ThoughtWorks®

OO Training

MOCKS & FOOTBALL GAME

周颖　(ybzhou@thoughtworks.com)
郑培真（pzzheng@thoughtworks.com)

TEST DOUBLES

- 定义
  - 测试替身：做了简化的，用于在测试中替换真实对象的对象统称
- 分类
  - Dummy
  - Fake
  - Stubs
  - Mocks
  - Spies
- 使用目的
  - 性能问题：真实实现包含巨量运算以及较慢的算法，影响测试性能
  - 场景不易复现：某些组件的行为与特殊场景有关：例如竞态条件、网络异常等
  - 非确定性：某些组件与物理环境有交互，如传感器等
  - 未实现：目前依赖的组件还未实现

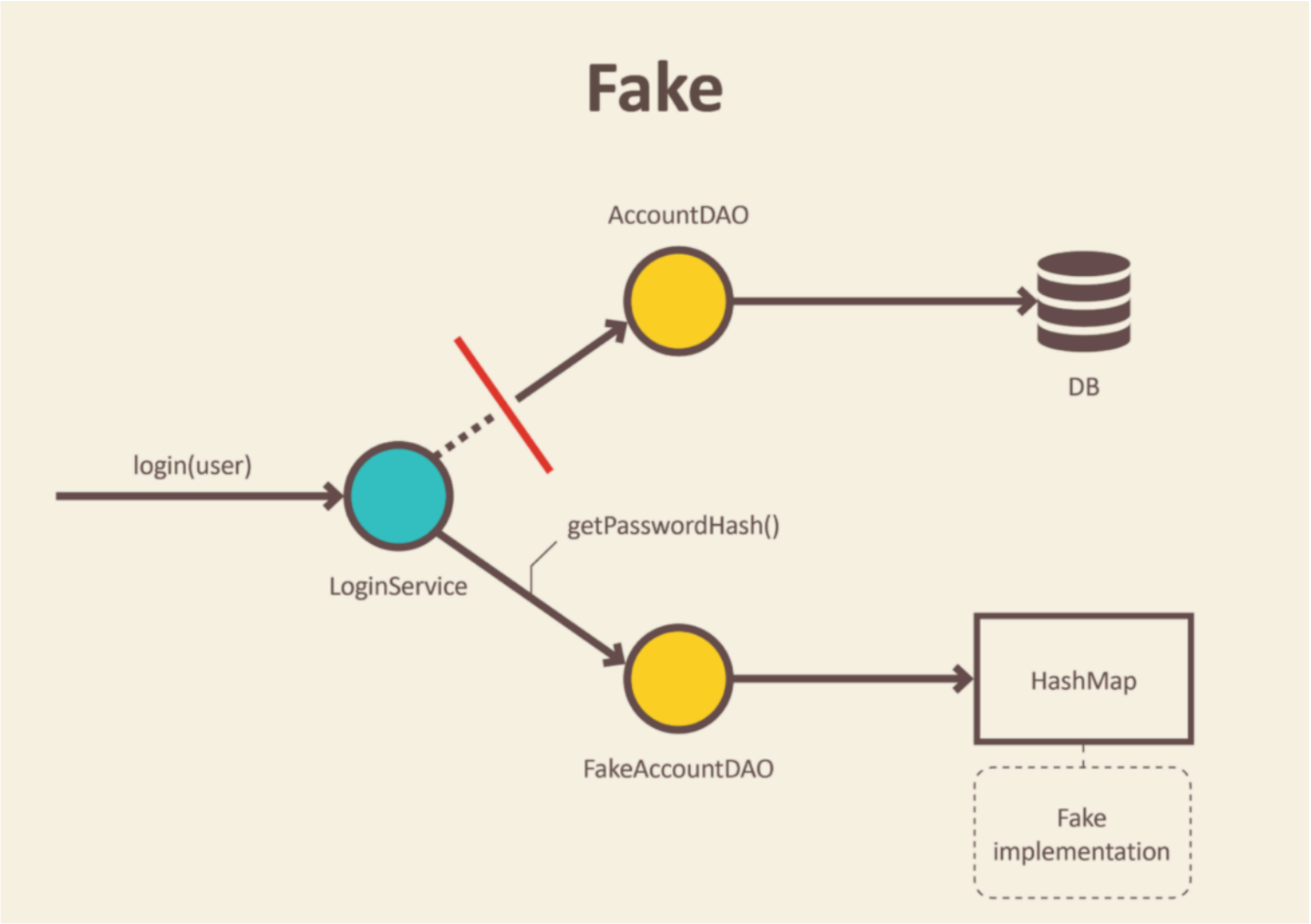
DUMMY

- 用于传递给调用者但永远不会被真实使用，常见于填充参数列表

```
@Test
public void test_how_many_customer_serviced() {
  Customer dummyCustomer = new Customer("name", "male");
  DriverSerice service = new DriverService();
  service.take(dummyCustomer);
  assertThat(service.getCountOfCustomer(), is(1));
}
```

FAKE

- 包含了生产环境下具体实现的简化版本对象，如内存数据库



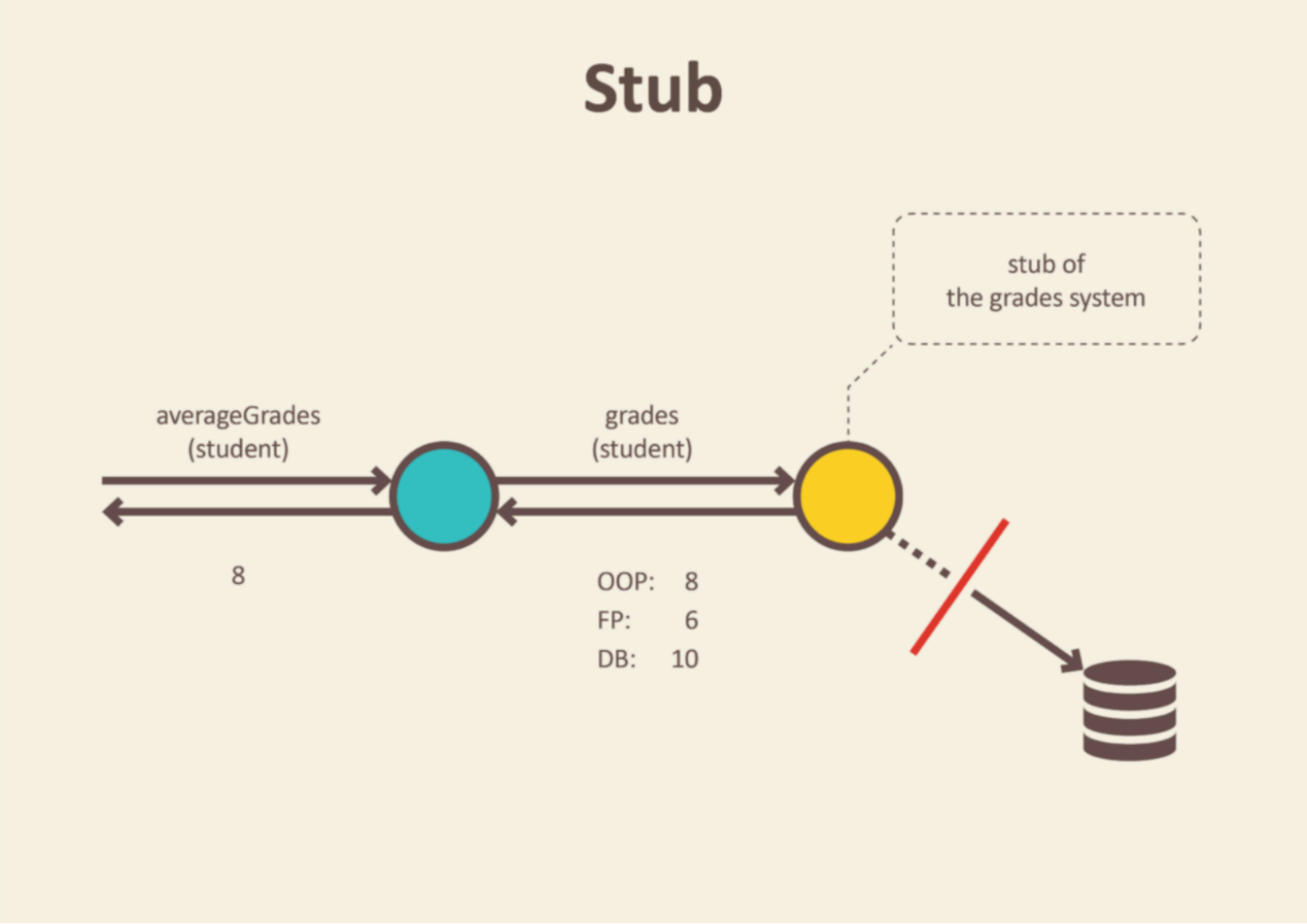
- 模拟 DB 的 HashMap 实现
- 可以用于集成测试
- 无需启动真实数据库
- 提高测试性能

FAKE

```
1
2   @Profile("transient")
3   public class FakeAccountRepository implements AccountRepository {
4
5           Map<User, Account> accounts = new HashMap<>();
6
7           public FakeAccountRepository() {
8                   this.accounts.put(new User("john@bmail.com"), new UserAccount());
9                   this.accounts.put(new User("boby@bmail.com"), new AdminAccount());
10          }
11
12          String getPasswordHash(User user) {
13                  return accounts.get(user).getPasswordHash();
14          }
15  }
```

STUBS

- 包含了预定义好的数据并且在测试时返回给调用者的对象，常用于不希望返回真实数据或造成其他副作用的场景



- 构造数据库数据并返回
- 用于单元测试
- 隔离数据库处理逻辑

STUBS

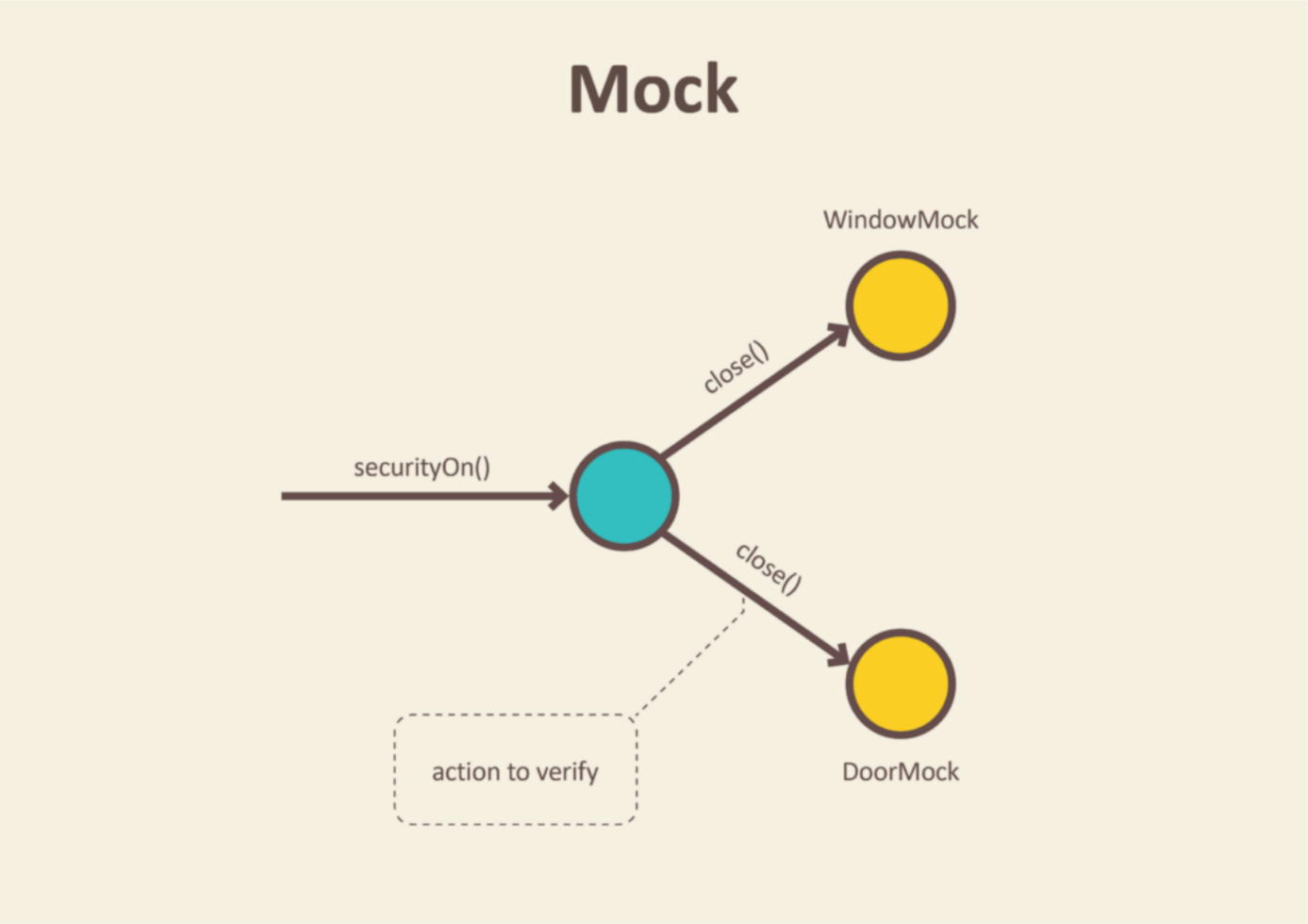
```
1
2  public class GradesService {
3      private final Gradebook gradebook;
4
5      public GradesService(Gradebook gradebook) {
6          this.gradebook = gradebook;
7      }
8
9      Double averageGrades(Student student) {
10         return average(gradebook.gradesFor(student));
11     }
12 }
```

```
1
2  public class GradesServiceTest {
3      private Student student;
4      private Gradebook gradebook;
5
6      @Before
7      public void setUp() throws Exception {
8          gradebook = mock(Gradebook.class);
9          student = new Student();
10     }
11
12     @Test
13     public void calculates_grades_average_for_student() {
14         when(gradebook.gradesFor(student)).thenReturn(grades(8, 6, 10)); //stubbing gra
15         double averageGrades = new GradesService(gradebook).averageGrades(student);
16         assertThat(averageGrades).isEqualTo(8.0);
17     }
18 }
```

- 只是返回结果，而对系统的状态不产生影响，没有副作用
- 属于 Query 类方法
- 对于 Query 类方法，测试时可以选择 Stub 对真实方法进行模拟
- 存在另一类方法，通过一系列 action，改变系统状态，而不期待返回值
- 属于 Command 类方法
- Command 类方法测试需要 Mock
- Command 和 Query 分开设计

MOCKS

- 仅记录调用信息的对象，测试中需要验证Mocks进行了符合预期的调用



- 不希望真的调用生产环境下的代码
- 测试中难以验证真实代码的执行效果
- 如：邮件发送服务的测试

MOCKS

```
1
2  public class SecurityCentral {
3      private final Window window;
4      private final Door door;
5
6      public SecurityCentral(Window window, Door door) {
7          this.window = window;
8          this.door = door;
9      }
10
11     void securityOn() {
12         window.close();
13         door.close();
14     }
15 }
```

```
1
2  public class SecurityCentralTest {
3      Window windowMock = mock(Window.class);
4      Door doorMock = mock(Door.class);
5
6      @Test
7      public void enabling_security_locks_windows_and_doors() {
8          SecurityCentral securityCentral = new SecurityCentral(windowMock, doorMock);
9          securityCentral.securityOn();
10         verify(doorMock).close();
11         verify(windowMock).close();
12     }
13 }
```

- 我们不想测试 securityOn 方法时真正 close window and door
- 我们只需要验证 window 和 door 的close方法都被调用即可
- 我们只关心 close 方法得到了调用
- close 方法实现的正确性，在 door 和 window 的单元测试里进行

SPIES

- 包含预定义好的数据返回给调用者，并同时记录调用信息，例如同时记录邮件发送条数的邮件服务（Theory）
- 对真实对象进行 mock，除 stub 方法外，均调用真实方法，属于部分 mock（Mockito）

```java
@Test
public void test_spy_list() {
  List list = new LinkedList();
  List spy = spy(list);
  when(spy.size()).thenReturn(100);

  spy.add("one");
  spy.add("two");

  assertThat(spy.get(0), is("one"));
  assertThat(spy.size(), is(100));

  verify(spy).add("one");
  verify(spy).add("two");
}
```

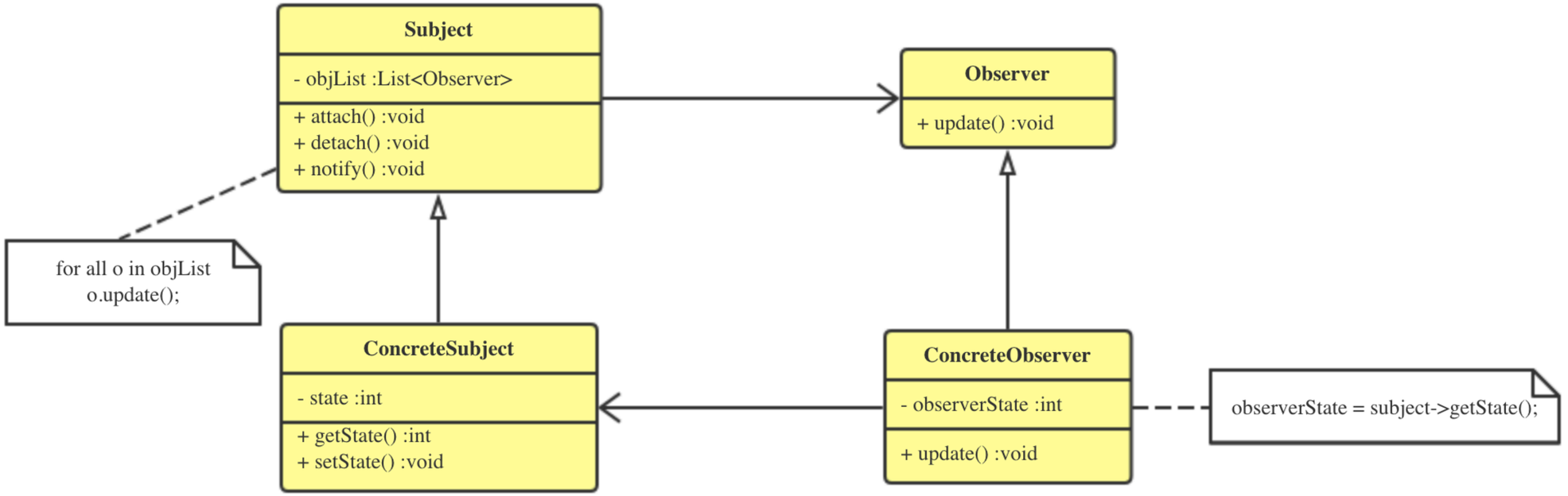
- Reals spies should used carefully and occasionally
- Partial mock is a code smell

FOOTBALL GAME

- 模拟一场足球比赛
- 每当有球队进球时
  - 球迷（Fan）：如果是支持的球队进球，会大喊”Hooray!“，如果是对方球队进球，大发出嘘声“Boooo!”
  - 解说员（Reporter）：大喊”Goooooal!“，并喊出进球的球队名字
  - 计分板（Score Board）：更新计分板的分数
- Task & Test

OBSERVER PATTERN

- 定义对象间的一种一对多依赖关系
- 每当一个对象状态发生改变时，其相关依赖对象皆得到通知并被自动更新



CODE ANALYSIS

```java
public interface Observable {
  void attach();
  void detach();
  void notify();
}
```

```java
public interface Observer {
  public void update();
}
```

```java
public class User implements Observer {
  private Observable observable = null;

  public User(Observable observable) {
    this.observable = observable;
  }

  @Override
  public void update() {
    buyDress();
    unsubscribe();
  }

  public void buyDress() {
    System.out.println("Got my new Red Dress");
  }

  public void unsubscribe() {
    observable.detach(this);
  }
}
```

```java
public class RedDress implements Observable {
  private List<Observer> users = new ArrayList<>();

  private boolean inStock = true;

  public boolean isInStock() {
    return inStock;
  }

  public void setInStock(boolean inStock) {
    this.inStock = inStock;
    notify();
  }

  @Override
  public void attach(Observer o) {
    users.add(o);
  }

  @Override
  public void detach(Observer o) {
    users.remove(o);
  }

  @Override
  public void notify() {
    // notify all the users
    users.forEach(User::update);
  }
}
```

BENEFITS & DRAWBACKS

- 优势
  - 实现表示层与数据逻辑层的分离、稳定的消息更新机制、抽象更新接口支持不同的表示层作为具体观察者
  - 观察目标与观察者之间建立一个抽象耦合
  - 支持广播通信
  - 符合”开闭原则“的要求
- 缺陷
  - 如果观察目标有很多直接或间接观察者，通知所有观察者会花费很多时间
  - 观察者之间有循环依赖的话，观察目标会触发循环调用，可能导致系统崩溃
  - 没有相应的机制让观察者知道观察的目标是怎么发生变化的

APPLICABLE SCENES

- 一个抽象模型有两个方面，其中一个方面依赖另一个方面
- 一个对象的改变导致其他一个或多个对象发生改变，而不知道具体数目
- 一个对象必须通知其他对象，而不知道这些对象是谁
- 需要再系统中创建一个触发链，A 对象的行为影响B对象，B 对象的行为影响C对象…，可以使用观察者模式创建一种链式触发机制

ASSIGNMENTS

- 小T、小Q、小L、小H是同事，他们都用同一款社交产品Circle
- 用户在Circle上可以发布自己的最新动态，而关注他的人都可以看到并点赞
- 每当有同学发布他在加班的消息时，他的同事们都会给他点赞
- 小L同学是个电子迷，每次有人发了有关电子产品的消息时，他都会点赞
- 小T和小Q同学喜欢研究投影仪，有人分享这些消息时，他们都会点赞
- 小Q同学爱研究比特币，他分享有关消息时，小T和小H会给他点赞
- 小L和小T都爱打篮球，有关篮球信息的分享，他们都会点赞
- 小H同学的家庭生活非常幸福，每当他分享家庭生活的时候，会收到大家的赞
- 小L和小H都是玩车一族，他们经常讨论有关话题，互相点赞也很多
- 每当有同学发的动态拿到2个或以上的赞时，都会在心里默默得意一下

ThoughtWorks®

QUESTIONS?