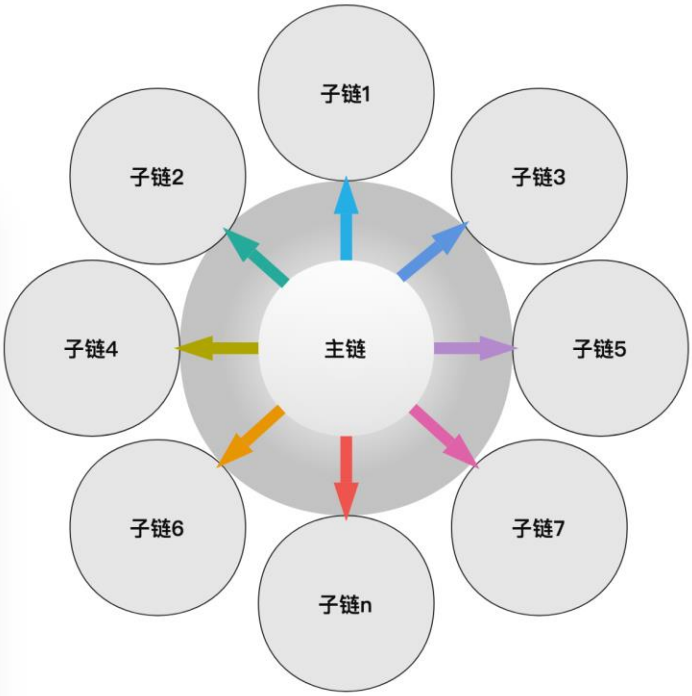
MANX 跨链技术

课题 1： **MANX** 主链和子链的结构设计

MANX 是一个分层子链架构，包含核心主链以及可配置生成子链，其中子链是受主链的协调约束的。主链是整个系统的核心基石，在主链中包含基础代币系统、核心共识机制以及用于配置和协调子链运行的跨链协议。在 MANX 的跨链协议中，包含两种场景：

1. 主链与子链之间的交互

   子链是通过 MANX 的生成模板配置定义出来的，可以拥有自己独立的共识以及存储机制。子链产生区块后，会按照一定间隔周期与主链通信，以使得主链可以获得子链的区块头以及其他统计信息，并能生成与子链相关的 Merkle 证明数据。MANX 主链为每一条生成的子链生成相应的 Merkle 证明数据。



主链与子链之间的交互

   如图所示，主链相当于是一个子链交互的 Hub，一个区块链结构的分布式 Hub。

2. 主链与外链之间的交互

对于更加广义的跨链通信，MANX 支持中继链的方式，通过中继链可以支持更加广泛且通用的跨链交互，可以实现与其他区块链系统交换资产数据以及部署跨链合约等。中继链在 MANX 中是以一条特殊的子链形式存在的。事实上，中继链是上述 Hub 形式的更一般形式，通过中继链也可以实现 MANX 主链与子链的数据交互，只不过，对于 Hub 形式在主链与子链的通信形式中效率更高。

课题 2： **MANX** 跨链功能应用

在 MANX 的跨链机制实现中，并不仅仅只是实现跨链资产交换，最终会实现如下的跨链功能应用：

1. 数据资产在多链之间的转移，包括 MANX 主链与侧链以及与外链之间
2. 多链之间的原子交换
3. 参与监听与验证某个链的事件信息
4. 建立多链之间的统一账户映射模型，便于管理多链资产

在整个跨链的设计中，MANX 主链将会保持功能的纯粹性，主要负责基本的 MANX 经济模型以及与侧链的协调，并为侧链的功能配置提供底层调用。也因此，MANX 主链会倾向于保持网络的分布式公正性。

课题 3： **MANX** 面向应用的分层侧链

每一条侧链可以以自己的业务逻辑为主，核心主链提供基础设施服务，理论上可以提供无限的性能扩展。侧链定义自己的业务逻辑，比如 MANX 提供的八种应用模板：网站，交易所，支付，保险，投顾，社交，游戏，电商。

各种应用的账本数据并不需要与核心主链存储在一起，而是拥有自己的独立存储，但是侧链的配置以及日常运行的区块头需要发送到核心主链，并且核心主链进行验证确认后存证。子链可以与主链全节点进行连接，也可以仅通过主链区块头进行连接。

侧链上运行的脚本等合约程序，其指令集必须来自于核心主链，也即是核心主链的子集，子链脚本程序可以通过协议通道的方式直接访问主链节点以及部署在主链上的合约。但是主链不一定能完全访问子链，侧链会拥有一个隔离模型，决定可以向主链以及外界开放的数据接口。

在 MANX 结构中，虽然具备分层链结构，但是每条链的运行是平等的，是异步解耦的。任何一条侧链的业务运行并不需要向主链进行频繁的查询调用或者验证，只需要每隔一段时间向自己的父链汇报区块头数据作为存证（父链节点会进行验证后存证）。

课题 4： **MANX** 与其它跨链技术的比较

项目名称	**MANX**	**Corda，Interledge**	**Polkdot，COSMOS**	**Atom Swap**	**WanChain Fusion，EKT**
原理	中继链+变量映射+监控/执行隔离策略+可控变量管理合约	公证人模式	中继器	哈希锁定	分布式私钥控制
公正性（去中心化程度）	最高	低	高	高	高
跨链资产转移	支持	支持	支持	仅支持跨链资产交换	支持
跨链智能合约	支持	不支持	支持	不支持	不支持
多链、多合约、多变量分布式业务	完全支持	不支持	部分支持	不支持	不支持

跨链机制比较

课题 5： **MANX** 分布式业务设计

分布式业务是指，事务的多个步骤分散在不同的区块链上执行，且保证整个事务的一致性。跨链资产交换是异构区块链多链融合的初级阶段，而分布式业务则是多链融合的高级阶段，分布式业务将资产交换的行为扩展成任意行为。跨链智能合约使分布式业务成为了可能，一个智能合约可以在多个不同的区块链上执行不同的部分，或者全部执行完毕，或者全部退回执行前的状态。这赋予了跨链协作和融合极大的想象力，将大幅扩张区块链的应用场景，打破目前阶段单链环境信息孤岛的壁垒，实现多链、多合约、多变量的协同适配和异构融合。

我们设置了 MANX 的分布式业务的实施方案，协调 k 个用户在 k 条公链之间发生分布式交易和业务。我们约定 k 条公链简写为$C_i, i = 1,2, \dots, k,$约定中继链简写为R。

分布式业务的参与对象分为三类：

1） 用户节点：即作为直接参与人，参与到分布式业务中的中继链和所涉及的各条公链状态机的更新；定义用户节点为$USER_i, i = 1,2, \dots, k$

2） 监控节点：监控分布式业务中所涉及的各条公链的数字资产和状态变量的更新的情况，并将更新情况打包成交易作为中继链智能合约的输入，驱动中继链合约状态机的执行和状态跳转。定义监控节点为$MON_i, i = 1,2, \dots$。

3） 执行节点：监控中继链智能合约状态机的执行结果，并将执行结果打包成交易作为所涉及各条公链合约的输入，驱动合约状态机的执行和状态跳转。定义执行节点为$EXE_i, i = 1,2, \dots$。

监控节点和执行节点的筛选：监控节点和执行节点筛选原则包括节点信用值、节点的保证金充足率和节点稳定性。其中节点信用值指节点之前作为监控节点或者执行节点的服务正确率，节点的保证金充足率是指节点是否有足够的 MANX 代币作为资产抵押物，节点稳定性是指节点在线服务的连续性。

账户创建要求：要求用户节点在分布式业务中有关联的各条公链拥有账户$USER_i(C_j), 0 \leq \mathrm{j} \leq \mathrm{k}$，在中继链中拥有账户$USER_i(R)$；同理，要求监控节点在中继链中拥有账户$MON_i(R)$，要求执行节点在中继链中拥有账户$EXE_i(R)$。

保证金机制：每个分布式业务均具有相应的价值度量，每次用户节点邀请监控节点和执行节点共同执行分布式业务时，需要在中继链智能合约中使监控节点和执行节点抵押一定的 MANX 代币作为保证金。同样地，用户节点需要在中继链智能合约中存入一定量的 MANX 代币作为合规执行分布式业务的保证金。

监控节点和执行节点的激励机制：每个分布式业务中，每个用户节点均需要在中继链合约中存入一定量的 MANX 代币作为监控节点和执行节点的预付服务费，实际服务费 =监控节点和执行节点的保证金 * 利率 * 合约执行周期 + 固定服务费，若实际服务费 > 预付服务费，则中继链合约根据监控节点的输入判定特定的用户节点的延迟过失，使其损失全部的预付服务费，其他用户的预付服务费原路退回。否则，该次业务执行完毕后，中继链合约按照实际服务费自动执行分配工作，并将剩余服务费（= 预付服务费 - 实际服务费）原路退回。

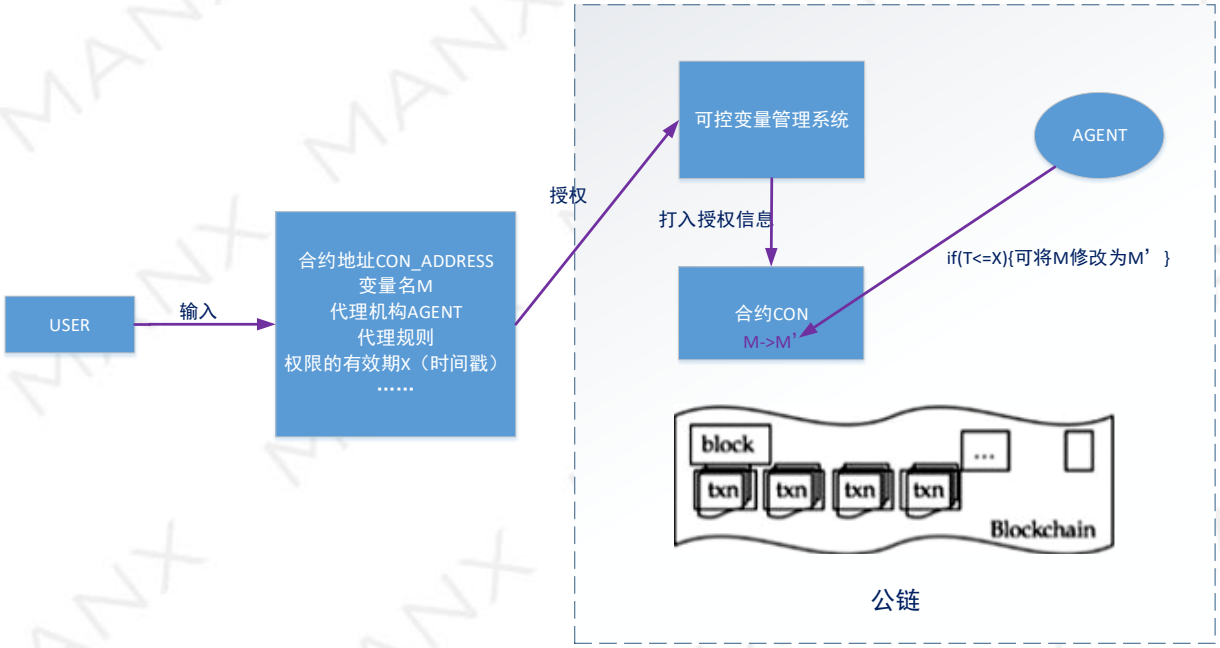
监控节点和执行节点的惩罚机制：每个分布式业务中，若因为某些监控节点或者执行节点未能及时执行相应的职责，则将按一定比例扣除其保证金，并作为其他合规节点的补偿。

中继链合约：中继链智能合约包括事件处理和保存的机制，以及一个完备的状态机，用于接受和处理各种智能合约，数据的状态处理在合约中完成。事件信息传入智能合约后，触发智能合约进行状态机判断。如果自动状态机中某个或某几个动作的触发条件满足，则由状态机根据预设信息选择合约动作的自动执行。

中继链合约状态机的原子性：中继链智能合约遵循原子性原则，状态变量或者按照逻辑全部更新至新的状态，或者全部退回至原始状态。

仲裁机制：MANXRelay 的仲裁机制通过内置 SPV 的机制来实现。一个外部的第三方，被称为 Relayer，发送一个交易到 MANXRelay 的智能合约，内容是最新的 k 条公链的区块头。MANXRelay 基于现存的区块头信息校验发送的区块头的有效性。如果校验通过，则加入到 MANXRelay 维护的 k 条公链的块头链。由此，在 MANXRelay 内建立了一个仲裁智能合约里，实现了一个内置的 SPV（简单支付校验）节点。当用户节点对监控节点发送的中继链合约输入的正确性有异议时，可以向 MANXRelay 的内部的仲裁智能合约，由该合约执行 SPV 验证，判定最终结果。若结果表明，监控节点作弊，则通过合约内置逻辑进行惩罚。

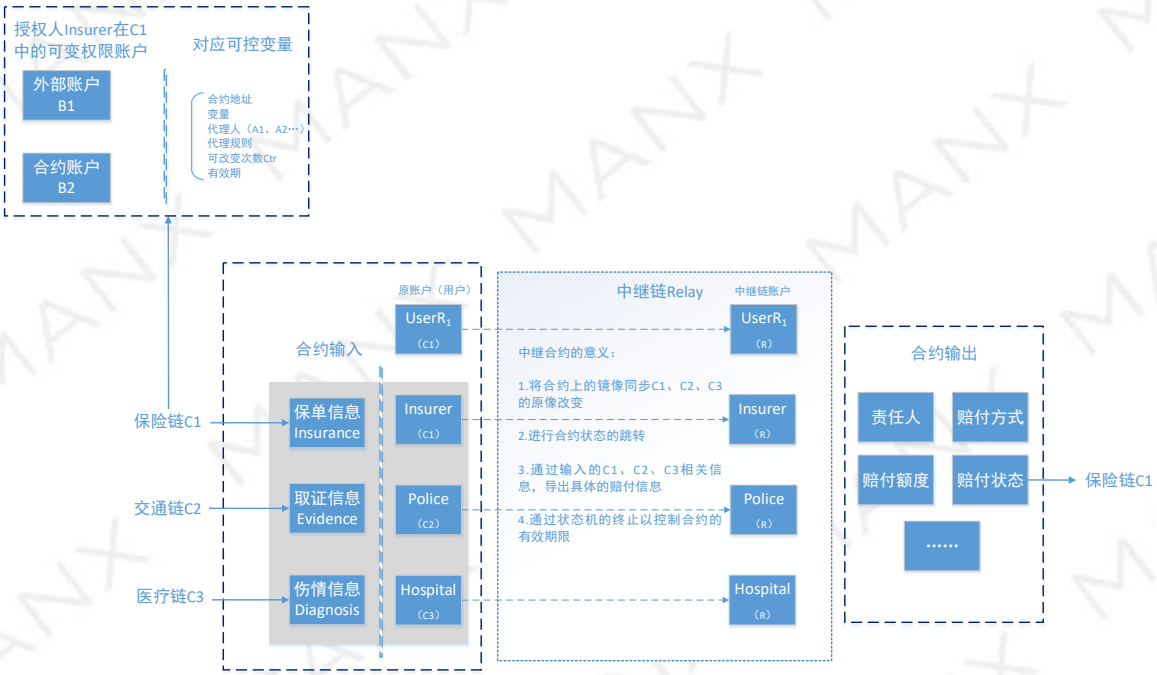
可控变量管理系统：任何用户 USER 都能在各个公链上建立这个管理系统，通过此系统，各用户可以将自己创建的合约中的变量或其它合约中自己拥有修改权限的变量的修改权限赋予其它代理节点，这个系统的使用权仅归 USER 所有。USER 每在公链 C 上创建一个合约 CON，CON 的地址便收录到可控变量管理系统中。用户节点将合约 CON 的地址 CON_ADDRESS、变量名称 M、代理节点 AGENT、代理规则和代理节点获得的修改权限的有效期 X 等等内容输入到可控变量管理系统中，随后执行授权。授权完成后，只要当前时间 T 在有效期 X 内（即$T \leq X$），则 AGENT 便可对 CON 中的 M 变量进行修改；若当前时间 T 超过有效期 X（即$T > X$），则 AGENT 失去对 M 的修改权。



可控变量管理系统

课题 6： **MANX** 分布式业务案例：

假设用户$USER_1$不幸碰上了一起交通事故，他之前已在保险公司购置有交通意外保险，其保单信息被保险公司 *INSURER* 存储在保险链 C1 上的保单合约 contract-policy 中，申请理赔的条件是有公安局 *POLICE* 出具的取证信息 *EVIDENCE* 和医院 *HOSPITAL* 出具的伤情诊断信息 *DIAGNOSIS*。而取证信息保存在交通链 C2 上，伤情诊断信息保存在医疗链 C3 上。如果$USER_1$想理赔成功，则须将 C1 上的保单信息、C2 上的取证信息和 C3 上的伤情诊断信息都放到一起才能得出理赔结果，最后还需将理赔结果更新至保险链 C1 才能完成理赔。这一事务由多方参与，是一项基于跨链智能合约开展的分布式业务。



保险、交通、医疗分布式业务整体框架图

详细流程如下：

1. 账户申请。$USER_1$已拥有 C1 账户$USER_1(C1)$，申请中继链账户$USER_1(R)$；公安局 POLICE 已有 C2 账户 POLICE$(C2)$，申请中继链账户 POLICE(R)；保险公司 INSURER 已有 C1 账户 $INSURER(C1)$，申请中继链账户$INSURER(R)$；医院 HOSPITAL 已有 C3 账户 $HOSPITAL(C3)$，申请中继链账户$HOSPITAL(R)$。

2. 中继链合约建立。$USER_1$，POLICE，$INSURER$，HOSPITAL 约定使用中继链 R 作为分布式业务运行环境，建立跨链智能合约，包含如下信息：

   2.1 分布式信息或状态一一映射关系，原有公链变量称为原像，中继链变量称为镜像，原像和镜像必须一致，原像作为输入到中继链的信息的有效性依据：

$$C_1.\text{contract}_{\text{policy}}.\text{insurer.policy} \leftrightarrow \text{R.this.insurer.policy}$$
$$C_2.\text{contract}_{\text{evidence}}.\text{police.evidence} \leftrightarrow \text{R.this.police.evidence}$$
$$C_3.\text{contract\_diagnosis.hospital.diagnosis} \leftrightarrow \text{R.this.hospital.diagnosis}$$

   2.2 设置用户节点$USER_1$，$POLICE$，HOSPITAL，INSURER 的保证金$\text{dep}(\text{user1})$，$\text{dep}(\text{police})$，$\text{dep}(\text{hospital})$，$\text{dep}(\text{insurer})$和预付服务费$\text{p}(\text{user1})$，$\text{p}(\text{police})$，$\text{p}(\text{hospital})$，$\text{p}(\text{insurer})$。

   2.3 监控节点和执行节点的匹配要求：节点信用值、节点的保证金充足率和节点稳定性。

   2.4 监控节点和执行节点的保证金要求：$\text{dep}(MON)$，$\text{dep}(EXE)$。

2.5 合约主逻辑：规定合约的输入和输出，状态机的跳转条件以及目标状态。输入为跳转条件所涉及的条件变量，输出为每个状态的结果变量。目标状态为跨链分布式业务中用户的最终诉求。

```
if (R.this.police.evidence == true &&R.this.insurer.policy == true &&R.this.hospital.diagnosis
        == true)
//若 evidence、policy 和 diagnosis 均已输入完成并且有效
  {R.this.insurer.compensate(责任人： null,    赔付方式： null,    赔付额度： null,
        赔付状态： null,    ……) → R.this.insurer.compensate(责任人： USER_1,
        赔付方式：支付 C1 代币 C1coin,赔付额度： 5 C1coin,赔付状态：未赔付,……)}
//得出理赔结果(目标变量 R.this.insurer.compensate)
else
  throw;
```

2.6 合约的激励和惩罚逻辑：根据合约状态机的状态和具体输入输出的执行情况，对每个用户节点、监控节点和执行节点进行相应的奖惩。

2.7 合约仲裁逻辑：规定在仲裁机制生效的情况下，参与多方的奖惩。

2.8 合约有效期限：若未能在有效期限内使中继链合约状态机跳转至目标状态，则保证金和资产原路返回，并且对于延误执行的节点进行扣除部分或者全部保证金的惩罚。

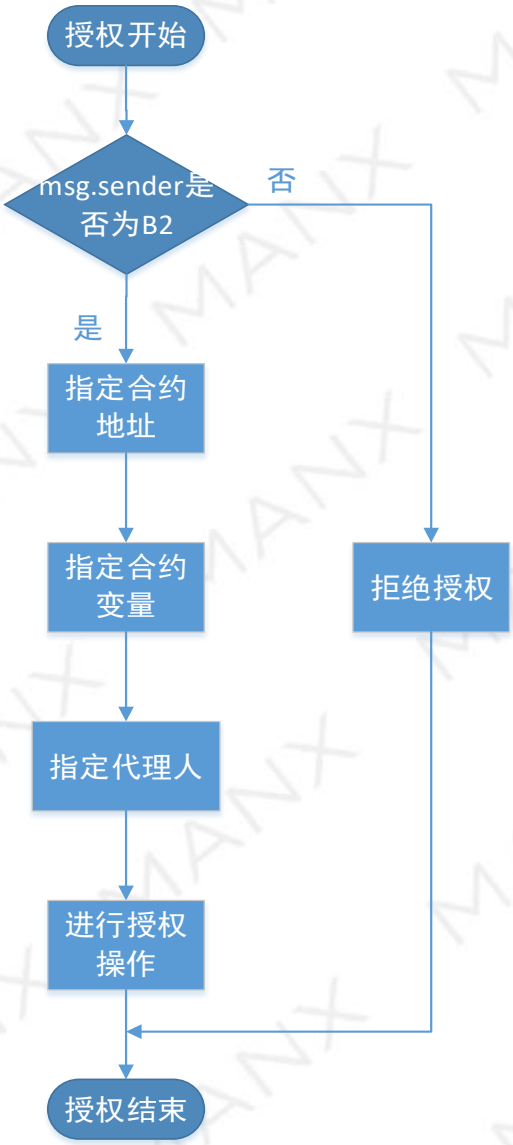
2.9 仲裁有效期限：在该有效期内，用户节点可以向仲裁合约提出仲裁请求。

3. 监控节点和执行节点加入：满足要求的监控节点和执行节点加入该合约，并且在输出涉及的公链中，执行节点共同申请基于（n,k）门限的执行账户，即对于 n 个执行节点来说，只需其中的 k 个执行节点同意，则可以将使用该账户发起交易。此处，输出相关变量是 R.this.insurer.compensate，则执行节点将在C_1中申请执行账户C_1.contract_policy.EXE_n(n ≥ 1)。

4. 授权输出代理：在 C1 中，INSURER 拥有两个可改变 contract_policy 的权限的账户，一个是外部账户 B1，一个是合约账户 B2。B1 是 INSURER 在 C1 上的账户，也可理解为公钥地址，B1 相当于 INSURER 在 C1 上的“身份证”；B2 是 INSURER 在 C1 上“管理变量修改权限的合约”的地址，简称为合约账户，这个合约的使用权仅归 INSURER 所有，B2 拥有改变 INSURER 在 C1 上所创建的合约中的变量的权限，而 INSURER 可以通过 B2 将这个权限赋予其它执行节点，权限的授予可细化到某个指定合约的指定变量。在整个理赔的过程中，INSURER 可以全程跟进，在中继链合约中得到目标变量$R.this.insurer.compensate$(责任人： $USER_1$, 赔付方式：支付$C1$ 代币$C1coin$,赔付额度： 5 $C1coin$,赔付状态：未赔付,……)后，亲自执行理

赔操作，将理赔金转给$USER_1$；INSURER 也可以通过 B2 委托一些执行节点 C1.contract_policy.EXE 执行理赔操作，大致操作如下：

```
//授权操作
if (msg.sender == B2) //若授权发起方是 B2，则可授权
{       addressOfcontract = contract_policy;  //指定合约地址
        variable=>[C1.contract_policy.insurer.compensate]   //指定合约变量
        Agent=>[C1.contract_policy.EXE1,....., C1.contract_policy.EXEn] //指定代理人
        Rule => {.....}  //代理规则                }
```



授权流程

5. 镜像处置。原像监控节点监控 C1、C2、C3 的原像(C1.contract_policy.insurer.policy、C2.contract_evidence.police.evidence、C3.contract_diagnosis.hospital.diagnosis)，一旦原像状态改变，则同步向中继链合约发送交易输入并要求处置镜像(R.this.police.evidence、R.this.insurer.policy、R.this.hospital.diagnosis)，若监控节点数达到一定比例，则合约按照原像的处置方法，处置镜像。这里，监控节点执行C1. contract_policy. insurer. policy → R. this. insurer. policy ， C2. contract_evidence. police. evidence → R. this. police. evidence ， C3. contract_diagnosis. hospital. diagnosis → R. this. hospital. diagnosis。

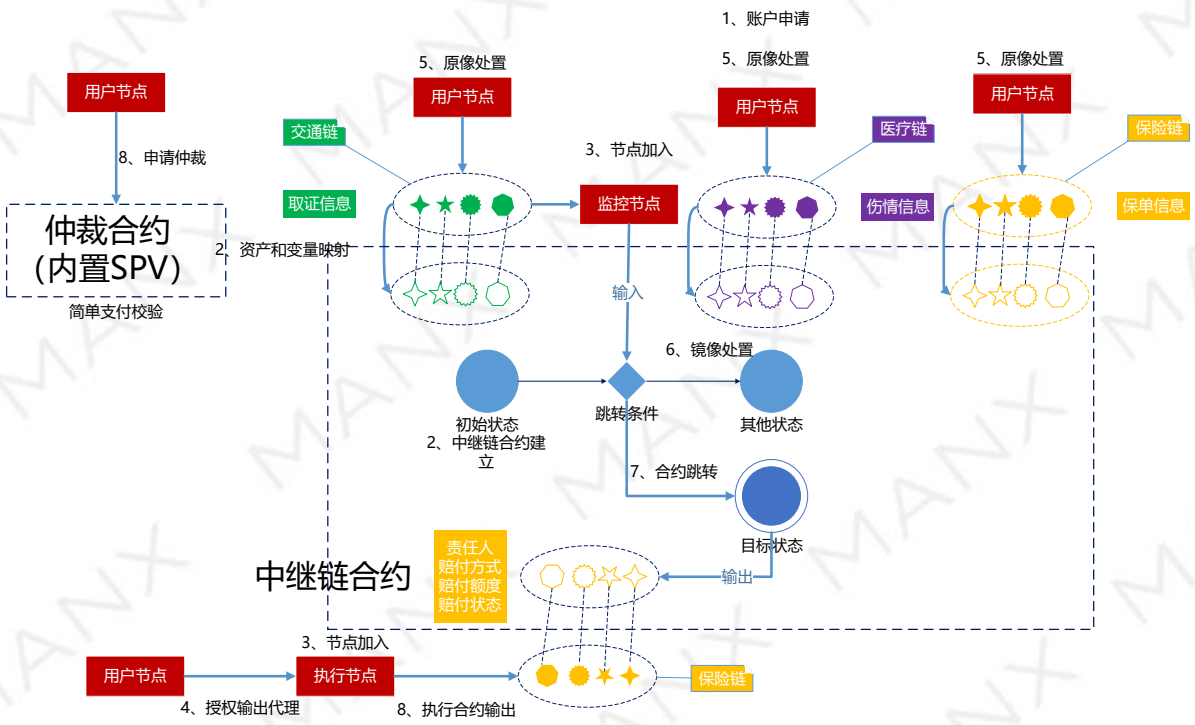
6. 中继链合约状态跳转。合约按照主逻辑执行状态机跳转，输出结果。这里，中继链状态跳转并获得目标变量 $R.this.insurer.compensate$(责任人：$USER_1$，赔付方式：支付$C1$ 代币$C1coin$,赔付额度：5 $C1coin$,赔付状态：未赔付,……)。

7. 原像处置：执行合约输出结果。用户节点或得到授权的执行节点监控中继链合约状态机的变化，在中继链合约得出目标变量后在原有公链对原像进行操作。若 INSURER 选择亲自对$USER_1$进行理赔，则 INSURER 在C_1执行{$C1$. contract_policy. $insurer.compensate$（责任人：$null$，赔付方式：$null$，赔付额度：$null$，赔付状态：$null$，……）→ $C1$. contract_policy. $insurer.compensate$（责任人：$USER_1$，赔付方式：支付$C1$ 代币$C1coin$，赔付额度：5 $C1coin$,赔付状态：未赔付,……)}，确认后 C1.contract_policy 自动执行$C1.insurer.5C1coin \rightarrow C1.user1.5C1coin$；若选择委托执行节点操作，例如委托了 n 个节点操作，则有 C_1. contract_policy. $EXE_m(0 < m \leq n)$ 执行{ $C1$. contract_policy. $insurer.compensate$（责任人：$null$，赔付方式：$null$，赔付额度：$null$，赔付状态：$null$，……）→ $C1$. contract_policy. $insurer.compensate$（责任人：$USER_1$，赔付方式：支付$C1$ 代币$C1coin$,赔付额度：5 $C1coin$,赔付状态：未赔付,……)}，只要$m \geq (n/2)$，则合约 C1.contract_policy 执行$C1.insurer.5C1coin \rightarrow C1.user1.5C1coin$。

8. 申请仲裁。在仲裁有效期内，用户节点若不同意监控节点向合约发送的任何输入，可以将包含相关信息的 SPV 证明提交至中继链上的仲裁合约，由中继链全体节点执行仲裁合约，并结合相应公链的区块头给出最终的仲裁结果。

9. 执行奖惩结果。经过仲裁有效期，合约面向用户节点、监控节点和执行节点进行交易费用、奖惩费用的结算，同时关闭中继链合约。

10. 合约逾期自动关闭。若超过合约有效期限，合约状态机并没有跳转至目标状态，则告知输入相关的用户节点原路退回至初始状态，并追究逾期责任和执行惩罚。


1、账户申请
5、原像处置
5、原像处置
5、原像处置
用户节点
用户节点
用户节点
用户节点
交通链
医疗链
保险链
8、申请仲裁
3、节点加入
取证信息
监控节点
伤情信息
保单信息
仲裁合约
（内置SPV）
2、资产和变量映射
输入
简单支付校验
6、镜像处置
初始状态
2、中继链合约建立
跳转条件
其他状态
7、合约跳转
目标状态
责任人
赔付方式
赔付额度
赔付状态
中继链合约
输出
3、节点加入
用户节点
执行节点
保险链
4、授权输出代理
8、执行合约输出

详细流程图