`Hashtable` and `Dictionary` are both used to store key-value pairs, but they have some significant differences in terms of their usage, performance, and features:

1. Type Safety:

- `Hashtable`: It is not type-safe and can store key-value pairs of any type of objects. This can lead to runtime type-related issues.

- `Dictionary`: It is type-safe and uses generics, which means you specify the data types of keys and values at compile-time, reducing the risk of type-related errors.

2. Performance:

- `Hashtable`: Performance characteristics can vary. It uses a hashing algorithm for key lookup, which can lead to efficient access in most cases. However, collisions (two different keys hashing to the same value) can impact performance.

- `Dictionary`: Typically provides better performance because it's type-safe and utilizes a more efficient hashing algorithm. Collisions are less common due to type-safety.

3. Null Values:

- `Hashtable`: Both keys and values can be `null`.

- `Dictionary`: Keys cannot be `null`, but values can be. This enforces a level of key integrity.

4. Usage:

- `Hashtable`: It is a non-generic collection primarily used in older .NET applications. It's not recommended for new development, as there are more modern and type-safe alternatives available.

- `Dictionary`: It is the preferred choice for new development due to type-safety and better performance. It is available in the `System.Collections.Generic` namespace.

5. Type Support:

- `Hashtable`: Supports storing objects of any type, including custom types and non-comparable objects.

- `Dictionary`: Requires key and value types to be known at compile-time, which enforces type safety.

6. Enumerating:

- `Hashtable`: It returns a collection of `DictionaryEntry` objects when enumerated. This requires additional casting to access the keys and values.

- `Dictionary`: Provides a more straightforward way to enumerate key-value pairs directly.

7. Scalability:

- `Hashtable`: Performance can degrade when it becomes large due to potential hash collisions.

- `Dictionary`: Offers more predictable and better scaling performance due to its more efficient hashing mechanism.

8. Obsolete:

- `Hashtable` is considered somewhat obsolete in modern .NET development, and it is recommended to use `Dictionary` for most use cases.

In summary, while both `Hashtable` and `Dictionary` can store key-value pairs, the `Dictionary` class is the preferred choice for most modern C# applications due to its type safety, better performance, and compatibility with generics. `Hashtable` may still be used in legacy codebases, but it is generally discouraged for new development in favor of `Dictionary`.