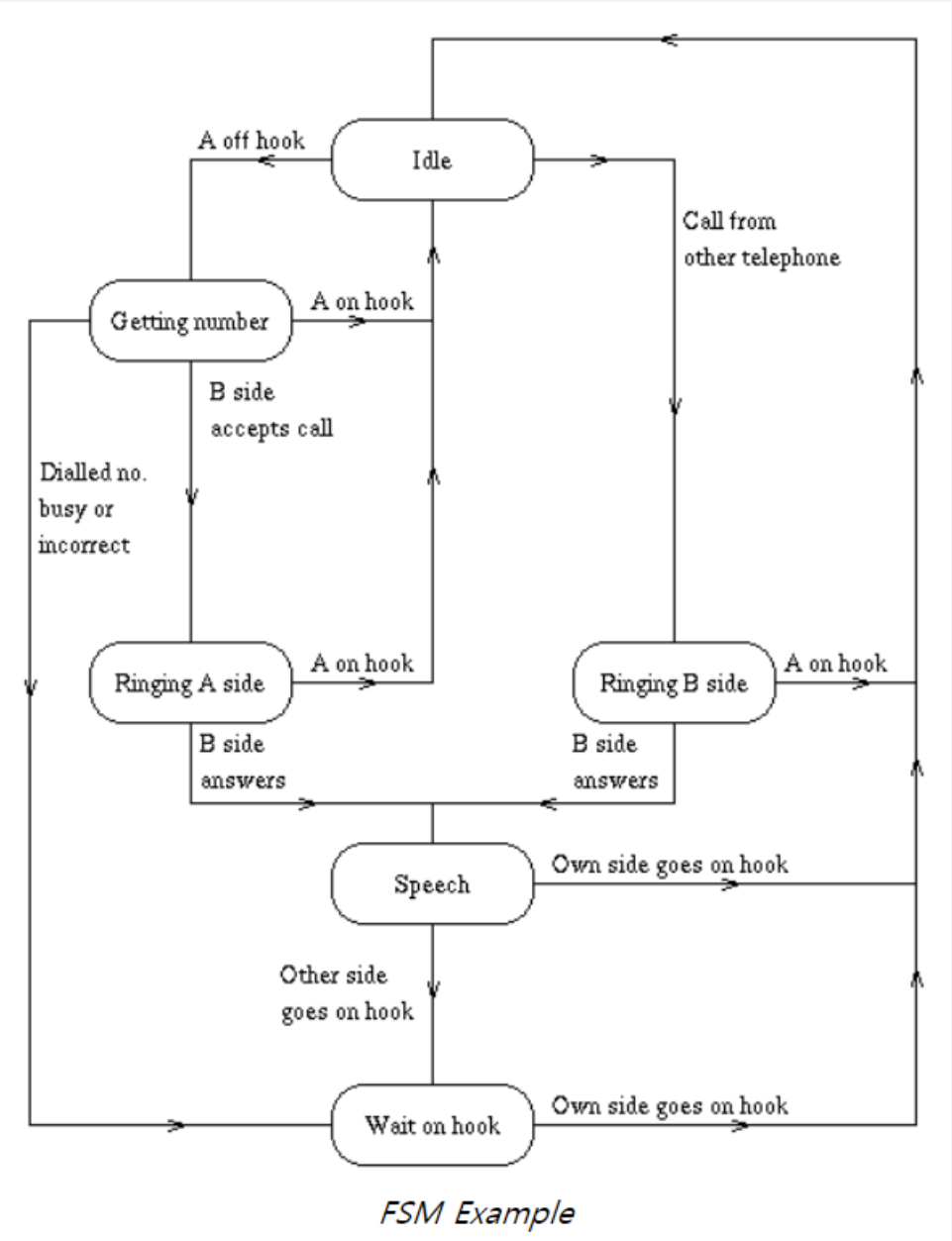
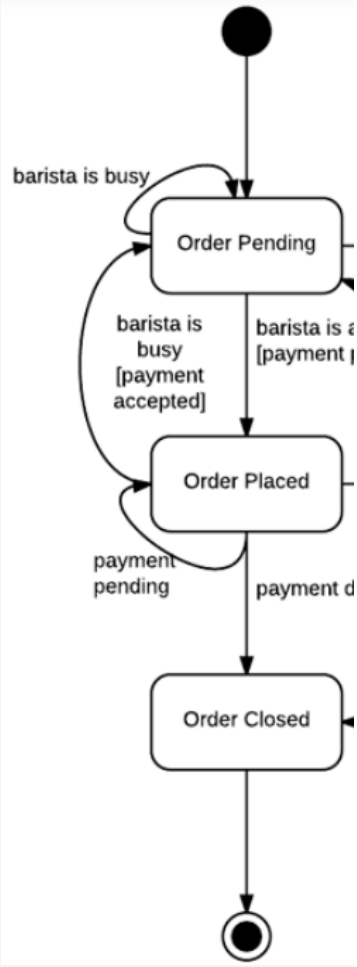
00.Finite State Machines

Actor API Become/Unbecome

AbstatractFSM .



FSM Example



.

http://erlang.org/documentation/doc-4.8.2/doc/design_principles/fsm.html

.

- , .
- .
- ( )

,

DBwrite DB .

1 . 100

100. 1 100 .

.

- .
- .
- ,
- (Fluse) .

-

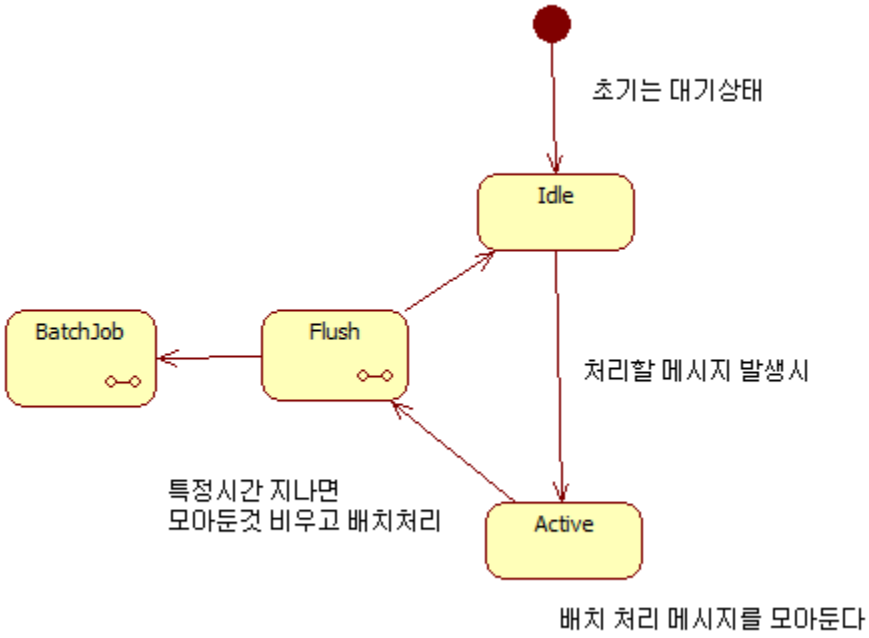
```
import akka.actor.AbstractFSM;
import akka.actor.ActorRef;
import akka.japi.pf.UnitMatch;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.io.Serializable;
import java.time.Duration;


public final class SetTarget  {
  private final ActorRef ref;

  public SetTarget(ActorRef ref) {
    this.ref = ref;
  }

  public ActorRef getRef() {
    return ref;
  }

  @Override
  public String toString() {
    return "SetTarget{" +
      "ref=" + ref +
      '}';
  }
}


public final class Queue  {
  private final Object obj;

  public Queue(Object obj) {
    this.obj = obj;
  }

  public Object getObj() {
    return obj;
  }

  @Override
  public String toString() {
    return "Queue{" +
```

```
      "obj=" + obj +
      '}';
  }
}


public final class Batch {
  private final List<Object> list;

  public Batch(List<Object> list) {
    this.list = list;
  }

  public List<Object> getList() {
    return list;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Batch batch = (Batch) o;

    return list.equals(batch.list);
  }

  @Override
  public int hashCode() {
    return list.hashCode();
  }

  @Override
  public String toString() {
    final StringBuilder builder = new StringBuilder();
    builder.append( "Batch{list=");
    list.stream().forEachOrdered(e -> { builder.append(e); builder.append(","); });
    int len = builder.length();
    builder.replace(len, len, "}");
    return builder.toString();
  }
}


public enum Flush {
  Flush
}


//states
enum State {
Idle, Active
}

//state data
interface Data {
}

enum Uninitialized implements Data {
Uninitialized
}

final class Todo implements Data {
private final ActorRef target;
private final List<Object> queue;

public Todo(ActorRef target, List<Object> queue) {
 this.target = target;
 this.queue = queue;
}
```

```
public ActorRef getTarget() {
 return target;
}

public List<Object> getQueue() {
 return queue;
}

@Override
public String toString() {
 return "Todo{" +
   "target=" + target +
   ", queue=" + queue +
   '}';
}

public Todo addElement(Object element) {
 List<Object> nQueue = new LinkedList<>(queue);
 nQueue.add(element);
 return new Todo(this.target, nQueue);
}

public Todo copy(List<Object> queue) {
 return new Todo(this.target, queue);
}

public Todo copy(ActorRef target) {
 return new Todo(target, this.queue);
}
}
```

, StateChartDiagram

.


초기는 대기상태
Idle
BatchJob
Flush
처리할 메시지 발생시
특정시간 지나면
모아둔것 비우고 배치처리
Active
배치 처리 메시지를 모아둔다

```
@Component
@Scope("prototype")
public class Buncher extends AbstractFSM<State, Data> {
        final static  com.psmon.cachedb.actors.fsm.State Idle = com.psmon.cachedb.actors.fsm.State.Idle;
        final static  com.psmon.cachedb.actors.fsm.Uninitialized Uninitialized = com.psmon.cachedb.actors.fsm.
Uninitialized.Uninitialized;
        final static public com.psmon.cachedb.actors.fsm.State Active = com.psmon.cachedb.actors.fsm.State.
Active;

        {
            startWith(Idle, Uninitialized);
            when(Idle,
              matchEvent(SetTarget.class, Uninitialized.class,
                (setTarget, uninitialized) ->
                  stay().using(new Todo(setTarget.getRef(), new LinkedList<>()))));

            onTransition(
              matchState(Active, Idle, () -> {
                // reuse this matcher
                final UnitMatch<Data> m = UnitMatch.create(
                  matchData(Todo.class,
                    todo -> todo.getTarget().tell(new Batch(todo.getQueue()), getSelf())));
                m.match(stateData());
              }).
              state(Idle, Active, () -> {/* Do something here */}));

            when(Active, Duration.ofSeconds(1L),
              matchEvent(Arrays.asList(Flush.class, StateTimeout()), Todo.class,
                (event, todo) -> goTo(Idle).using(todo.copy(new LinkedList<>()))));

            whenUnhandled(
              matchEvent(Queue.class, Todo.class,
                (queue, todo) -> goTo(Active).using(todo.addElement(queue.getObj()))).
                anyEvent((event, state) -> {
                  log().warning("received unhandled request {} in state {}/{}",
                    event, stateName(), state);
                  return stay();
                }));

            initialize();
          }
}
```

```
        @Test
        public void contextLoads() {
                ActorSystem system = context.getBean(ActorSystem.class);
                SpringExtension ext = context.getBean(SpringExtension.class);
                fsmTest(system,ext);
                TestKit.shutdownActorSystem(system , scala.concurrent.duration.Duration.apply(5, TimeUnit.
SECONDS ) ,true );
        }

        protected void fsmTest(ActorSystem system,SpringExtension ext) {
            new TestKit(system) {

                {
                final ActorRef buncher =
                  system.actorOf(ext.props("buncher"));

                final ActorRef probe = getRef();

                buncher.tell(new SetTarget(probe), probe);
                buncher.tell(new Queue(42), probe);
                buncher.tell(new Queue(43), probe);
                LinkedList<Object> list1 = new LinkedList<>();
                list1.add(42);
                list1.add(43);
                expectMsgEquals(new Batch(list1));
                buncher.tell(new Queue(44), probe);

                        buncher.tell(Flush.Flush , probe);
                buncher.tell(new Queue(45), probe);
                LinkedList<Object> list2 = new LinkedList<>();
                list2.add(44);
                expectMsgEquals(new Batch(list2));
                LinkedList<Object> list3 = new LinkedList<>();
                list3.add(45);
                expectMsgEquals(new Batch(list3));
                system.stop(buncher);
              }};
        }
```

,

, .

DBWrite

```
@Component("DBWriteActor")
@Scope("prototype")
public class DBWriteActor extends AbstractActor{
    private final LoggingAdapter log = Logging.getLogger(getContext().system(), "DBWriteActor");

        @Autowired
        ItemBuyLogRepository itemBuyLogRepository;

    @Override
    public Receive createReceive() {
      return receiveBuilder()
        .match(  com.psmon.cachedb.actors.fsm.Batch.class , s -> {
          log.info("Received ItemBuyLog message: {}", s.toString()  );
          List<ItemBuyLog> insertList = new ArrayList<>();
          s.getList().forEach( item-> {
                  ItemBuyLog itemLog = (ItemBuyLog)item;
                  insertList.add(itemLog);
          });
          itemBuyLogRepository.save( insertList );
        })
        .matchAny(o -> log.info("received unknown message - {}", o.getClass().getName()  ))
        .build();
    }
}
```

> *DB Row*
>
> *JPA .*
>
> *JPA(ORM) DB DB.*
>
> *: 02-DBHANDLE with JPA*

```
        protected void fsmDBWriteTest(ActorSystem system,SpringExtension ext) {
            new TestKit(system) {
                {
                final ActorRef buncher =
                  system.actorOf(ext.props("buncher"));

                final ActorRef dbWriter =
                            system.actorOf(ext.props("DBWriteActor"));

                final ActorRef probe = getRef();

                buncher.tell(new SetTarget(dbWriter), dbWriter);

                        for(int i=0;i<200;i++) {
                                String buyTime = String.format("2018-%02d-%02d", i%10+1,i%
20+1);
                                UserInfo buyUser = userInfoRepository.findAll().get(i%50);
                                GameItem buyItem = gameItemRepository.findAll().get(i%
50);
                                ItemBuyLog addBuyLog = new ItemBuyLog(buyTime, buyItem,
buyUser);
                                buncher.tell(new Queue(addBuyLog), dbWriter);
                        }

                        buncher.tell(Flush.Flush , dbWriter);

                         system.stop(buncher);
                }};
        }
```

> *Flush Row List DB .*
>
> *, Flush .*