Mocks

Современные приложения редко работают в изоляции, чаще всего они интегрируются с множеством сторонних сервисов. Поэтому остро встает вопрос тестирования этой интеграции в рамках интеграционного или функционального тестирования.

Первое решение, которое приходит в голову — тестировать напрямую на реальном сервисе. В некоторых случаях это может быть обосновано, но часто приводит к следующим проблемам:

- **Тесты замедляются достаточно сильно**. Доступ к внешнему сервису обычно осуществляется по сети с использованием публичных протоколов, что гораздо медленнее локальной интеграции. Внешний сервис обычно не такой быстрый и подвержен собственной нагрузке, во время которой может замедляться еще сильнее. В результате я встречал на практике замедление интеграционного набора тестов с 10 минут до пары часов, причем время выполнения сильно плавало.
- **Тесты становятся менее стабильными**. Как я уже говорил, внешние сервисы подвержены нагрузкам, иногда недоступны из-за обновлений, «падают» при неверном обновлении разработчиками сервиса, ну и сетевые ошибки также не исключены. Все это очень сильно влияет на ваш тестовый набор и его стабильность. Тесты «падают» неожиданно, на изучение проблемы уходит много времени и члены команды начинают недоверчиво относиться к тестам в целом. В результате сама полезность тестирования может быть поставлена под сомнение.
- Вы просто **не имеете возможность получить некоторые ответы внешнего сервиса** (ошибки, специфические данные и т.д.), что может сказаться на покрытии вашего кода тестами, а значит надежности регрессионного тестирования интеграционными тестами. Ведь нужно покрывать не только позитивные, но и негативные сценарии.
- Реальный сервис не всегда имеет тестовый интерфейс и вам приходится использовать совершенно реальный. А это **не всегда бесплатно** и любые **ошибки в вашем коде подвергают риску реальные данные** на внешнем сервисе. Благодаря этой проблеме можно легко удалить очень важные данные или заблокировать доступ из реального приложения. И это не говоря о том, что много денег может быть потрачено на «неожиданное» использование внешнего сервиса.

Заглушки

Заглушки - это компьютерные программы, которые служат временной заменой вызываемого модуля basin и выдают тот же результат, что и фактический продукт или программное обеспечение.

Заглушки тестов - это программы, которые имитируют поведение программных компонентов (или модулей), от которых зависит модуль, проходящий тестирование. Заглушки теста предоставляют готовые ответы на вызовы, сделанные во время теста, обычно вообще не реагируя ни на что , кроме того, что запрограммировано для теста.[1] Они в основном используются в нисходящем подходе инкрементного тестирования.

Рассмотрим компьютерную программу, которая запрашивает базу данных для получения суммарной цены всех продуктов, хранящихся в базе данных. В этом примере запрос

выполняется медленно и потребляет большое количество системных ресурсов. Это уменьшает количество запусков тестов в день. Во-вторых, тесты могут включать значения, отличные от тех, которые в настоящее время находятся в базе данных. Метод (или вызов), используемый для выполнения этого, - *get_total()* . В целях тестирования исходный код в *get_total()* может быть временно заменен простым оператором, который возвращает определенное значение. Это будет заглушка для теста.
Доступно несколько платформ тестирования, а также программное обеспечение, которое генерирует заглушки тестов на основе существующего исходного кода и требований к тестированию. Заглушки и драйверы - это два типа тестовых жгутов. Тестовые жгуты представляют собой совокупность программного обеспечения и тестовых данных, которые сконфигурированы таким образом, что можно тестировать программный модуль, моделируя различные наборы условий, контролируя поведение и результаты.
Заглушки и драйверы являются фиктивными модулями и создаются только для целей тестирования.
Заглушки используются при нисходящем подходе к тестированию, когда основной модуль готов к тестированию, но вспомогательные модули еще не готовы. Таким образом, на простом языке заглушки - это "вызываемые" программы, которые вызываются для проверки функциональности основного модуля.
Например, в ситуации, когда у вас есть три разных модуля: Login, Home, User. Предположим, что модуль входа готов к тестированию, но два второстепенных модуля Home и User, которые вызываются модулем входа, еще не готовы к тестированию. В это время написан фрагмент фиктивного кода, который имитирует вызываемые методы Home и User. Эти фиктивные фрагменты кода являются заглушками.
С другой стороны, драйверы - это те, которые являются "вызывающими" программами. Драйверы используются в методе тестирования снизу вверх. Драйверы - это фиктивный код, который используется, когда вспомогательные модули готовы, но основной модуль все еще не готов.
Берем тот же пример, что и выше. Предположим, на этот раз пользовательский и домашний модули готовы, но модуль входа в систему не готов к тестированию. Теперь, поскольку Home и User возвращают значения из модуля входа, поэтому записывается фиктивный фрагмент кода, который имитирует модуль входа. Затем этот фиктивный код называется драйвером.
Заглушка в распределенных вычислениях - это фрагмент кода, который преобразует параметры, передаваемые между клиентом и сервером во время удаленного вызова процедуры (RPC).
Основная идея RPC заключается в том, чтобы позволить локальному компьютеру (клиенту) удаленно вызывать процедуры на другом компьютере (сервере). Клиент и сервер используют разные адресные пространства, поэтому параметры, используемые при вызове функции (процедуры), должны быть преобразованы, в противном случае значения этих параметров не могут быть использованы, поскольку указатели на параметры в памяти одного компьютера будут указывать на разные данные на другом компьютере. Клиент и сервер также могут использовать разные представления данных, даже для простых параметров (например, для целых чисел с большим и меньшим порядком). Заглушки выполняют преобразование параметров, поэтому удаленный вызов процедуры выглядит как вызов локальной функции для удаленного компьютера.
Библиотеки-заглушки должны быть установлены как на стороне клиента, так и на стороне сервера. Клиентский stub (иногда называемый прокси-сервером) отвечает за преобразование (сортировку) параметров, используемых при вызове функции, и деконверсию результатов, переданных с сервера после выполнения функции. Серверный скелет, заглушка на стороне сервера, отвечает за деконверсию параметров, передаваемых клиентом, и преобразование результатов после выполнения функции.
Заглушки могут быть созданы одним из двух способов:

- Вручную: В этом методе разработчик RPC предоставляет набор функций трансляции, из которых пользователь может создавать свои собственные заглушки. Этот метод прост в реализации и может обрабатывать очень сложные типы параметров.

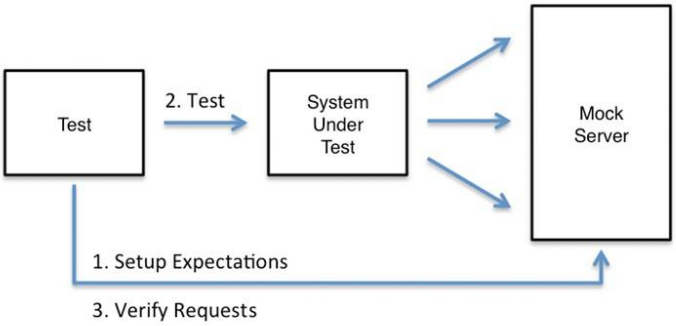
- Автоматически: это наиболее часто используемый метод генерации stub. Он использует язык описания интерфейса (IDL) для определения интерфейса между клиентом и сервером. Например, определение интерфейса содержит информацию, указывающую, является ли каждый аргумент входным, выходным или обоими; только входные аргументы должны быть скопированы с клиента на сервер, и только выходные элементы должны быть скопированы с сервера на клиент.

Говорят, что серверная программа, которая реализует процедуру в интерфейсе, *экспортирует* интерфейс, а клиентская программа, которая вызывает процедуры из интерфейса, *импортирует* интерфейс. При написании распределенного приложения программист сначала пишет определение интерфейса, используя IDL, затем программисты могут написать клиентскую программу, которая импортирует интерфейс, и серверную программу, которая экспортирует интерфейс. Определение интерфейса обрабатывается с помощью компилятора IDL для создания компонентов, которые могут быть объединены с клиентскими и серверными программами, без внесения каких-либо изменений в существующие компиляторы. В частности, из интерфейса для каждой процедуры в интерфейсе компилятор генерирует соответствующие операции сортировки и отмены сортировки в каждой процедуре-заглушке и заголовочный файл, который поддерживает типы данных в определении интерфейса. Заголовочный файл включен в исходные файлы как клиентской, так и серверной программ, клиентские процедуры-заглушки компилируются и связываются с клиентской программой, а серверные процедуры-заглушки компилируются и связываются с серверной программой. Компилятор IDL может быть разработан для обработки определений интерфейса для использования с разными языками, позволяя клиентам и серверам, написанным на разных языках, взаимодействовать с помощью удаленных вызовов процедур. Для достижения цели прозрачности семантики разработчики сделали RPC похожим на LPC, используя концепцию заглушек, которые скрывают фактическую реализацию RPC от программ, интерфейсов к базовой системе RPC.

«Мок» в программировании - это объект-заглушка, реализующий заданный аспект моделируемого программного окружения.

Заглушки обеспечивают постоянные ответы на вызовы, сделанные во время теста, обычно вообще не реагируя ни на что, кроме того, что запрограммировано для теста. Заглушки могут также записывать информацию о вызовах, такую как заглушка шлюза электронной почты, которая запоминает сообщения, которые она «отправила», или, возможно, только сколько сообщений она «отправила».

Заглушки для запросов



Запускается тест.

- В первую очередь, тест готовит себе данные – обращается к специально запущенному приложению, передает ему информацию о том, что собирается отправить в него такие-то данные и ожидает получить на них такие-то ответы.
- Далее тест обращается к вашей системе, чтобы она отправила запросы не на реальный сервер, а на заглушку.
- Заглушка присылает ответы, а тест верифицирует: правильные ли данные получила система.

Существует несколько видов объектов, которые позволяют симулировать поведение реальных объектов во время тестирования:

1. Dummy — пустые объекты, которые передаются в вызываемые методы, но не используются. Предназначены лишь для заполнения параметров методов.
2. Fake — объекты, которые имеют реализации, но в таком виде, который делает их неподходящими для использования в рабочей ситуации.
3. Stub — предоставляют заранее заготовленные ответы на вызовы во время теста и не отвечают ни на какие другие вызовы, которые не требуются в тесте.
4. Mock — объекты, которые заменяют реальный объект в условиях теста и позволяют проверять вызовы своих методов. Содержат заранее подготовленные описания вызовов, которые они ожидают получить. Применяются в основном для тестирования поведения пользователя.

Относительно веб-сервисов можно сказать, что мок-сервис позволяет переопределить реальный сервис и подставить вместо него упрощенную реализацию, которая работает нужным разработчику образом и дает доступ к собственным данным и настройкам. Мы как бы создаем у себя аналог удаленного веб-сервиса, который отвечает на вызовы нашего приложения.

Правда, тут возникают две сложности.

Во-первых, моки надо создать. Надо устанавливать дополнительный софт, описывать логику, загружать нужные данные в макет ответа. Не то чтобы это сложно, но это требует

усилий и времени. Может быть, кому-то будет проще сохранить результат вызова в файл и при тестировании обращаться не к заглушке, а к файлу. В небольших проектах с малым количеством тестов такой подход вполне оправдан.

Во-вторых, веб-сервис может поменяться. Например, программист написал рабочую логику для заглушки, а через какое-то время веб-сервис стал работать по-другому (а разработчика, как часто бывает, об этом никто не уведомил). О том, что в работе веб-сервиса что-то поменялось он узнал, когда выкатил обновление на рабочую базу.

Вариантов мок-серверов много, перечислю самые популярные.

- Один из первых, который мне попадался – это Postman, в нем есть возможность мокирования. Postman – одно из мастхэв-решений, если вы пишете интеграцию. Работать с запросами в нем очень удобно. И также в нем есть собственная возможность мокирования. Достаточно слабенькая, но есть.
- SoapUI – достаточно мощное решение, там есть запись ваших запросов, их воспроизведение, упаковка в сценарии. Достаточно популярное решение для мокирования.
- Еще есть сервис WireMock.
- И есть JSON-Server – легкое решение, в нем есть база данных, основанная на JSON. Он умеет сохранять данные и преобразовывать.