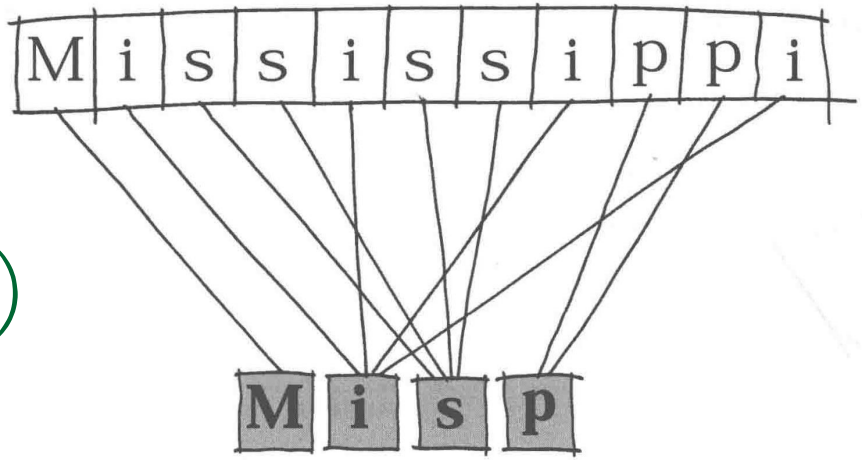
Flyweight (享元, Structural Pattern)

"Share objects and avoid waste"



Kai SHI

Flyweight Pattern

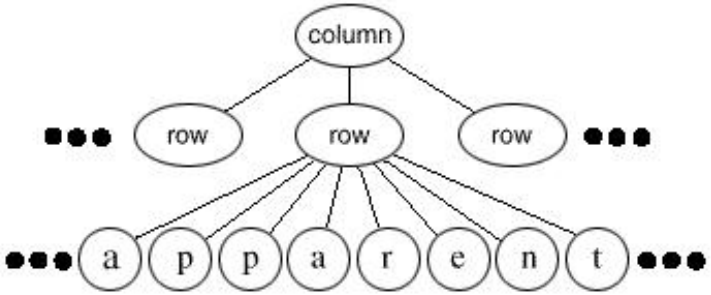
- Flyweight means “a boxer between a junior flyweight and a bantamweight, with a maximum weight of 50.81 kg”
- Intent
  - Use sharing to support large numbers of finegrained objects efficiently. (享元模式以共享的方式高效地支持大量的细粒度对象)
  - Avoid “**new**” instances by sharing instances as much as possible. “**new**” not only consumes memory, but also takes time. (“通过尽量共享实例来避免new出实例。new不仅消耗内存，还会花费时间)”

Motivation (1/2)

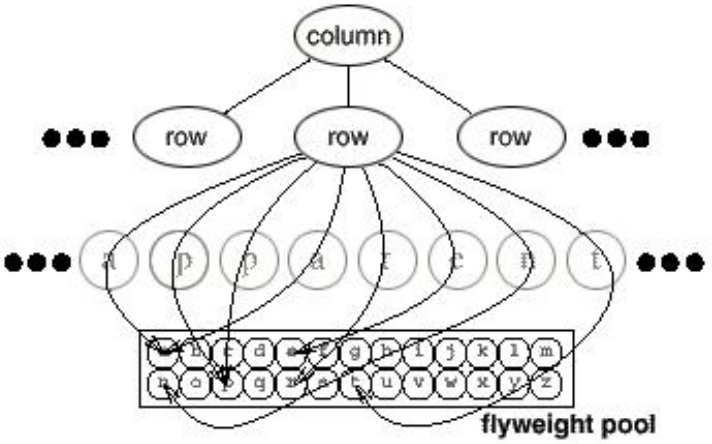
- Some applications could benefit from using objects throughout their design, but a naive implementation would be prohibitively expensive.
- Object-oriented document editors typically use objects to represent embedded elements like tables and figures.
- The drawback of such a design is its cost. Even moderate-sized documents may require hundreds of thousands of character objects, which will consume lots of memory and may incur unacceptable run-time overhead.

Motivation (2/2)

- Logically there is an object for every occurrence of a given character in the document:



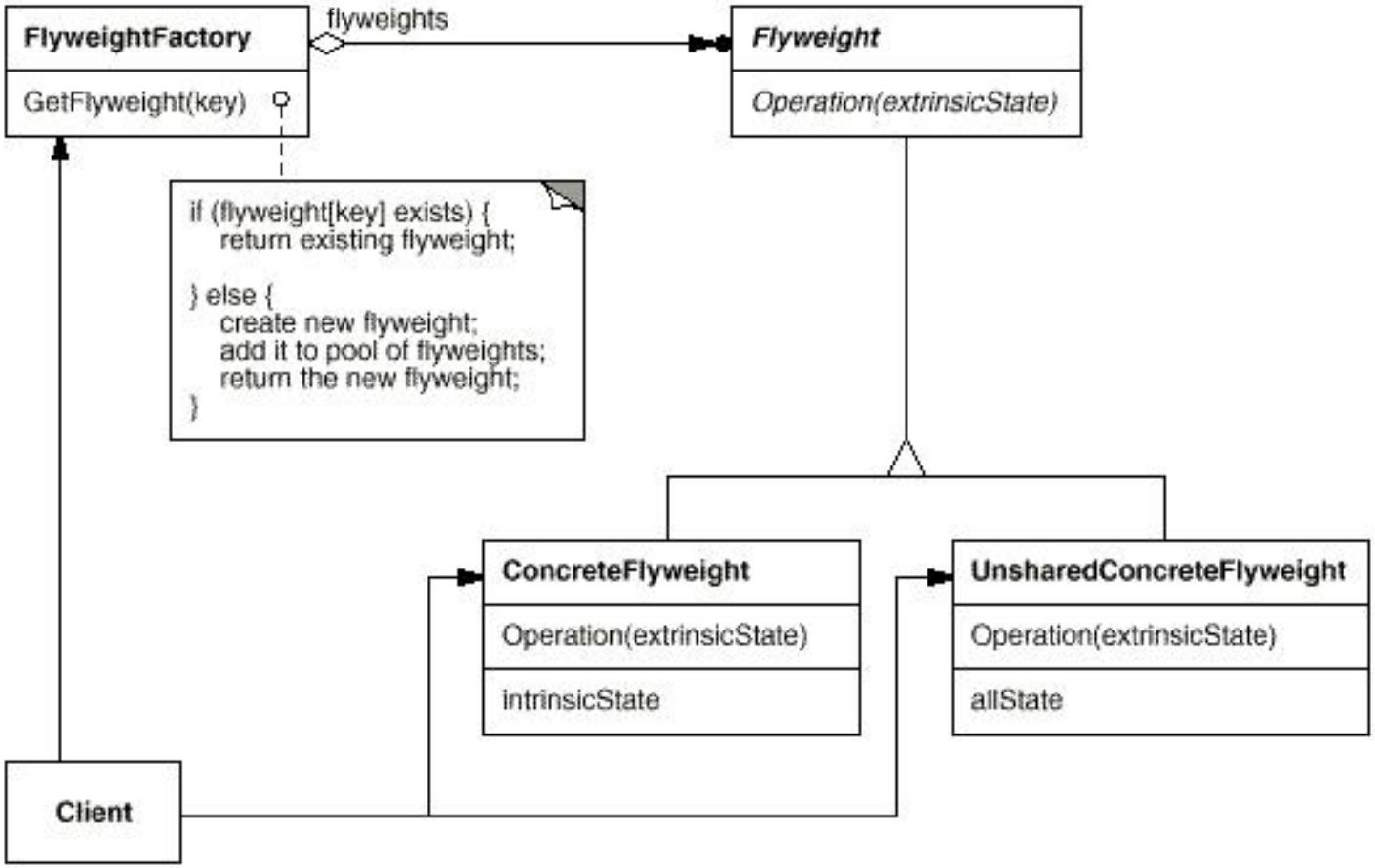
- A flyweight is a shared object that can be used in multiple contexts simultaneously.
- Each occurrence of a particular character object refers to the same instance in the shared pool of flyweight objects:



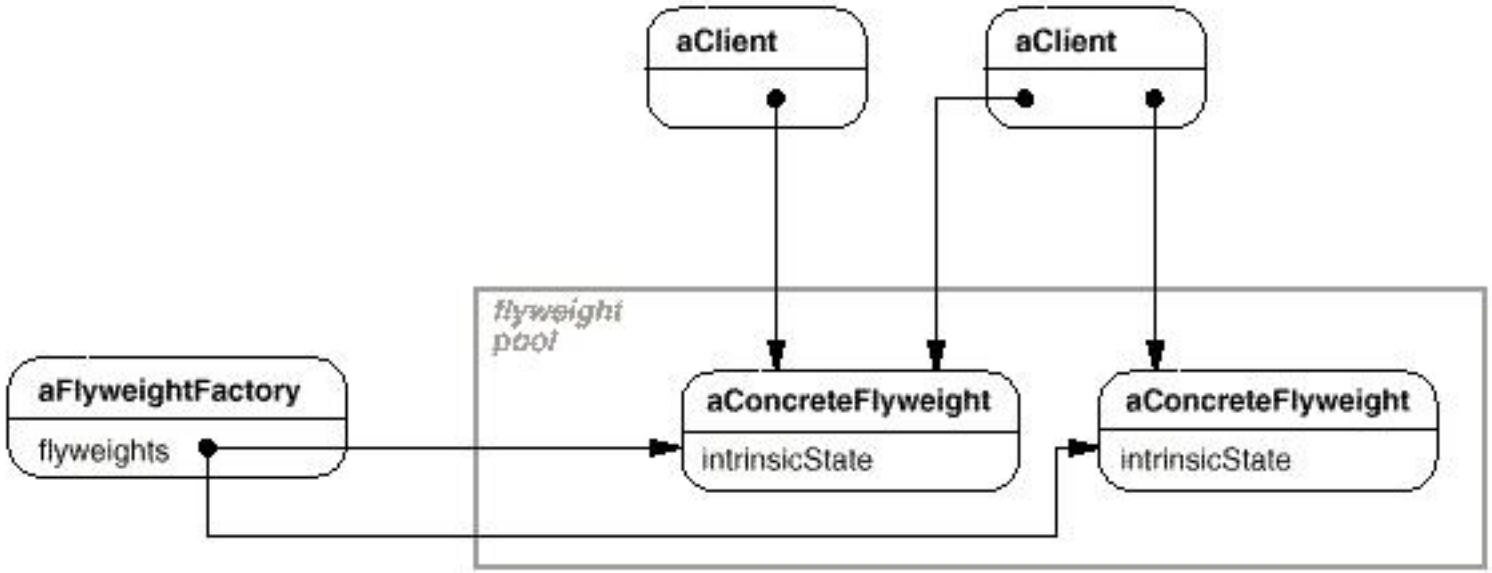
Applicability

- An application uses a large number of objects.
- Storage costs are high because of the sheer quantity (绝对数量) of objects, e.g., Chinese（汉字）.
- Most object state can be made extrinsic (外部).
  - Many groups of objects may be replaced by relatively few shared objects once extrinsic state is removed.
- The application doesn't depend on object identity.

Structure


FlyweightFactory
flyweights
Flyweight
GetFlyweight(key)
Operation(extrinsicState)
if (flyweight[key] exists) {
return existing flyweight;
} else {
create new flyweight;
add it to pool of flyweights;
return the new flyweight;
}
ConcreteFlyweight
Operation(extrinsicState)
intrinsicState
UnsharedConcreteFlyweight
Operation(extrinsicState)
allState
Client

The object diagram shows how flyweights are shared:



Participants

- Flyweight
  - Declares an interface through which flyweights can receive and act on extrinsic (外部的) state.
- ConcreteFlyweight
  - Implements the Flyweight interface and adds storage for intrinsic state, if any.
  - A ConcreteFlyweight object must be sharable.
  - Any state it stores must be intrinsic (内在的); that is, it must be independent of the ConcreteFlyweight object's context.
- UnsharedConcreteFlyweight
  - Not all Flyweight subclasses need to be shared.
  - The Flyweight interface enables sharing; it doesn't enforce it.
- FlyweightFactory
  - Creates and manages flyweight objects, and ensures that flyweights are shared properly.
  - When a client requests a flyweight, the FlyweightFactory object supplies an existing instance or creates one, if none exists.
- Client
  - Maintains a reference to flyweight(s).
  - Computes or stores the extrinsic state of flyweight(s).

Collaborations

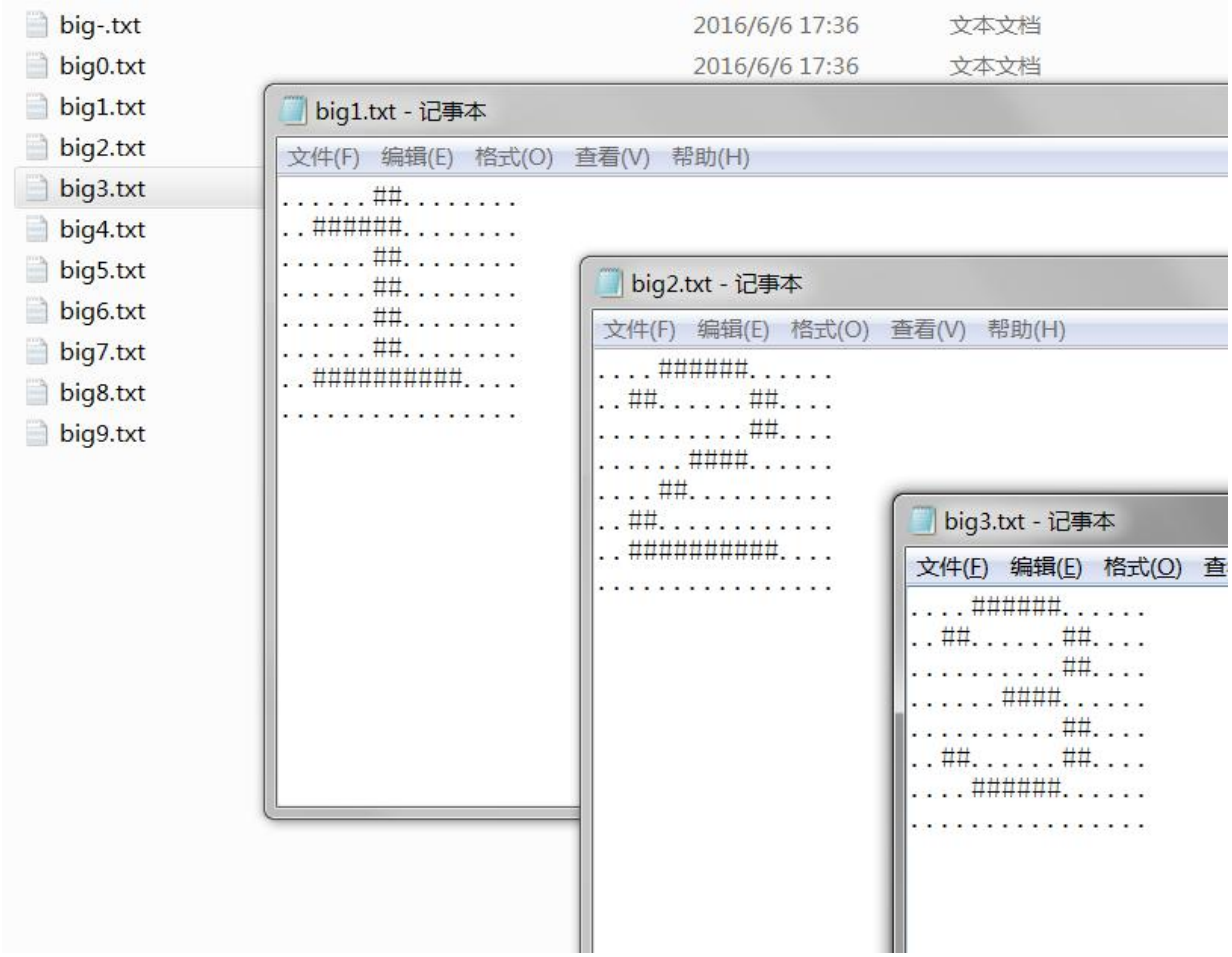
- State that a flyweight needs to function must be characterized as either intrinsic or extrinsic.
  - Intrinsic state is stored in the ConcreteFlyweight object;
  - Extrinsic state is stored or computed by Client objects. Clients pass this state to the flyweight when they invoke its operations.
- Clients should not instantiate ConcreteFlyweights directly. Clients must obtain ConcreteFlyweight objects exclusively from the FlyweightFactory object to ensure they are shared properly.

Consequences

- Flyweights may introduce run-time costs associated with transferring, finding, and/or computing extrinsic state.
- Storage savings are a function of several factors:
  - The reduction in the total number of instances that comes from sharing
  - The amount of intrinsic state per object
  - Whether extrinsic state is computed or stored.
- The more flyweights are shared, the greater the storage savings.
- The greatest savings occur when the extrinsic state can be computed rather than stored. Then you save on storage in two ways: Sharing reduces the cost of intrinsic state, and you trade extrinsic state for computation time.

Example: 非flyweight VS flyweight

Code: flyweight.bigchar.Main



Implementation Issue 1: Removing extrinsic state (1/2)

- The pattern's applicability is determined largely by how easy it is to identify extrinsic state and remove it from shared objects.
- Removing extrinsic state won't help reduce storage costs.
- Ideally, extrinsic state can be computed from a separate object structure, one with far smaller storage requirements.

Implementation Issue 1: Removing extrinsic state (2/2)

- (Best) The clients store and pass the extrinsic state to the flyweight by parameters.
- (Better) Remove the extrinsic state and associated behaviors (codes) from the flyweight to clients together.
- Encapsulate the extrinsic state and corresponding behaviors (codes) to build new classes.

Implementation Issue 2: Managing shared objects

- Because objects are shared, clients shouldn't instantiate them directly. FlyweightFactory lets clients locate a particular flyweight.
- FlyweightFactory objects often use an associative store to let clients look up flyweights of interest. The manager returns the proper flyweight given its code, creating the flyweight if it does not already exist.
- Sharability also implies some form of reference counting or garbage collection to reclaim a flyweight's storage when it's no longer needed. However, neither is necessary if the number of flyweights is fixed and small.