테스트 커버리지

- 테스트 범위를 측정하는 테트 품질 측정 기준이며, 테스트의 정확성과 신뢰성을 향상시키는 역할
- 테스트 케이스의 부족한 부분을 파악하고 추가적인 테스트 케이스를 개발할 수 있음

테스트 커버리지의 종류

1. 구문 커버리지 (Statement Coverage)

- 모든 구문에 대해 한 번 이상 수행하는 테스트 커버리지
- `테스트 케이스 집합에 의해 실행된 문장의 수 / (전체 실행 가능한 프로그램 문장의 수) * 100%`

```
if (x > 0) {
        y = x * 2;
} else {
        y = x / 2;
}
```

- x의 값을 3으로 설정한 경우 -> 첫 번째 구문이 True가 되어 두 번째 구문이 실행
- x의 값을 -2 또는 0으로 설정한 경우 -> 첫 번째 구문이 False가 되어 네 번째 구문이 실행

이처럼 모든 구문을 한 번씩 실행하는 것이 구문 커버리지를 충족하는 것이다.

2. 결정 커버리지 (Decision Coverage)

- 결정 포인트 내의 전체 조건식이 적어도 한 번은 참(T)과 거짓(F)의 결과를 수행하는 테스트 커버리지
- 분기 커버리지(Branch Coverage)라고도 함
- `(테스트 케이스 집합에 의해 실행된 결정의 결과 수 / 전체 프로그램 결과 수) * 100%`

```
if (x > 1 || y > 3) {
        z = z * x
} elif (x < 2 && y > 1) {
        z = z + 1
}
```

테스트 케이스	x > 1 || y > 3 (A 분기문)	x < 2 && y > 1 (B 분기문)
x=1.5, y=2, z=2	T	T
x=3, y=2, z=1	T	F
x=1, y=3, z=3	F	F

첫 번째와 두 번째 테스트 케이스만 수행하면 B 분기만 결정 커버리지를 만족하고, A 분기문의 결정 커버리지는 만족하지 못한다. 따라서 세 번째 테스트 케이스까지 모두 수행해야지 결정 커버리지를 100% 만족하게 된다.

3. 조건 커버리지 (Condition Coverage)

- 결정 포인트 내의 개별 조건식이 적어도 한 번은 참(T)과 거짓(F)의 결과를 수행하는 테스트 커버리지
- `(테스트 케이스 집합에 의해 실행된 조건의 결과 수 / 전체 프로그램 조건의 결과 수) * 100%`

```
if (x > 1 || y > 3) {
        z = z * x
} elif (x < 2 && y > 1) {
        z = z + 1
}
```

테스트 케이스	x > 1 (A 분기문)	y > 3 (A 분기문)	x < 2 (B 분기문)	y > 1 (B 분기문)
x=3, y=0.5, z=2	T	F	F	F
x=1, y=3, z=3	F	F	T	T
x=0.5, y=4, z=1	F	T	T	T

첫 번째, 두 번째 테스트 케이스까지 수행할 경우 `y > 3`이 True가 되는 경우를 테스트하지 못한다. 조건 커버리지의 경우 개별 조건식이 적어도 한 번은 참(T)과 거짓(F)의 결과가 되는 테스트 케이스를 가져야하기 때문에, 세 번째 테스트 케이스까지 수행해야지 조건 커버리지를 100% 만족하게 된다.

커버리지 측정

내가 작성한 테스트 코드의 커버리지가 몇 %인지 확인하기 위해서 `JaCoCo` 플러그인을 사용해볼 수 있다.

Configuration

코드 커버리지 계산을 원하는 프로젝트에 다음과 같이 `JaCoCo` 플러그인을 적용해준다.

```
plugins {
    id 'jacoco'
}
```

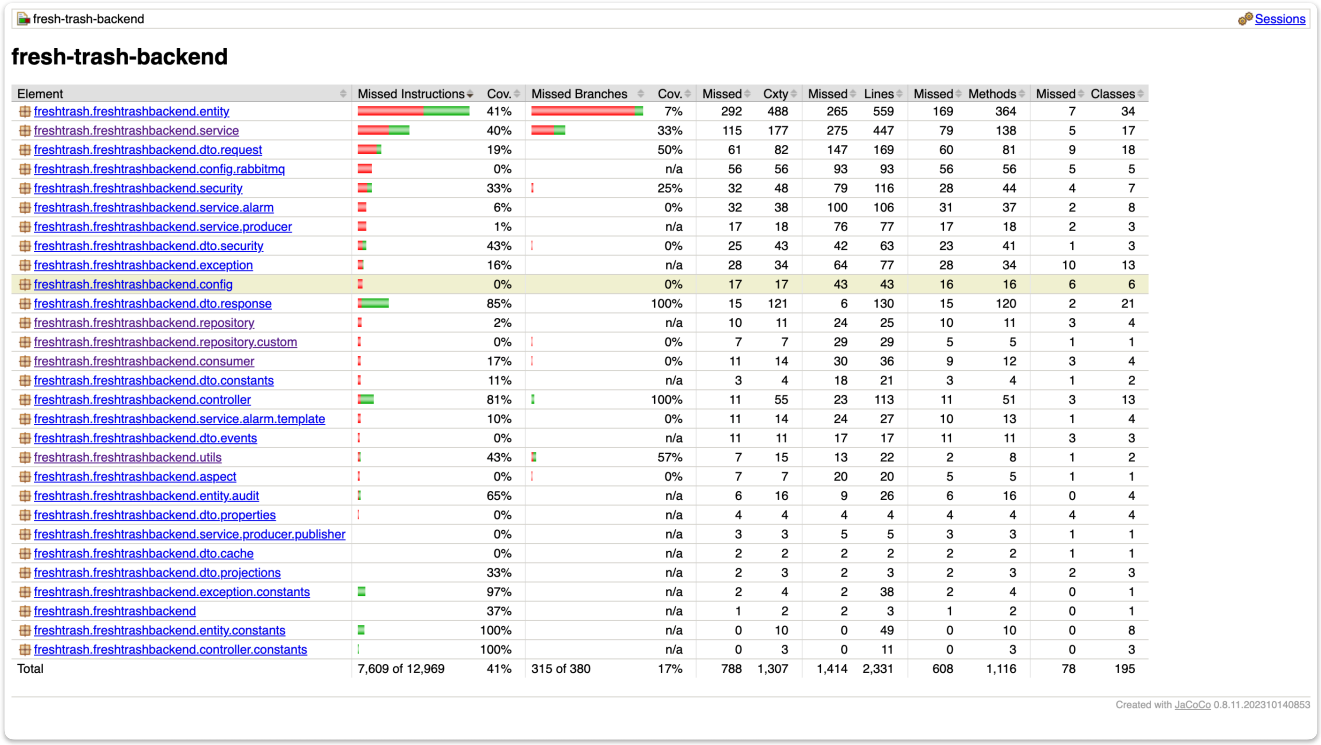
그리고 버전과 report가 생성될 directory를 지정해준다. directory를 따로 지정해주지 않으면 `layout.buildDirectory.dir("reports/jacoco")`으로 설정된다.

```
jacoco {
    toolVersion = "0.8.12"
    reportsDirectory = layout.buildDirectory.dir('customJacocoReportDir')
}
```

report는 XML, CSV, HTML 형식으로 생성할 수 있고 아래처럼 어떤 형식으로 생성할 것인지, 어떤 directory에 생성할 것인지 설정할 수 있다.

```
jacocoTestReport {
    reports {
        xml.required = false
        csv.required = false
        html.outputLocation = layout.buildDirectory.dir('jacocoHtml')
    }
}
```

여기까지 설정해주고 빌드를 해주면 `jacocoHtml`에 `index.html`이 생성되고 열어보면 다음과 같이 각 패키지별로 커버리지를 확인할 수 있다.


fresh-trash-backend

Sessions

fresh-trash-backend

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
freshtrash.freshtrashbackend.entity		41%		7%	292	488	265	559	169	364	7	34
freshtrash.freshtrashbackend.service		40%		33%	115	177	275	447	79	138	5	17
freshtrash.freshtrashbackend.dto.request		19%		50%	61	82	147	169	60	81	9	18
freshtrash.freshtrashbackend.config.rabbitmq		0%		n/a	56	56	93	93	56	56	5	5
freshtrash.freshtrashbackend.security		33%		25%	32	48	79	116	28	44	4	7
freshtrash.freshtrashbackend.service.alarm		6%		0%	32	38	100	106	31	37	2	8
freshtrash.freshtrashbackend.service.producer		1%		n/a	17	18	76	77	17	18	2	3
freshtrash.freshtrashbackend.dto.security		43%		0%	25	43	42	63	23	41	1	3
freshtrash.freshtrashbackend.exception		16%		n/a	28	34	64	77	28	34	10	13
freshtrash.freshtrashbackend.config		0%		0%	17	17	43	43	16	16	6	6
freshtrash.freshtrashbackend.dto.response		85%		100%	15	121	6	130	15	120	2	21
freshtrash.freshtrashbackend.repository		2%		n/a	10	11	24	25	10	11	3	4
freshtrash.freshtrashbackend.repository.custom		0%		0%	7	7	29	29	5	5	1	1
freshtrash.freshtrashbackend.consumer		17%		0%	11	14	30	36	9	12	3	4
freshtrash.freshtrashbackend.dto.constants		11%		n/a	3	4	18	21	3	4	1	2
freshtrash.freshtrashbackend.controller		81%		100%	11	55	23	113	11	51	3	13
freshtrash.freshtrashbackend.service.alarm.template		10%		0%	11	14	24	27	10	13	1	4
freshtrash.freshtrashbackend.dto.events		0%		n/a	11	11	17	17	11	11	3	3
freshtrash.freshtrashbackend.utils		43%		57%	7	15	13	22	2	8	1	2
freshtrash.freshtrashbackend.aspect		0%		0%	7	7	20	20	5	5	1	1
freshtrash.freshtrashbackend.entity.audit		65%		n/a	6	16	9	26	6	16	0	4
freshtrash.freshtrashbackend.dto.properties		0%		n/a	4	4	4	4	4	4	4	4
freshtrash.freshtrashbackend.service.producer.publisher		0%		n/a	3	3	5	5	3	3	1	1
freshtrash.freshtrashbackend.dto.cache		0%		n/a	2	2	2	2	2	2	1	1
freshtrash.freshtrashbackend.dto.projections		33%		n/a	2	3	2	3	2	3	2	3
freshtrash.freshtrashbackend.exception.constants		97%		n/a	2	4	2	38	2	4	0	1
freshtrash.freshtrashbackend		37%		n/a	1	2	2	3	1	2	0	1
freshtrash.freshtrashbackend.entity.constants		100%		n/a	0	10	0	49	0	10	0	8
freshtrash.freshtrashbackend.controller.constants		100%		n/a	0	3	0	11	0	3	0	3
Total	7,609 of 12,969	41%	315 of 380	17%	788	1,307	1,414	2,331	608	1,116	78	195

Created with JaCoCo 0.8.11.202310140853

Coverage Counter

위 표에서 Instruction, Branch 등 각 칼럼은 커버리지 metric을 의미한다. `JaCoCo` 공식 문서에서는 `Coverage Counter`라고 하는데, 각각 어떤 부분을 측정하는 것을 의미하는지 살펴보자.

- Instructions
  - 가장 기본적인 커버리지 metric으로 자바 바이트코드의 명령어 단위로 커버리지를 측정
  - 코드가 얼마나 실행되었는지를 나타냄

- Branches
  - 조건문(if, switch 등)의 각 분기가 테스트되었는지를 측정
- Lines
  - 코드의 각 라인이 얼마나 실행되었는지를 측정
  - 하나의 라인에 여러 명령어가 있을 수 있기 때문에 이 경우 라인 커버리지와 명령어 커버리지가 다를 수 있음
- Methods
  - 메소드 단위로 커버리지를 측정
- Classes
  - 클래스 단위로 커버리지를 측정
- Cyclomatic Complexity(Cxty)
  - 소프트웨어의 복잡도를 측정하는 metric
  - Cxty 값이 노을수록 더 많은 테스트 케이스가 필요

커버리지 제외

DTO 객체, Exception 등 커버리지를 측정하지 않아도되는 대상을 지정하여 제외시켜줄 수 있다.

```
jacocoTestReport {
    reports {
        xml.required = false
        csv.required = false
        html.outputLocation = layout.buildDirectory.dir('jacocoHtml')
    }

    afterEvaluate {
        classDirectories.setFrom(files(classDirectories.files.collect {
            fileTree(dir: it, exclude: [
                    "**/config/*",
                    "**/dto/*",
                    "**/exception/*",
                    "**/security/*",
                    "**/entity/*",
                    "**/constants/*",
            ])
        }))
    }
}
```

jacocoTestCoverageVerification

설정한 규칙에 따라 특정 커버리지를 만족하는지 확인할 수 있다. 만족하지 않는다면 빌드에 실패한다.

- element: BUNDLE, PACKAGE, CLASS, SOURCEFILE, METHOD
- counter: INSTRUCTION, LINE, BRANCH, COMPLEXITY, METHOD, CLASS
- value: TOTALCOUNT, COVEREDCOUNT, MISSEDCOUNT, COVEREDRATIO, MISSEDRATIO

```
jacocoTestCoverageVerification {
    violationRules {
        rule {
            enabled = true
            element = 'CLASS'

            limit {
                counter = 'INSTRUCTION'
                value = 'COVEREDRATIO'
                minimum = 0.8
            }
        }
        }
}
```

test 실행 후 항상 report

테스트가 끝나고 항상 jacoco를 실행시키려면 다음과 같이 설정해준다.

```
test {
    finalizedBy jacocoTestReport
}

jacocoTestReport {
    dependsOn test
    ...
}
```

`finalizedBy`로 테스트가 실행되고 `jacocoTestReport` task가 실행되도록 해준 것이다.

Appendix

Cyclomatic Complexity

공식적으로 Cyclomatic Complexity 계산식은 `M = E - N + 2`가 된다.

- E: 그래프의 엣지 수 (Edges)를 의미
- N: 그래프의 노드 수 (Nodes)를 의미

`JaCoCo`는 E를 branch 수, N을 결정 포인트(desision points)로 동일하게 보며 `M = B(branch) - D(decision point) + 1` 로 계산한다고 한다.

[reference]

- https://docs.gradle.org/current/userguide/jacoco_plugin.html#sec:jacoco_report_configuration
- https://hello-judy-world.tistory.com/213
- https://groups.google.com/g/jacoco/c/wtse-rTNIoI
- https://www.jacoco.org/jacoco/trunk/doc/counters.html
- https://www.eclemma.org/jacoco/trunk/doc/check-mojo.html
- https://about.codecov.io/blog/mutation-testing-how-to-ensure-code-coverage-isnt-a-vanity-metric/