Linux 源码中创建型设计模式分析
一、原型模式
1. 应用场景
进程创建：通过 `fork()` 系统调用复制父进程。
2. 说明
父进程的`task_struct`被克隆生成子进程，避免重新初始化复杂数据结构
3.UML 类图
```
@startuml
class Prototype {
    + clone(): Prototype
}

class task_struct {
    + pid: int
    + clone(): task_struct
}

class Client {
    + create_process(): void
}

Prototype <|-- task_struct
Client --> Prototype : uses
@enduml
```
4.顺序图
```
@startuml
actor User
participant "fork()" as fork
participant "copy_process()" as copy
participant "task_struct" as task

User -> fork: 调用 fork()
fork -> copy: 创建子进程
copy -> task: clone() 父进程 task_struct
task --> copy: 返回子进程 task_struct
copy --> fork: 返回 PID
fork --> User: 子进程创建成功
@enduml
```
二、工厂方法模式

1. 应用场景
Slab 缓存分配：`kmem_cache_create`动态创建内存缓存对象
2. 说明
根据对象类型动态创建内存缓存
3.类图
```
@startuml
class Creator {
    + factory_method(): Product
}

class SlabFactory {
    + factory_method(): kmem_cache
}

class Product {
    + allocate(): void
}

class kmem_cache {
    + allocate(): void
}

Creator <|-- SlabFactory
Product <|-- kmem_cache
SlabFactory --> kmem_cache
@enduml
```
4.顺序图
```
@startuml
actor Client
participant "kmem_cache_factory" as Factory
participant "kmem_cache" as Product

Client -> Factory: 调用 create_cache()
activate Factory
Factory -> Product: 初始化 kmem_cache
activate Product
Product --> Factory: 返回实例
deactivate Product
Factory --> Client: 返回 kmem_cache 对象
deactivate Factory
@endumlS
```

```
## <font face="仿宋" font color = black>三、单例模式</font>
### 1. 应用场景
全局唯一进程：`init_task`是系统的第一个进程
### 2. 说明
系统启动时初始化唯一的初始进程，确保全局唯一性
### 3.类图
```
@startuml
class Singleton {
    - static instance: Singleton
    + get_instance(): Singleton
}

class init_task {
    - static instance: task_struct
    + get_instance(): task_struct
}

Singleton <|-- init_task
@enduml
```
### 4.顺序图
```
@startuml
actor Client
participant "init_task" as Singleton

Client -> Singleton: 首次调用 get_instance()
activate Singleton
Singleton -> Singleton: 检查 instance == null?
activate Singleton
Singleton -> Singleton: 创建 init_task 实例
deactivate Singleton
Singleton --> Client: 返回实例
deactivate Singleton

Client -> Singleton: 后续调用 get_instance()
activate Singleton
Singleton --> Client: 直接返回已有实例
deactivate Singleton
@enduml
```
```