(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 119856170 A
(43)申请公布日 2025.04.18

(21)申请号 202380027955.3

(22)申请日 2023.03.14

(30)优先权数据
FR2202221 2022.03.14 FR

(85)PCT国际申请进入国家阶段日
2024.09.14

(86)PCT国际申请的申请数据
PCT/EP2023/056430 2023.03.14

(87)PCT国际申请的公布数据
WO2023/174914 FR 2023.09.21

(71)申请人 宝格丽钟表股份有限公司
地址 瑞士

(72)发明人 恩尼奥·皮奇里洛
达维德·巴尔迪

(74)专利代理机构 北京银龙知识产权代理有限公司 11243
专利代理师 金成哲 张会娟

(51)Int.Cl.
G06F 21/10(2013.01)
H04L 9/00(2022.01)
G06Q 20/12(2012.01)
H04L 9/32(2006.01)

权利要求书2页 说明书24页 附图1页

(54)发明名称

用于管理与对象相关联的权限的方法

(57)摘要

本发明涉及一种用于管理与对象(1),例如钟表(1)或珠宝物品相关联的权限的方法,该方法包括以下步骤:-创建第一数字通证,该第一数字通证与对对象的第一权限、特别是所有权相关联,第一数字通证使用第一区块链协议进行存储和验证;-创建第二数字通证、特别是非同质化数字通证,该第二数字通证与第二权限、特别是访问与对象相关联的数字内容、特别是多媒体内容的权限相关联,第二数字通证使用第二区块链协议进行存储和验证;-创建将上述权限绑定到第一数字通证和第二数字通证的智能合约,使得对第一数字通证和第二数字通证中的一个的所有权在没有对第一数字通证和第二数字通证中的另一个的所有权的情况下不能被转移。

CN 119856170 A

1.一种与对象(1)相关联的权限的管理方法,所述对象(1)例如为钟表(1)或珠宝物品,所述方法包括以下步骤:

-创建第一数字通证,所述第一数字通证与对所述对象的第一权限、特别是所有权相关联,所述第一数字通证借助于第一区块链协议进行存储和认证,

-创建第二数字通证、特别是非同质化数字通证,所述第二数字通证与第二权限、特别是对与所述对象相关联的数字内容、特别是多媒体内容的访问权限相关联,所述第二数字通证借助于第二区块链协议进行存储和认证,

-创建链接所述第一数字通证上的权限和所述第二数字通证上的权限的智能合约,使得所述第一数字通证和所述第二数字通证中的一个上的所有权在没有所述第一数字通证和所述第二数字通证中的另一个上的所有权的情况下不能被转移。

2.根据权限要求1所述的管理方法,其中所述第一区块链协议和所述第二区块链协议是相同的类型,或者所述第一区块链协议和所述第二区块链协议是相同的区块链协议。

3.根据权限要求1或2所述的管理方法,其中所述智能合约涉及借助于区块链协议创建第三数字通证,所述第三数字通证包括的信息包括:

-包含在所述第一数字通证中的至少一条信息,特别是例如关于所述对象的信息和/或所述对象的序列号和/或创建所述第一数字通证的交易的标识符和/或真实性证书的激活日期和/或产品的销售日期和/或所述第一数字通证的标识符,以及

-包含在所述第二数字通证中的至少一条信息,特别是例如所述第二数字通证的标识符。

4.根据前述权限要求中任一项所述的管理方法,其中所述智能合约被配置为使得:

-转移所述第一数字通证上的所有权涉及转移所述第二数字通证上的所有权和/或转移所述第三数字通证上的所有权,和/或

-转移所述第二数字通证上的所有权涉及转移所述第一数字通证上的所有权和/或转移所述第三数字通证上的所有权,和/或

-转移所述第三数字通证上的所有权涉及转移所述第一数字通证上的所有权和/或转移所述第二数字通证上的所有权。

5.根据前述权限要求中任一项所述的管理方法,其中所述第一数字通证和/或所述第二数字通证与数字钱包相关联,特别是与同一数字钱包相关联。

6.根据权限要求5所述的管理方法,其中所述数字钱包或所述数字钱包之一与以下相关联:

-公钥,或

-与所述公钥相关联的代码(4),

所述公钥或代码被写在:

-所述对象上、特别是钟表(1)的部件(2)上、特别是钟表(1)的发条盒(2)上或发条盒(2)的棘轮(3)上,或

-与所述对象一起出售的有形介质上,或

-数字文档中,例如计算机文件,或消息中,特别是MMS或SMS。

7.根据权限要求6所述的管理方法,其中所述公钥或与所述公钥相关联的所述代码通过印刷或通过雕刻或通过粘接标签来写入。

8.根据权限要求6或7所述的管理方法,其中与所述公钥相关联的所述代码是机器可读代码,例如NFC标签或条形码,特别是例如矩阵码或QR码。

9.根据前述权限要求中任一项所述的管理方法,其中所述数字钱包或所述数字钱包之一与以下相关联:

-私钥,或

-与所述私钥相关联的代码,

所述私钥或与所述私钥相关联的代码被写在:

-附加到所述对象的有形介质上,特别是卡片上、标签上或物理文档上,和/或

-数字文档中,例如计算机文件,或消息中,特别是MMS或SMS。

10.根据权限要求9所述的管理方法,其中与所述私钥相关联的所述代码是机器可读代码,例如条形码,特别是例如NFC标签或例如矩阵码或QR码。

11.一种对象(1)或附加到所述对象的有形介质和/或附件,特别是钟表、特别是钟表(1)的发条盒(2)或发条盒(2)的棘轮(3)作为用于数字钱包的公钥或与公钥相关联的代码的介质的用途。

12.一种允许管理与对象(1)相关联的权限的设备,特别是分布式计算架构,所述设备包括实现根据权限要求1至10中任一项所述的方法的硬件和/或软件元件,特别是被设计为实现根据权限要求1至10中任一项所述的方法的硬件和/或软件元件。

13.一种计算机程序产品,其包括记录在计算机可读介质上的程序代码指令,所述程序代码指令用于当在计算机上运行所述程序时实现根据权限要求1至10中任一项所述的方法的步骤。

14.一种计算机可读数据记录介质,其上记录有计算机程序,所述计算机程序包括用于实现根据权限要求1至10中任一项所述的方法的程序代码指令。

用于管理与对象相关联的权限的方法

技术领域

[0001] 本发明涉及一种用于管理与对象相关联的权限的方法。本发明还涉及这样的对象在用于管理附加到该对象上的权限的方法中的用途。本发明还涉及一种用于管理与这样的对象相关联的权限的设备。本发明还涉及一种用于管理与这样的对象相关联的权限的程序产品。本发明还涉及一种专用于实现用于管理与这样的对象相关联的权限的方法的数据记录介质。最后,本发明涉及一种来自携带这种计算机程序产品的数据介质的信号。

背景技术

[0002] 在某些领域,特别是奢侈品领域,需要保存已经制造和销售的商品的登记簿。在商品的整个存在期间记录权限链也可能有意义。然而,一旦产品被投入市场,就不能保证这样的记录。只有当产品的不同所有者使其自己为登记簿的所有者所知并且更具体地为保存该登记簿的制造商所知时,才能保存登记簿。

[0003] 本发明的提出

[0004] 本发明的目的是提供一种用于管理与对象相关联的权限的方法,该方法克服了上述缺点并且改进了已知的现有技术的管理方法。特别地,本发明提供了一种用于管理与对象相关联的权限的方法,该方法允许保证可追溯性和真实性,同时还允许管理与商品相关联的权限和/或附加到对象的服务。

发明内容

[0005] 根据本发明,一种方法允许管理与对象(例如钟表或珠宝物品)相关联的权限。该方法包括以下步骤:

[0006] -创建第一数字通证,该第一数字通证与对对象的第一权限、特别是所有权相关联,第一数字通证借助于第一区块链协议进行存储和认证,

[0007] -创建第二数字通证、特别是非同质化数字通证,该第二数字通证与第二权限、特别是对与对象相关联的数字内容、特别是多媒体内容的访问权限相关联,第二数字通证借助于第二区块链协议进行存储和认证,

[0008] -创建链接第一数字通证上的权限和第二数字通证上的权限的智能合约,使得第一数字通证和第二数字通证中的一个上的所有权在没有第一数字通证和第二数字通证中的另一个上的所有权的情况下不能被转移。

[0009] 第一区块链协议和第二区块链协议有利地是相同的类型,或者第一区块链协议和第二区块链协议有利地是相同的区块链协议。

[0010] 智能合约可以涉及借助于区块链协议创建第三数字通证,该第三数字通证包括的信息包括:

[0011] -包含在第一数字通证中的至少一条信息,特别是例如关于对象的信息和/或对象的序列号和/或创建第一数字通证的交易的标识符和/或真实性证书的激活日期和/或产品的销售日期和/或第一数字通证的标识符,以及

[0012] -包含在第二数字通证中的至少一条信息,特别是例如第二数字通证的标识符。

[0013] 优选地,智能合约被配置为使得转移第一数字通证上的所有权涉及转移第二数字通证上的所有权和/或转移第三数字通证上的所有权,和/或

[0014] 转移第二数字通证上的所有权涉及转移第一数字通证上的所有权和/或转移第三数字通证上的所有权,和/或

[0015] 在第三数字通证上转移所有权涉及在第一数字通证上转移所有权和/或在第二数字通证上转移所有权。

[0016] 第一数字通证和/或第二数字通证可以与数字钱包相关联,特别是与同一数字钱包相关联。

[0017] 数字钱包或数字钱包之一可以与以下相关联:

[0018] -公钥,或

[0019] -与公钥相关联的代码,

[0020] 公钥或代码被写在:

[0021] -对象上、特别是钟表的部件上、特别是钟表的发条盒上或发条盒的棘轮上,或

[0022] -与对象一起出售的有形介质上,或

[0023] -数字文档中,例如计算机文件,或消息中,特别是MMS(Multimedia Messaging Service(多媒体消息服务)的首字母缩写)或SMS(Short Messaging Service(短消息服务)的首字母缩写)。

[0024] 公钥或与公钥相关联的代码可以通过印刷或通过雕刻或通过粘接标签来写入。

[0025] 与公钥相关联的代码可以是机器可读代码,例如NFC标签或条形码,特别是例如矩阵码或QR码。

[0026] 数字钱包或数字钱包之一可以与以下相关联:

[0027] -私钥,或

[0028] -与私钥相关联的代码,

[0029] 私钥或与私钥相关联的代码被写在:

[0030] -附加到对象的有形介质上,特别是卡片上、标签上或物理文档上,和/或

[0031] -数字文档中,例如计算机文件,或消息中,特别是MMS或SMS。

[0032] 与私钥相关联的代码可以是机器可读代码,例如NFC标签或条形码,特别是例如矩阵码或QR码。

[0033] 本发明还涉及对象或附加到对象的有形介质或附件,特别是钟表、特别是钟表的发条盒或发条盒的棘轮作为用于数字钱包的公钥或与公钥相关联的代码的介质的用途。

[0034] 根据本发明,一种允许管理与对象相关联的权限的设备,特别是分布式计算架构,该设备包括实现上面限定的方法的硬件和/或软件元件,特别是被设计为实现上面限定的方法的硬件和/或软件元件。

[0035] 根据本发明,一种设备包括用于实现上面限定的方法的步骤的装置。

[0036] 根据本发明,一种计算机程序产品包括记录在计算机可读介质上的程序代码指令,该程序代码指令用于当在计算机上运行所述程序时实现上面限定的方法的步骤。

[0037] 本发明还涉及一种可从通信网络下载和/或被记录在可由计算机读取和/或可由计算机执行的数据介质上的计算机程序产品。该程序产品包括指令,当该程序由计算机执

行时,该指令引导计算机实现上面限定的方法的步骤。

[0038] 本发明还涉及一种计算机可读数据记录介质,其上记录有计算机程序,该计算机程序包括用于实现上面限定的方法的程序代码指令。

[0039] 本发明还涉及一种计算机可读记录介质,其包括指令,所述指令在由计算机执行时使得计算机实现上面限定的方法的步骤。

[0040] 本发明还涉及一种来自携带上面限定的计算机程序产品的数据介质的信号。

附图说明

[0041] 在以下参照附图非限制性地给出的一个特定实施方式的描述中详细描述了本发明的主题、特征和优点。

[0042] 图1是钟表的视图,对钟表的权限按照根据本发明的方法的一种执行模式来管理。

具体实施方式

[0043] 在应用于为钟表1的对象1的上下文中解释了下面描述的管理方法的执行模式。然而,该方法可以应用于任何类型的物理或无形的对象、产品或服务。它还可以特别地应用于珠宝物品。

[0044] 对象是根据该方法管理其权限的主要元素。对象可以与附加元素或辅助元素一起销售,该附加元素或辅助元素也可以是任何类型,例如物理或无形的对象、产品或服务。

[0045] 在所描述的实施方式中,虽然主要元素是钟表,但是辅助元素包括允许访问数据的服务,特别是可以包含图像、音频内容和视频的多媒体数据。例如,这些数据可以用图表说明将对象投入市场的创意项目的历史。例如,这些数据可以包含与参与项目的那些人的面谈,例如与艺术总监的面谈或与直接向钟表所有者提议的技术总监的面谈。

[0046] 辅助元素还可以包含与钟表一起出售的数字或物理艺术作品,该数字或物理艺术作品例如是由与制造商合作的艺术家,特别是在创建钟表的过程中与制造商合作的艺术家制作的。辅助元素还可以是附件对象,例如钟表的显示壳。

[0047] 一旦制造商生产了主要元素(例如,由钟表组成的对象)和辅助元素(例如,允许访问数据的服务),就实现了作为本发明主题的方法的执行模式。

[0048] 该方法允许管理对主要元素的权限。特别地,可以管理对主要元素的所有权。该所有权可以被划分为裸所有权(bare-ownership)和用益权。可以设想与主要元素相关联的任何其他权限的管理。

[0049] 该方法还允许管理对辅助元素的权限。特别地,可以管理对辅助元件的所有权。该所有权可以被划分为裸所有权和用益权。可以设想与辅助元素相关联的任何其他权限的管理。对辅助元素的这些权限可以被认为是与主要元素相关联的权限,辅助元素与主要元素相关联。

[0050] 辅助元素和主要元素之间的关联可以是任何性质的,特别是功能性的、商业性的、概念性的或艺术性的。该关联还可以仅是权限管理方法实现的结果。

[0051] 在管理方法的第一步骤中,创建或创造与对钟表的第一权限(特别是所有权)相关联的第一数字通证。例如,第一数字通证是非同质化通证(NFT)。第一数字通证借助于第一区块链协议进行存储和认证。第一协议例如是区块链协议AURA。

[0052] 在管理方法的第二步骤中,创建或创造第二数字通证,该第二数字通证与第二权限、特别是对与钟表相关联的数字内容、特别是多媒体内容的访问权限相关联。如上所述,第二权限可以是对辅助元件的权限。例如,第二数字通证是非同质化通证。第二数字通证借助于第二区块链协议进行存储和认证。第二协议例如是区块链协议POLYGON。

[0053] 例如,第二权限可以是对托管在属于制造商的互联网站点上的数字内容的访问权限。因此,第二权限可以是网站的用益权,而网站的裸所有权由制造商保留。

[0054] 第一区块链协议和第二区块链协议可以是相同的类型,或者实际上第一区块链协议和第二区块链协议可以是相同的区块链协议。

[0055] 第二步骤可以以各种方式来实现。

[0056] 例如,在第一执行模式中,制造商一完成钟表就触发创造第二数字通证的操作,或者当制造商的钟表被出售给第一客户或第一所有者时触发创造第二数字通证的操作。

[0057] 例如,在第二执行模式中,是第一客户或第一所有者在其已经拥有钟表之后触发第二数字通证的创造。

[0058] 为了实现第二执行模式:

[0059] -公钥,或

[0060] -与公钥相关联的代码,

[0061] 被写在钟表1上,特别是写在钟表的部件2上,特别是写在钟表1的发条盒2上或发条盒2的棘轮3上。

[0062] 替代地,可以给钟表提供有形介质,例如文档、标签、证书、显示壳或在其上写入公钥或与公钥相关联的代码的任何附件。替代地,也可以给钟表提供包含公钥或与公钥相关联的代码的数字文档,例如计算机文件,或消息,特别是MMS或SMS。该数字文档可以在购买钟表时移交或发送。

[0063] 此外,在钟表的销售期间,

[0064] -私钥,或

[0065] -与私钥相关联的代码,

[0066] 可以被提供有形介质,例如文档、标签、证书、显示壳或其上写入私钥或与私钥相关联的代码的任何附件。替代地,可以给钟表提供包含私钥或与私钥相关联的代码的数字文档,例如计算机文件,或消息,特别是MMS或SMS。该数字文档可以在购买钟表时移交或发送。优选地,私钥或与私钥相关联的代码的递送是安全的。例如,当私钥或与私钥相关联的代码被附加到文档或更一般地附加到任何物理介质时,它被覆盖有对其进行掩蔽的涂层,使得只有第一所有者在移除涂层之后才能够看到它。

[0067] 密钥中的任一个或两个和/或与密钥相关联的代码中的任一个或两个可以通过印刷或通过雕刻或粘接标签来写入。

[0068] 优选地,与密钥相关联的代码中的任一个或两个是可经由射频读取的代码或由机器可光学读取的代码,例如NFC标签(NFC代表近场通信)或条形码,特别是例如矩阵码或QR码的条形码。

[0069] 因此,在购买钟表之后,钟表的第一所有者可以通过访问公钥和私钥来触发第二数字通证的创造。为此,她或他可以导航到专用网站并输入公钥和私钥以开始创造第二数字通证。

[0070] 在创建时,第一数字通证和第二数字通证与一个或两个数字钱包相关联。它们一创建就与第一所有者的数字钱包相关联,或者在销售钟表时从制造商的数字钱包转移到第一所有者的数字钱包。第一数字通证和第二数字通证可以与不同的数字钱包相关联。有利地,第一数字通证和第二数字通证可以与相同的数字钱包相关联。

[0071] 上述公钥和私钥有利地是第一所有者的数字钱包的公钥和私钥,它们在将制造商的钟表销售给第一所有者之后包含第一数字通证和第二数字通证。

[0072] 有利地,在管理方法的第三步骤中,创建链接第一数字通证上的权限和第二数字通证上的权限的智能合约,使得第一数字通证和第二数字通证中的一个上的所有权在没有第一数字通证和第二数字通证中的另一个上的所有权的情况下不能被转移。换句话说,第一数字通证和第二数字通证不能没有彼此地传递。同样优选地,假设第一数字通证和第二数字通证最初与第一数字钱包相关联,则它们仅可以一起转移以在转移之后与第二数字钱包相关联。

[0073] 优选地,智能合约涉及借助于第三区块链协议创建或创造第三数字通证。第三区块链协议优选地与第一区块链协议或第二区块链协议相同。

[0074] 第三数字通证有利地包含如下信息:

[0075] -包含在第一数字通证中的至少一条信息,特别是例如关于钟表的信息和/或钟表的序列号和/或创建第一数字通证的交易的标识符和/或真实性证书的激活日期和/或钟表的销售日期和/或第一数字通证的标识符,以及

[0076] -包含在第二数字通证中的至少一条信息,特别是例如第二数字通证的标识符。

[0077] 替代地,可以同时或几乎同时创建第一数字通证和第二数字通证(以及可选的第三数字通证)。在这种情况下,第一数字通证可以被创建为包括与第二数字通证相关的数据,并且第二数字通证可以被创建为包括与第一数字通证相关的数据。因此,第一通证和第二通证可以在创建时链接。

[0078] 优选地,智能合约被配置为使得:

[0079] -转移第一数字通证上的所有权涉及转移第二数字通证上的所有权和/或转移第三数字通证上的所有权,和/或

[0080] -转移第二数字通证上的所有权涉及转移第一数字通证上的所有权和/或转移第三数字通证上的所有权,和/或

[0081] -转移第三数字通证上的所有权涉及转移第一数字通证上的所有权和/或转移第二数字通证上的所有权。

[0082] 实际上,当触发一个数字通证从一个数字钱包到另一个数字钱包的转移时,自动触发另一个数字通证或其他数字通证的转移。有利地,在允许这种转移的软件应用程序中,当请求转移一个数字通证时,该应用程序发送警告该转移对其他数字通证的后果的消息。

[0083] 同样有利地,智能合约可以被配置为限制一个或多个数字通证的转移,例如以便:

[0084] -使一个或多个数字通证的转移取决于第三方(例如制造商)的授权,

[0085] -限制一个或多个数字通证的转移频率,例如,将数字通证的转移限制为每年一次转移,

[0086] -禁止一个或多个数字通证的转移。

[0087] 特别地,这些限制中的任一个可以在钟表丢失或盗窃的情况下实现。

[0088] 在禁止一个或多个数字通证的转移的情况下,可以规定包含数字通证的整个数字钱包能够从一个所有者转移到另一个所有者。

[0089] 通过上述方法,可以管理对例如钟表的主要元素的权限。特别地,可以在区块链协议上认证和跟踪过去的各种交易。

[0090] 有利地,区块链协议上的所有交易都可以由制造商保存。为此,特别地,制造商可以管理架构的网络节点,从而允许执行用于构造数字通证的存储和认证中涉及的区块链的处理操作。

[0091] 注意,所有者有利地使用与公钥相关联并附加到钟表本身上的代码来连接到提供上述数字内容的网站。例如,使用智能电话读出附加到钟表上的QR码允许电话自动连接到提供内容的网站。

[0092] 本发明还涉及对象和/或附加到对象的有形介质和/或附件,特别是钟表1,特别是钟表1的发条盒2或发条盒2的棘轮3作为用于数字钱包的公钥或与公钥相关联的代码4的介质的用途。

[0093] 特别是分布式计算架构的设备的实施方式允许管理与对象相关联的权限。该实施方式包括实现上述方法的硬件和/或软件元件。

[0094] 计算机程序产品的实施方式包括记录在计算机可读介质上的程序代码指令,该程序代码指令用于当在计算机上运行所述程序时实现上述方法的步骤。

[0095] 下面详细描述用于实现所描述的管理方法的程序产品代码的示例。

[0096]

```
pragma solidity ^0.8.0;

interface IERC721Metadata {
```

```
    function name() external view returns (string memory_name);

    function symbol() external view returns (string memory_symbol);

    function tokenURI(uint256 _tokenId) external view returns (string memory);
}

interface IERC721 {
    event Transfer(
    address indexed_from,
    address indexed_to,
    uint256 indexed_tokenId
);
```

[0097]

```
event Approval(
    address indexed_owner,
    address indexed_approved,
    uint256 indexed_tokenId
);
event ApprovalForAll(
    address indexed_owner,
    address indexed_operator,
    bool_approved
);

function balanceOf(address_owner) external view returns (uint256);

function ownerOf(uint256_tokenId) external view returns (address);
```

```
function safeTransferFrom(
    address_from,
    address_to,
    uint256_tokenId,
    bytes calldata data
) external;

function safeTransferFrom(
    address_from,
    address_to,
    uint256_tokenId
) external;

function transferFrom(
    address_from,
    address_to,
    uint256_tokenId
) external;

function approve(address_approved, uint256_tokenId) external;

function setApprovalForAll(address _operator, bool _approved) external;

function getApproved(uint256_tokenId) external view returns (address);

function isApprovedForAll(address_owner, address_operator)
    external
    view
    returns (bool);
```

[0098]

```
}

interface IERC721Receiver {
  function onERC721Received(
      address_operator,
      address_from,
      uint256_tokenId,
      bytes calldata_data
  ) external returns (bytes4);
}

interface ILuxochainNFT {
event OwnershipTransferred(address indexed, address indexed);
```

[0099] `function owner() external view returns (address);`

```
function exists(uint256 tokenId) external view returns (bool);

function getStorageType() external view returns (uint256);

function isLuxochainNFT() external view returns (bool);

function transferOwnership(address newIssuer) external;

function totalSupply() external view returns (uint256);

function count() external view returns (uint256);

event TokenFreezed(uint256 tokenId, address unfreezableAddress);
```

```
    event TokenUnfreezed(uint256 tokenId, address newOwner);

    function safeMint(
        uint256 tokenId,
        address to,
        string calldata tokenMetadataURI
    ) external;

    function multipleSafeMint(
        uint256[] calldata_tokensIds,
        address_to,
        string[] calldata_tokenMetadataURIs
    ) external;
```

[0100]

```
    function freezeToken(uint256 tokenId, address unfreezeAddress) external;

    function unfreezeToken(uint256 tokenId, address newOwner) external;

    function isTokenFreezed(uint256 tokenId) external view returns (bool);

    function burn(uint256 tokenId) external;
}

contract LUXO_BLG_NFT is IERC721, IERC721Metadata, ILuxochainNFT {
    bytes4 private constant _INTERFACE_ID_ERC721_METADATA =
0x5b5e139f;
    bytes4 private constant_INTERFACE_ID_ERC721 = 0x80ac58cd;
    bytes4 private constant_INTERFACE_ID_ERC165 = 0x01ffc9a7;
```

```
        uint256 private_totalSupply;
        uint256 private_maxTotalSupply;
        uint256 private_count;

        string private_name;
        string private_symbol;

        address private_issuer;

        mapping(address => mapping(address => bool))
    private_operatorApprovals;
        mapping(uint256 => address) private_tokenApprovals;
        mapping(uint256 => address) private_owners;
        mapping(address => uint256) private_balances;
[0101]  mapping(uint256 => string) private_metadataURIs;
        mapping(uint256 => bool) private_freezed;
        mapping(uint256 => address) private_unfreezeAddresses;
        bool private _isFreeMintable;

        constructor(
            string memory name_,
            string memory symbol_,
            bool isFreeMintable_,
            uint256 maxTotalSupply_
        ) {
        _name = name_;
        _symbol = symbol_;
        _IsFreeMintable = isFreeMintable_;
        _issuer = msg.sender;
```

```
        _maxTotalSupply = maxTotalSupply_;
    }

    function isLuxochainNFT() public view virtual override returns (bool) {
        return true;
    }

    function getStorageType() public view virtual override returns (uint256)
{
        return 1;
    }

    function owner() public view virtual override returns (address) {
        return _issuer;
[0102]  }

    function transferOwnership(address newIssuer) public virtual override {
        require(msg.sender == _issuer, "Not issuer");
        require(newIssuer != address(0), "New issuer is the zero address");
        _issuer = newIssuer;
        emit OwnershipTransferred(msg.sender, newIssuer);
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        virtual
        returns (bool)
    {
```

```
        return
            interfaceId == _INTERFACE_ID_ERC721_METADATA //
            interfaceId == _INTERFACE_ID_ERC721 //
            interfaceId == _INTERFACE_ID_ERC165;
    }

    function balanceOf(address _owner)
        public
        view
        virtual
        override
        returns (uint256)
    {
        return _balances[_owner];
    }

    function ownerOf(uint256 tokenId)
        public
        view
        virtual
        override
        returns (address)
    {
    return _owners[tokenId];
    }

    function name() public view virtual override returns (string memory) {
        return _name;
    }
```

[0103]

```
function symbol() public view virtual override returns (string memory) {
    return_symbol;
}

function totalSupply() external view override returns (uint256) {
    return_totalSupply;
}

function count() external view override returns (uint256) {
    return_count;
}

function tokenURI(uint256 tokenId)
    public
    view
    virtual
    override
    returns (string memory)
{
    return_metadataURIs[tokenId];
}

function transferFrom(
    address from,
    address to,
    uint256 tokenId
) public virtual override {
    require(
```

[0104]

```
            _isApprovedOrOwner(msg.sender, tokenId),
            "Transfer caller is not owner nor approved"
        );
        _transfer(from, to, tokenId);
    }

    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId
    ) public virtual override {
        safeTransferFrom(from, to, tokenId, "");
    }
```

[0105]

```
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId,
        bytes memory _data
    ) public virtual override {
        _safeTransfer(from, to, tokenId, _data);
    }

    function _safeTransfer(
        address from,
        address to,
        uint256 tokenId,
        bytes memory _data
    ) internal virtual {
```

```
        _transfer(from, to, tokenId);
        require(
            _checkOnERC721Received(from, to, tokenId, _data),
            "Transfer to non ERC721Receiver implementer"
        );
    }

    function _transfer(
        address from,
        address to,
        uint256 tokenId
    ) internal virtual {
        require(to != address(0), "Transfer to the zero address");
        require(!_freezed[tokenId], "Token is freezed");
[0106]  require(
            _isApprovedOrOwner(msg.sender, tokenId),
            "Sender cannot transfer token"
        );
        _balances[from] -= 1;
        _balances[to] += 1;
        _owners[tokenId] = to;
        _approve(address(0), tokenId);
        emit Transfer(from, to, tokenId);
    }

    function exists(uint256 tokenId)
        public
        view
        virtual
```

```
        override
        returns (bool)
    {
        return_owners[tokenId] != address(O);
    }
    function_safeMint(
        uint256 tokenId,
        address to,
        string calldata tokenMetadataURI
    ) internal virtual {
        _owners[tokenId] = to;
        _metadataURIs[tokenId] = tokenMetadataURI;
        emit Transfer(address(0), to, tokenId);
    }
```

[0107]

```
    function safeMint(
        uint256 tokenId,
        address to,
        string calldata tokenMetadataURI
    ) external override {
        require(
            msg.sender ==_issuer // _isFreeMintable,
            "Not Issuer nor free mint"
        );
        require(to != address(0), "Mint to the zero address");
        require(!exists(tokenId), "Token already minted");
        _totalSupply += 1;
        require(_maxTotalSupply >= _totalSupply, "Max total supply
reached");
```

```
        _count += 1;
        _balances[to] += 1;
        _safeMint(tokenId, to, tokenMetadataURI);
    }

    function multipleSafeMint(
        uint256[] calldata tokensIds,
        address to,
        string[] calldata tokenMetadataURIs
    ) external override {
        require(
            msg. sender == _issuer // _isFreeMintable,
            "Not Issuer nor free mint"
        );
```

[0108]

```
        require(
            tokensIds.length == tokenMetadataURIs.length,
            "Different number of tokens and metadata provided"
        );
        require(to != address(0), "Mint to the zero address");
        require(
            _maxTotalSupply >= _totalSupply + tokensIds.length,
            "Max total supply reached"
        );
        for (uint256 i = 0; i < tokensIds.length; i++) {
            uint256 tokenId = tokensIds[i];
            require(!exists(tokenId), "Token already minted");
            _safeMint(tokenId, to, tokenMetadataURIs[i]);
    }
    _count += tokensIds.length;
```

```
        _totalSupply += tokensIds.length;
        _balances[to] += tokensIds.length;
        }

        function freezeToken(uint256 tokenId, address unfreezeAddress)
            external
            override
        {
            require(
                _isApprovedOrOwner(msg.sender, tokenId),
                "Cannot freeze token: permission denied"
            );
            require(unfreezeAddress != address(0), "Invalid unfreeze address");
            _freezed[tokenId] = true;
[0109]      _unfreezeAddresses[tokenId] = unfreezeAddress;
            emit TokenFreezed(tokenId, unfreezeAddress);
        }

        function unfreezeToken(uint256 tokenId, address newOwner)
            external
            override
        {
            require(
                _unfreezeAddresses[tokenId] == msg.sender,
                "You're not allowed to unfreeze this token"
            );
            require(newOwner != address(0), "Invalid new owner address");
            delete _freezed[tokenId];
            delete _unfreezeAddresses[tokenId];
```

```
        _owners[tokenId] = newOwner;
        emit TokenUnfreezed(tokenId, newOwner);
    }

    function isTokenFreezed(uint256 tokenId)
        public
        view
        virtual
        override
        returns (bool)
    {
        return_freezed[tokenId];
    }
```

[0110]

```
    function burn(uint256 tokenId) external override {
        require(!_freezed[tokenId], "Token is freezed");
        address__owner = ownerOf(tokenId);
        require(_isApprovedOrOwner(msg.sender, tokenId), "Permission
denied");
        _totalSupply -= 1;
        _balances[__owner] -= 1;
        delete_owners[tokenId];
        _approve(address(0), tokenId);
        emit Transfer(__owner, address(0), tokenId);
    }

    function isContract(address account) internal view returns (bool) {
        return account.code.length > 0;
    }
```

```
    function _checkOnERC721Received(
        address from,
        address to,
        uint256 tokenId,
        bytes memory _data
    ) private returns (bool) {
        if (!isContract(to)) return false;
        try
            IERC721Receiver(to).onERC721Received(
                msg.sender,
                from,
                tokenId,
                _data
            )
```

[0111]

```
        returns (bytes4 retval) {
            return retval == IERC721Receiver.onERC721Received.selector;
        } catch (bytes memory reason) {
            if (reason.length == 0) {
                revert("Transfer to non ERC721Receiver implementer");
            } else {
                assembly {
                    revert(add(32, reason), mload(reason))
                }
            }
        }
    }
    function _isApprovedOrOwner(address spender, uint256 tokenId)
        internal
```

```
        view
        virtual
        returns (bool)
    {
        require(exists(tokenId), "Operator query for nonexistent token");
        address __owner = ownerOf(tokenId);
        return (spender == __owner //
            getApproved(tokenId) == spender //
            isApprovedForAll(__owner, spender));
    }

    function _setApprovalForAll(
        address __owner,
        address operator,
[0112]  bool approved
    ) internal virtual {
        require(__owner != operator, "Approve to caller");
        _operatorApprovals[__owner][operator] = approved;
        emit ApprovalForAll(__owner, operator, approved);
    }

    function _approve(address to, uint256 tokenId) internal virtual {
        _tokenApprovals[tokenId] = to;
        emit Approval(ownerOf(tokenId), to, tokenId);
    }

    function approve(address to, uint256 tokenId) public virtual override {
        address __owner = ownerOf(tokenId);
        require(to != __owner, "Approval to current owner");
```

```
        require(
            msg.sender == __owner // isApprovedForAll(__owner, msg.sender),
            "Approve caller is not owner nor approved for all"
        );
        _approve(to, tokenId);
    }

    function getApproved(uint256 tokenId)
        public
        view
        virtual
        override
        returns (address)
    {
[0113]  require(exists(tokenId), "Approved query for nonexistent token");
        return_tokenApprovals[tokenId];
    }

    function setApprovalForAll(address operator, bool approved)
        public
        virtual
        override
    {
        _setApprovalForAll(msg.sender, operator, approved);
    }

    function isApprovedForAll(address__owner, address operator)
        public
        view
```

```
        virtual
        override
        returns (bool)
[0114]  {
            return_operatorApprovals[__owner][operator];
        }
    }
```

[0115] 本发明还涉及一种计算机可读数据记录介质,其上记录有计算机程序,该计算机程序包括用于实现上述方法的程序代码指令。

[0116] 本发明还涉及一种来自携带上述计算机程序产品的数据介质的信号。

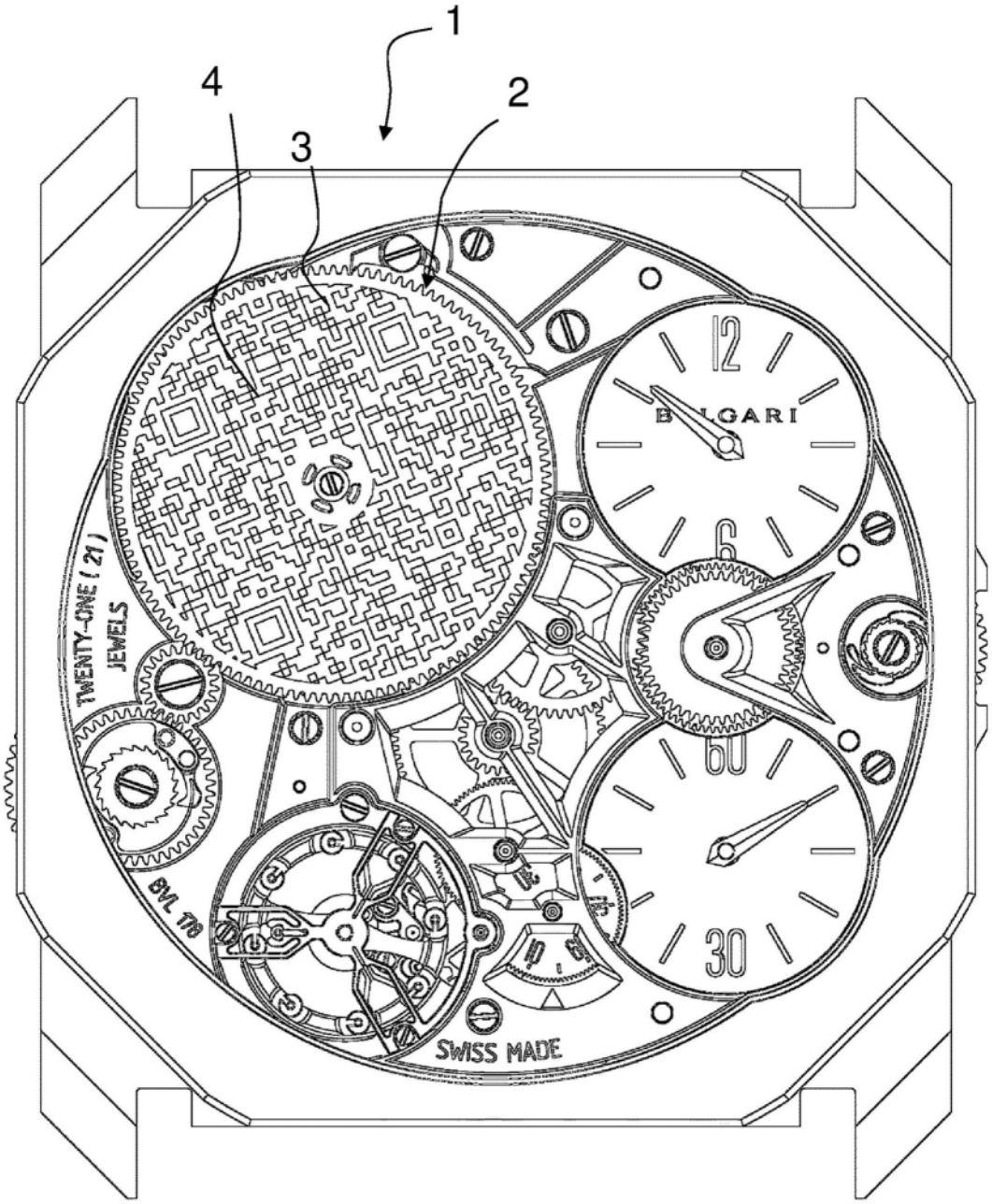


图1