모듈화와 액터

한병일
lucismh@gmail.com

무엇을 하려고 ?

1. 콘텐츠 간에 의존성과 결합도를 낮추고 싶다

2. 로컬에서 원격으로 바꾸더라도 작동되게

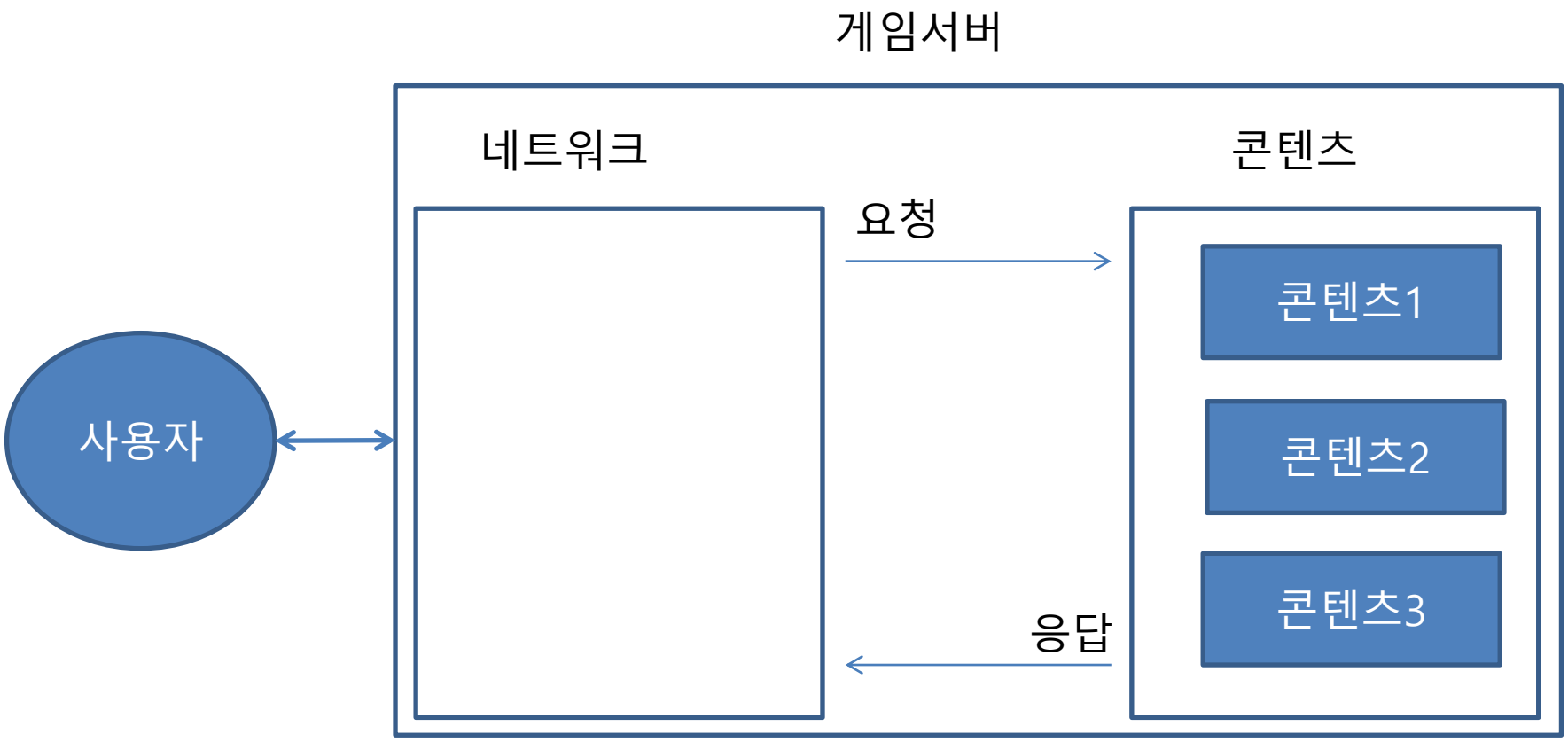
모듈화

: 독립적으로 기능을 수행 하도록 짜여진 코드의 묶음

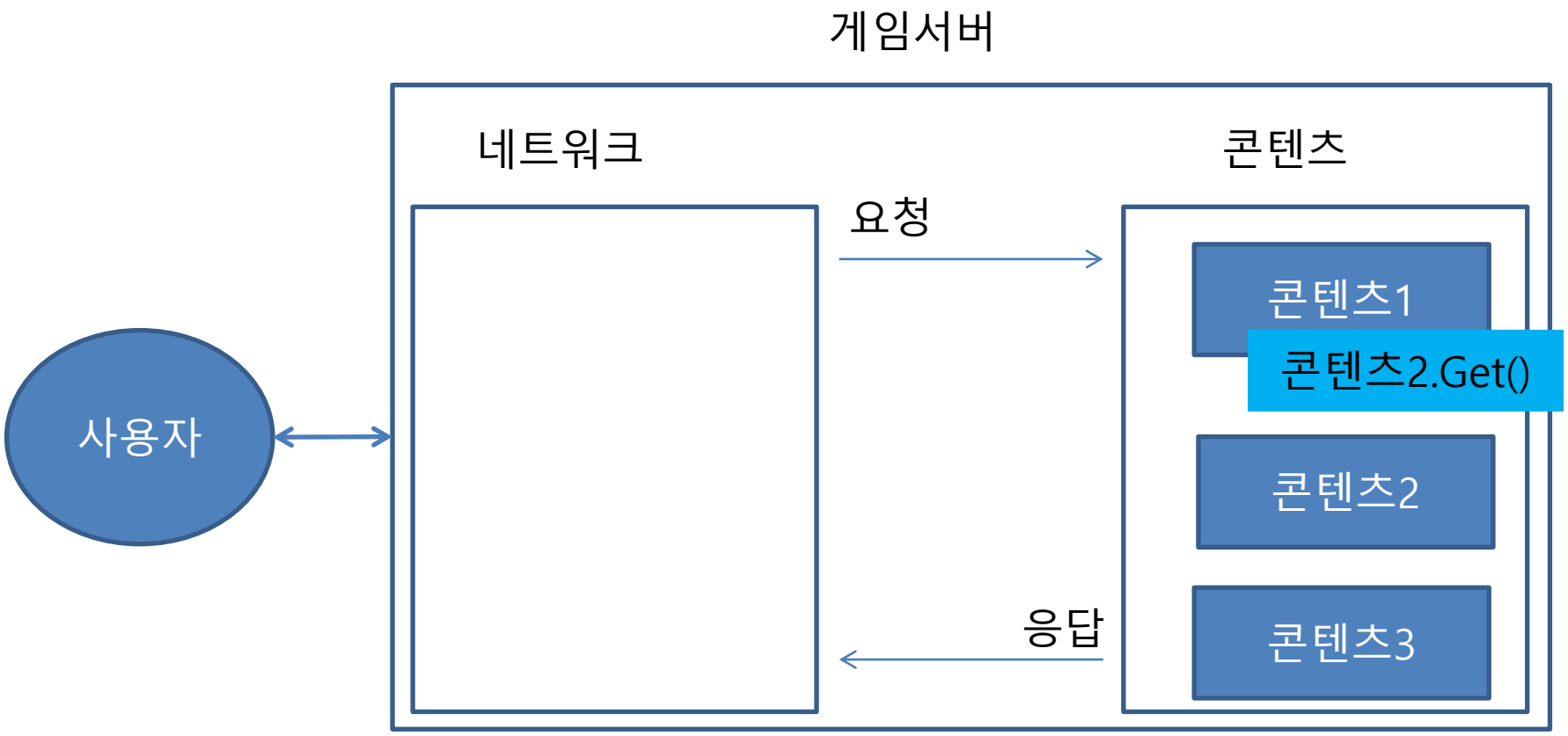
액터, 액터 모델 ? 액티브 오브젝트 ?

: 능동적 객체
: 외부에서 직접적으로 객체에 접근을 못하게 하며,
: 자원을 공유 하지 않는다.

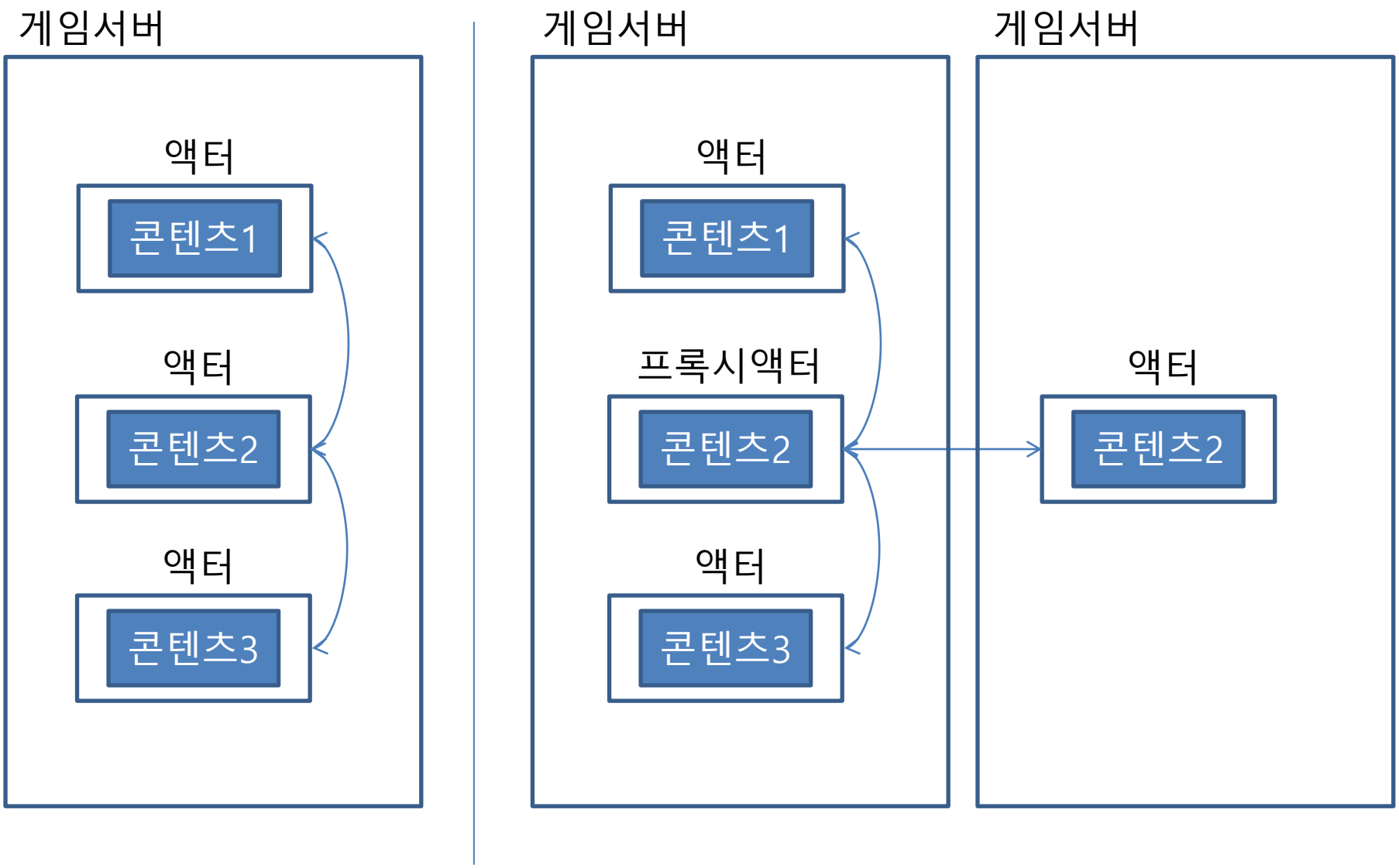
구조



구조



구조

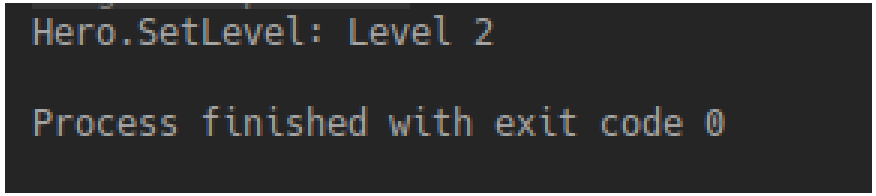

게임서버
액터
콘텐츠1
액터
콘텐츠2
액터
콘텐츠3
게임서버
액터
콘텐츠1
프록시액터
콘텐츠2
액터
콘텐츠3
게임서버
액터
콘텐츠2

샘플 코드

```
type Hero struct {
        id     int
        level int
        nodeName string
}

func (h *Hero) GetLevel() int {
        return h.level
}

func (h *Hero) SetLevel(level int) {
        h.level = level
        fmt.Printf("Hero.SetLevel: Level %d ₩n", level)
}

func (h *Hero) GetNodeName() string {
        return h.nodeName
}
```

```
type Training struct {
        nodeName string
}

func (t *Training) TrainingHero(hero *Hero) {
        level := hero.GetLevel()
        level++
        hero.SetLevel(level)
}

func (t *Training) Do(level int) int {
        newLevel := level+1
        fmt.Printf("Training.Do: oldLevel %d, nowLevel %d ₩n", level, newLevel)
        return newLevel
}

func (t *Training) GetNodeName() string {
        return t.nodeName
}
```

```
func main() {
	hero := samplestruct.NewHero(1, 1, "Hero")
	training := samplestruct.NewTraining("Training")

	training.TrainingHero(hero)
}

/*func (t *Training) TrainingHero(hero *Hero) {
		level := hero.GetLevel()
		level++
		hero.SetLevel(level)
}*/
```



```
func main() {

        hero := samplestruct.NewHero(1, 1, "Hero")
        training := samplestruct.NewTraining("Training")

        heroAID := actor.StartActor(hero)
        trainingAID := actor.StartActor(training)

        //GetLevel() int
        results, _ := actor.Call(heroAID, (*samplestruct.Hero).GetLevel)

        // Do(level int) int
        results, _ = actor.Call(trainingAID, (*samplestruct.Training).Do, results[0])

        //SetLevel(level int)
        _, _ = actor.Call(heroAID, (*samplestruct.Hero).SetLevel, results[0])
}
```

```
Training.Do: oldLevel 1, nowLevel 2
Hero.SetLevel: Level 2
```

```
/*actor.Call(trainingAID, "Do", results[0])*/
/*actor.Call(trainingAID, "Do", heroAID, "SetLevel", results[0])*/
```

```
func main() {

        ctxMain, _ := context.WithCancel(context.Background())

        actor.StartWebServer(ctxMain,"9999")

        training := samplestruct.NewTraining("Training")

        _ = actor.StartActor(training)
}
```

```
func main() {

        hero := samplestruct.NewHero(1, 1, "Hero")
        //training := samplestruct.NewTraining("Training")

        heroAID := actor.StartActor(hero)
        //trainingAID := actor.StartActor(training)

        trainingAID := actor.StartWebActor(nil, "Training", "http://localhost:9999")

        //GetLevel() int
        results, _ := actor.Call(heroAID, (*samplestruct.Hero).GetLevel)

        // Do(level int) int
        results, _ = actor.Call(trainingAID, (*samplestruct.Training).Do, results[0])

        //SetLevel(level int)
        _, _ = actor.Call(heroAID, (*samplestruct.Hero).SetLevel, results[0])
}
```

```
func main() {
        ...

        // Do(level int) int
        results, _ = actor.Call(trainingAID, (*samplestruct.Training).Do, results[0])
```

```
Actor Server Start :9999
Training.Do: oldLevel 1, nowLevel 2
```

```
        //SetLevel(level int)
        _, _ = actor.Call(heroAID, (*samplestruct.Hero).SetLevel, results[0])
}
```

```
Hero.SetLevel: Level 2
```

샘플 액터 라이브러리 설명

```
type AID struct {
        ActorID  uint64
        NodeName string
}

type IActorReceiver interface {
        GetNodeName() string
}

type iActor interface {
        IActorReceiver
        getReceiver() reflect.Value
        getAID() *AID
        setAID(aid *AID)
        call(function interface{}, args ...interface{}) ([]interface{}, error)
}
```

샘플 액터 라이브러리 설명

```
type Actor struct {
        aid         *AID
        receiver    reflect.Value
        queue       *list.List
        inChan      chan *ActorCall
}
func (a *Actor) call(function interface{}, args ...interface{}) ([]interface{}, error) {
        done := make(chan *ActorCall, 0)
        a.inChan <- a.makeActorCall(done, function, args...)
        actorCall, _ := <-done
        return actorCall.GetResults()
}
func (a *Actor) process(actorCall *ActorCall) {
                                //reflect.Value.Call
        actorCall.Results = actorCall.Function.Call(actorCall.Args)
        if actorCall.Done != nil {
                actorCall.Done <- actorCall
        }
}
```

샘플 액터 라이브러리 설명

```
func (a *WebActor) call(function interface{}, args ...interface{}) ([]interface{}, error) {
        req := defaultActorSystem.createRequest(a, function, args...)

        var buffer bytes.Buffer
        enc := gob.NewEncoder(&buffer)
        enc.Encode(req)

        httpReq, err := http.NewRequest("POST", a.address+"/Call", &buffer)
        if err != nil {
                return nil, err
        }
        .....
}
```