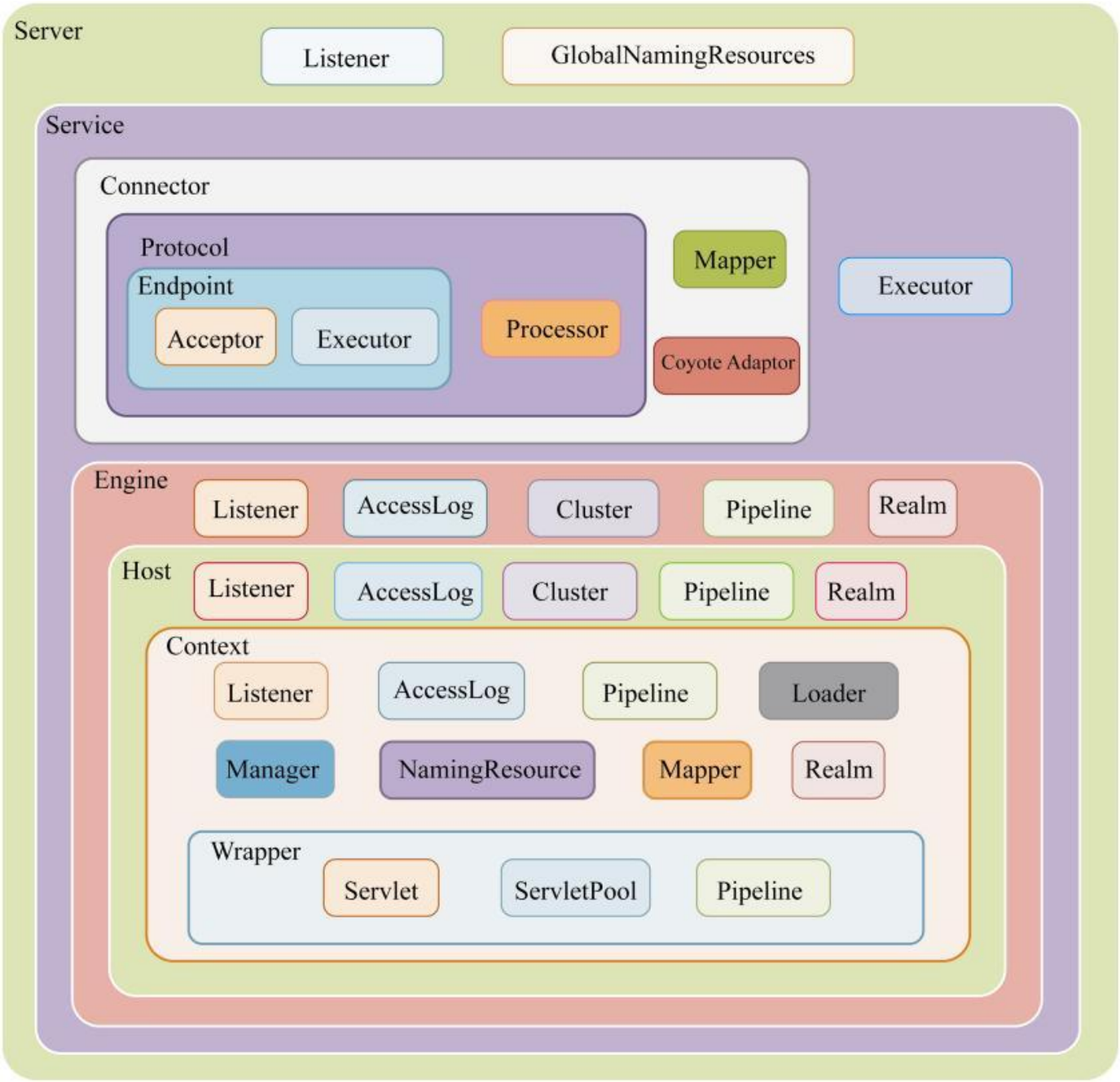
五、Tomcat的架构、生命周期、事件监听

Tomcat的架构



Tomcat生命周期

Tomcat架构是一种树状的层级管理结构，组件会有自己的父节点，也可能会有自己的孩子节点，每个节点都是组件，每个组件都有生命周期，为了管理方便，子节点的生命周期都是交由父节点来管理的。

每个组件生命周期的管理主要由一个接口org.apache.catalina.Lifecycle和一个枚举org.apache.catalina.LifecycleState来表示。

Lifecycle

org.apache.catalina.Lifecycle接口定义了组件所有执行的动作，核心的有三个：

1. init()，组件进行初始化
2. start()，启动组件
3. stop()，停止组件
4. destroy()，销毁组件
5. getState()，获取组件当前状态

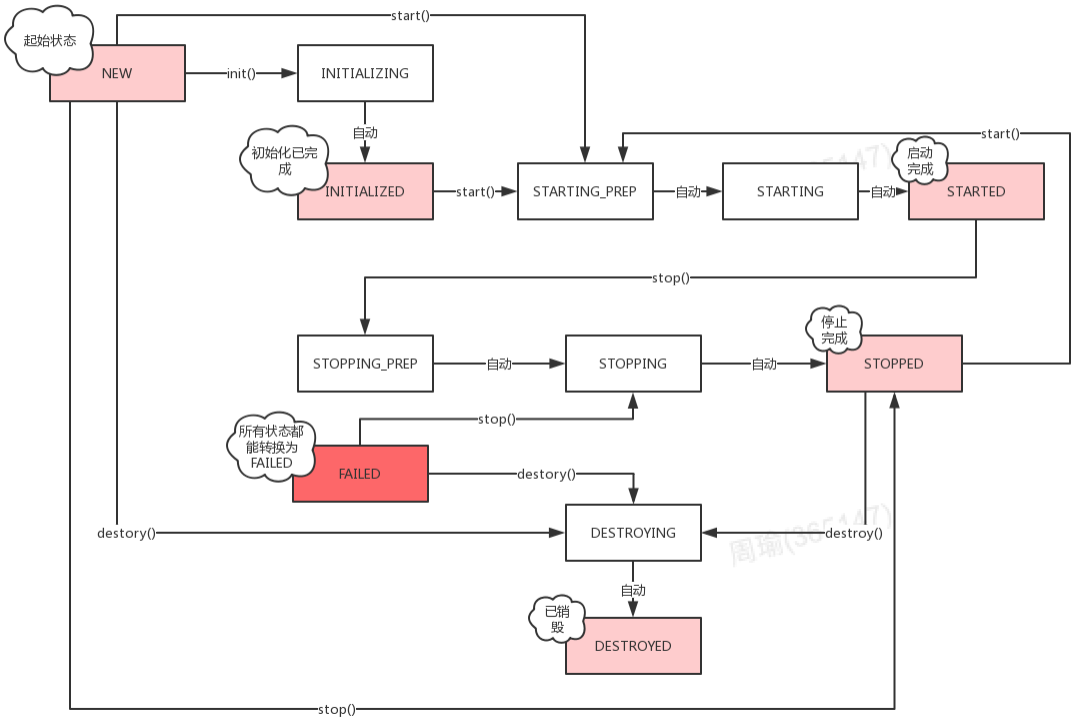
LifecycleState

org.apache.catalina.LifecycleState是一个枚举，表示组件的所有生命周期。

```
NEW(false, null),
INITIALIZING(false, Lifecycle.BEFORE_INIT_EVENT),
INITIALIZED(false, Lifecycle.AFTER_INIT_EVENT),
STARTING_PREP(false, Lifecycle.BEFORE_START_EVENT),
STARTING(true, Lifecycle.START_EVENT),
STARTED(true, Lifecycle.AFTER_START_EVENT),
STOPPING_PREP(true, Lifecycle.BEFORE_STOP_EVENT),
STOPPING(false, Lifecycle.STOP_EVENT),
STOPPED(false, Lifecycle.AFTER_STOP_EVENT),
DESTROYING(false, Lifecycle.BEFORE_DESTROY_EVENT),
DESTROYED(false, Lifecycle.AFTER_DESTROY_EVENT),
FAILED(false, null);
```

- 枚举值表示状态
- 第一个参数表示当前状态下组件可不可用
- 第二个参数表示当变为当前状态时出发相应事件

生命周期流转



1. 所有状态都能转变为FAILED
2. 一个组件在STARTING_PREP、STARTING、STARTED状态调用start()方法不会产生影响
3. 一个组件在NEW状态调用start()方法时，会先调用init()方法
4. 一个组件在STOPPING_PREP、STOPPING、STOPPED状态调用stop方法不会产生影响
5. 一个组件在NEW状态调用stop()方法是，会将状态直接改为STOPPED。当组件自己启动失败去停止时，需要将子组件也进行停止，尽管某些子组件还没有启动。
6. 其他状态相互转换都会抛异常
7. 合法的状态转换发生时都会触发相应的LifecycleEvent事件，非合法的转换不会触发事件。

Tomcat事件监听

事件触发

Tomcat中每个组件的状态会发送变化，变化的时候会抛出一些事件，Tomcat支持定义事件监听器来监听并消费这些事件。

事件执行

实现事件监听功能的类为org.apache.catalina.util.LifecycleBase。每个组件都会继承这个类。

该类中有一个属性：`List<LifecycleListener> lifecycleListeners`；该属性用来保存事件监听器，也就是说每个组件拥有一个事件监听器列表。
该类中有一个方法：

```
protected void fireLifecycleEvent(String type, Object data) {
    LifecycleEvent event = new LifecycleEvent(this, type, data);
    for (LifecycleListener listener : lifecycleListeners) {
        listener.lifecycleEvent(event);
    }
}
```

当组件的状态发生变化时，会调用fireLifecycleEvent触发事件执行。比如当Server初始化时，会调用：

```
setStateInternal(LifecycleState.STARTING_PREP, null, false);
```

该方法内部会执行：

```
String lifecycleEvent = state.getLifecycleEvent();
if (lifecycleEvent != null) {
    fireLifecycleEvent(lifecycleEvent, data);
}
```

事件监听器

程序员可以自定义事件监听器，只需实现LifecycleListener接口即可，比如：

```
class NamingContextListener implements LifecycleListene{...}
class FrameworkListener implements LifecycleListener{...}
```

定义好事件监听器后，每个组件就可以调用父类LifecycleBase中的addLifecycleListener()方法添加事件监听器到该组件的监听器列表中。

总结

虽然说是事件监听，但实际上并不是异步触发，而是主动调用事件监听器。