**比特币：一种点对点的电子现金系统**

中本聪

satoshin@gmx.com

www.bitcoin.org

**摘要。** 一个纯点对点版本的电子现金将允许在线支付从一方直接发送到另一方，而无需通过金融机构。数字签名是解决方案的一部分，但如果仍然需要一个受信任的第三方来防止双重消费，主要的好处就会失去。我们提出了一个使用点对点网络来解决双支出问题的解决方案。网络时间戳通过将事务哈希到一个正在进行的基于哈希的工作证明链中，形成一个不重新做工作证明就无法更改的记录。最长的链不仅可以证明所目睹的事件序列，而且还可以证明它来自最大的CPU功率池。只要大部分CPU功率由不合作攻击网络的节点控制，它们就会产生最长的链，并超过攻击者。网络本身需要最小的结构。消息是在最大努力的基础上广播的，节点可以随意离开并重新加入网络，接受最长的工作证明链作为它们离开时发生了什么的证明。

1. **项目简介**

互联网上的商业已经几乎完全依赖金融机构作为值得信赖的第三方来处理电子支付。虽然该系统对于大多数事务都工作得很好，但它仍然存在着基于信任的模型的固有弱点。完全不可逆的交易是不可能的，因为金融机构不能避免调解纠纷。调解成本增加了交易成本，限制了实际交易规模，切断了小型临时交易的可能性，丧失对不可逆服务支付不可逆能力的成本更大。随着逆转的可能性，对信任的需求就会蔓延。商人们必须小心他们的客户，骚扰他们以获取比他们所需要的更多的信息。一定比例的欺诈行为被认为是不可避免的。这些成本和支付不确定性可以通过使用物理货币亲自避免，但没有没有受信任的一方通过通信渠道支付的机制。

所需要的是一个基于加密证明而不是信任的电子支付系统，允许任何两个愿意交易的一方直接进行交易，而不需要受信任的第三方。计算上不实际逆转的交易将保护卖家免受欺诈，常规托管机制可以很容易地实现来保护买家。在本文中，我们提出了一个使用点对点分布式时间戳服务器来生成事务时间顺序的计算证明。只要诚实的节点比任何攻击者节点集体控制更多的CPU功率，系统就是安全的。

1

1. **交易记录**

我们将一个电子硬币定义为一个数字签名链。每个所有者通过数字签名前一个交易的哈希和下一个所有者的公钥，并将它们添加到硬币的末尾，将硬币转移到下一个所有者。收款人可以验证签名，以验证所有权链。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 交易记录 |  |  | 交易记录 |  |  | 交易记录 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 所有者1的 |  |  |  |  | 所有者2的 |  |  |  |  | 所有者3的 |  |  |
|  |  |  | 公钥 |  |  |  |  | 公钥 |  |  |  |  | 公钥 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | 散列 |  |  |  |  |  |  |  | 散列 |  |  |  |  |  | 散列 |  |  |
|  |  |  |  |  |  |  |  |  |  |  | ~~验证~~ |  |  |  |  |  |  |  |  |  |  |  | ~~验证~~ |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 所有者0的 |  |  |  |  | 所有者1的 |  |  |  |  | 所有者2的 |  |  |
|  |  |  | 签名 |  |  |  |  | 签名 |  |  |  |  | 签名 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | 签字 | 签字 |  |
|  |  |  | 所有者1的 |  |  |  |  | 所有者2的 |  |  |  |  | 所有者3的 |  |  |
|  |  |  | 私钥 |  |  |  |  | 私钥 |  |  |  |  | 私钥 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



当然，问题是收款人无法核实其中一个所有者没有重复硬币。一个常见的解决方案是引入一个受信任的中央权力机构，或铸币厂，以检查每笔交易的双倍支出。在每笔交易之后，硬币必须归还给铸币厂才能发行新硬币，只有直接从铸币厂发行的硬币才不会重复使用。这个解决方案的问题是，整个货币系统的命运取决于经营造币厂的公司，每一笔交易都必须通过它们，就像一家银行一样。

我们需要一种方式，让收款人知道，以前的所有者没有签署任何早期的交易。为了我们的目的，最早的交易是重要的，所以我们不关心以后的尝试。确认没有交易的唯一方法是了解所有交易。在以铸币厂为基础的模型中，造币厂知道所有的交易，并决定哪一个先到达。为了在没有受信任方的情况下完成这一点，交易必须公开宣布[1]，我们需要一个系统让参与者就接收顺序的单一历史达成一致。收款人需要证明，在每笔交易发生时，大多数节点都同意它是第一次收到的节点。

1. **时间戳记器服务器**

我们提出的解决方案从一个时间戳服务器开始。时间戳服务器的工作原理是获取一个项的哈希，并广泛发布哈希，例如在报纸或Usenet帖子[2-5]。时间戳证明，数据必须已经存在，显然，为了进入散列。每个时间戳在其哈希中包含以前的时间戳，形成一个链，每个附加的时间戳强化了前面的时间戳。



 散列和散列 



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 区块 |  |  |  |  |  |  |  | 区块 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 项目名称 |  | 项目名称 |  | ... |  |  |  | 项目名称 |  | 项目名称 |  | ... |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

2

1. **工作证明书**

为了在点对点的基础上实现一个分布式的时间戳服务器，我们将需要使用一个类似于亚当回来的Hashcash[6]的工作证明系统，而不是报纸或Usenet的帖子。工作证明涉及扫描一个值，当哈希时，例如使用SHA256时，哈希以一些零位开始。所需的平均工作以所需的零位数的指数，可以通过执行单个散列来验证。

对于我们的时间戳网络，我们通过在块中增加一次9次来实现证明，直到找到一个值，给块的哈希所需的零位。一旦花费了CPU精力以使其满足工作证明，则如果不重新执行工作，就无法更改块。由于后面的块在它之后，更改块的工作将包括重做它之后的所有块。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 区块 |  |  |  |  |  |  |  | 区块 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 前置式散列 |  | 一次 |  |  |  |  | 前置式散列 |  | 一次 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | Tx |  | Tx |  |  | ... |  |  |  |  | Tx |  | Tx |  |  | ... |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



工作证明也解决了在多数决策中确定表示的问题。如果多数人是基于一个IP地址-一票，它可能会被任何能够分配许多IP的人所颠覆。工作证明本质上是一张CPU的一票。多数决策由最长的链表示，它有最大的工作证明。如果大部分CPU功率由诚实的节点控制，诚实的链将增长最快，并超过任何竞争的链。要修改过去的工作块，攻击者必须重做工作块和之后所有工作块的工作证明，然后赶上并超越诚实节点的工作。稍后我们将展示，随着后续块的添加，一个较慢的攻击者追赶的概率呈指数级减少。

为了补偿不断增长的硬件速度和对运行节点的不同兴趣，工作证明难度是由平均每小时块数的移动平均值确定的。如果它们生成得太快，难度就会增加。

1. **“网络”**

运行该网络的步骤如下：

1. 新的事务将被广播到所有的节点上。
2. 每个节点都将新的事务收集到一个块中。
3. 每个节点都可以为其块找到一个困难的工作证明。
4. 当节点找到工作证明时，它会将该块广播到所有节点。
5. 只有在该块中的所有事务都有效且尚未使用时，节点才会接受该块。
6. 节点通过在链中创建下一个块，使用所接受的块的散列作为前一个散列来表示它们对块的接受程度。

节点总是认为最长的链是正确的，并将继续努力扩展它。如果两个节点同时广播下一个块的不同版本，则一些节点可能首先接收一个或另一个。在这种情况下，他们会处理他们收到的第一个分支，但会保存另一个分支，以防它变得更长。当找到下一个工作证明时，一个分支就会中断，一个分支就会变长；在另一个分支上工作的节点将切换到较长的节点。

3

新的事务广播并不一定需要到达所有的节点。只要它们到达许多节点，它们不久就会进入一个块。块广播也可以容忍被丢弃的消息。如果一个节点没有接收到一个块，那么当它收到下一个块并意识到它错过了一个块时，它就会请求它。

1. **激励措施**

按照惯例，一个块中的第一个交易是一个特殊的交易，它开始了一个由块的创建者拥有的新硬币。这增加了对节点支持网络的激励，并提供了一种最初分发硬币进入流通的方式，因为没有中央权力来发行它们。不断地增加大量的新硬币，类似于金矿工人消耗资源来增加黄金的流通。在我们的例子中，所消耗的是CPU的时间和电力。

该激励措施也可以提供交易费用。如果交易的输出值小于其输入值，则差额是添加到包含交易的块的激励值中的交易费用。一旦预先确定数量的硬币进入流通，激励措施就可以完全过渡到交易费用，并且完全没有通货膨胀。

这种激励措施可能有助于鼓励节点保持诚实。如果一个贪婪的攻击者能够收集比所有诚实的节点更多的CPU能力，他将必须选择使用它来欺骗人们，或者使用它来生成新的硬币。他应该发现，遵守规则，这些规则拥有比其他人更多的新硬币，比破坏制度和他自己财富的有效性更有利可图。

1. **正在回收磁盘空间**

一旦硬币中的最新交易被埋在足够的块下，可以丢弃之前的交易以节省磁盘空间。为了在不中断块的散列的情况下，事务被哈列在MerkreTree[7][2][5]中，只有根包含在块的哈列中。然后可以通过折断树枝来压实。不需要存储内部哈希值。



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 区块 | 块标题（块散列） | 区块 | 块标题（块散列） |  |
|  |  |  |
|  | 前置式散列 | 一次 |  | 前置式散列 | 一次 |  |
|  | 根哈列 |  |  | 根哈列 |  |  |
| 散希01 | 散希23 |  | 散希01 | 散希23 |  |
| 散希0 | 哈希1 | 哈希2 | 哈希3 |  |  | 哈希2 | 哈希3 |  |
| Tx0 | Tx1 | Tx2 | Tx3 |  |  |  | Tx3 |  |
| 隐藏在汞树中的事务处理 | 从街区修剪Tx0-2后 |  |

没有事务的块头将约为80个字节。如果我们假设每10分钟生成一次块，则每年生成80字节\*6\*24\*365=4.2MB。截至2008年，计算机系统通常销售2GB内存，而摩尔定律预测目前每年增长1.2GB，即使块头必须保存在内存中，存储也不应该是问题。

4

1. **简化的支付验证**

不运行完整的网络节点就可以验证支付。用户只需要保留最长工作证明链的块头的副本，他可以通过查询网络节点得到，直到他确信自己有最长的链，并获得将事务链接到时间戳的块的Merkle分支。他不能自己检查交易，但通过链接到链中的一个地方，他可以看到网络节点接受了它，并在进一步确认网络接受后添加块。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 最长的工作证明链 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 方块标头 |  |  |  | 方块标头 |  |  |  | 方块标头 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 前置式散列 |  |  | 一次 |  |  |  |  | 前置式散列 |  |  | 一次 |  |  |  |  | 前置式散列 |  |  | 一次 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | 墨根 |  |  |  |  |  |  | 墨根 |  |  |  |  |  |  | 墨根 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



散希01 散希23

Tx3的墨克尔分支

哈希2 哈希3

Tx3

因此，只要诚实的节点控制网络，验证就是可靠的，但如果网络被攻击者覆盖，验证就更容易被攻击。虽然网络节点可以自己验证事务，但只要攻击者能够继续覆盖网络，简化的方法就会被攻击者编造的事务所愚弄。防止这种情况的一种策略是，当网络节点检测到无效的块时，接受它们的警报，提示用户的软件下载完整的块，并提醒事务以确认不一致。频繁付款的企业可能仍然希望运行自己的节点，以获得更独立的安全和更快的验证。

1. **组合分值**

虽然可以单独处理硬币，但在转让中的每一分钱单独交易都是不拙的。为了允许分割和组合值，事务处理包含多个输入和输出。通常情况下，会有一个来自更大的以前交易的单一输入，或者有多个组合较小金额的输入，最多两个输出：一个用于支付，另一个返回更改，如果有的话，返回给发件人。



交易记录

在 输出

在 ...

...

应该注意的是，一个交易依赖于几个交易，而这些交易更依赖于许多交易，这在这里不是一个问题。从来不需要提取事务历史记录的完整独立副本。

5

**10.ᾏ隐私权**

传统的银行模式通过限制相关当事人和受信任的第三方访问信息，从而实现了一定程度的隐私性。公开宣布所有交易的必要性排除了这种方法，但隐私仍然可以通过打破另一个地方的信息流来维护：通过保持公钥匿名。公众可以看到，有人正在向其他人发送一个金额，但没有链接交易到任何人的信息。这类似于证券交易所发布的信息水平，其中个别交易的时间和规模，“磁带”，被公开，但没有告诉双方是谁。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 传统的隐私权模式 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | 受信任的人 |  |  |  |  |  |  |  |  |
|  | 身份信息 |  |  |  | 交易记录 |  |  |  |  |  |  |  |  | 交易对方 |  |  | 公共 |  |
|  |  |  |  |  |  |  |  |  | 第三方： |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 新的隐私权模式 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 身份信息 |  |  |  | 交易记录 |  |  |  |  |  | 公共 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



作为附加防火墙，每个事务都应该使用一个新的密钥对，以防止它们链接到共同所有者。一些链接仍然是不可避免的，这必然表明它们的输入属于同一所有者。风险是，如果显示了一个密钥的所有者，链接可能会显示属于同一所有者的其他事务。

**11.ᾏ计算结果**

我们考虑攻击者试图比诚实链更快地生成替代链的场景。即使完成了，它也不会让系统进行任意更改，比如用空创造价值，或者获取从来不属于攻击者的钱。节点不会接受无效的事务作为支付，而诚实的节点也永远不会接受包含它们的块。攻击者只能试图改变他自己的一笔交易来收回他最近花费的钱。

诚实链和攻击者链之间的竞争可以被描述为一个二项式随机游走。成功事件是将诚实链扩展一个区块，通过+1增加其领先优势，失败事件是将攻击者的链扩展一个区块，缩小差距-1。

攻击者从给定的赤字中赶上的概率类似于赌徒的毁灭问题。假设一个信用无限的赌徒开始于赤字，并可能进行无限次的尝试，以试图达到盈亏平衡。我们可以计算他达到盈亏平衡的概率，或者攻击者赶上诚实链，如下[8]：

一个诚实的节点找到下一个块

攻击者找到下一个块的概率

qz=攻击者会赶上后面的概率

*qz*={*q* /1*pz* *如果：如果： pp*≤q*q*}

6

给定我们假设p>q，概率随着攻击者必须赶上的块数量呈指数下降。随着他的赔率的增加，如果他不能早早地幸运地向前冲，他的机会就会变得越来越小，因为他会变得越来越落后。

我们现在考虑一个新事务的收件人需要等待多长时间，才能充分确定发件人不能更改事务。我们假设发件人是一个攻击者，他想让收件人相信他支付了他一段时间，然后在一段时间后把它还给自己。当这种情况发生时，接收方会收到警告，但发送方希望现在会太晚。

接收方生成一个新的密钥对，并在签名前不久将公密钥给发送者。这可以防止发送方提前准备一串块链，直到他足够幸运地提前走得足够远，然后在此时执行事务。一旦事务被发送，不诚实的发送人开始秘密地在包含一个他的事务版本的并行链上工作。

收件人等待事务添加到块并且在块之后链接z块。他不知道攻击者所取得的确切进展量，但假设诚实的块占用了每个块的平均预期时间，攻击者的潜在进展将是具有预期值的泊松分布：

=z*qp*



为了获得攻击者现在仍然能赶上的概率，我们将他可能取得的每个进展量的泊松密度乘以他从那时起就能赶上的概率：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ∞ | *k* | *e* | −  |  |  | *z−k* |  | } |  |
| ∑ |  |  |  |  | ⋅ | *q/p* | *如果是k≤z* |  |
|  | *k！* |  |  |
| *k=0* |  |  | { | 1 | *如果取值为kz* |  |

重新排列，以避免对分布的无限尾部求加。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *z* | *k* | −  |  |  |
| 1−∑ |  | *e* | 1−q/p*z−* *k* |  |
| *k！* |  |
| *k=0* |  |  |

正在转换为C代码。。

#包括<math.h>

双攻击者成功概率（双q，intz）

{

双p=1.0-q；

双层板=z\*（q/p）；

双和=1.0；

国际、k；

对于(k=0；k<=z；k++)

{

双毒松=exp（=）；

对于(i=1；i<=k；i++)

泊松\*=lambda/i；

sum-=泊松\*（1波（q/p，z-k））；

}

回和

}

7

在运行一些结果时，我们可以看到概率随z呈指数下降。

q=0.1

z=0 P=1.0000000

z=1 P=0.2045873

z=2 P=0.0509779

z=3 P=0.0131722

z=4 P=0.0034552

z=5 P=0.0009137

z=6 P=0.0002428

z=7 P=0.0000647

z=8 P=0.0000173

z=9 P=0.0000046

z=10 P=0.0000012

q=0.3

z=0 P=1.0000000

z=5 P=0.1773523

z=10 P=0.0416605

z=15 P=0.0101008

z=20 P=0.0024804

z=25 P=0.0006132

z=30 P=0.0001522

z=35 P=0.0000379

z=40 P=0.0000095

z=45 P=0.0000024

z=50 P=0.0000006

解决P小于0.1%。

P<0.001

q=0.10 z=5

q=0.15 z=8

q=0.20 z=11

q=0.25 z=15

q=0.30 z=24

q=0.35 z=41

q=0.40 z=89

q=0.45 z=340

**12.ᾏ试验结论**

我们提出了一种不依赖于信任的电子交易系统。我们从通常由数字签名制成的硬币框架开始，它提供了强大的所有权控制，但没有办法防止双重等待是不完整的。为了解决这个问题，我们提出了一个使用工作证明来记录事务的公共历史，如果诚实的节点控制了大多数CPU功率，攻击者就会变得不现实。该网络的非结构化简单性非常健壮。节点同时工作，几乎没有协调。它们不需要被识别，因为消息不被路由到任何特定的地方，只需要尽最大努力进行传递。节点可以随意离开并重新加入网络，接受工作证明链作为他们离开时发生的事情的证明。他们用CPU能力投票，通过扩展有效块来表达他们和拒绝使用有效块来表达他们对有效块的接受。任何必要的规则和激励机制都可以通过这种共识机制来执行。

8

**参考文献**

1. W.Dai，b-muny”，http://www.weidai.com/bmoney.txt，1998。
2. H.Massias，X.S.阿维拉，和J.-J。羽毛，“具有最小信任要求的安全时间戳服务的设计”，第20届信息论研讨会，1999年5月。
3. S.Haber，W.S.Stornetta，“如何给数字文档进行时间戳”，在密码学杂志，第3卷，第2期，第99-111页，1991年。
4. D. 拜耳，S.哈伯尔，《提高数字时间戳的效率和可靠性》，《序列二：通信、安全和计算机科学的方法》，第329-334页，1993页。
5. S.Haber，W.S.Stornetta，“比特字符串的安全名称”，在第四届ACM计算机和通信安全会议论文集，第28-35页，1997年4月。
6. A.ᾏ回来了，“哈什卡什——一种拒绝服务的反击措施”，http://www.hashcash.org/papers/hashcash.pdf，2002年。
7. R.C.Merkle，“公钥密码系统的协议”，在Proc中。1980年安全与隐私研讨会，IEEE计算机学会，第122-133页，1980年4月。
8. W.Feller，《概率论及其应用的介绍》，1957年。

9