

```
class SoloRunFake:public TrickCard{
    Q_OBJECT
public:
    SoloRunFake(Suit suit, int number, bool aggressive):TrickCard(suit, number, aggressive){
        setObjectName("solo_run");
    }
    virtual QString getSubtype() const{
        return "single_target_trick";
    }
};
class NoCard:public Card{
    Q_OBJECT
public:
    NoCard(Suit suit, int number, bool aggressive):Card(suit, number, aggressive){
        setObjectName("no_card");
    }
    virtual QString getType() const{
        return "no_card";
    }
```

```
    virtual QString getSubtype() const{
        return "no_card";
    }
    virtual CardType getTypeId() const{
        return Basic;
    }
};
```

最后也是最重要的一条，我们要找到可是的地方来调用 askForSoloRun，也就是伤害即将造成的地方，在 server\gamerule 的 DamageDone 事件下加上红色部分就大功告成了：

```
case DamageDone:{
            DamageStruct damage = data.value<DamageStruct>();
            if(room->askForSoloRun(damage))
                damage.damage = 0;
            room->sendDamageLog(damage);
            room->applyDamage(player, damage);
            if(player->getHp() <= 0){
                room->enterDying(player, &damage);
            }
            break;
        }
```

下面我们来总结一下被动锦囊的实现：

1. 完成锦囊的类，这步可以完全仿照无懈可击
2. 在 room 类中完成 askForXXXX 函数
3. 在 serverPlayer 类中完成 hasXXXX 函数
4. 在 TrustAI 类中完成 askForXXXX 函数
5. 在 Client 类中添加回调函数 askForXXXX（在 askForNullification 下方），并完成 askForXXXX 函数
6. 有必要的卡牌子类需要自行添加
7. 在合适的相应时机调用 askForXXXX 函数