主讲：唐老师（Delphi Tang）

狄泰软件
D.T.SOFTWARE

第21课

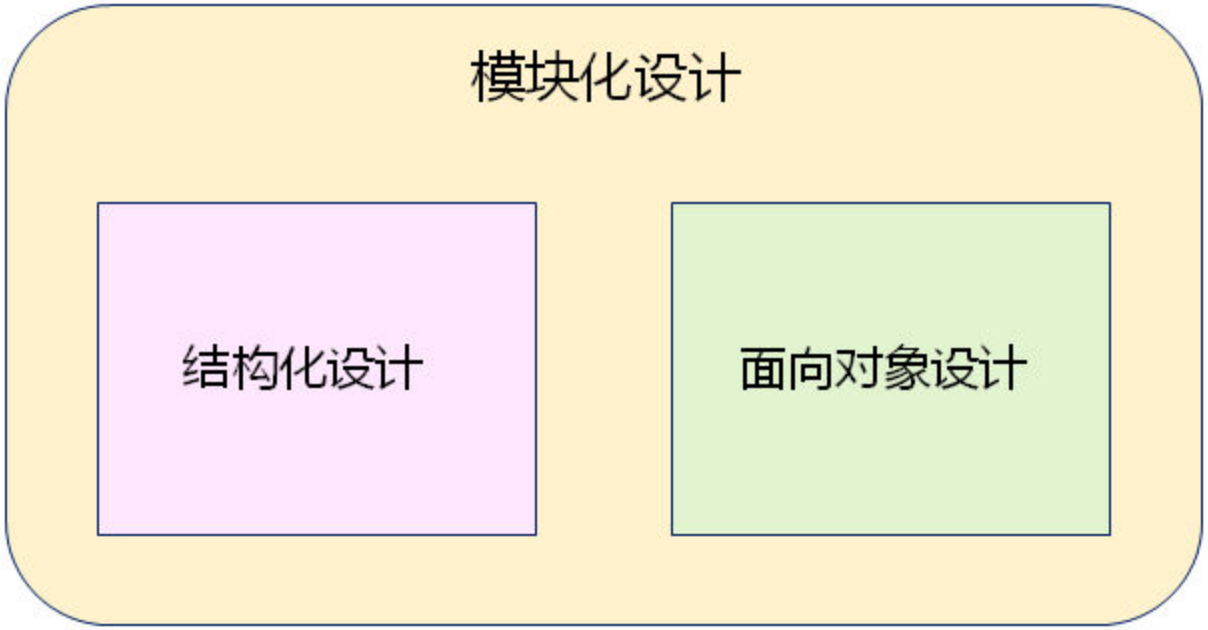
系统模块管理的设计

系统模块管理的设计

- 系统模块化设计
  - 将系统中有关联的部分组合在一起，构成具有特定功能的子系统
  - 模块的内部组成具有较强的耦合性，模块本身具有一定通用性
  - 不同的模块间可以进行相互组合与依赖，进而构成不同的产品

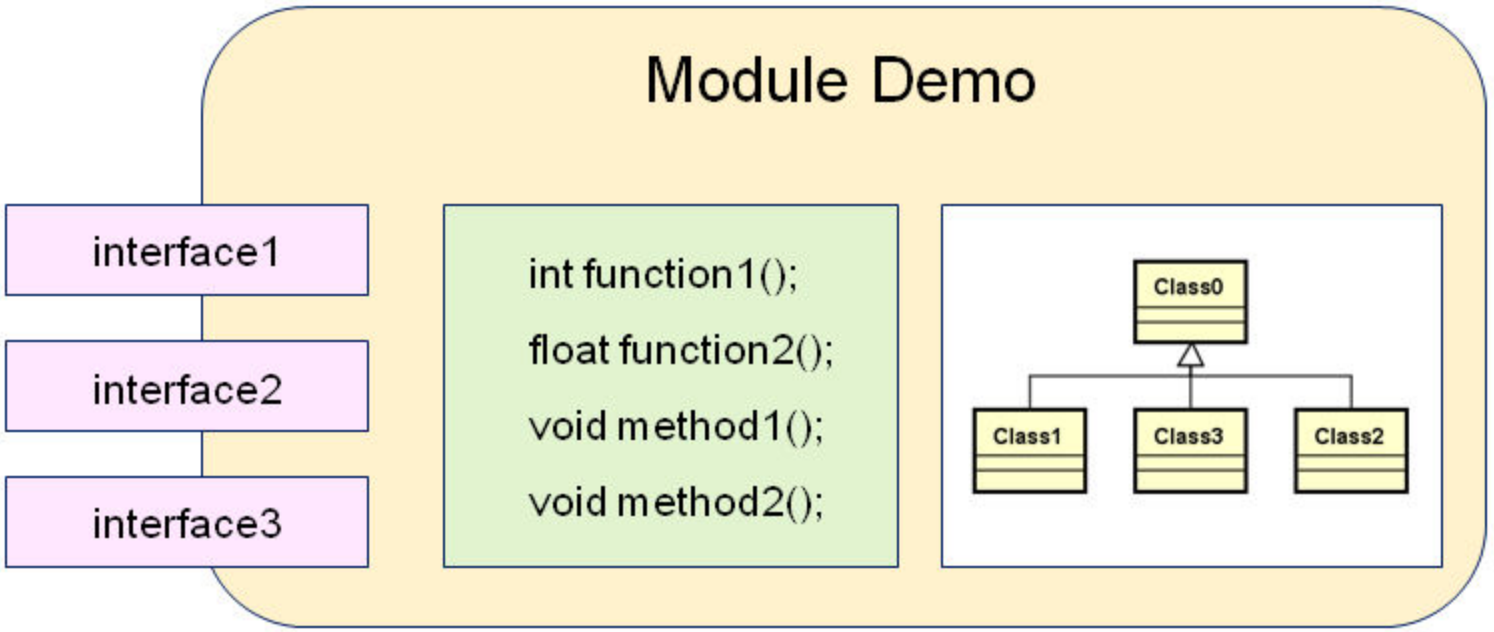
系统模块管理的设计

- 模块化设计



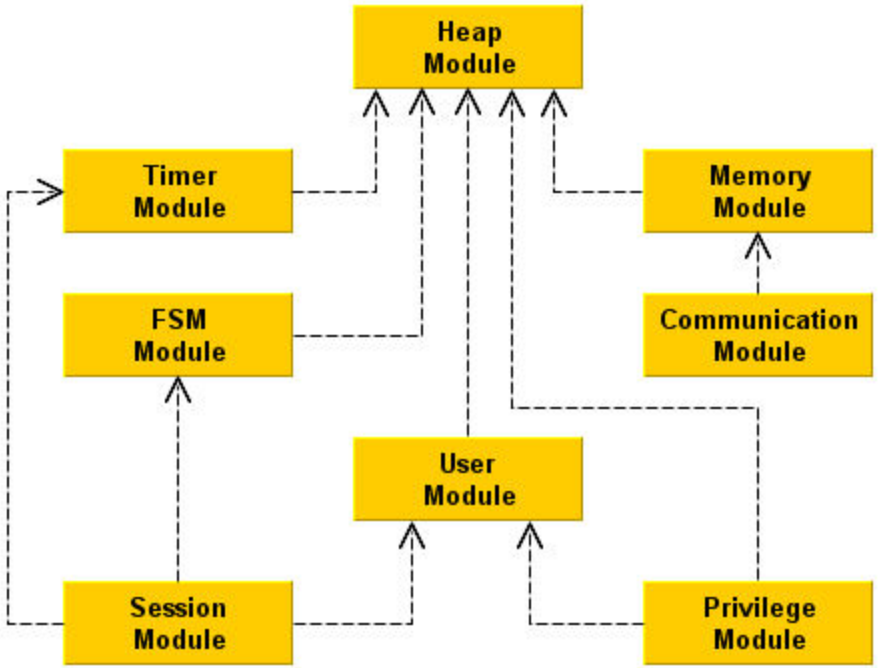
系统模块管理的设计

- 模块化设计示例



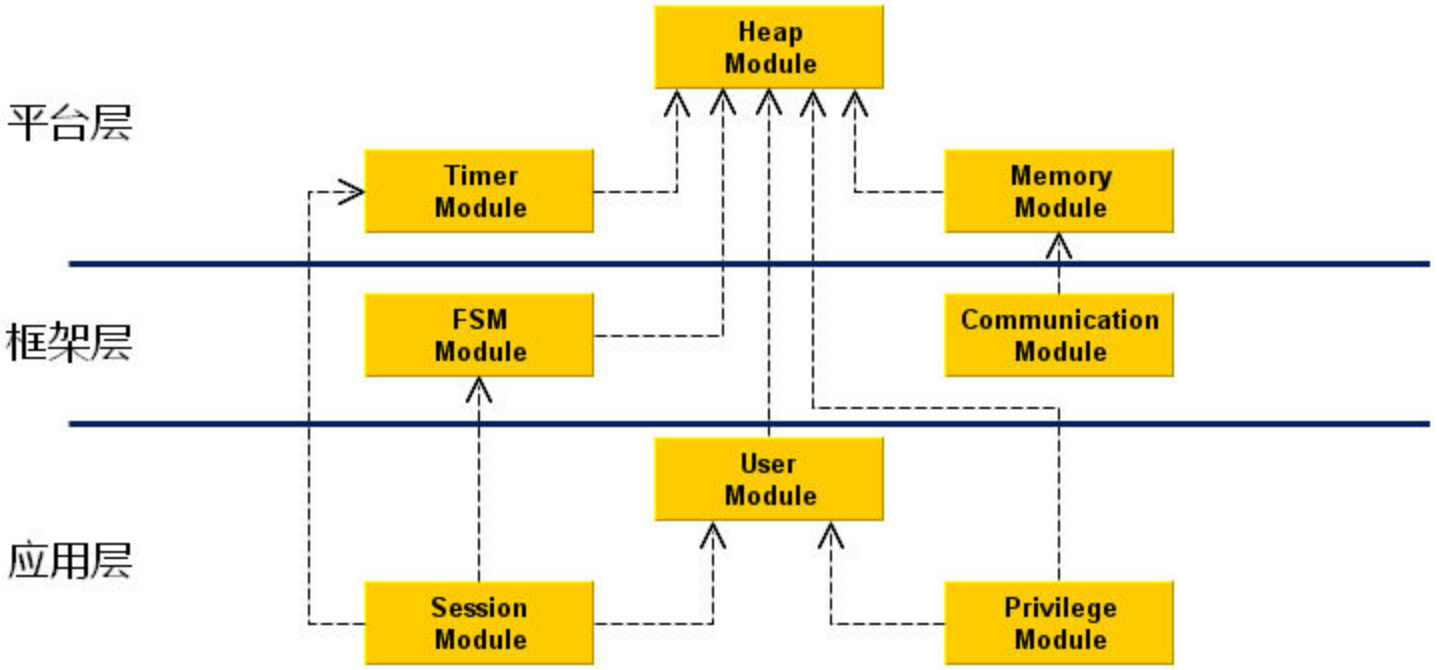
系统模块管理的设计

- 各个模块间需要相互依赖，进而完成产品功能
- 根据依赖关系能够将模块分为不同的层



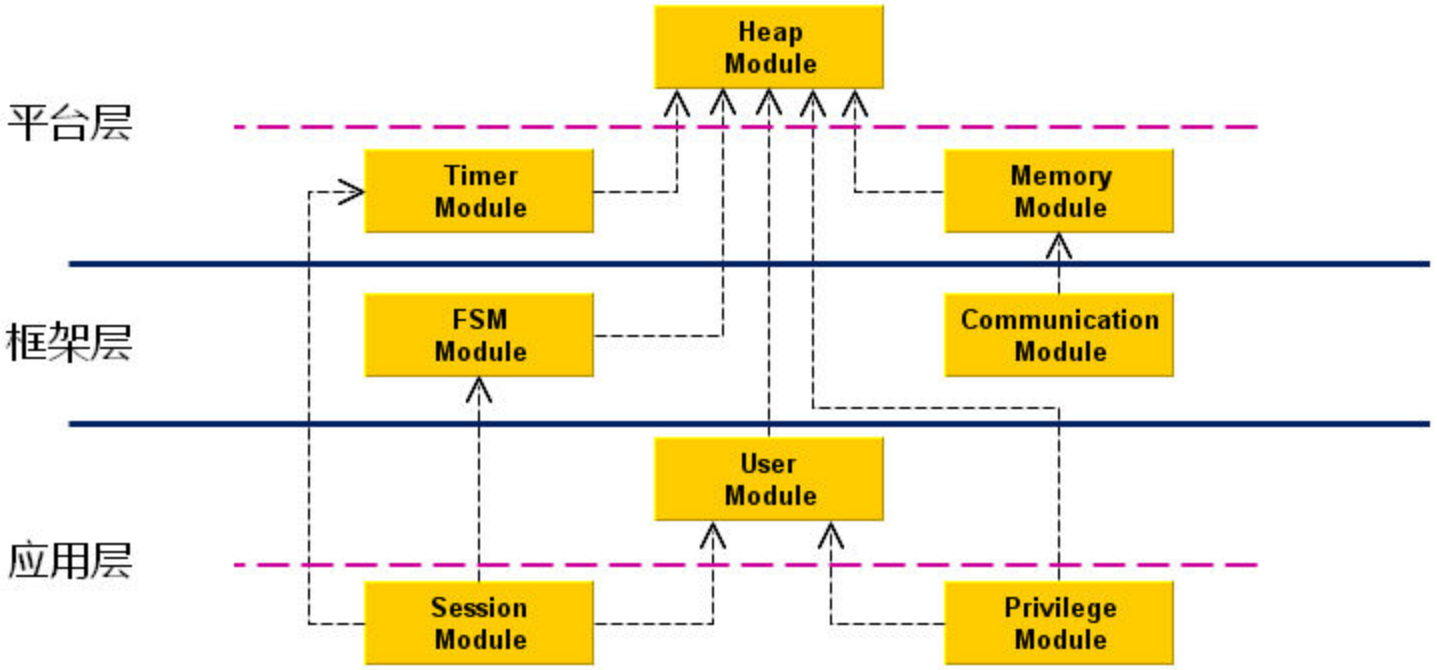
系统模块管理的设计

- 模块间的分层
  - 硬件层，系统层，平台层，框架层，应用层



系统模块管理的设计

- 模块的分级（更细的设计粒度）
  - 同一层中的模块根据依赖关系能够继续分级

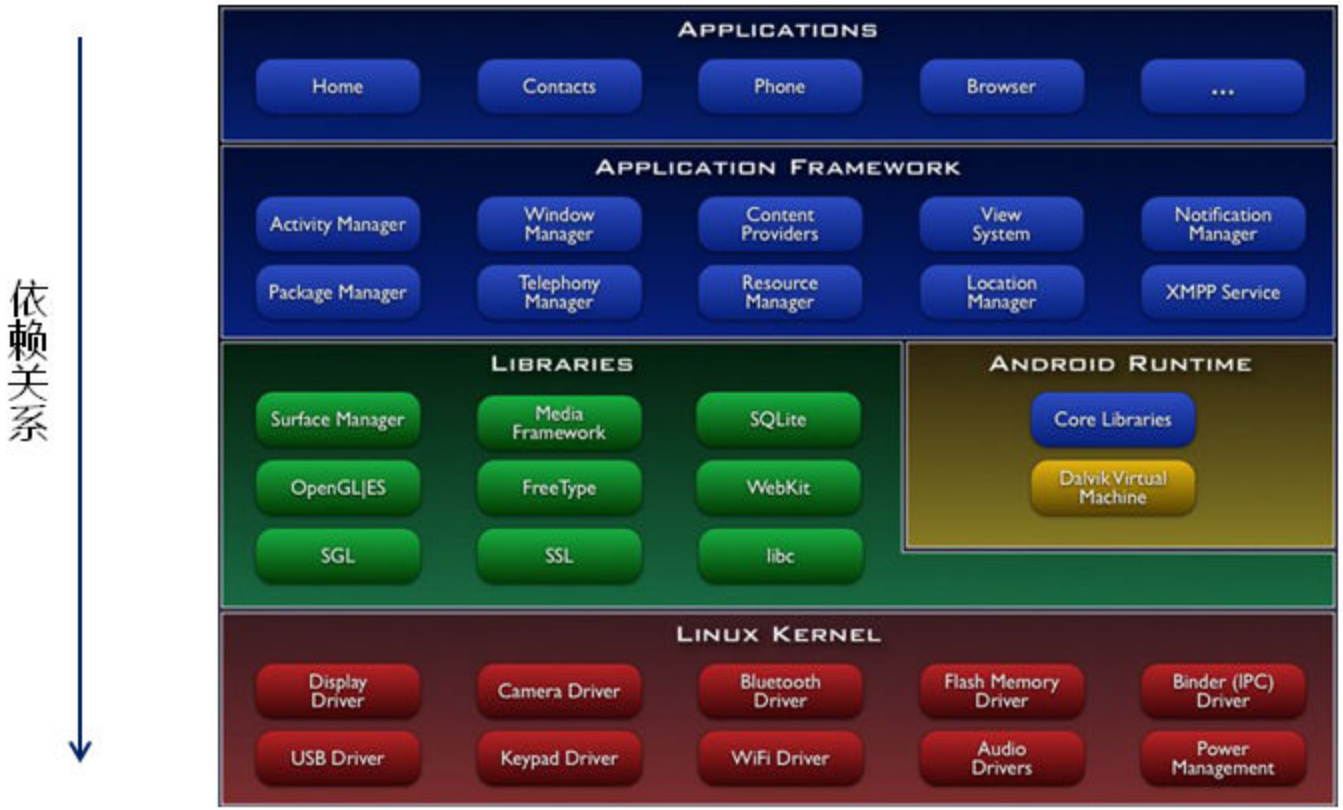


系统模块管理的设计

- 分层与分级的意义
  - 模块间的依赖关系决定了初始化的前后顺序
  - 被依赖的模块必须先初始化（底层先于上层初始化）
    - 如：
      - 硬件层模块先于系统层模块初始化
      - 框架层模块先于应用层模块初始化

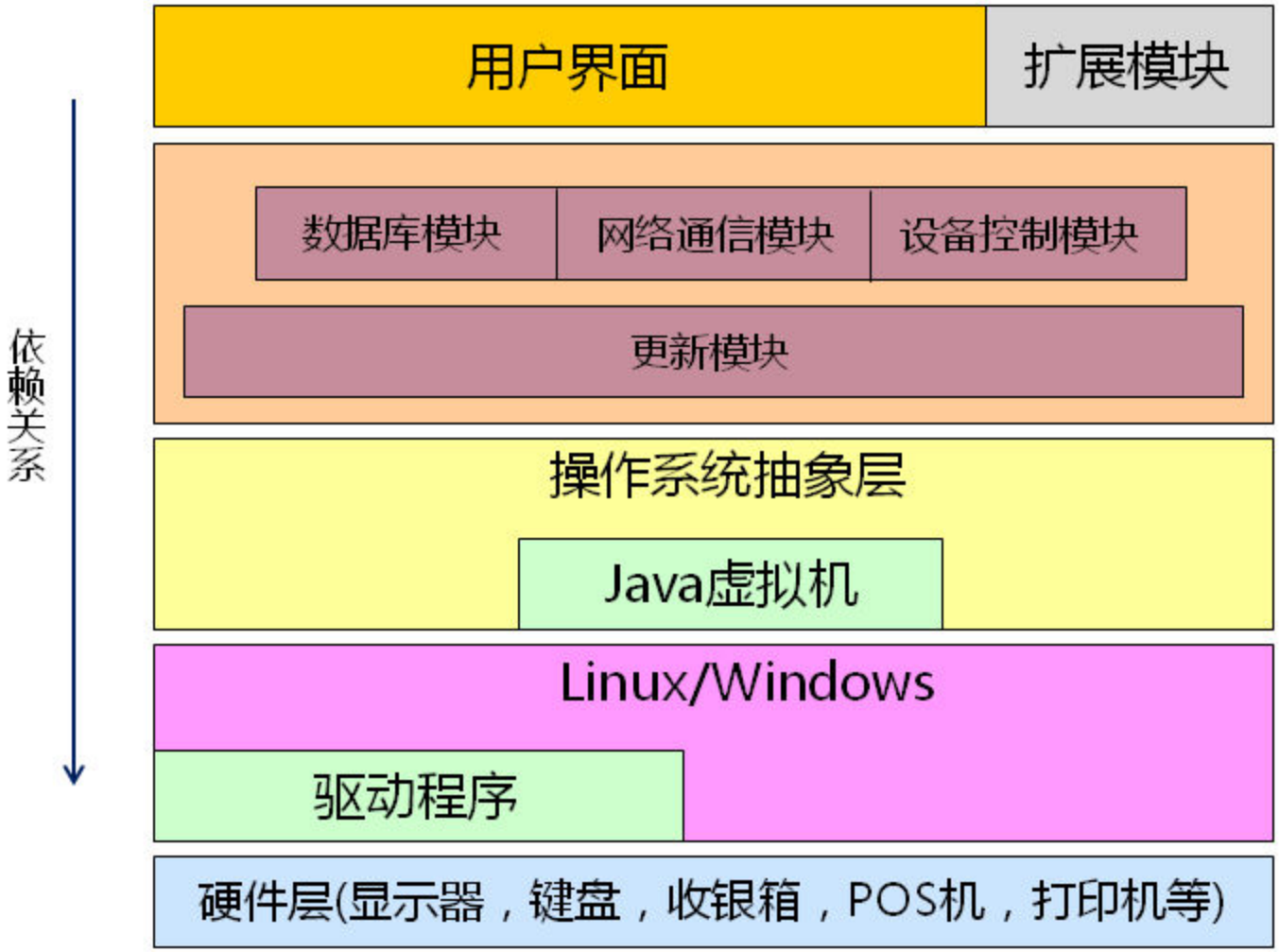
系统模块管理的设计

- 系统架构示例一



系统模块管理的设计

- 系统架构示例二



系统模块管理的设计

- 设计时需要思考的问题

如何在代码中定义模块？

如何定义的层级关系（依赖关系）？

如何确定模块的初始化顺序？

系统模块管理的设计

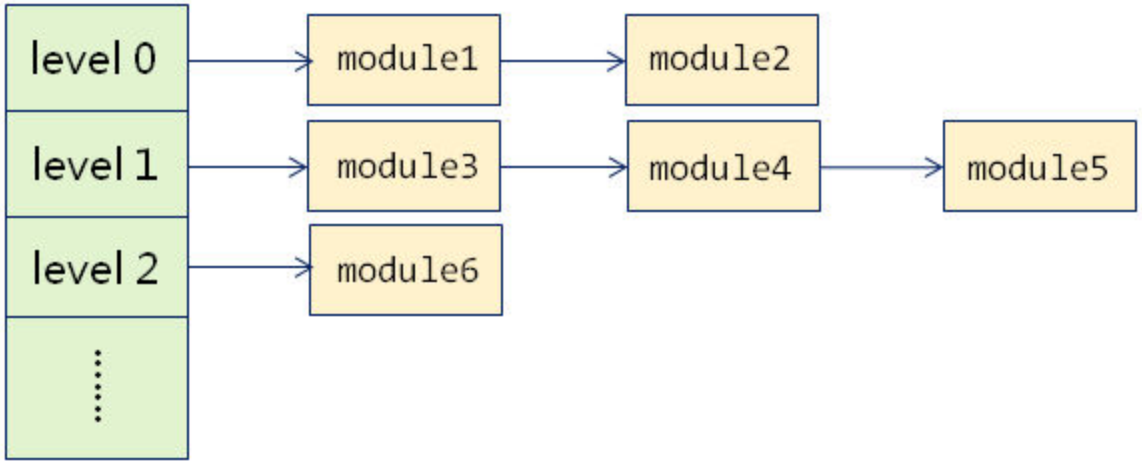
- 模块的定义

```
typedef enum {
    MODULE_MODULE,          // for module management
    MODULE_INTERRUPT,       // for interrupt management
    MODULE_DEVICE,          // for device management
    MODULE_CLOCK,           // for clock management
    MODULE_CONSOLE,         // for device of console
    MODULE_CTRLC,           // for handling Ctrl+C on Linux/Cygwin
    MODULE_FLASH,           // for device of flash
    MODULE_TIMER,           // for timer management
    MODULE_TASK,            // for task
    MODULE_SYNC,            // for task sync object management
    MODULE_SEMAPHORE,       // for semaphore management
    MODULE_MUTEX,           // for mutex management
    MODULE_QUEUE,           // for queue management
    MODULE_HEAP,            // for heap management
    MODULE_MPOOL,           // for memory pool management

    // ……

} module_t;
```

系统模块管理的设计

- 模块的描述及组织方式

```
typedef struct {
    dll_node_t  node_;                  // 链表结点
    const char *p_name_;                // 模块名
    module_callback_t  callback_;       // 模块回调函数
    bool  is_registered_;               // 注册标记
} module_init_t;
```



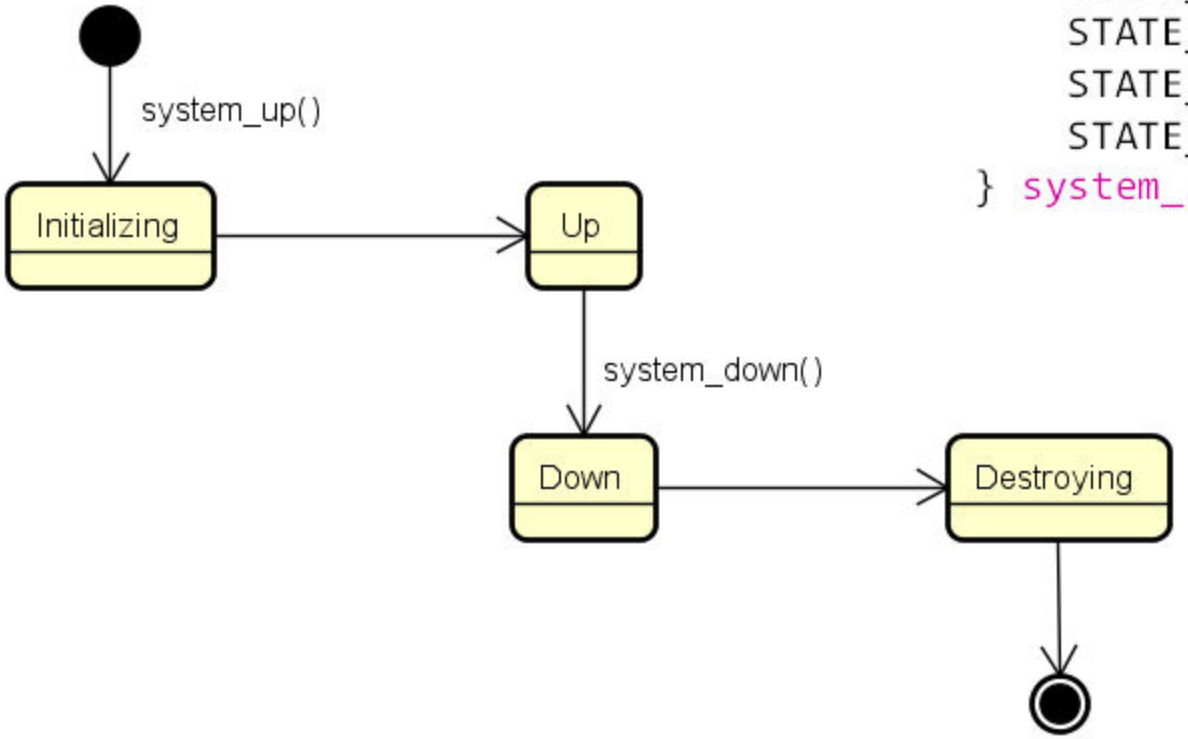
系统模块管理的设计

- 层级关系的定义

```
typedef enum {
    LEVEL_FIRST,
    // ...

    // for platform layer
    PLATFORM_LEVEL0,
    // ...
    PLATFORM_LEVEL7,

    // for framework layer
    FRAMEWORK_LEVEL0,
    // ...
    FRAMEWORK_LEVEL7,

    // for application layer
    APPLICATION_LEVEL0,
    // ...
    APPLICATION_LEVEL7,


    LEVEL_COUNT,
    LEVEL_LAST = (LEVEL_COUNT - 1)
} init_level_t;
```

系统模块管理的设计

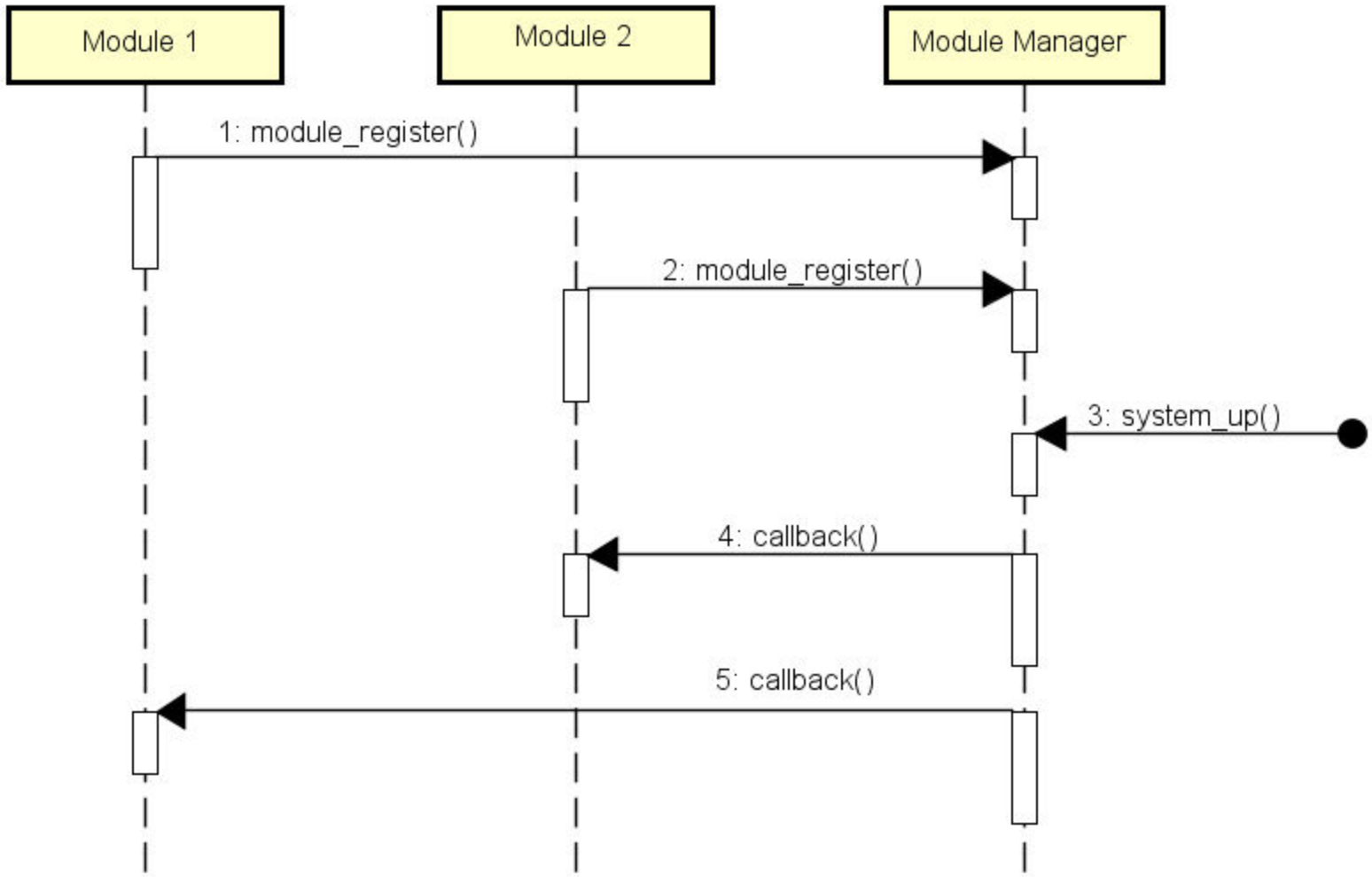
- 状态设计




```
typedef enum {
    STATE_INITIALIZING,
    STATE_UP,
    STATE_DOWN,
    STATE_DESTROYING
} system_state_t;
```

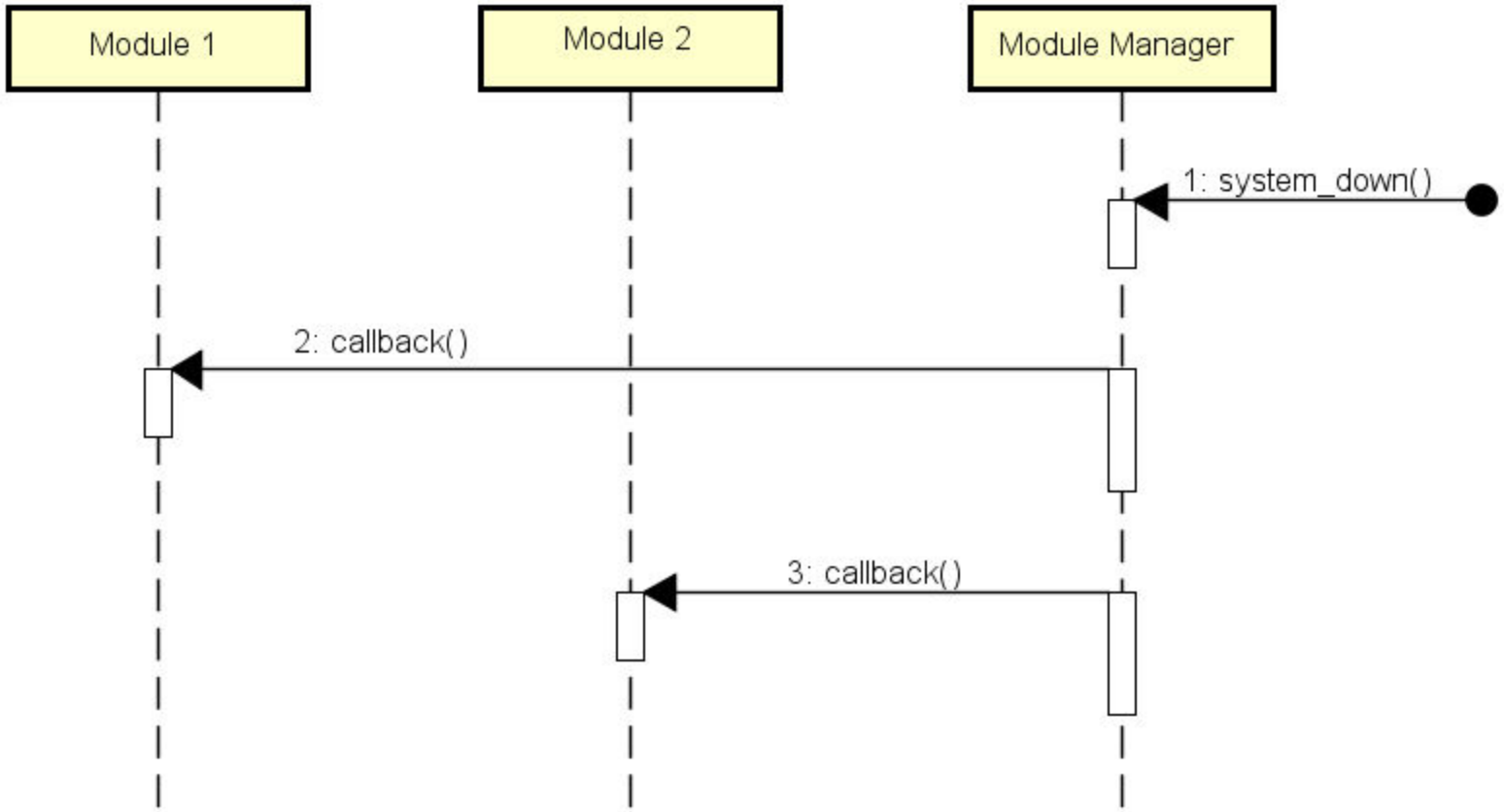
系统模块管理的设计

- 模块的初始化



系统模块管理的设计

- 模块的销毁



系统模块管理的设计

- 实现要点
  - 每一个模块对应一个 ID 和一个结构体变量（module_init_t）
  - 模块需要注册后才能被初始化（ module_register ）
  - 模块提供一个回调函数（ module_callback_t ）用于接收事件
  - 所有的模块根据层级关系组织于不同链表中
    - 同一个链表中的模块没有依赖关系
    - 整个系统从最底层（最上层）的模块开始进行初始化（销毁）

实例分析

模块的组织，初始化与销毁

源码分析

小结

- 模块设计是要遵从强内聚弱耦合的原则
- 模块之间可以相互依赖，并进行模块层级的划分
- 模块管理是为了系统中各个模块的有序启动和停止
- 模块设计时需要考虑资源的分配和释放问题

问与答

- Q / A

狄泰微信公众号

狄泰软件 D.T.SOFTWARE

在这里收获的是未来，而不只是技术！

交流群: 199546072