The Move Book

Sui introduces four distinct ownership types for objects: single owner, shared state, immutable shared state, and object-owner. Each model offers unique characteristics and suits different use cases, enhancing flexibility and control in object management.

Note that ownership does not control the confidentiality of an object — it is always possible to read the contents of an on-chain object from outside of Move. You should never store unencrypted secrets inside of objects.

See the Storage Functions chapter for details on how to change the owner or ownership type of an object.

The account owner, also known as the single owner model, is the foundational ownership type in Sui. Here, an object is owned by a single account, granting that account exclusive control over the object within the behaviors associated with its type. This model embodies the concept of true ownership , where the account possesses complete authority over the object, making it inaccessible to others for modification or transfer. This level of ownership clarity is a significant advantage over other blockchain systems, where ownership definitions can be more ambiguous, and smart contracts may have the ability to alter or transfer assets without the owner's consent.

Single owner model has its limitations: for example, it is very tricky to implement a marketplace for digital assets without a shared state. For a generic marketplace scenario, imagine that Alice owns an asset X, and she wants to sell it by putting it into a shared marketplace. Then Bob can come and buy the asset directly from the marketplace. The reason why this is tricky is that it is impossible to write a smart contract that would "lock" the asset in Alice's account and take it out when Bob buys it. First, it will be a violation of the single owner model, and second, it requires a shared access to the asset.

To facilitate a problem of shared data access, Sui has introduced a shared ownership model. In this model, an object can be shared with the network. Shared objects can be read and modified by any account on the network, and the rules of interaction are defined by the implementation of the object. Typical uses for shared objects are: marketplaces, shared resources, escrows, and other scenarios where multiple accounts need access to the same state.

Sui also offers the frozen object model, where an object becomes permanently read-only. These immutable objects, while readable, cannot be modified or moved, providing a stable and constant state accessible to all network participants. Frozen objects are ideal for public data, reference materials, and other use cases where the state permanence is desirable.

The last ownership model in Sui is the object owner . In this model, an object is owned by another object. This feature allows creating complex relationships between objects, storing large heterogeneous collections, and implementing extensible and modular systems. Practically speaking, since the transactions are initiated by accounts, the transaction still accesses the parent object, but it can then access the child objects through the parent object.

A use case we love to mention is a game character. Alice can own the Hero object from a game, and the Hero can own items: also represented as objects, like a "Map", or a "Compass". Alice may take the "Map" from the "Hero" object, and then send it to Bob, or sell it on a marketplace. With object owner, it becomes very natural to imagine how the assets can be structured and managed in relation to each other.

In the next section we will talk about transaction execution paths in Sui, and how the ownership models affect the transaction execution.

Account Owner (or Single Owner)

The account owner, also known as the single owner model, is the foundational ownership type in Sui. Here, an object is owned by a single account, granting that account exclusive control over the object within the behaviors associated with its type. This model embodies the concept of true ownership , where the account possesses complete authority over the object, making it inaccessible to others for modification or transfer. This level of ownership clarity is a significant advantage over other blockchain systems, where ownership definitions can be more ambiguous, and smart contracts may have the ability to alter or transfer assets without the owner's consent.

Single owner model has its limitations: for example, it is very tricky to implement a marketplace for digital assets without a shared state. For a generic marketplace scenario, imagine that Alice owns an asset X, and she wants to sell it by putting it into a shared marketplace. Then Bob can come and buy the asset directly from the marketplace. The reason why this is tricky is that it is impossible to write a smart contract that would "lock" the asset in Alice's account and take it out when Bob buys it. First, it will be a violation of the single owner model, and second, it requires a shared access to the asset.

To facilitate a problem of shared data access, Sui has introduced a shared ownership model. In this model, an object can be shared with the network. Shared objects can be read and modified by any account on the network, and the rules of interaction are defined

by the implementation of the object. Typical uses for shared objects are: marketplaces, shared resources, escrows, and other scenarios where multiple accounts need access to the same state.

Sui also offers the frozen object model, where an object becomes permanently read-only. These immutable objects, while readable, cannot be modified or moved, providing a stable and constant state accessible to all network participants. Frozen objects are ideal for public data, reference materials, and other use cases where the state permanence is desirable.

The last ownership model in Sui is the object owner . In this model, an object is owned by another object. This feature allows creating complex relationships between objects, storing large heterogeneous collections, and implementing extensible and modular systems. Practically speaking, since the transactions are initiated by accounts, the transaction still accesses the parent object, but it can then access the child objects through the parent object.

A use case we love to mention is a game character. Alice can own the Hero object from a game, and the Hero can own items: also represented as objects, like a "Map", or a "Compass". Alice may take the "Map" from the "Hero" object, and then send it to Bob, or sell it on a marketplace. With object owner, it becomes very natural to imagine how the assets can be structured and managed in relation to each other.

In the next section we will talk about transaction execution paths in Sui, and how the ownership models affect the transaction execution.

Shared State

Single owner model has its limitations: for example, it is very tricky to implement a marketplace for digital assets without a shared state. For a generic marketplace scenario, imagine that Alice owns an asset X, and she wants to sell it by putting it into a shared marketplace. Then Bob can come and buy the asset directly from the marketplace. The reason why this is tricky is that it is impossible to write a smart contract that would "lock" the asset in Alice's account and take it out when Bob buys it. First, it will be a violation of the single owner model, and second, it requires a shared access to the asset.

To facilitate a problem of shared data access, Sui has introduced a shared ownership model. In this model, an object can be shared with the network. Shared objects can be read and modified by any account on the network, and the rules of interaction are defined by the implementation of the object. Typical uses for shared objects are: marketplaces, shared resources, escrows, and other scenarios where multiple accounts need access to the same state.

Sui also offers the frozen object model, where an object becomes permanently read-only. These immutable objects, while readable, cannot be modified or moved, providing a stable and constant state accessible to all network participants. Frozen objects are ideal for public data, reference materials, and other use cases where the state permanence is desirable.

The last ownership model in Sui is the object owner . In this model, an object is owned by another object. This feature allows creating complex relationships between objects, storing large heterogeneous collections, and implementing extensible and modular systems. Practically speaking, since the transactions are initiated by accounts, the transaction still accesses the parent object, but it can then access the child objects through the parent object.

A use case we love to mention is a game character. Alice can own the Hero object from a game, and the Hero can own items: also represented as objects, like a "Map", or a "Compass". Alice may take the "Map" from the "Hero" object, and then send it to Bob, or sell it on a marketplace. With object owner, it becomes very natural to imagine how the assets can be structured and managed in relation to each other.

In the next section we will talk about transaction execution paths in Sui, and how the ownership models affect the transaction execution.

Immutable (Frozen) State

Sui also offers the frozen object model, where an object becomes permanently read-only. These immutable objects, while readable, cannot be modified or moved, providing a stable and constant state accessible to all network participants. Frozen objects are ideal for public data, reference materials, and other use cases where the state permanence is desirable.

The last ownership model in Sui is the object owner . In this model, an object is owned by another object. This feature allows creating complex relationships between objects, storing large heterogeneous collections, and implementing extensible and modular systems. Practically speaking, since the transactions are initiated by accounts, the transaction still accesses the parent object, but it can then access the child objects through the parent object.

A use case we love to mention is a game character. Alice can own the Hero object from a game, and the Hero can own items: also

represented as objects, like a "Map", or a "Compass". Alice may take the "Map" from the "Hero" object, and then send it to Bob, or sell it on a marketplace. With object owner, it becomes very natural to imagine how the assets can be structured and managed in relation to each other.

In the next section we will talk about transaction execution paths in Sui, and how the ownership models affect the transaction execution.

Object Owner

The last ownership model in Sui is the object owner . In this model, an object is owned by another object. This feature allows creating complex relationships between objects, storing large heterogeneous collections, and implementing extensible and modular systems. Practically speaking, since the transactions are initiated by accounts, the transaction still accesses the parent object, but it can then access the child objects through the parent object.

A use case we love to mention is a game character. Alice can own the Hero object from a game, and the Hero can own items: also represented as objects, like a "Map", or a "Compass". Alice may take the "Map" from the "Hero" object, and then send it to Bob, or sell it on a marketplace. With object owner, it becomes very natural to imagine how the assets can be structured and managed in relation to each other.

In the next section we will talk about transaction execution paths in Sui, and how the ownership models affect the transaction execution.

Summary

In the next section we will talk about transaction execution paths in Sui, and how the ownership models affect the transaction execution.

Next Steps

In the next section we will talk about transaction execution paths in Sui, and how the ownership models affect the transaction execution.