# Глава 14 Модель программирования RESTful ABAP

"Я не думаю, что изменил бы что-то в рэпе. Рэп не имеет границ, и мне это нравится".

- Meek Mill

Возможно, вы знакомы с термином "запланированное устаревание", означающим, что продукт предназначен для того, чтобы устареть через год - или даже несколько месяцев - после его выпуска, чтобы вам пришлось покупать новый. Например, смартфоны работают по такой бизнес-модели.

Программное обеспечение в некоторой степени похоже на это в том смысле, что циклы выпуска стали настолько быстрыми, что десять различных компаний могут выпускать десять различных версий одного и того же программного обеспечения. Это превращает поддержку в кошмар и сбивает людей с толку. Когда SAP хочет запутать людей, он может быть очень хорош в этом. Один из хороших способов - запутать сообщество разработчиков ABAP настолько, что его члены не знают, приходят они или уходят - это выпускать новую модель программирования с каждым релизом, что, похоже, и происходит в настоящее время.

На момент написания статьи существует четыре модели программирования SAP, о которых я знаю:

1. Процедурное программирование.

2. Объектно-ориентированное программирование.

3. Модель программирования ABAP для SAP S/4HANA. Эта модель предполагает использование CDS представлений для доступа к базе данных, BOPF для бизнес-логики и SAPUI5 для пользовательского интерфейса. Это рекомендуемый способ программирования в локальной системе SAP S/4HANA. До этого момента именно на этой модели была сосредоточена данная книга.

4. Модель программирования RESTful ABAP (RAP). На момент написания книги эта модель всего несколько месяцев и доступна только для ABAP в облаке. В этой среде вы вынуждены программировать с использованием RAP. Она имеет ту же структуру, что и модель программирования ABAP для SAP S/4HANA, но BOPF заменен аналогичной, но более продвинутой концепцией. RAP будет доступен в локальной среде примерно в сентябре 2019 года.

797

В этой главе сначала мы рассмотрим, что такое SAP Cloud Platform, среда ABAP (далее ABAP в облаке) (раздел 14.1). Затем мы объясним модель RAP.

Примечание

При использовании ABAP в облаке вы вынуждены использовать RAP. Буквально нет другого способа программировать. Когда RAP появится для SAP S/4HANA в локальной версии, он будет необязательным, хотя, несомненно, будет настойчиво продвигаться компанией SAP. Имейте в виду, SAP настойчиво подталкивает программистов ABAP к использованию модели OO с 2000 года, и посмотрите, к чему это привело.

В разделе 14.2 вы увидите, чем RAP отличается от концепций, представленных ранее в этой книге, в отношении уровня базы данных, уровня бизнес-логики и уровня пользовательского интерфейса. И это являются эволюцией, а не полной заменой концепций.

Затем основная часть главы (раздел 14.3 - раздел 14.6) будет посвящена тому, как построить монструозное транзакционное приложение бизнес-объектов с использованием RAP. Новые вещи могут быть пугающими, поэтому, придерживаясь повседневных вещей, таких как монстры, с которыми все знакомы, надеюсь, облегчит переход.

## 14.1 ABAP в облаке

Как мы все знаем, Джек посадил в землю несколько волшебных бобов, и на следующий день вырос огромный бобовый стебель. Он взобрался на него и на вершине нашел облако. Так родились облачные вычисления. Последней вещью, которая взобралась по бобовому стеблю в облако, является язык ABAP. Об этом было объявлено на SAP TechED 2017 в Лас-Вегасе техническим директором SAP, а первая часть была выпущена 4 сентября 2018 года, скорее всего, для того, чтобы о ней можно было рассказать на SAP TechED 2018.

Первая версия сосредоточена на расширении SAP S/4HANA Cloud с помощью совершенно нового ABAP кода. Последующие версии будут направлены на миграцию существующего Z-кода и поддержку SAP S/4HANA на локальном уровне.

Прежде всего, важно отметить, что для программирования с использованием ABAP в облаке вам нужна специальная учетная запись в SAP Cloud Platform, а это стоит довольно много денег каждый месяц, поэтому вы хотите, чтобы организация, на которую вы работаете, платила за это!

Более того, по крайней мере, на момент написания статьи, бесплатной пробной версии не существует, поскольку SAP утверждает, что объем памяти для каждого экземпляра слишком велик, чтобы предоставить бесплатную пробную версию системы, как это можно сделать для SAP Cloud Appliance Library.

798

Следующее большое изменение заключается в том, что единственный способ написания ABAP в SAP Cloud Platform - через ABAP в Eclipse. SE80 не имеет возможности заглянуть внутрь. Eclipse будет общаться с платформой ABAP в облаке через эквивалент узла SICF - то есть, через URL, что фактически является способом общения с локальными системами.

До сих пор, если вы хотели написать пользовательский код для SAP S/4HANA Cloud, вы должны были делать это на JavaScript, Java или - как однажды сказал эксперт в области программирования Генри Форд - "на любом

языке, если это не ABAP" на платформе SAP Cloud Platform, а затем подключаться к основной ERP-системе через определенные APis.

Это связано с тем, что в облачной системе SAP должна быть свободна в обновлении стандартной кодовой базы через регулярные промежутки времени и при этом не нарушать никакой клиентский код. Единственный способ сделать это - разделить стандартный код и код клиента большой кирпичной стеной, что и представляют собой APis.

В связи с этим возникла ужасная перспектива для клиентов, которые хотят перейти на SAP S/4HANA Cloud (а SAP хочет, чтобы все перешли на SAP S/4HANA Cloud), придется переписывать весь свой пользовательский Z-код на Java или что-то подобное. Так что с ABAP в облаке вам не придется переписывать весь ваш пользовательский код: просто вырежьте и вставьте его, и все будет хорошо! Ура! Но, к сожалению, нет. Вам все равно придется переписывать большую часть кода, а возможно, и весь код, и я расскажу, почему.

Первое, что нужно понять, это то, что эта новая среда программирования собирается решать вопрос о четком разделении кода бизнес-логики и кода пользовательского интерфейса.

В этой главе вы увидите, что бизнес-логика по-прежнему пишется на ABAP, но совсем по-другому, чем раньше. Версия ABAP в облаке будет подмножеством текущего языка ABAP, к которому вы привыкли, с удаленными устаревшими командами и - это очень важный момент - без UI команд вообще. Никаких утверждений WRITE, никаких CALL SCREEN, никаких ALV, никаких Web Dynpro, и так далее. Также нет доступа к файловой системе - потому что нет никакой файловой системы!

В этот момент, без сомнения, вы начинаете понимать, почему ваша текущая кодовая база должна быть несколько переписана. Дальше - хуже: единственные компоненты ABAP в облаке - это ABAP и BASIS, а не APPL, потому что это отдельная система ABAP. Таким образом, если вы начнете использовать термины вроде KNA1 или MARA в своем пользовательском коде, компилятор не поймет, о чем вы говорите (поскольку эти таблицы существуют только в ERP-системе) и выдаст вам синтаксическую ошибку. Думаю, теперь вы понимаете, в чем проблема и почему, простите за каламбур, просто вырезать и вставить не получится.

Если быть более точным, в прошлом основные таблицы и Z-таблицы жили в одной базе данных, и стандартный код SAP ABAP и код Z ABAP жили в одной базе данных. Теперь этого нет. Теперь основная ERP-система и ее таблицы, такие как KNA1, находятся в одной базе данных SAP HANA, а ваш

799

код Z ABAP и ваши таблицы Z ABAP находятся в совершенно другой базе данных SAP HANA, на которой работает ABAP в облаке. Поэтому в ABAP в облаке вы можете выполнить SELECT на ZTMONSTER, потому что облако ABAP знает, что это такое, но не на KNA1, потому что это только живет в базе данных ERP-системы. Вы все еще можете иметь эквивалент Z-дополнений в стандартных таблицах, но добраться до них гораздо сложнее.

Вы можете сказать себе: "Как я могу получить данные о клиенте, не выполнив SELECT в KNA1? Очевидно, что если бы вы не могли общаться с ERP-системой в какой-то форме, то вы не сможете использовать ABAP в облаке для расширения ERP-системы; вам придется писать совершенно новую ERP-систему с нуля, что было бы глупо, хотя некоторые циники могут сказать, что это было бы улучшением. На самом деле вы узнаете, как это сделать в разделе 14.2.1.

Пользовательский интерфейс (на данный момент) будет SAPUI5 и управляться аннотациями из представления CDS. Здесь также не будет системы изменений и транспорта; контроль версий будет осуществляться с помощью abap-Git, о котором вы узнали в главе 2.

Вся эта концепция, без сомнения, кажется ужасным кошмаром. Вы, вероятно, думаете про себя, что если то, что я только что сказал, правда, то переход на SAP S/4HANA Cloud означает переписывание двадцатилетних пользовательских разработок - то есть, начать все с нуля, поэтому вы ни за что не перейдете на SAP S/4HANA Cloud.

Однако есть и хорошие новости. Во-первых, SAP активно ищет отзывы о ABAP в облаке в его нынешнем виде, так что, если вы посмотрите и вам не понравится, вступайте и высказывайте свое мнение. Это не похоже на открытый исходный код, где вы можете сами все исправить, но, по крайней мере, SAP прислушивается.

Во-вторых, вы можете уже сейчас начать готовить свой код к переходу, что включает в себя написать свой Z-код так, чтобы он следовал основным принципам ОО - а именно, "разделению задач". Вы потеряете код пользовательского интерфейса, поэтому убедитесь, что вы перенесли всю логику UI в специальные классы UI (которые оборачивают экраны Dynpro в функциональные блоки внутри класса, скажем). Получение данных сильно изменится, поэтому убедитесь, что у вас есть выделенные классы для доступа к базе данных.

В главах, посвященных модульному тестированию, вы узнали, что вам все равно придется все это сделать, чтобы сделать ваш код пригодным для тестирования, поэтому конечный смысл здесь в том, что если вы перепишете свой код в соответствии с принципами ОО, то это даст вам не только краткосрочное преимущество в виде тестируемости, но и долгосрочное преимущество - возможность переноса вашей бизнес-логики Z-кода в облако относительно безболезненно.

Компания SAP выпустила вариант проверки ABAP Test Cockpit под названием *SAP\_CP\_READINESS\_REMOTE* с помощью которого вы можете проанализировать ваш текущий Z-код, чтобы узнать, какая часть его будет сломана, если он

800

мигрировал на ABAP в облаке. Вам нужна центральная система, на которой будут выполняться проверки с вашей реальной ERP-системой, и эта центральная система должна быть 7.52 или выше. Затем это просто вопрос установки SAP Notes 2682626 и 2684665. Если только ваш текущий Z-код действительно надежен, вы, скорее всего, получите ряд ошибок и предупреждений длиной как сто миллионов слоновьих хоботов, связанных вместе. Однако, по крайней мере, вы будете знать в какой ситуации вы находитесь, что не так плохо, как эти бедные слоны.

Предположим, что ваш пользовательский Z-код настолько хорошо написан, что скопировать его не составляет труда. Конечный результат миграции вашей основной ERP-системы на SAP S/4HANA Cloud и вашего Z-кода на ABAP в облаке будет то, что четыре раза в год ваш стандартный код SAP (как в среде ERP, так и в среде разработки) будет обновляться самостоятельно без необходимости проходить через невероятно дорогостоящий проект обновления.

Проект обновления занимает от шести месяцев до года, и вы можете отправлять улучшения и дополнения к вашему пользовательскому коду в производство, когда вам этого захочется. Не будет никакой зависимости между этими двумя циклами. Более того, ваша среда разработки всегда будет иметь последнюю версию.

На мой взгляд, день, когда большинство клиентов SAP окажутся в такой утопической ситуации еще далеко - около тридцати лет - но путешествие уже началось, и эта книга в конце концов, в названии этой книги есть слово "будущее".

Тем не менее, и ABAP в облаке, и RAP уже здесь и сейчас, и к тому времени, когда вы будете читать эту книгу, я уверен, что, по крайней мере, один "ранний адепт" будет использовать их продуктивно. Итак, давайте перейдем к тому, чем RAP отличается от того, что было раньше, а затем погрузимся в жуткие подробности того, как программировать с его помощью.

Этот человек на бета-версии

Важно отметить, что и ABAP в облаке в целом, и RAP в частности являются совершенно новыми и представляют собой версию 0.1, так сказать. Компании-разработчики находятся под огромным давлением, стремясь выпустить продукты как можно скорее, поэтому первая версия чего бы то ни было будет содержать пробелы и часто ошибки. Печальная правда заключается в том, что, если любая компания ждет, пока "это" станет совершенным, прежде чем выпустить в свет, что бы это ни было, конкуренты выпустят версию с ошибками первыми и захватят рынок.

В данном случае преимущество заключается в том, что по своей природе ABAP in the cloud является облачным продуктом и, следовательно, может быстро выходить на рынок, и поэтому может иметь быстрые циклы выпуска, потому что клиенту не нужно делать обновление вручную или импортировать пакеты поддержки. В данном случае цикл выпуска новых функций каждые три месяца, и я не сомневаюсь, что выявленные ошибки будут исправляться по мере их обнаружения.

801

То, о чем вы прочитаете в этой главе, описывает ABAP в облаке и RAP такими, какими они были в первый месяц их появления - то, что можно назвать минимально жизнеспособным продуктом. Основополагающие концепции прочны и не изменятся, но у меня нет сомнений в том, что через два года (или даже раньше) технические детали того, как вы делаете вещи, будут выглядеть совсем по-другому.

## 14.2 Изменения в модели программирования

Этот раздел посвящен тому, чем RAP отличается от модели программирования ABAP для SAP S/4HANA, о которой шла речь в остальной части этой книги.

Три центральных столпа не изменились: по-прежнему существует четкое разделение между уровнем базы данных, уровнем бизнес-логики и уровнем пользовательского интерфейса. Однако в каждом случае программирование изменилось в меньшей или большей степени, поэтому давайте поговорим об изменениях в каждом слое по очереди.

### 14.2.1 Изменения в уровне базы данных

Как упоминалось ранее, основное изменение здесь заключается в том, что вы больше не можете использовать операторы SELECT для доступа к таблицам базы данных в основной ERP-системе, такой как KNA1. Для ваших собственных Z таблиц, вы можете продолжать использовать SELECT или UPDATE, потому что они находятся в той же базе данных, что и ваш облачный ABAP-код. Поскольку вам всегда будет нужен доступ как к чтению данных из ERP-системы, так и к их обновлению (например, при отпуске товаров), должен существовать какой-то механизм для этого.

В данном случае механизм связи осуществляется через APIs, включенные в белый список. Что это такое, спросите вы? Ну, вы знаете, что 95% стандартных ФМ SAP помечены как "не выпущенные", но мы все равно их используем? Представьте себе, если бы вы не могли этого делать. Представьте, если бы в один прекрасный день каждый невыпущенный функциональный блок в ваших Z-программах начал вызывать синтаксическую ошибку. Представляю, как бы вы взвыли; я знаю, что взвыл бы. Белый список в ABAP cloud похож на этот ужасный сценарий. Вместо того, чтобы иметь возможность вызывать любой стандартный SAP ФМ или класс, который вам захочется, вы должны выбрать их из списка одобренных. В свою очередь, SAP на 100% гарантирует, что эти вызовы будут работать после обновления основной системы.

Если вы хотите отправлять и/или получать данные из ERP-системы, вы должны использовать API, включенный в белый список. Определение RESTful API согласно <https://searchmicro-services.techtarget.com/definition/RESTful-API> следующее: "RESTful API - это

802

программный интерфейс (API), использующий HTTP-запросы для получения, ввода, вывода и удаления данных".

Звучат ли эти термины знакомо? Они должны напомнить вам о том, как вы тестировали OData сервисы, созданные в SEGW, которые взаимодействуют с приложениями SAPUI5 в главе 12.

Если быть еще более точным, SAP заявила, что, если она хочет внести таблицу в белый список, она создаст представление CDS для нее (доступ на чтение), а для доступа на запись предложит типизированный API на основе классов. На сайте сказано, что в какой-то момент в будущем SAP также будет поддерживать вызовы RFC, что может заставить вас быть немного менее нервным.

Эта ситуация - общение с API - не является концом света, хотя на первый взгляд может показаться, что это не так. Вопрос, который вы должны задать себе, заключается в том, перевешивает ли стабильность, которую вы получаете в результате такого подхода, первоначальную боль от того, что вы делаете все по-другому. Суть стабильности в том, что она позволяет вам вносить изменения, а в наши дни возможность быстрее адаптироваться не позволит вам быстро пойти по пути динозавра или Blockbuster Video.

С вашими Z-таблицами вы можете выполнять pushdown кода любым удобным для вас способом, как описано в Главе 7. Ничего не изменилось - за исключением, как вы увидите, случаев, когда речь идет о бизнес-объектах, которые должны вращаться вокруг представления CDS.

### 14.2.2 Изменения на уровне бизнес-логики

Изменения в этой области немного более радикальны. BOPF, как описано в главе 8, больше нет. Он был заменен чем-то, что, хотя и является концептуально идентичным, но реализовано совершенно по-другому технически. На случай, если у вас случится сердечный приступ, когда вы прочитаете, что BOPF теперь находится в "режиме обслуживания", когда вы последние три года писали все свои новые приложения с использованием BOPF - как рекомендует SAP, не бойтесь: SAP утверждает, что ваши инвестиции в безопасность оправданы и что с 2020 года будет встроенная интеграция между RAP и BOPF, хотя она говорит, что как только RAP будет доступен для вас, не используйте BOPF для любых новых разработок.

На момент написания статьи новой концепции бизнес-объектов всего несколько дней, и поэтому она несомненно, получит радикальные изменения в будущих релизах. В этом есть два направления: использование языка бизнес-определения и идея о том, что бизнес-объекты являются гражданами первого класса.

**Язык бизнес-определения**

В главе 7 вы видели, что основное преимущество представления CDS перед 100% идентичным запросом SQL заключается в том, что представление CDS выражает ту же информацию "семантически".

803

Например, INNER JOIN на VBAK/VBAP - это то, как база данных (машина) думает о соединении двух таблиц, в то время как "ассоциация" между двумя понятиями (заголовок заказа и заказ - это то, как человек думает о связи между двумя таблицами.

Таким образом, определение CDS читается как обычный английский, но компилируется в точно такой же SQL-запрос в фоновом режиме, как и тот, который генерируется более техническим Open-SQL-запросом.

Когда ABAP в облаке вышел в сентябре 2018 года, SAP раскрыла концепцию бизнес-определения, которое заключается в определении бизнес-объекта в текстовом виде, подобно DDL, определяющему представление CDS. Теперь вы можете описать природу бизнес-объекта (например, какое поведение он способен выполнять) в определении BDL (которое само по себе является специализированным типом представления CDS).

В начальном выпуске RAP у вас есть только концепция так называемого неуправляемого бизнес-определения, при котором вы описываете поведение бизнес-объекта в BDL, но фактически реализуете логику, используя то, что на первый взгляд кажется традиционным ABAP-кодом, хотя и с некоторыми новыми причудами. В будущем будет введена концепция управляемых бизнес определений, с помощью которых, предположительно, BDL сама будет генерировать код для управления поведением. Многие программисты ABAP (включая меня) находят эту идею довольно пугающей, но мы должны подождать, пока SAP действительно скажет нам, что это значит, прежде чем впадать в безумную панику.

Стоит отметить, что с 1950-х годов компании-разработчики программного обеспечения обещают, что "на следующей неделе" они представят возможность написать спецификацию на простом английском языке и программа сгенерируется сама, не требуя ни строчки кода. До сих пор это так и не произошло, поэтому, подобно делению ядерного заряда, фактическая реализация этого обещания всегда кажется десятилетней перспективой.

**Бизнес-объекты как граждане первого класса**

Компания SAP также разработала довольно туманную концепцию, согласно которой бизнес-объект должен быть гражданином первого класса. Этот термин означает (в мире, не связанном с SAP), что он (бизнес-объект) может быть передан в качестве параметра, быть результатом функционального метода, и быть присвоенным переменной. По сути, в RAP определение бизнес-объекта выполняется через конструкции языка ABAP, и у вас есть типизированные интерфейсы. Это контрастирует с BOPF, в котором вы определяете метаданные, генерирующие классы с типовыми интерфейсами, поэтому вы не можете иметь экземпляр объекта BOPF в качестве параметра RETURNING, например.

Я бы утверждал, что это делает мой отдельный класс модели монстра гражданином первого класса: он имеет типизированный интерфейс и представляет бизнес-объект. Более того, как вы увидите

804

позже, как ABAP-интерфейсы, используемые RAP, определяют, каким должен быть их правильный тип: очень странно и происходит под капотом, как будто по черной магии. Как это работает, станет ясно позже в этой главе.

### 14.2.3 Изменения в слое пользовательского интерфейса

Эта область практически не изменилась. Идея всегда заключалась в том, что технология пользовательского интерфейса меняется настолько быстро, что необходимо полностью отделить бэкенд от фронтенда, заставив уровень пользовательского интерфейса использовать сервисы OData, предоставляемые бэкендом. На данный момент компания SAP предлагает SAPUI5 в качестве предпочтительной технологии пользовательского интерфейса, но в наши дни и эпоху что-то большее и лучшее может появиться в любую секунду.

Ранее в этой книге вы заметили, что модель декларирует информацию, которую в прошлом можно было считать работой пользовательского интерфейса, например, порядок расположения полей, альтернативные имена и практически вся информация о типе каталога полей ALV.

В RAP такое поведение пользовательского интерфейса полностью контролируется аннотациями в представлениях CDS.

Уровень пользовательского интерфейса является тупым и занимается исключительно тем, чтобы приложение выглядело как можно красивее, насколько это возможно. Идея заключается в том, что если во фронтенде нет никаких "умностей", то вы можете переключить технологию по щелчку выключателя и не менять бэкенд вообще.

Тем не менее, некоторые люди испытывают глубокий дискомфорт от управления пользовательским интерфейсом путем кодирования в том, что кажется слоем базы данных (представление CDS). Можно утверждать, что модель данных не должна знать, как с ее данными будут взаимодействовать во фронтенде, особенно если модель предназначена для использования многими различными приложениями. Контраргумент заключается в том, что представление CDS находится на одном уровне с уровнем базы данных, потому что однажды (возможно) представление CDS сможет потреблять источники, отличные от таблицы/представления базы данных, например, сервисы OData.

Независимо от того, нравится ли вам концепция аннотаций пользовательского интерфейса в представлении CDS или нет, одна вещь, которая есть в RAP - вы получаете своего рода WYSIWYG-представление того, как будет выглядеть ваше приложение, создаваемое на лету по мере добавления аннотаций, специфичных для пользовательского интерфейса, к вашему CDS.

Работает это так: вы определяете свое представление CDS с помощью ABAP в Eclipse. Находясь в экране определения CDS, в любой момент вы можете нажать *Open Fiori Elements App Preview*, чтобы посмотреть, как будет выглядеть приложение на основе текущих аннотаций в представлении CDS.

Это очень похоже на инструмент тестирования в BOPF, с помощью которого вы могли получить список монстров на лету и выполнить над ними действия, например, создать нового, и получить

805

результат сохранения в базе данных. Большим отличием является пользовательский интерфейс, который представляет собой SAPUI5 в отличие от SAP GUI.

Забегая вперед еще раз, скажу, что в какой-то момент в будущем аннотации пользовательского интерфейса не нужно будет кодировать вручную в определении представления CDS. У вас будет визуальный редактор WYSIWYG, в котором вы будете перетаскивать поля и изменять их свойства, и будет сгенерирован файл аннотаций для конкретного приложения.

## 14.3 Структура приложения

Хорошо, давайте сделаем несколько монстров! В облаке! Как обычно, вы хотите иметь приложение, которое показывает список монстров и позволяет создавать, обновлять и удалять их (хотя монстры не очень любят, когда их удаляют), а также заставлять определенного монстра выполнить какое-либо действие, например, завыть на луну. До сих пор в книге вы выполняли следующее упражнение, используя BOPF, ALV, WDA и SEGW/UI5. Теперь настало время попробовать выполнить то же самое упражнение, используя RAP в ABAP в облаке.

В каждом случае - и RAP не будет исключением - многократно используемый класс модели монстра можно использовать для всей бизнес-логики, обновлений, действий и так далее, потому что он на 100% независим от фреймворка. Поэтому, если вы пишете многократно используемые классы бизнес-объектов, вы можете перейти от (скажем) BOPF к (скажем) RAP, просто ознакомившись с тем, как работает новый фреймворк и "подключить" свой существующий класс и использовать паттерн адаптера, чтобы позволить ему взаимодействовать с новым фреймворком. Это было бы невозможно, если бы у вас был, например, BOPF- или WDA-специфические конструкции в вашем классе модели. Каждому новому фреймворку нужен код для адаптации к классу монстра. Класс монстра остается неизменным. Если бы вы были достаточно сумасшедшим, вы могли бы написать приложение, используя класс модели монстра, в котором пользовательский интерфейс выводился бы в виде операторов WRITE - но давайте смотреть в будущее, а не

прошлое.

Действительно хороший способ проектирования любого фреймворка - это структурировать его в виде слоеного пирога; вы неоднократно видели это в этой книге и, скорее всего, посчитали это чрезмерно сложным. Каждый слой знает о том, что находится над ним и под ним, и не имеет никаких представлений ни о чем другом, что происходит в структуре. Таким образом, вы можете заменить любой отдельный слой с минимальными усилиями - что было бы довольно сложно в настоящем торте, но в программном обеспечении это не проблема.

RAP устроен следующим образом:

- Основа слоеного пирога состоит из представлений CDS. Некоторые из них доступны только для чтения, но интересны те, которые определены как бизнес-объекты, такие как монстры, которые могут иметь определения поведения и реализации. Определение поведения само по себе является

806

особым типом представления CDS - это расширение базового представления CDS. Представления CDS не заботятся о том как они будут потребляться.

- Над этим находится определение сервиса, которое в основном представляет собой список представлений CDS, которые "открыты" сервисом. Здесь вы можете добавлять и удалять представления CDS по своему усмотрению.

- Над этим находится привязка сервиса. Она связывает определение службы с техническим протоколом. На данный момент это OData (и самое интересное, что это все еще OData V2, в то время как SAP уже много лет говорит, что поддержка OData V4 в SAP не за горами). Идея заключается в том, что, если однажды SAP решит, что OData - это куча старого мусора, и отныне SAP будет плевать на нее и на лошадь, на которой она приехала, здесь вы можете просто изменить привязку на любой новый протокол, который работает без ущерба для других слоев.

- На вершине пирога находится потребитель, который обычно представляет собой слой пользовательского интерфейса, но также может быть прямым межмашинным взаимодействием без участия пользовательского интерфейса. Это было бы плохо, потому что большинство машин не понимают, насколько вкусен торт.

## 14.4 Кодирование транзакционного приложения для бизнес-объектов

Давайте перейдем к кодированию представления CDS, специфичного для бизнес-объектов (раздел 14.4.1) и связанного с ним определения поведения (раздел 14.4.2) и реализации (раздел 14.4.3), что является фактическим кодом ABAP, который вам нужно написать. Затем мы сохраним изменения, которые сделанные нами изменения (раздел 14.4.4).

### 14.4.1 Кодирование представлений CDS бизнес-объектов

Ранее в книге вы видели представления CDS с аннотациями, помечающими их как бизнес-объекты. В таких случаях, в зависимости от того, какие именно аннотации вы добавили, BOPF определение и/или определение SEGW генерировалось автоматически на основе определения CDS.

В RAP больше не существует ни BOPF, ни SEGW, поэтому использование прежних аннотаций, связанных с этими технологиями, больше не имеет никакого эффекта. Как вы увидите, вместо этого определение поведения может присоединяться к представлению CDS, так сказать, извне. Это делает представление CDS более многоразовым, поскольку его можно использовать как для чтения, так и для транзакционных приложений.

В RAP есть новое ключевое слово, которое вы должны указать в представлении CDS, говоря, что это корень всего зла... погодите, это неправильно. На самом деле, это "корень" бизнес-объекта - так что

807

для монстра корневой сущностью будет заголовок монстра, который может иметь ассоциации с предметами монстра, питомцами монстра и так далее. В определении DDL вы должны сказать *DEFINE ROOT VIEW* вместо просто *DEFINE VIEW*.

В листинге 14.1 давайте максимально придерживаться рекомендованных SAP соглашений об именовании насколько это возможно. Для имени представления SQL поставьте "V" в начале, чтобы сказать, что это представление, потому что оно имеет те же имена, что и определения таблиц и структур. Укажите, что это корневая сущность, а также придерживайтесь отраслевого стандарта называть примеры "MY\_NEW\_THIS" или "MY\_NEW\_THAT": "MY\_NEW\_MONSTERS" в данном случае вы должны начинать с "Z", это довольно сложно, но вы можете справиться с этим, удалив слово "NEW" и подчеркивания между словами.

С именем представления CDS намного проще, потому что у вас есть тридцать символов для игры. Однако и здесь давайте следовать предложенному SAP именованию, как показано ниже:

- Имя представления CDS начинается с буквы "I", чтобы указать, что это представление интерфейса.

- Имя представления CDS заканчивается символом "\_U", чтобы указать, что результирующий бизнес-объект является неуправляемым, что означает, что вы сами кодируете операции CRUD.

В середине постарайтесь, чтобы оставшаяся часть имени представления CDS была как можно ближе к имени основного представления SQL, насколько это возможно. На этот раз не нужно удалять подчеркивания, разделяющие слова.

Определение представления CDS наполнено аннотациями, которые управляют поведением пользовательского интерфейса. Вы можете иметь десятки для каждого поля в списке выбора. Самые важные из них (для того, чем вы сейчас занимаетесь) следующие:

- *@UI.FACET*

Здесь вы говорите, что собираетесь позволить пользователю щелкнуть строку в обзоре и вызвать появится подробный экран.

- *@UI.LINEITEM*

Здесь вы управляете тем, как поле отображается на экране обзора.

- *@UI.IDENTIFICATION*

Здесь вы управляете отображением поля на экране детализации.

- *@UI.SELECTIONFIELD*

Здесь вы указываете, какие поля будут отображаться в качестве фильтров (опций выбора) в верхней части экрана.

- *#FOR\_ACTION*

Это означает, что в верхней части экрана появится кнопка, позволяющая пользователю выполнить действие над выбранным монстром - в данном случае заставить его выть на луну.

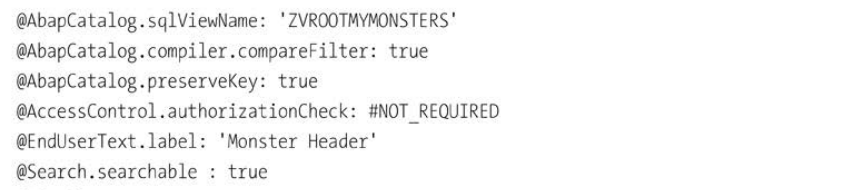
808

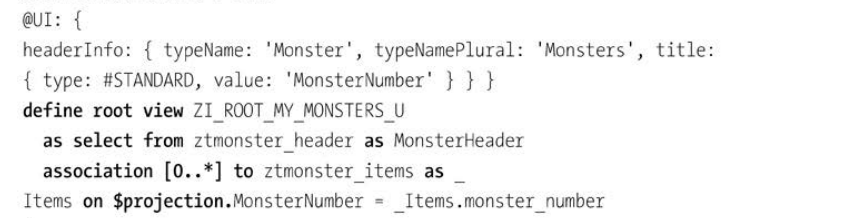
- *@SEARCH.SEARCHABLE*

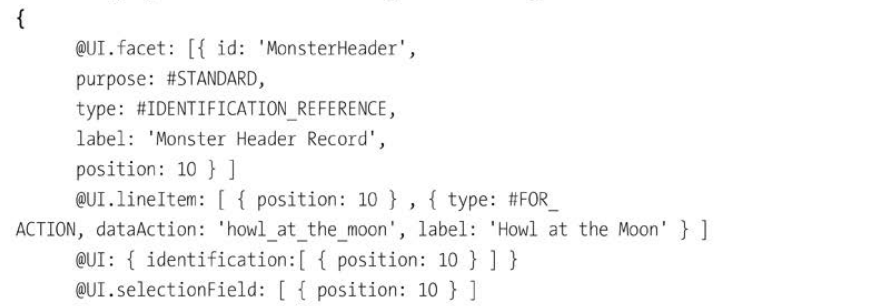
Это заставляет появиться поле вверху, чтобы включить поиск по столбцам, подобно бинокль на экране ALV.

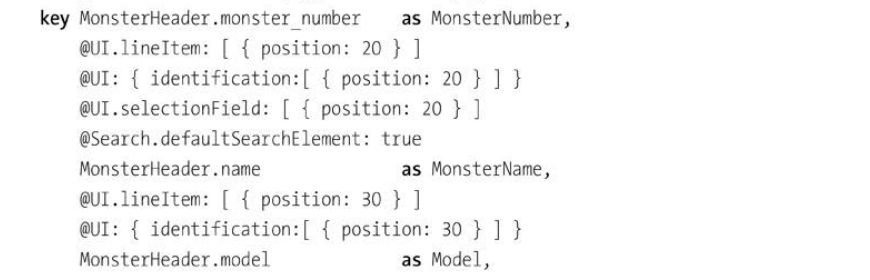
- *@SEARCH.DEFAULTSEARCHELEMENT*

Это означает, что вы хотите включить этот столбец в поиск по столбцам.

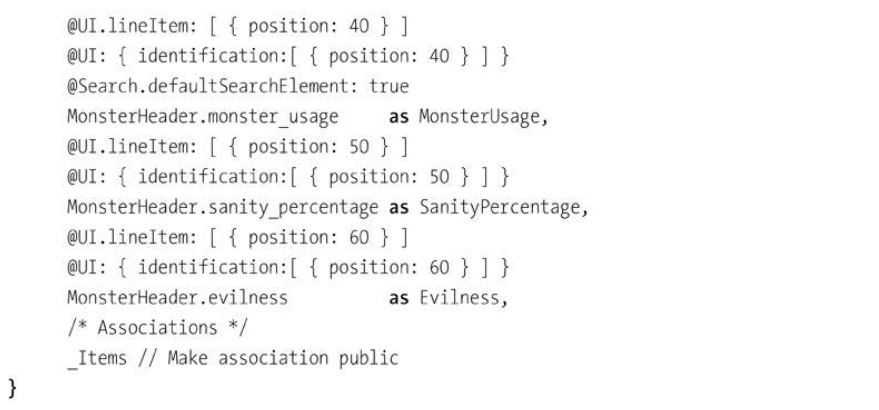








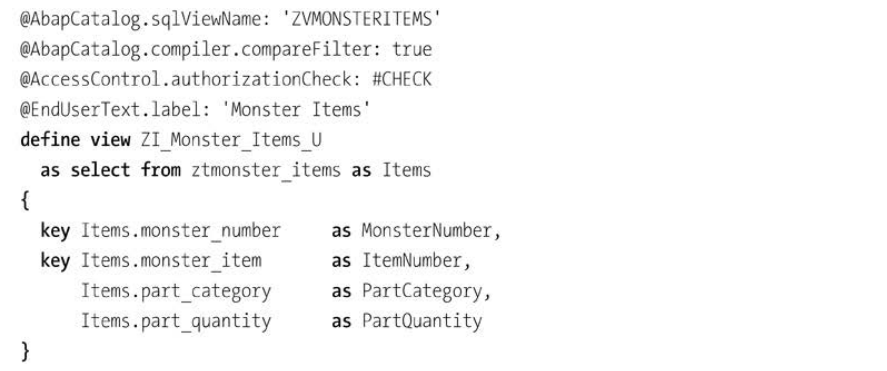
809



Listing 14.1 RAP Monster Root CDS View Defintion

В листинге 14.1 видно, что корневой узел монстра объявляет ассоциацию, такую, что монстр содержит от нуля до многих элементов. С точки зрения бизнеса, монстр, не содержащий ни одного элемента, является виртуальным монстром, потому что он еще не собран.

Поэтому, естественно, вам нужно создать представление CDS и для элементов монстра, то есть представление CDS будет ZI\_Monster\_Items\_U. Чтобы не усложнять ситуацию, представление CDS будет как можно более простым, просто перечисляющим все поля в основной таблице базы данных (листинг 14.2).



Listing 14.2 Monster Item CDS View Defintion

810

На этом с определениями вида CDS покончено. До сих пор, кроме новой идеи корневого представления, ничего не изменилось с того времени, когда представления CDS впервые появились - а это было более пяти лет назад. Теперь пришло время использовать эти представления CDS для создания полноценного бизнес-объекта-монстра.

### 14.4.2 Кодирование определения поведения

Определение поведения - это не более и не менее чем список действий, доступных для бизнес-объекта: CRUD-операции и любые пользовательские действия, специфичные для данного бизнес-объекта. Вы можете представить это как список публичных методов интерфейса, используемого Z-классом, который представляет бизнес-объект.

В RAP существует очень четкое разделение проблем, с целью сделать каждый компонент как можно более пригодным для повторного использования. В качестве примера, представление CDS монструозного элемента не знает, что его будет использовать корневое представление заголовка монстра, и поэтому может быть использовано любое другое представление CDS.

Корневое представление заголовка монстра не обязательно должно иметь определение поведения, но оно может иметь его (и только его) по желанию. В данном случае это именно то, что вам нужно.

Точно так же, как представление CDS определяется с помощью объекта ***DDL*** (Data Definition Language) в ABAP, поведение определяется с помощью ***BDL*** (Business Definition Language).

Чтобы создать такого зверя, зайдите в ABAP в Eclipse и выберите определение DDL для вашего корневого монстра CDS, откройте контекстное меню и выберите *New Behavior Definition*. Как вы можете себе представить, если ваша версия ниже 7.53, вы получите ошибку "не поддерживается".

Как показано на рисунке 14.1, у вас нет выбора в имени, которое выделено серым цветом. Единственное, что вы можете изменить, это описание, и предложенное описание имеет такой смысл, что менять его не стоит. Корневая сущность на рисунке 14.1, естественно, является рассматриваемым представлением CDS. Вы получаете красивую большую букву "B" в правом верхнем углу.

Нажмите кнопку NEXT, выберите транспортный запрос, и вы попадете на экран, который выглядит точно так же, как и тот, где вы вводите код определения представления CDS. На этот раз код, который вам нужно написать (как минимум), очень прост. Все, что вам нужно, это перечислить необходимые CRUD-операции (и вам не нужно указывать READ, потому что это само собой разумеется), хотя вам придется написать код этого поведения позже) и любые пользовательские действия.

811

В данном случае вы хотите, чтобы монстр мог выть на луну. Голый код для определения поведения показан в листинге 14.3; обратите внимание, что вы используете псевдоним, чтобы скрыть техническое имя со всеми маркерами "I" и "U" от внешнего мира.



Listing 14.3 Behavior Definiton

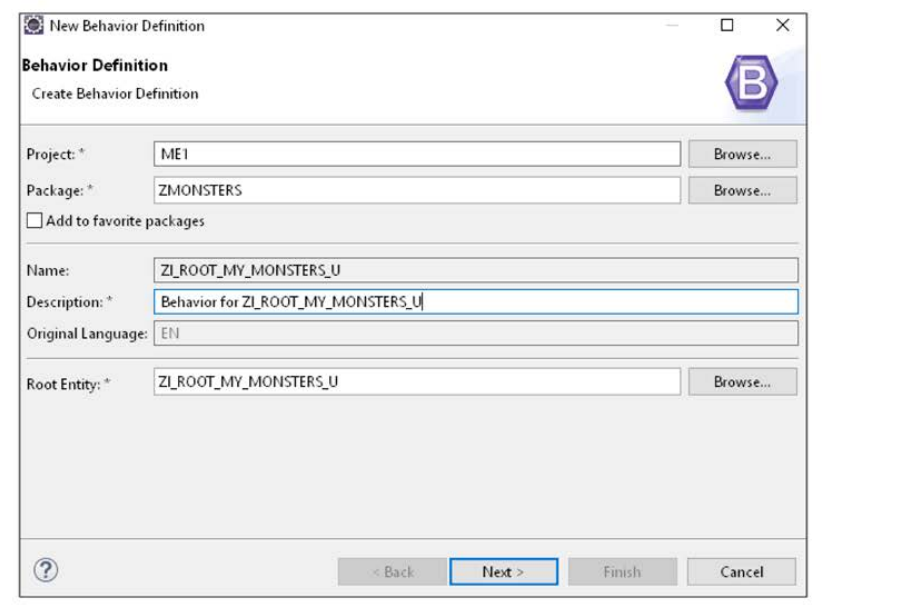


Figure 14.1 Behavior Defnition

Вам может быть интересно, что за странные вещи находятся в конце определения действия. Действие имеет входной и выходной параметры: на данный момент вы не говорите, что именно

812

входные и выходные параметры, а скорее то, к какому представлению CDS они относятся. В этом случае вы не хотите делать ничего причудливого, например, возвращать другой тип представления CDS в качестве результата, поэтому *$self* означает, что корневое представление CDS является входным и выходным параметром, а *[0 ... 1]* означает, что выходной параметр вам не нужен.

Позже вы будете добавлять дополнительные элементы к этому определению, но давайте пока не будем путаться. Чтобы попытаться связать определение поведения с ОО-программированием на данном этапе, вы объявили что-то похожее на определение класса, но класса еще нет (люди всегда обвиняют меня в отсутствии класса), так что сейчас самое время пойти и создать его.

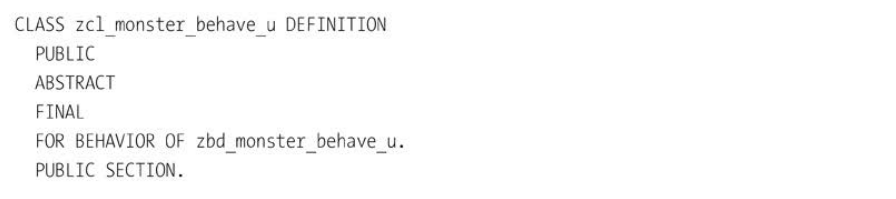
### 14.4.3 Кодирование реализации поведения

Класс, который вы собираетесь создать, - это не просто обычный класс или класс для сада. Это так называемый пул поведения. В большинстве случаев это обычный класс ABAP, но кодировать его совсем иначе, чем вы привыкли.

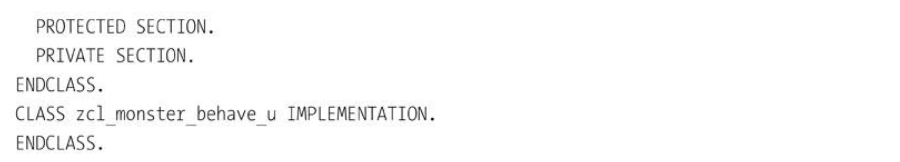
Вы создаете класс поведения (именно так я его называю; я не могу заставить себя использовать слово *pool*) с помощью мастера в ABAP в Eclipse. Сначала вам нужно выбрать определение поведения узла, который вы создали в предыдущем разделе, а затем использовать контекстное меню и выбрать *Create Behavior Implementation*.

Появившееся окно (рисунок 14.2) сообщит вам, что вы создаете ABAP-класс и спрашивает вас об имени этого класса, но вас также спрашивают об определении поведения, которое, конечно же, задается по умолчанию. Назовите этот класс *ZCL\_MONSTER\_BEHAVE\_U*, чтобы быть последовательным что было сделано ранее.

Нажмите кнопку Next, чтобы продолжить, и будет создан скелет класса ABAP. Возможно, вы помните ранее в этой книге, что вы можете определить класс как *FOR TESTING*; в данном случае он определен как *FOR BEHAVIOR*. Это будет самый пустой класс, который вы когда-либо видели в своей жизни (листинг 14.4), и вы оставите его таким: в RAP все кодирование выполняется в локальных классах в рамках глобального класса, хотите верьте, хотите нет. Вы заметите, что класс является абстрактным, поэтому вы не можете создать его экземпляр, потому что его методы не будут вызываться обычным способом, а скорее с помощью черной магии - что, честно говоря, так же работает и в BOPF.



813



Listing 14.4 Generated Behavior Defiintion Class



Figure 14.2 Behavior Implementation

В Eclipse вы даже не можете увидеть это определение (хотя оно там есть); вы переходите прямо на экран, где показаны локальные классы. Это происходит потому, что вы не должны всё испортить в определении глобального класса, поэтому вас принудительно останавливают от этого.

Что касается этих локальных классов, вы обнаружите, что не только пустая дефиниция класса на глобальном уровне, но и в разделе *local types* глобального класса два локальных класса также были сгенерированы в пустом состоянии.

814

При создании класса поведения создаются два локальных класса: один наследуется от CL\_ABAP\_BEHAVIOR\_HANDLER и имеет отношение к действиям пользователя в транзакции, а другой наследуется от CL\_ABAP\_BEHAVIOR\_SAVER, который, как это ни удивительно, имеет отношение к сохранению конечного состояния транзакции в базе данных.

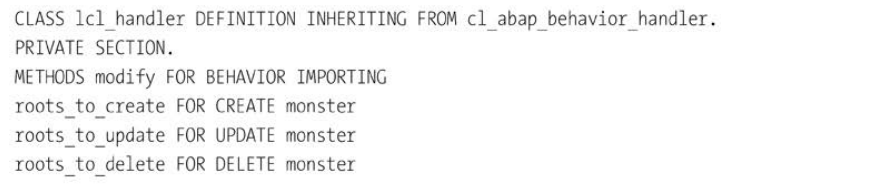
Почему существует два класса? Ну, традиционно, при кодировании программы Dynpro для создания/изменения бизнес-объекта вы храните данные транзакции в памяти (во внутренних таблицах), пока пользователь играет с транзакцией, а затем сохраняете эти данные в базу данных, когда пользователь нажимает кнопку "Сохранить". Никто не посчитает это коммерческой тайной. Как вы можете себе представить, в RAP все точно так же, с той лишь тонкой разницей, что вместо внутренней таблицы как таковой, текущее состояние измененного объекта хранится в транзакционном буфере (истребителе вампиров). Вы должны написать код такого буфера самостоятельно, и через минуту вы создадите локальный класс для буфера.

Я требую разговора с менеджером

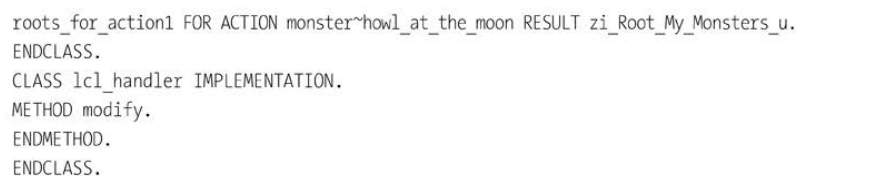
Обратите внимание, что кодирование таких вещей, как транзакционный буфер и методы CRUD вручную необходимо только для неуправляемого сценария, который является единственным способом на данный момент вы можете программировать в RAP. В одном из будущих выпусков будет также доступен управляемый состаренный сценарий, в котором много кода, относящегося к методам CRUD и транзакционного буфера, будет генерироваться автоматически.

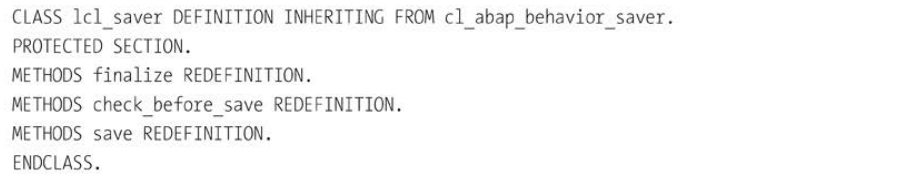
Важно то, что вы не можете создавать экземпляры вручную или вызывать какие-либо из методов в классе LCL\_HANDLER или LCL\_SAVER. Это может сделать только фреймворк.

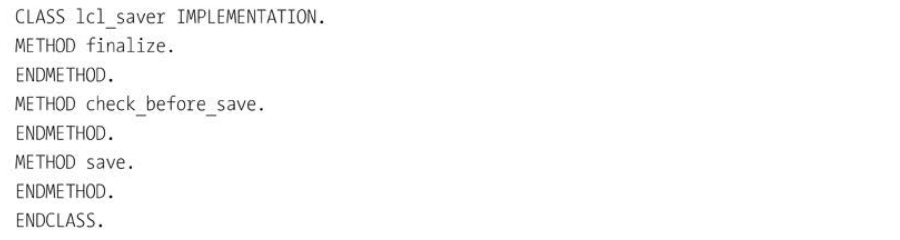
SAP придумала забавные названия для этих двух классов, а именно: фаза взаимодействия – *interaction phase* (в которой вы возитесь с буфером транзакции) и последовательность сохранения - *save sequence* (в которой вы сохраняете буфер в базе данных). Сгенерированный код будет выглядеть как в листинге 14.5. Обратите внимание на повторное использование слова *ROOTS*, которое не имеет никакого отношения к Алексу Хейли; вместо этого оно означает, что каждое действие может быть применено к одному или нескольким бизнес-объектам (монстрам, в данном случае) одновременно, в случае, если вы выбираете шесть монстров из списка и затем нажали кнопку, чтобы заставить их всех завыть на луну. Метод *MODIFY* обрабатывает все поведение, которое вы определили.



815







Listing 14.S Generated Local Classes

Код в листинге 14.5 был сгенерирован системой, и в последующих разделах вы увидите, как адаптировать его к текущему случаю использования.

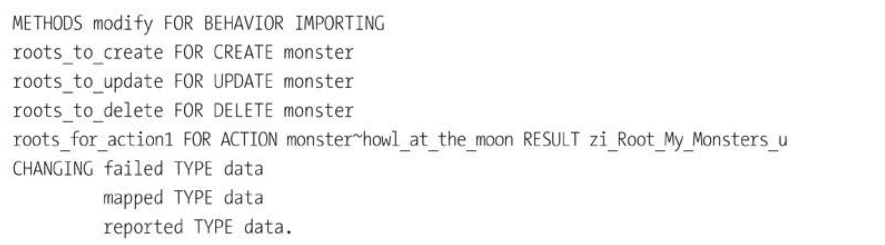
Возможно, сгенерированный код будет использовать имя представления CDS (ZI\_ROOT\_MY\_MONSTERS\_U), а не псевдоним (MONSTERS). Если это так, система немедленно выдаст ошибку и вам придется изменить имя на псевдоним.

Однако сейчас я хотел бы обратить ваше внимание на очень странную часть программы RAP. Глядя на определение метода *MODIFY*, можно подумать, что у него есть только параметры *IMPORTING*. На самом деле, у него есть три изменяющих параметра, но вы не должны их объявлять! Это так называемые неявные параметры; они являются частью сигнатуры и добавляются автоматически и незаметно.

Мне эта идея ни капельки не нравится. К счастью, нет никаких правил, запрещающих явно добавить три параметра в определение, чтобы было более очевидно, что происходит. Подпись пересмотренная сигнатура выглядит как в листинге 14.6. Все три параметра - это таблицы: *FAILED* хранит все бизнес-объекты, которые не удалось изменить, *MAPPED* хранит объекты, которые были созданы, а *REPORTED* содержит сообщения об ошибках. Позже в этом разделе, по мере того как вы будете работать

816

через возможные методы транзакционного поведения, вы узнаете, как заполнить их правильно. Обратите внимание, что все они типизированы обобщенно. При выполнении проверки синтаксиса компилятор проверяет фактический список полей из представления CDS.



Listing 14.6 MODIFY Defintion with Implicit Parameters Made Explicit

Наиболее зоркие из вас заметят, что сгенерированный код не обеспечивает все операции CRUD; операция *READ* отсутствовала. Мы решим эту проблему позже.

Теперь давайте перейдем сначала к кодированию транзакционного буфера для хранения текущего состояния бизнес-объектов монстра, а затем к рассмотрению различных методов, вызываемых основным методом *MODIFY*: создание, обновление и так далее.

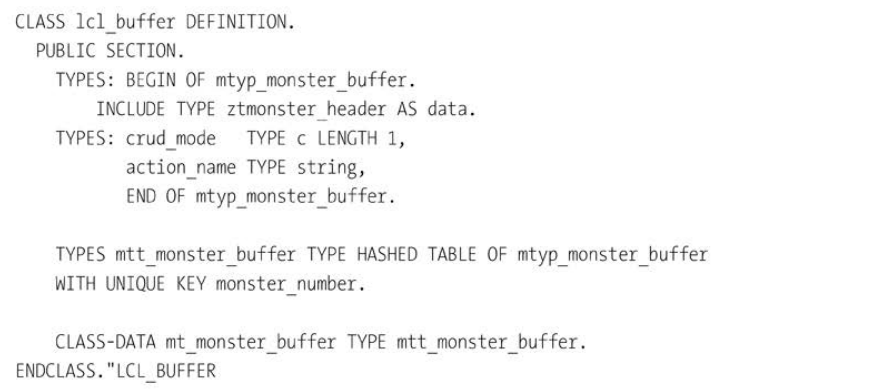
**Буферный класс**

Вся работа в RAP выполняется локальными классами, поэтому буферный класс также будет локальным классом тоже. Вы можете назвать его как угодно, хотя бы LCL\_BUFFER, вероятно, имеет больше смысла, чем LCL\_ BANANA. В традиционном мире SAP вы привыкли к структурам типа VBAPVB, в которой есть все поля VBAP плюс поле "знак" в конце, в котором говорится, что происходит с записью. Эта структура используется для определения внутренней таблицы в транзакции ***VA01***, чтобы она знала, что делать во время сохранения:

какие строки добавить, какие изменить, а какие удалить. Нет необходимости изобретать колесо и делать его квадратным, поэтому определение класса буфера включает в себя создание такого определения - но я не против написать на новом колесе "Я - колесо", так что вместо бессмысленного термина типа SIGN или FLAG, мне нравится называть односимвольное поле в конце CRUD\_MODE. Поскольку могут существовать как действия, так и методы CRUD, было добавлено дополнительное поле было добавлено для хранения имени любого выполняемого действия (действий).

Добавьте кодировку, подобную листингу 14.7, в самом начале локальных классов. перед определением класса LCL\_HANDLER. Класс не имеет методов, поэтому нет соответствующей реализации. Таблица буферов является статической, поэтому никогда не нужно создавать экземпляр класса класса буфера.

817



Listing 14.7 Class LCL\_BUFFER Defintion

**Создание**

Теперь вам предстоит вручную закодировать различные методы CRUD. Для того, чтобы еще больше удручить вас, в следующем релизе будет управляемый сценарий, с помощью которого животные леса будут фактически генерировать весь код создания/обновления/удаления за вас.

Но пока что вам придется делать это вручную, поэтому сейчас самое время рассмотреть некоторые новые ABAP конструкции. В листинге 14.8 вы увидите, как некоторый код добавляется в раздел *PRIVATE* локального класса LCL\_HANDLER. Во-первых, у вас есть куча определений данных, которые все выглядят идентичны и в каждом случае определяют таблицу экземпляров корневого представления CDS. Они затем используются в методе *CREATE*. В метод входят детали монстра (монстров), которого вы хотите создать. Допустим, вы создаете шесть монстров и у вас есть три неудачи. Данные успешных монстров хранятся в таблице *MAPPED*, а неудачи попадают в таблицу под названием *FAILED*. Если бы таблица успехов имела в своем названии фразу типа SUCCESS, то вы могли бы догадаться, что она означает, и где было бы веселье в этом? Загадочные термины типа MAPPED гораздо более забавны. На самом деле, это не так глупо, как кажется. Во время создания у нового монстра еще нет номера, поэтому система присваивает ему временный номер, например, %SADL\_CID\_1. После успешного создания монстр получает новый номер, который сохраняется в базе данных, а таблица сопоставления связывает (сопоставляет) временный номер монстра с окончательным номером монстра.

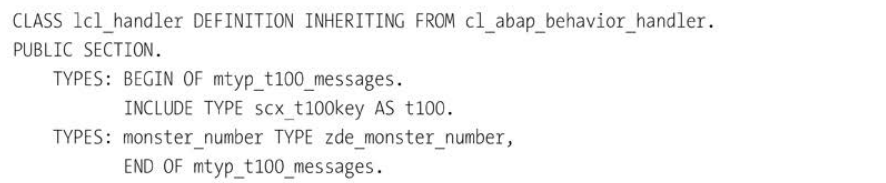
Скорее всего, вам также понадобится немного информации о том, почему некоторые монстры не удалось создать, поэтому в сигнатуре метода *create* есть таблица сообщений. Вы можете напечатать ее как угодно, например, в виде таблицы T100 или BAPIRET2 -

818

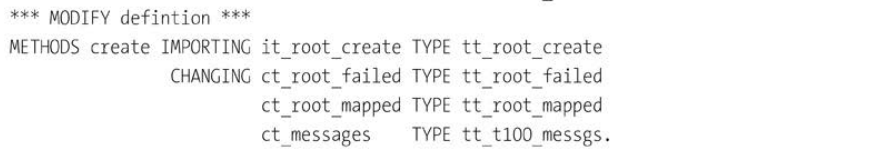
любая структура возврата, которую вы использовали при создании многократно используемого класса бизнес-объекта. В классе модели монстра ошибки возвращаются как экземпляры классов исключений классов, которые реализуют интерфейс T100, поэтому таблица *SCX\_T100KEY* - это то, что будет использоваться в данном примере. Структура *SCX\_T100KEY* включена в белый список ABAP в облаке, поэтому вы можете использовать ее, и вы хотите добавить номер монстра, чтобы обеспечить связь между сообщением об ошибке и бизнес-объектом, который сгенерировал это сообщение. Но вы не можете (пока) создавать типы таблиц Z в ABAP в облаке, поэтому вы определяете пользовательский тип таблицы тип, TT\_ROOT\_MESSGS.

Обратите внимание, что вместо *EXPORTING* параметров, что можно было бы вполне ожидать, результаты возвращаются через *CHANGING* параметров. Это происходит потому, что вы можете смешивать и сопоставлять операции - создать монстра, затем изменить его во время одной и той же транзакции - так что

результаты каждой операции объединяются.







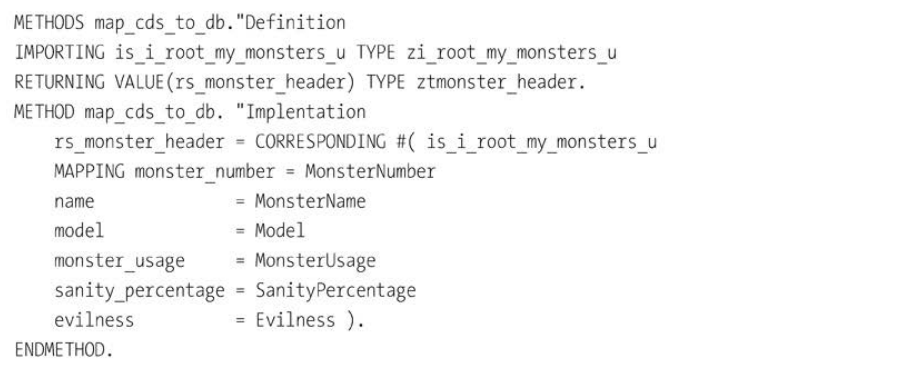
Listing 14.8 Definition of CREATE Method

То, что происходит дальше, просто уморительно. Вы помните, что лучшей практикой является определение псевдонимов в представлениях CDS, потому что вы не хотите, чтобы внешний мир видел такие странные вещи, как KUNNR или AUFNR; вы хотите, чтобы вместо них они видели Customer и Internal Order.

Проблема в том, что для обновления базы данных вам нужно сделать обратное: данные, поступающие с экрана, используют значимые имена, которые нужно сопоставить обратно

819

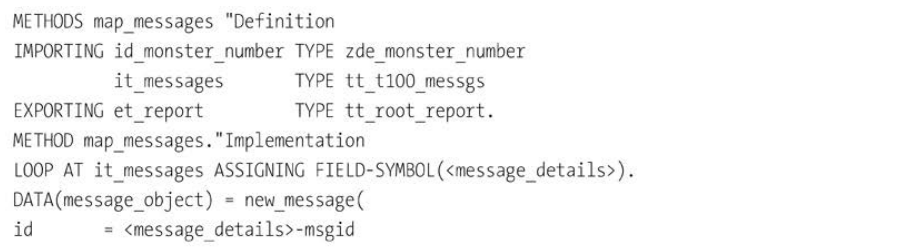
с именами полей базы данных. Это дает прекрасную возможность в Листинге 14.9, чтобы увидеть, как новое ключевое слово *CORRESPONDING* ABAP используется в гневе.



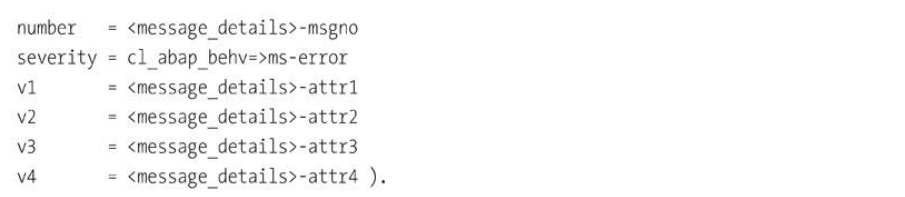
Listing 14.9 Mapping CDS Field Names to Database Field Names

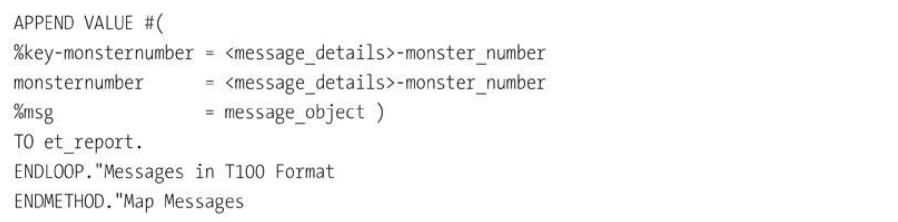
Если вы программируете в ABAP уже несколько лет, вы, несомненно, знаете о том, что распространенной задачей отображения сообщений из одного формата (например, SYST) в другой (например, BAPIRET2). Причина, по которой вам приходится делать это на регулярной основе, заключается в том, что SAP удалось придумать буквально миллион различных сокращений для номера сообщения: MSGNO, MSGNR и так далее. Теперь у нас есть новая структура сообщений для RAP, содержащая точно такую же информацию, как и все десятки структур сообщений об ошибках, которые были до этого. И вы будете в восторге, узнав, что SAP удалось придумать новые вариации для всех имен полей еще раз. Это действительно гениальная работа.

В листинге 14.10 показано, как строится так называемая таблица ошибок отчета. Каждая строка дважды содержит первичный ключ (в данном случае номер монстра), за которым следует объект, содержащий сообщения об ошибках. Метод NEW\_MESSAGE доступен благодаря тому, что класс LCL\_HANDLER наследуется от суперкласса.



820





Listing 14.10 Mapping T100 Messages to RAP Format

Далее обратимся к методу *CREATE*, который будет вызван методом *MODIFY*. Во многом это просто еще одна вариация того, что вы видели в главе 7 с BOPF или в главе 12 с сервисом SEGW, который потребляется SAPUI5. В обоих случаях фактический метод *CREATE* обрабатывается многократно используемым классом-монстром, который полностью отделен от использующей его технической структуры. Здесь действует точно такой же принцип, и снова это просто вопрос отображения фреймворка и специфических структур в общие структуры, используемые классом модели монстра, и обратно.

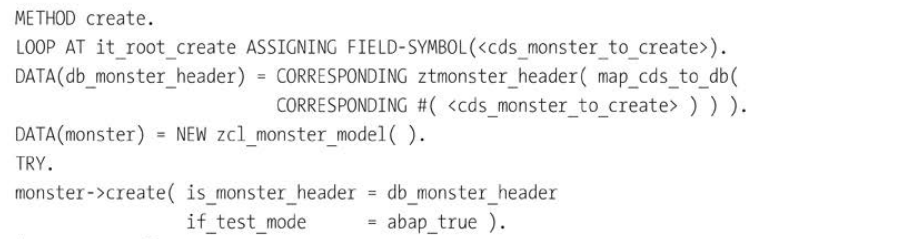
В методе *CREATE* (листинг 14.11) сначала данные отображаются из формата CDS, специфичного для представления в формат таблицы базы данных. Вы заметите в строке, которая выполняет сопоставление что ключевое слово *CORRESPONDING* встречается дважды подряд. Одно из них сопоставляет поля из входящего представления CDS в импортирующий параметр метода сопоставления, а другое сопоставляет поля из возвращаемого параметра метода сопоставления в локальную структуру. Это прекрасный пример использования нового синтаксиса ABAP, который, хотя и является очень компактным, может иметь тенденцию полностью скрывать то, что происходит в мире.

Далее вызывается класс модели монстра для фактического создания монстра (в тестовом режиме, как это делается с BAPI, потому что вы пока не хотите реального COMMIT). Наконец, мы обрабатываем ошибки и успехи.

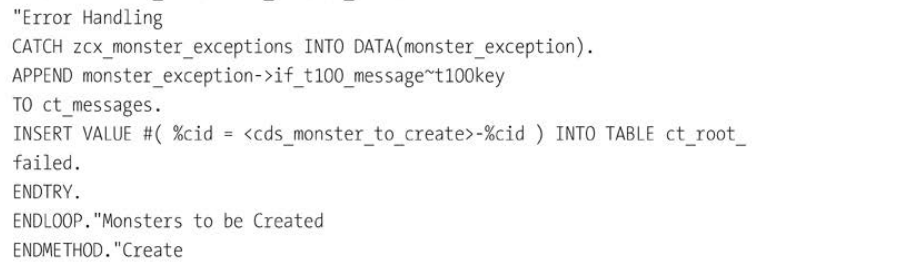
Вы заметите, что в таблицах FAILED и MAPPED заполняется переменная забавного вида. Она называется %CID и не имеет никакого отношения к британской полиции, а представляет собой временный идентификатор, который монстр получает до того, как у него появится свой номер.

821

Если запись может быть создана успешно - она передается в таблицу *MAPPED*, чтобы связать фактический номер монстра, который попадет в базу данных, с временным номером и с транзакционный буфер, чтобы он мог быть создан по-настоящему во время обработки *SAVE*.





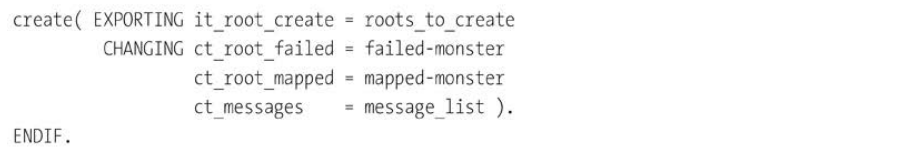


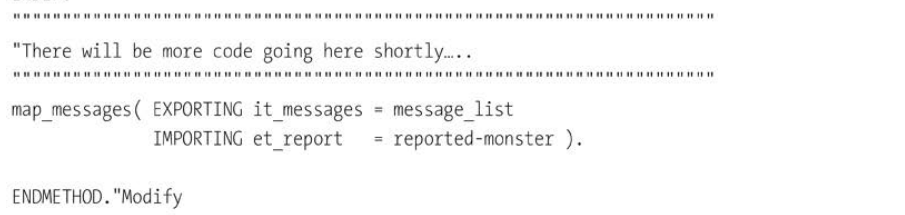
Listing 14.11 Implementation of CREATE Method

Ранее, в листинге 14.5, вы объявили определение метода *MODIFY*, который заботится обо всех поведении, таких как создание, обновление, удаление и любые пользовательские действия. Теперь у нас достаточно определенных методов, чтобы начать собственно кодирование метода *MODIFY*. В листинге 14.12, выполняется вызов метода *CREATE*, а затем вы выполняете отображение любых возвращаемых сообщений в таблицу *REPORTED*, чтобы они могли быть возвращены в пользовательский интерфейс для чтобы фреймворк совершил черную магию и показал ошибки конечному пользователю.



822





Listing 14.12 Calling CREATE Method from MODIFY Method

**Обновление**

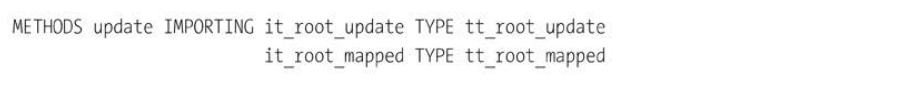
Позже, в разделе 14.5.1, вы увидите, что при использовании опции *Preview* в определении службы вы получаете практически полностью функционирующее приложение SAPUI5, по крайней мере, в том смысле, что появляется список монстров и вы можете перейти к одностраничному представлению монстра. В списке обзора волшебным образом появляется кнопка *Create*, а при просмотре отдельного монстра вверху появляется кнопка *Edit*, а внизу - кнопка *Save*. Чтобы заставить работать функцию сохранения после редактирования, необходимо написать метод *UPDATE*.

Первым шагом будет добавление нового определения данных в начало метода LCL\_HANDLER следующим образом:

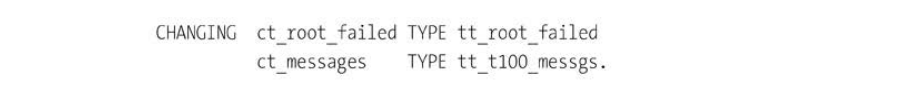


Далее вы пишете определение метода *UPDATE*. Как видно из листинга 14.13, оно на 99,9% идентично определению метода *CREATE*. Одно отличие заключается в том, что таблица *MAPPED* на этот раз является параметром *IMPORTING*, а не *CHANGING*, поскольку при обновлении вы не создаете временные номера монстров во время обработки, которые будут отображены на конечные номера монстров при сохранении.

Однако можно предположить (но не очень вероятно), что вы одновременно создаете и изменяете бизнес-объект. В этом случае результат операции создания (который будет находиться в таблице сопоставления) нужно передать в операцию обновления.



823



Listing 14.13 Defintion of \_UPDATE Method

Следующая концепция покажется очень знакомой программистам старой школы ABAP. Если вы хотите использовать BAPI для обновления заказа на продажу или чего-то подобного, то вы должны сообщить BAPI, какие именно поля вы изменяете, причина в том, что изменение значение поля на пустое является действительным изменением, и вы хотите избежать ситуации, в которой поля, для которых вы предоставили новые значения, изменяются, а все остальные поля остаются пустыми.

Как вы знаете, решение заключается в том, чтобы иметь структуру BAPEX с теми же именами полей, что и в структуре BAPE, но при этом каждое поле должно быть определено как поле с одним символом. Затем вы ставите крестик напротив полей, которые хотите изменить, а не отмеченные поля BAPI оставляет в покое.

Вы будете рады узнать, что в RAP это работает на 100% так же. Единственное отличие заключается в том, что вместо того, чтобы вручную определять структуру BAPEX через SE11, она автоматически генерируется системой (больше черной магии) на основе определения представления CDS. Внутри IT\_ROOT\_UPDATE в каждой строке будет компонент под названием %CONTROL, и это структура возможных имен полей, которые могут быть изменены, каждое из которых набирается как поле с одним символом.

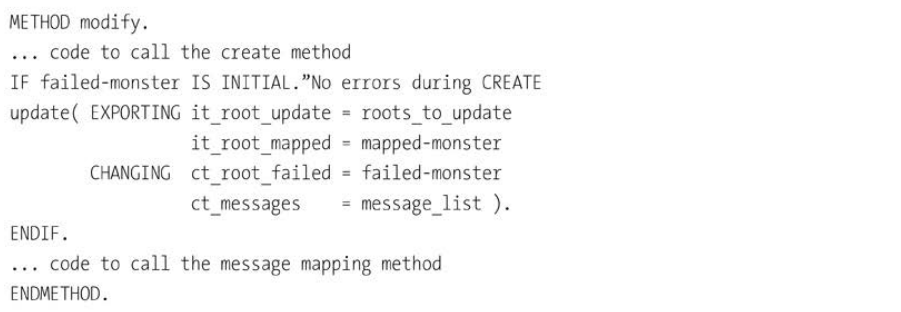
Точнее говоря, каждая строка таблицы обновления начинается с поля %CID\_REF, которое хранит временный ID объекта, если он еще не создан в базе данных, затем все поля представления CDS, и, наконец, структура %CONTROL со всеми именами полей, на этот раз в виде элемента данных ABP\_BEHV\_FLAG, который сам по себе не является булевым, а хранит константы типа CL\_ABAP\_BEHV=>FLAG\_CHANGED.

Еще в главе 3 вы узнали о новой конструкции XSDBOOL ABAP, которая превращает логическое выражение либо в ABAP\_TRUE, либо в ABAP\_FALSE. Вот идеальная возможность увидеть это в действии, когда вы определяете, какие поля будут обновлены, когда окончательная структура передается в класс многоразовой модели монстра.

В листинге 14.14 видно, что код для вызова метода UPDATE добавлен к методу MODIFY, сразу после вызова метода CREATE. Вы должны убедиться, что что метод CREATE не имел сбоев перед вызовом метода UPDATE, а если метод CREATE был вызван, то таблица MAPPED, возвращенная из метода CREATE передается в метод UPDATE.

824

Поскольку таблица сообщений является изменяемым параметром, любые сообщения из метода UPDATE добавляются к сообщениям из метода CREATE, а в конце метода MODIFY они отображаются в последний параметр сообщения, возвращаемый в пользовательский интерфейс.



Listing 14.14 Adding \_UPDATE Method Call to MODIFY Method

В листинге 14.15 показан код самого метода UPDATE. В этом случае нет необходимости сопоставлять поля представления CDS с полями базы данных, чтобы получить структуру, содержащую новые значения после того, как пользователь изменил некоторые из них. Вместо этого вы будете получать существующую запись в формате базы данных либо из буфера, либо из самой базы данных.

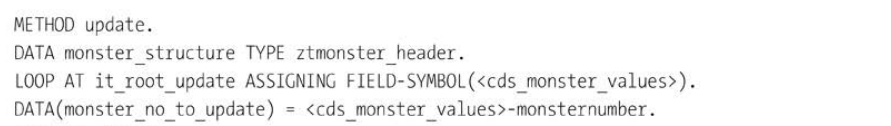
Итак, первое, что происходит в методе, - это некоторая подколка, чтобы справиться с ситуацией, когда изменяемый объект пока только создан в памяти.

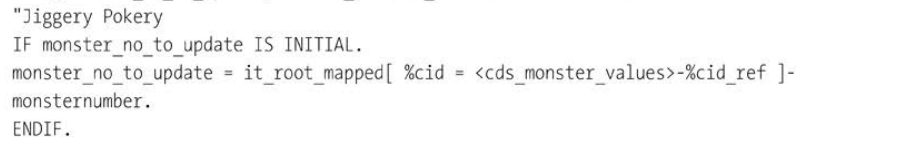
Затем присваивается символ поля, указывающий на существующую или новую строку в транзакционном буфере, которая представляет запись, подлежащую изменению. Если такой записи нет, то вы получаете ее из базы данных и присваиваете ей символ поля.

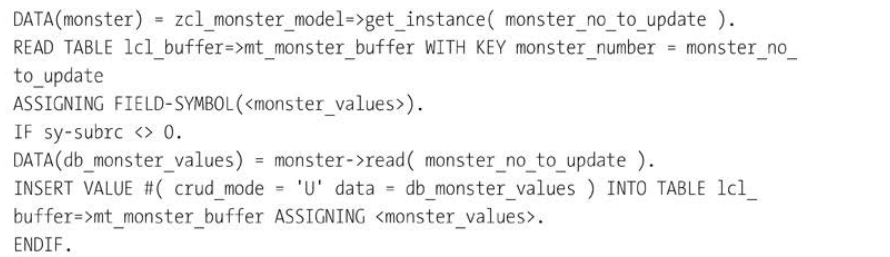
Таким образом, отправной точкой является символ поля, содержащий существующие значения, и любое отдельное значение в этой структуре перезаписывается новым значением только в том случае, если управляющая структура говорит, что оно было изменено - именно здесь вы используете своего друга XSDBOOL.

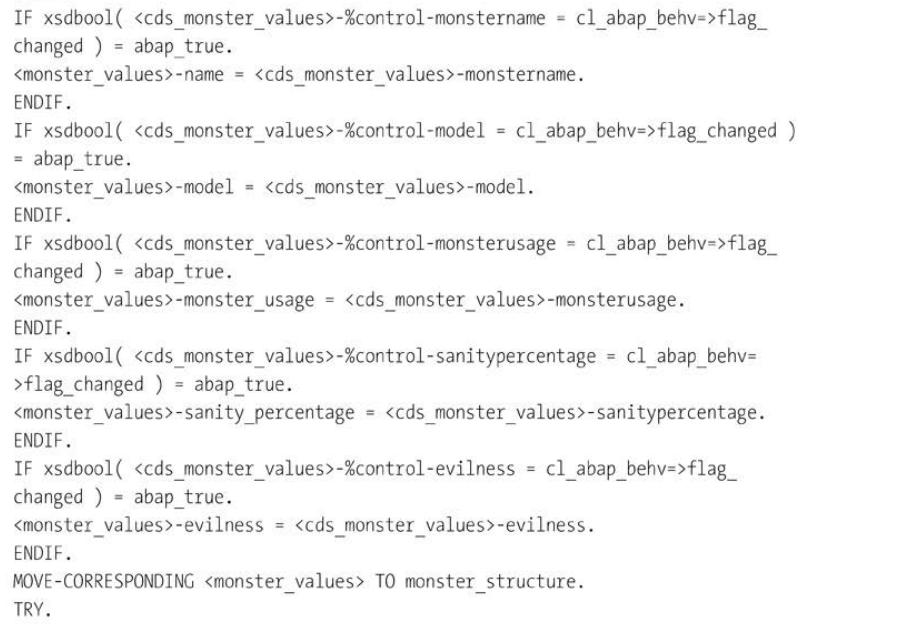
Далее следует вызов метода CHANGE класса модели монстра - снова в тестовом режиме. Если обновление прошло успешно, то ура, у нас уже есть новая обновленная запись в транзакционном буфере; если нет, то запись удаляется из транзакционного буфера, а все такие записи, которые потерпели неудачу при обновлении, передаются из метода путем добавления в таблицы параметров CHANGING - FAILED и CT\_MESSAGES.

825









826



Listing 14.15 Implementation of UPDATE Method

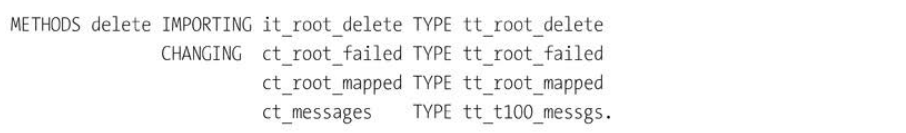
**Удаление**

Здесь фреймворк волшебным образом создаст две кнопки Delete: одна находится на экране обзора, где вы можете выбрать монстра и удалить его, а другая - на экране детализации, когда вы просматриваете отдельного монстра (звучит болезненно для монстра).

Как и раньше, первым шагом будет добавление нового определения данных в начало класса LCL\_HANDLER следующим образом:



Я уверен, что вы уже поняли, что это за шаблон. На этот раз определение класса \_DELETE имеет три параметра CHANGING, такие же, как и метод CREATE, предположительно потому, что маловероятно, что вы захотите создать бизнес-объект и удалить его в одной транзакции. Тем не менее, как вы увидите через минуту, вы все равно проверяете эту ситуацию в коде метода DELETE. И снова FAILED означает то, что вы подумали, а MAPPED - успех. Определение класса *delete* (который вы также добавляете в секцию определения LCL\_HANDLER) показано в листинге 14.16.

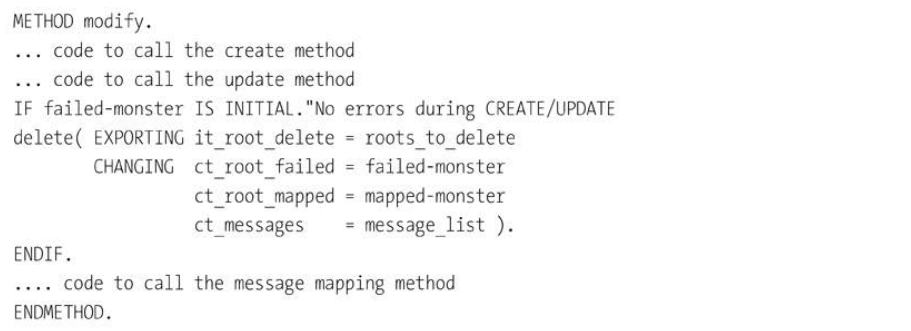


Listing 14.16 Defintion of DELETE Method

827

Как вы можете себе представить. это самое простое из всех поведений, которые вы до сих пор кодировали. потому что в программировании, как и в жизни, к сожалению, гораздо легче разрушать вещи, чем создавать.

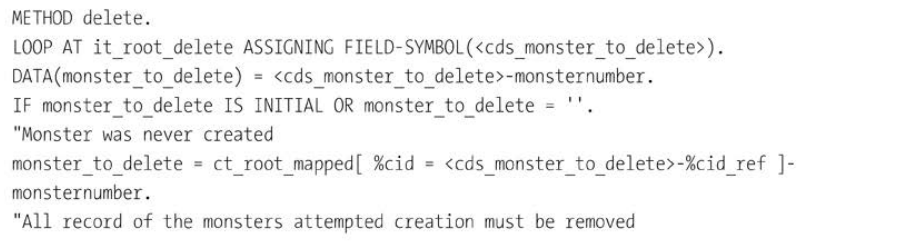
Как и раньше, вызов метода DELETE добавляется к методу MODIFY после вызовов методов CREATE и UPDATE, как показано в листинге 14.17.



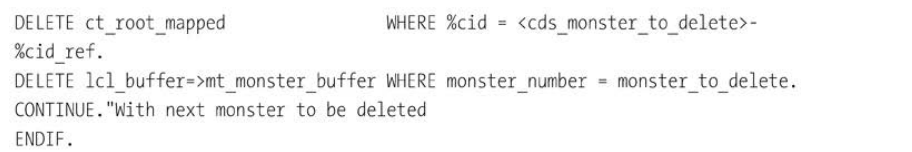
Listing 14.17 Adding DELETE Method to MODIFY Method

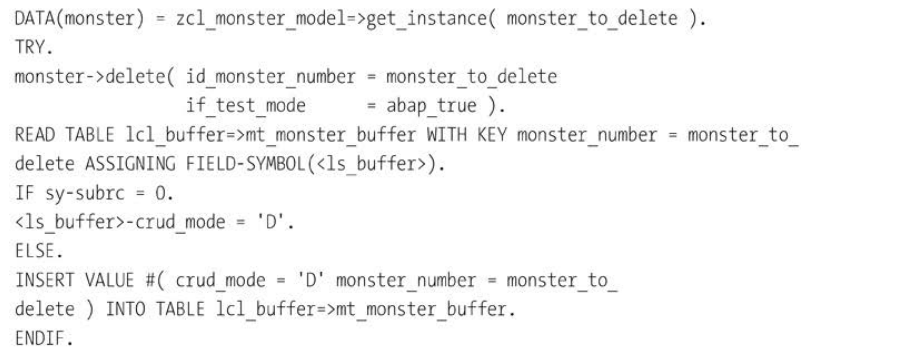
Код для удаления объекта монстра показан в листинге 14.18. Сначала задается номер удаляемого монстра. Если он пустой, то монстр еще не создан, поэтому извлекается временный номер, а затем данный монстр стирается из таблицы MAPPED (и транзакционного буфера), поскольку удалять несуществующего монстра бессмысленно.

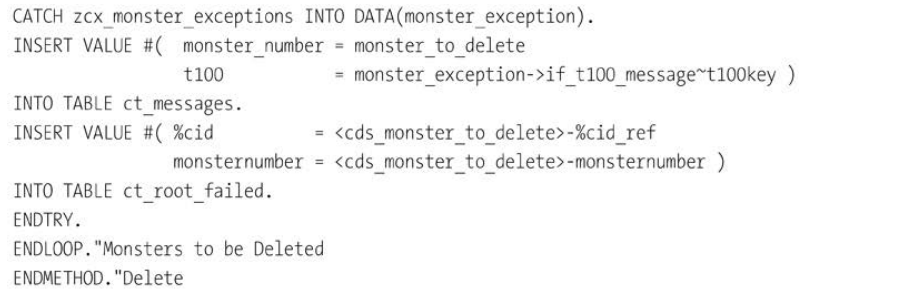
Далее следует вызов класса модели монстра для удаления монстра (в тестовом режиме), а затем та же обработка успехов/ошибок, которую вы видели в двух предыдущих разделах. Здесь необходимо учесть возможность того, что монстр был изменен, а затем удален в одной транзакции. В этом случае значение CRUD\_MODE меняется с U на D.



828







Listing 14.18 Implementation of \_DELETE Method

**Действия**

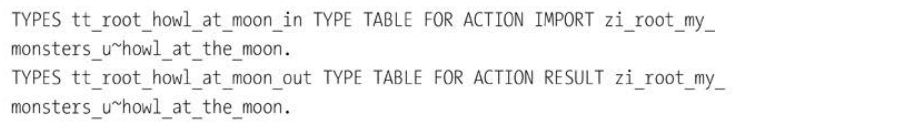
Другие модели поведения, которые вы уже рассмотрели, являются общими для многих различных типов бизнес-объектов: все они должны быть созданы, обновлены и (иногда) удалены.

Действия, с другой стороны, почти всегда специфичны для определенного типа бизнес-объектов. В данном случае действие - это вой на луну, что имеет смысл для монстра, но не очень для бизнес-объекта в модуле FI, представляющего основные средства, или бизнес-объекта в модуле HR, представляющего менеджера. (Если, конечно, менеджер, о котором идет речь, на самом деле не монстр - что встречается довольно часто - или не безумный лай - что встречается столь же часто - или и то, и другое).

829

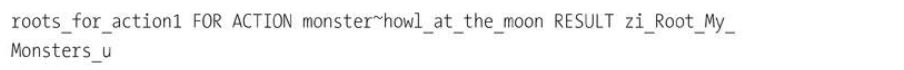
Когда вы определяете действие в определении поведения и добавляете соответствующую аннотацию в определение представления CDS, на экране обзора пользовательского интерфейса волшебным образом появляется кнопка с названием вашего действия. Затем вы можете выбрать одного или нескольких монстров и заставить их завыть на луну. Естественно, сначала нужно написать код.

Первый шаг - добавить еще два определения TYPE в начало класса LCL\_HANDLER. Одно из них - это таблица импорта действия, а другое - таблица экспорта, и они определяются следующим образом:



И снова вы выбираете имена таблиц, но определение строк этих таблиц генерируется системой. Первым компонентом является временный идентификатор (%CID\_ REF), затем идет фактический первичный ключ (ключи) из представления CDS, например, MonsterNumber. Тип экспортируемой строки имеет дополнительный параметр, остроумно названный PARAM, для отправки результатов, таких как количество воплей, которые действительно имели место.

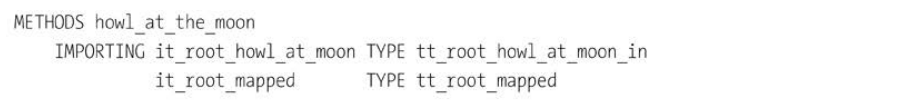
Когда класс LCL\_SAVER был впервые создан, результат для действия был задан как общее значение, которое было просто именем представления CDS, например, так:



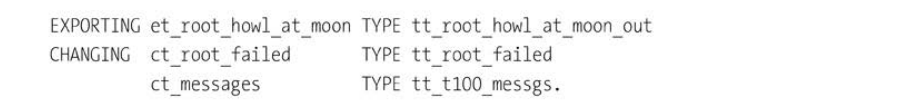
Теперь измените это, чтобы отразить тип таблицы экспортируемых параметров, который вы только что определили:



ROOTS\_FOR\_ACTION1 содержит список выбранных монстров для действия и типизируется на основе таблицы IMPORTING, определенной ранее. Следующий шаг - создание определения для метода HOWL\_AT\_THE\_MOON. Он содержит параметры IMPORTING и EXPORTING, набранные на основе двух новых определений, бонусный параметр IMPORTING с отображением временных ID на фактические ID, и два параметра CHANGING для хранения объектов (монстров), которые не удалось сохранить. объектов (монстров), которые не смогли завыть, а также любые сообщения об ошибках. Результат выглядит как в листинге 14.19.

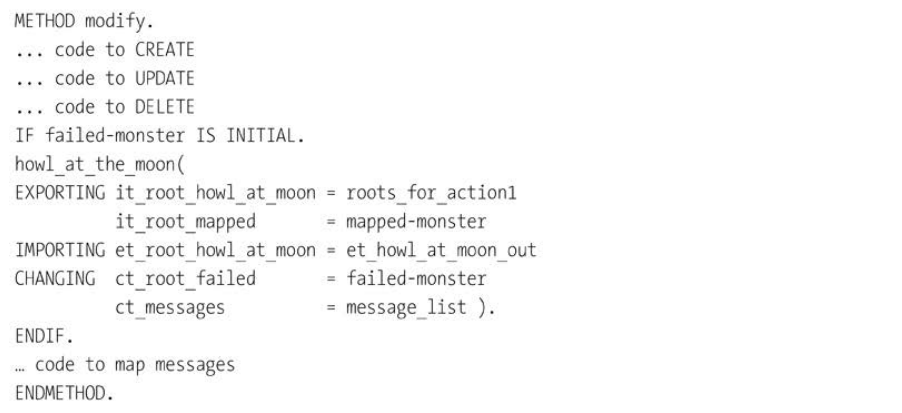


830



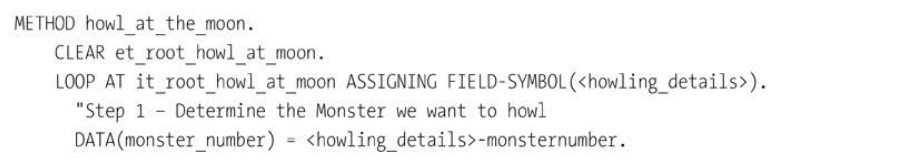
Listing14.19 HOWL\_AT\_THE\_MOON Method Definition

На этом этапе вы можете догадаться, каковы два следующих шага: все верно, далее вы добавляете некоторый код в основной метод MODIFY после уже закодированных методов поведения и перед отображением сообщений в конце, на этот раз для вызова метода *howl* *at the moon* метод. Вы можете увидеть это в листинге 14.20

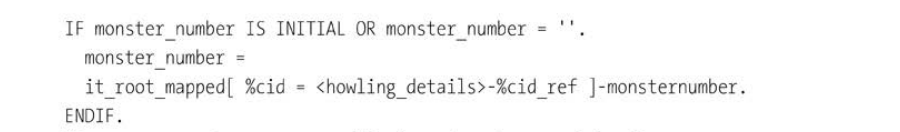


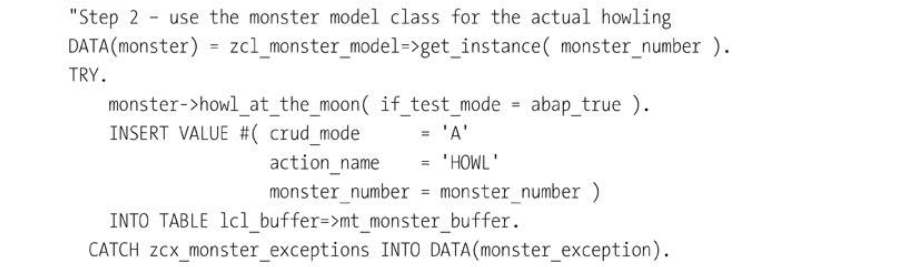
Listing 14.20 Adding HOWL\_AT\_THE\_MOON Method to MODIFY Method

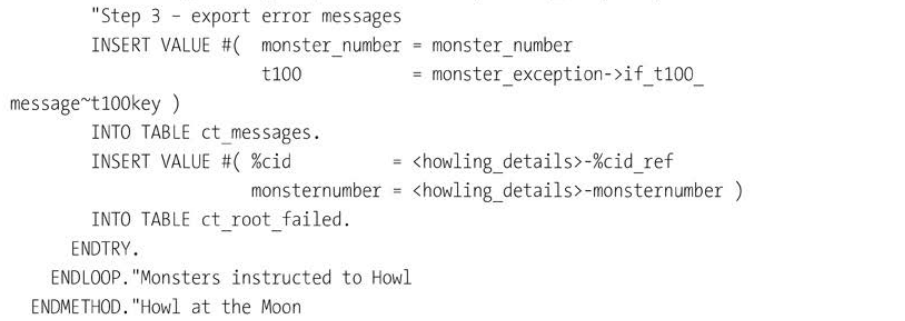
Как и раньше, при кодировании метода действия вы передаете фактическую работу классу модели монстра. Она просто обертывается в код, специфичный для фреймворка: сначала уже знакомая нам подколка для получения первичного ключа, если монстр еще не создан, затем вызов класса модели монстра, затем заполнение таблиц сообщений и неудачных монстров. В этом случае у монстра может не быть головы и, следовательно, он не может завыть, поэтому будет выдано исключение, содержащее сообщение о том, что будет вызвано исключение, содержащее соответствующее сообщение. Результат показан в Листинг 14.21.



831







Listing 14 .21 Implementation of HOWL\_AT\_THE\_MOON Method

Как видите, я добавил фиктивное значение в поле CRUD - а именно, A для действия. Некоторые действия обновляют базу данных, а другие - нет, но для подстраховки вы выполняете их все в конце транзакции вместе с добросовестными операциями CRUD.

**Чтение**

При генерации кода для класса LCL\_HANDLER был сгенерирован метод MODIFY, но не метод READ. Поэтому вам придется добавить его вручную. Опять же, здесь есть так называемые неявные параметры IMPORTING и CHANGING; проверка синтаксиса не будет жаловаться, если вы их не объявите, но я предпочитаю объявлять их явно, чтобы было очевидно, что они там есть. Меня очень смущает, когда данные передаются в параметр, который не указан в сигнатуре метода.

832

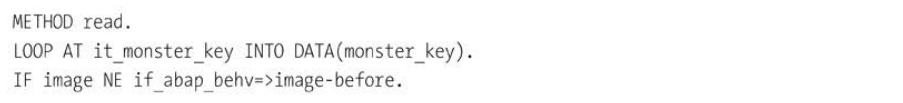
После метода MODIFY в LCL\_HANDLER добавьте код в листинге 14.22. Параметры IMAGE/FAILED/REPORTED являются неявными, а то, что обычно называется параметром EXPORTING или RETURNING, вместо этого является параметром RESULT. Как и раньше, FAILED перечисляет номера монстров, которые не удалось прочитать, а REPORTED содержит все сообщения об ошибках. Параметр IMAGE поступает от фреймворка. Если он говорит TRANSACTIONAL, то он хочет, чтобы вы прочитали транзакционный буфер; если он говорит BEFORE, то он хочет, чтобы вы прочитали базу данных (а затем обновили транзакционный буфер).

Вы можете задаться вопросом об IT\_MONSTER\_KEY и ET\_MONSTER: где находятся определения типов для этих параметров? Опять же, это работает по волшебству; типы строк вычисляются на основе определения представления CDS. В случае с ключевой таблицей у вас есть %PID, который является временным номером, за которым следует фактическое ключевое поле (поля), а затем %CONTROL, в которой указывается, какие поля должны быть возвращены. Вы видели структуру управления ранее, когда кодировали метод обновления. Каждое поле вводится как поле с одним символом, говорящее о том, актуально оно или нет. Тип строки exporting (result) параметра имеет %PID, за которым следует каждое поле в представлении CDS.

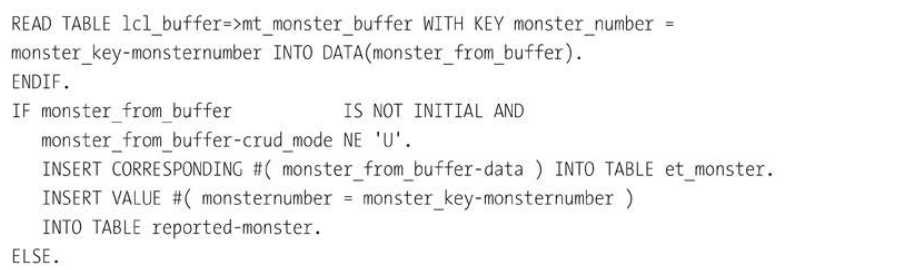


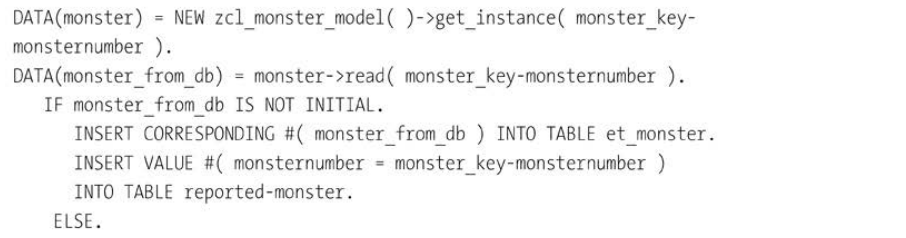
Listing 14.22 Definition of READ Method

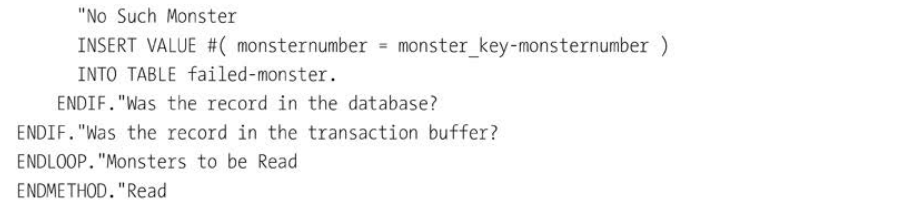
Реализация метода READ (листинг 14.23) довольно проста, как вы можете себе представить. Единственное решение, которое необходимо принять, - это чтение из буфера транзакций или из базы данных. Входящий параметр IMAGE является примером новой функции перечисления ABAP для определения списка констант; возможные значения - BEFORE или TRANSACTIONAL. Если входящий параметр IMAGE имеет значение BEFORE (то есть, до создания буфера), то вы не читаете буфер, а сразу переходите к чтению базы данных; в противном случае вы читаете базу данных, только если записи нет в буфере. Если запись не существует в буфере, то номер несуществующего монстра добавляется в таблицу FAILED. И наоборот, в таблицу REPORTED попадает список номеров монстров, которые существуют.



833







Listing 14.23 Implementation of READ Method

**Блокировка**

Одним из строительных блоков традиционного программирования SAP является концепция ENQUEUE: когда один пользователь изменяет объект, например, заказ на продажу или монстра, вы не хотите, чтобы другой пользователь одновременно изменял тот же объект, что привело бы к странным результатам, основанным на том, кто сохранил запись первым.

Поэтому вы блокируете запись, как только первый пользователь начинает ее редактировать, и все остальные, кто пытается редактировать ту же запись, получают сообщение об ошибке: "Монстр XXXX обрабатывается пользователем MODOQ". Все это не является новостью, и очевидно, что вы хотите иметь такую же функциональность в RAP. Но как этого добиться?

Во-первых, и это самое главное, механизм блокировки в ABAP точно такой же, как и раньше. Вы по-прежнему вызываете функциональный блок ENQUEUE, сгенерированный из объекта блокировки на основе

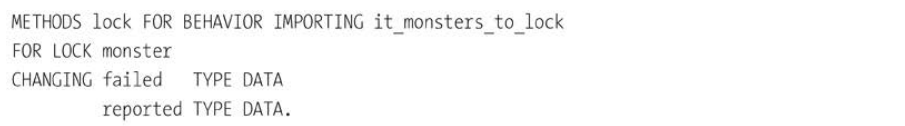
834

вашего бизнес-объекта. Функциональные модули все еще разрешены в новом смелом мире, как бы удивительно это ни казалось. Этот ФМ просто заворачивается в метод LOCK.

Однако, как вы видели, методы вызываются не вручную, а автоматически, когда приложение SAPUI5 запущено. В данном случае метод LOCK вызывается, когда пользователь нажимает кнопку Save после изменения некоторых данных на экране. Вам может показаться это необычным; в мире SAP GUI блокировка проверяется при нажатии кнопки Edit, до изменения каких-либо данных. Мы вернемся к этому позже, но сначала давайте рассмотрим, как описать этот метод.

Если вам нужна блокировка (а вы наверняка хотите), то вы должны добавить слова LOCK MASTER в начало определения вашего поведения (перед первой открывающей скобкой). В противном случае метод LOCK не будет вызван автоматически. Термин master здесь относится не к смертельному врагу из "Доктора Кто", а скорее к тому факту, что то, что вы блокируете, является корневым узлом. В будущих выпусках вы также сможете блокировать зависимые бизнес-объекты, подобно тому, как вы блокируете родительские продажи в настоящее время, когда вы открываете поставку в SAP GUI для предотвращения несоответствий, например, когда кто-то снижает количество заказа меньше, чем в поставке, которую вы собираетесь выпустить.

После изменения определения поведения пришло время добавить еще немного кода. После определения метода READ в классе LCL\_HANDLER, вы пишете определение метода LOCK, как в листинге 14.24.



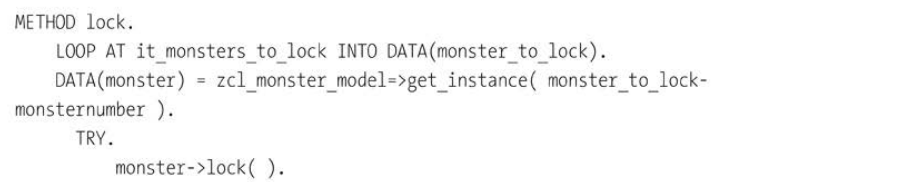
Listing 14.24 Definition of LOCK Method

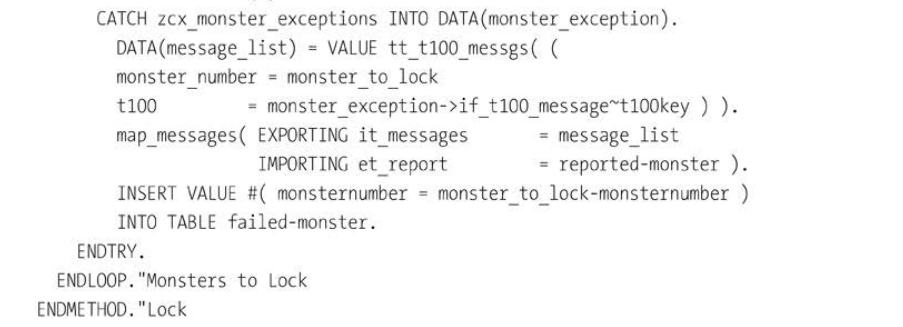
Вам может быть интересно, как набирается таблица IMPORTING; в конце концов, вы же не сами ее набирали. Вы, наверное, уже догадались: фреймворк сам набирает таблицу. В данном случае это внутренняя таблица, содержащая все поля в определении представления CDS. Вам не нужны такие странные вещи, как %CID (и, следовательно, параметр MAPPED), потому что запись уже существует в базе данных.

Следуя той же схеме, что и раньше, реализация метода не представляет сложности: операция блокировки передается ZCL\_MONSTER\_MODEL, который, в свою очередь, вызывает функциональный блок ENQUEUE функциональный блок. Вы снова разблокируете объект после выполнения метода SAVE, к которому мы к которому мы перейдем через минуту.

835

Вы можете увидеть кодировку в листинге 14.25, которая на 99% состоит из обработки ошибок. Вы должны сделать отображение сообщения об ошибке самостоятельно, точно так же, как вы бы сделали это в конце метода MODIFY.





Listing 14.25 Definition of LOCK Method

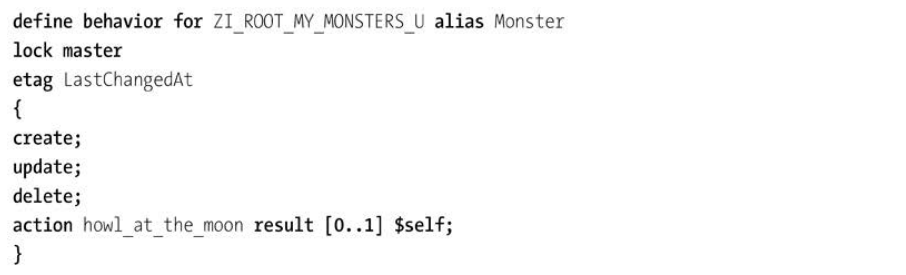
Возвращаясь к тому, когда именно метод LOCK вызывается фреймворком: как вы видели, это отличается от того, что происходит в SAP GUI. В традиционных программах Dynpro вы проверяете, существует ли блокировка, когда пользователь нажимает кнопку Edit, но здесь RAP проверяет только тогда, когда пользователь изменил поля и попытался нажать кнопку Save.

Нам, программистам старой школы ABAP, это кажется очень странным, но мне достоверно известно, что это "так и есть", когда речь идет о веб-приложениях, нравится вам это или нет. Очевидная проблема, которая приходит на ум, заключается в следующем: Что если горбун получит телефонный звонок от злого землевладельца с просьбой снизить уровень разумности монстра с 75% до 50%? Он делает изменение в режиме редактирования, а затем уходит по своим делам и забывает нажать кнопку Сохранить кнопку. Помните, что в этот момент метод LOCK еще не был вызван. Затем снова звонит злой землевладелец, и второй горбун принимает звонок. На этот раз землевладелец хочет, чтобы уровень здравомыслия снизился до 1%. Второй горбун делает изменение, сохраняет измененные данные, и все хорошо.

Затем первый горбун возвращается, понимает, что не сохранил свои изменения, и нажимает кнопку Сохранить. Повысится ли уровень здравомыслия с 1% до 50%? Можно подумать, что да, не так ли?

836

Как выяснилось, в веб-приложениях это решается путем обеспечения наличия временной метки "последнее изменение в" в представлении CDS (подобно полям ERDAT/ERZET, которые мы знаем и любим из традиционных ABAP-программ). Это становится, говоря языком Интернета, ETAG. В определении поведения вы добавляете оператор *etag LastChangedAt* после оператора *lock* *master*. Измененный код выглядит как листинг 14.26



Listing 14.26 Behavior Definition with Locking and ETAG Handling

В методах CREATE и UPDATE необходимо убедиться, что поле LASTCHANGEDAT обновлено текущей меткой времени. Как и во многих других случаях в RAP, волшебство происходит под покровом тайны. В данном случае, если вы объявили поле *etag* в определении и заполнили значения в методах, то фреймворк волшебным образом позаботится о том, чтобы сущности (монстры) не обновлялись вне последовательности.

### 14.4.4 Сохранение измененных бизнес-объектов

В RAP рекомендуется разделять метод, создающий новый экземпляр (монстра), и метод, сохраняющий его в базе данных. Теоретически, ничто не мешает вам даже не беспокоиться о буфере и выполнять фактическое обновление базы данных в LCL\_HANDLER, имея все методы в LCL\_SAVER пустыми - но это было бы не очень разумно, поскольку вы бы боролись с фреймворком, а не работали с ним.

Фактическое обновление базы данных - все, что включает COMMIT WORK - должно выполняться в методе SAVE класса LCL\_SAVER. Логика здесь такова, что методы FINALIZE (то, что BOPF назвал бы производной для добавления вычисляемых полей и тому подобного) и CHECK\_BEFORE\_SAVE (то, что BOPF назвал бы валидацией) вызываются перед методом SAVE, чтобы убедиться, что все хорошо и прекрасно.

В терминах RAP, методы create/update/ и т.д. обновляют транзакционный буфер транзакции в фазе взаимодействия, и это изменение находится в памяти только до тех пор, пока не произойдет фактический

837

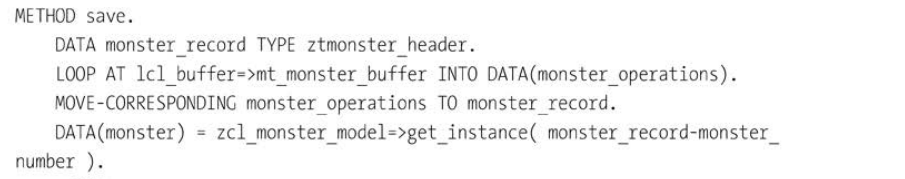
Commit, который выполняется в базе данных в конце последовательности сохранения. BOPF делает аналогичное различие; вы можете вспомнить из главы 8, что существуют отдельные методы BOPF для изменения данных в памяти и для изменения данных в базе данных.

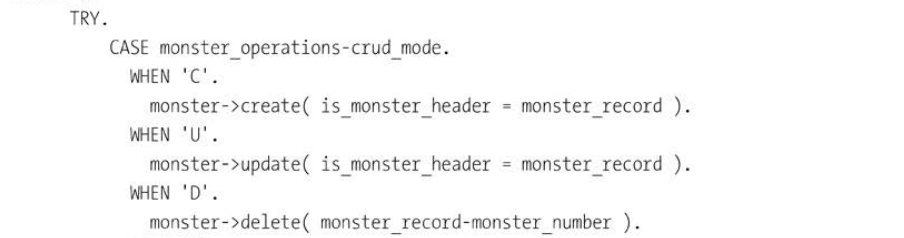
В методе SAVE, Листинг 14.27, ему просто нужно запросить транзакционный буфер по одной операции за раз и, если есть записи для этой операции, передать их классу monster для выполнения - на этот раз по-настоящему. Вас может озадачить полное отсутствие обработки ошибок. Идея заключается в том, что к моменту выполнения операции SAVE вы выполнили так много проверок, что ее невозможно провалить, так же как невозможно утонуть "Титанику". Вот почему RAP не предоставляет таблицу ошибок в сигнатуре метода SAVE. Думаю ли я, что это хорошая идея? Вам придется спросить капитана "Титаника".

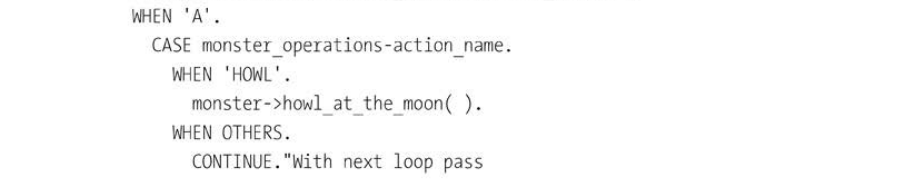
Причина в том, что в текущей реализации фреймворка конечный пользователь абсолютно ничего не сможет сделать, если во время SAVE произойдет какая-либо ошибка, поэтому дамп - это подходящая реакция, чтобы разработчик мог исследовать ситуацию. Если бы я правил миром, то фреймворк мог бы перехватывать любые ошибки и отображать их конечному пользователю.

Возможно, когда-нибудь RAP позволит это сделать.

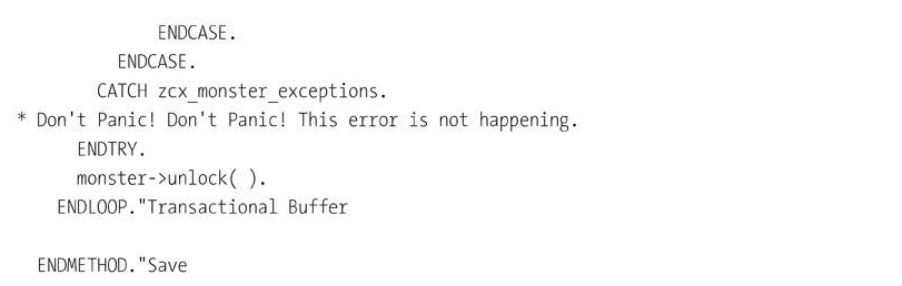
В любом случае, после того как транзакция была сохранена в базе данных, вызывается метод UNLOCK, как и следовало ожидать в программе Dynpro.







838



Listing 14.27 Implementation of SAVE Method

## 14.5 Определения служб и привязки

Теперь перейдем к следующим шагам - созданию определения службы (раздел 14.5.1) и привязки сервиса (раздел 14.5.2), чтобы ваше приложение-монстр могло быть использовано слоем пользовательского интерфейса (который будет рассмотрен в разделе 14.5.3).

### 14.5.1 Создание определения службы

В главе 12 вы узнали, как создать сервис с помощью SEGW, добавляя сущности и указывая отношения между ними. Здесь сущности представляют собой представления CDS, в которых отношения (ассоциации) задаются в определениях представлений CDS.

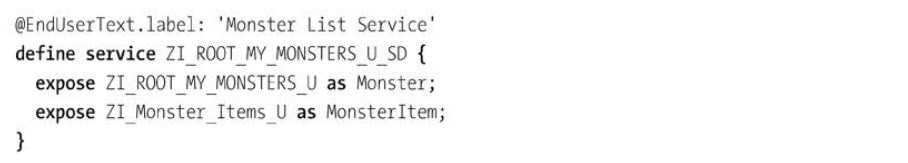
Новый объект репозитория определений сервиса - это, по сути, список представлений CDS, которые могут быть использованы пользовательским интерфейсом или другим приложением. Вы также сопоставляете глупые внутренние технические имена, предназначенные для разработчиков ABAP, с человекочитаемыми именами, которые будут использоваться внешним миром.

Чтобы создать определение службы, войдите в Eclipse и пройдите по пути меню ***New - Other ABAP Repository Object - Business Services - Service Definition***.

Появится окно, показанное на рисунке 14.3. Здесь вам нужно только ввести имя, например, "ZI\_ROOT\_MY\_MONSTERS\_U\_SD", и описание, а затем нажать кнопку Next.

На следующем экране появится всплывающее окно с запросом на транспортировку. Вы можете закончить на этом или нажать «Далее», чтобы получить очень простой шаблон. Шаблон не очень полезен в настоящее время на данный момент, но я уверен, что со временем появятся другие. Как только вы закончите мастера, у вас теперь есть что-то похожее на определение CDS представления DDL, но в этот раз на этот раз вы просто перечисляете задействованные представления CDS, как в листинге 14.28.

839



Listing 14.28 Monster Service Definition

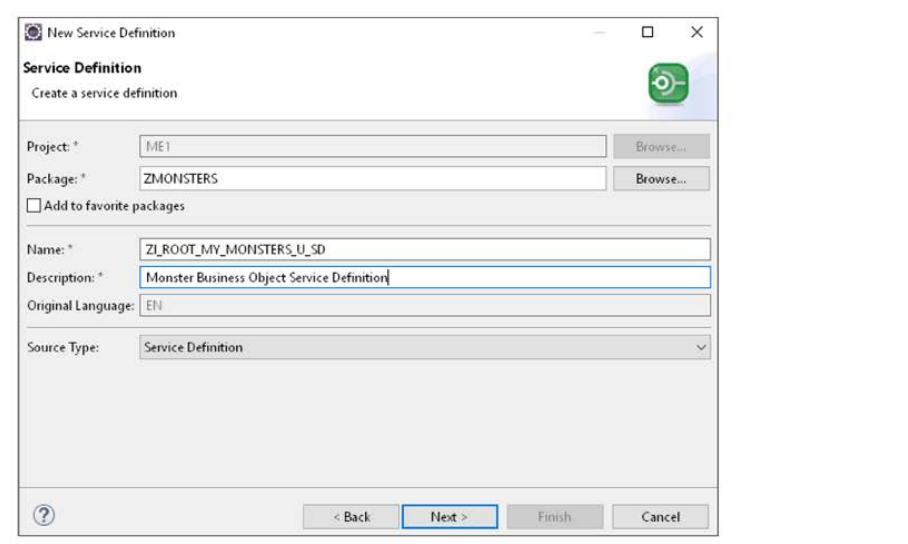


Figure 14.3 Creating Business Service Defintion

Убедитесь, что вы активировали определение службы. В отличие от SE80, ABAP в облаке не может распознать неактивные объекты, поэтому всегда активируйте объекты, как только они перейдут в состояние, свободное от ошибок.

### 14.5.2 Создание привязки службы

Я всегда люблю сильно упрощать вещи, поэтому здесь я собираюсь определить привязку сервиса как то, будет ли сервис потребляться пользовательским интерфейсом или будет потребляться другим приложением, без всякого назойливого человека на пути. Довольно часто вам нужно и то, и другое, поэтому вам нужно создать две привязки; в этом примере вы просто придерживаетесь сценария пользовательского интерфейса.

840

В Eclipse установите курсор на только что созданное определение службы и щелкните правой кнопкой мыши, чтобы перейдите по пути меню ***New - Service Binding***. Появится окно, показанное на рисунке 14.4.



Figure 14.4 Creating Monster Service Binding

Здесь вы вводите имя привязки, например, "ZI\_ROOT\_MY\_MONSTERS\_U\_SBV2", где "V2" в конце не имеет ничего общего с ракетами, а скорее указывает на то, что вы собираетесь выбрать опцию OData V2 для типа привязки на том же экране. Другой вариант был бы A2X, для связи между приложениями. Надеюсь, в невероятно близком будущем SAP включит опцию OData V4, которая намного лучше и занимает меньше места при отправке сообщений.

На этом экране вы также указываете, о каком определении службы идет речь, а затем нажмите NEXT, чтобы продолжить.

На следующем экране (после всегда популярного экрана запроса транспорта) вы увидите информацию, которую вы только что ввели на предыдущем экране, но теперь у вас есть кнопка *Publish Locally* (Опубликовать локально) - кнопку, которую вам нужно нажать, так что сделайте это. На данном этапе служба может использоваться реальными приложениями, а не только встроенным тестовым фреймворком.

841

После того, как вы нажали на эту кнопку, в нижней части экрана (рисунок 14.5) вы увидите URL службы в левой части (запишите этот URL; он понадобится вам в следующем шаге), а в правой части что-то похожее на то, что вы видели в BOPF, с сущностями и отношениями между ними, вплоть до имен навигационных путей, например, to\_Monster\_Items.

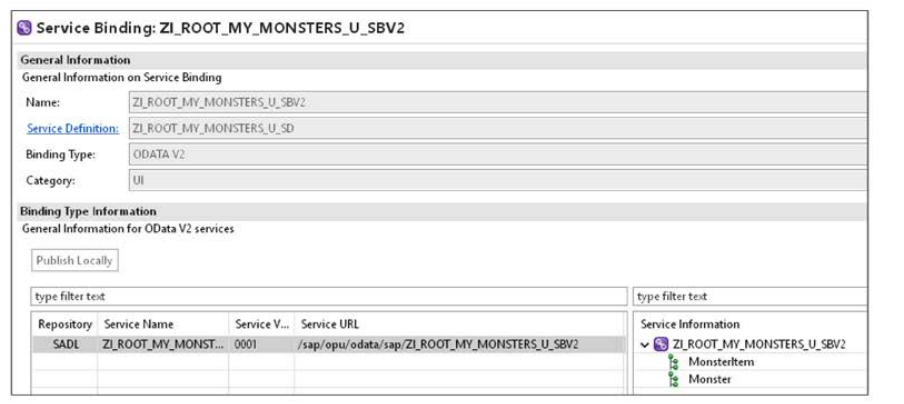


Figure 14.5 Publishing Service Binding

### 14.5.3 Создание приложения SAPUI5 для потребления приложения

Как я уже упоминал ранее, многие вещи - например, неявные параметры CHANGING - в RAP работают с помощью черной магии. Фреймворк делает многие вещи неявно, без необходимости кодировать их, что некоторые люди считают замечательным, а другие - ужасающим.

Следующая часть магии заключается в том, что на данный момент, если вы поместите курсор в поле *Service Information* на сущности Monster, щелкните правой кнопкой мыши и выберите опцию *Open Fiori Elements App Preview*, вам будет предложено войти в систему, а затем, подождите, вы увидите: пустой экран! Не правда ли, впечатляет! Удивительно, что они могут сделать с технологиями в наши дни. На самом деле, если у вас нет аннотаций, указывающих, в каком положении находится каждое поле на экране, то экран действительно будет пустым. Вам нужно будет щелкнуть значок *Settings* справа, который выглядит как шестеренка, и тогда вы сможете выбрать столбцы, которые вы хотите видеть на экране. Вы также можете нажать кнопку *Adapt Filters*, чтобы выбрать некоторые поля выбора в верхней части экрана. Если у вас много аннотаций, то все это происходит автоматически. В любом случае результат выглядит так, как показано на рисунке 14.6.

842

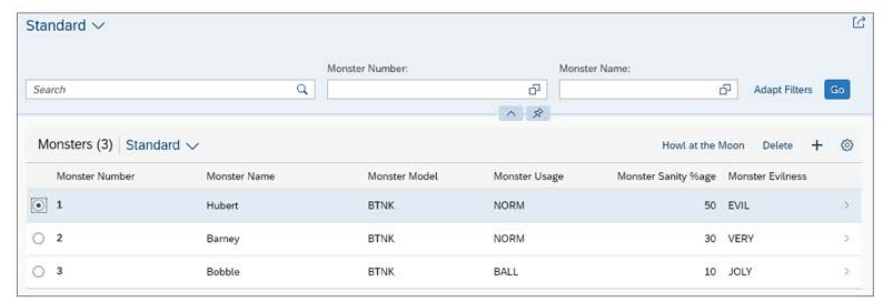


Figure 14.6 SAP Fiori List Preview: List Display

На рисунке 14.6 (список монстров) теперь есть различные значки, представляющие определенные вами ранее действия: удалить, выть на луну и так далее. Действие "Создать" представлено знаком плюс. Если вы нажмете его, откроется новая страница со всеми полями заголовка монстра для заполнения, а на этой странице, в свою очередь, есть кнопки Сохранить и Отмена внизу. Если вы щелкните любое поле в любой строке списка, то откроется экран детализации элемента, которое выглядит как показано на рисунке 14.7.



Figure 14.7 SAP Fiori List Preview: Detail Screen

Если чего-то не хватает или вам не нравится внешний вид предварительного просмотра - например, на рисунке 14.7 колонка Monster Sanity слишком широкая - тогда вы можете начать изменять аннотации пользовательского интерфейса в определении представления CDS, пока вы не станете счастливым кроликом. Существует пол миллиарда различных аннотаций, и вы можете изменить практически все.

На этом этапе у вас, похоже, есть полностью функционирующее приложение, которое выглядит так, как вы хотите, и в котором вы можете создавать/обновлять/удалять монстров и заставлять их выть на

843

луну. Однако это только для тестирования; вам необходимо создать соответствующее приложение SAPUI5. Это делается в SAP Web IDE. Вы узнали об этом в главе 12, когда использовали SAP Web IDE для создания монструозных приложений SAPUI5, но вкратце напомним, что вы подключаете SAP Web IDE к различным пунктам назначения, где, собственно, и находится ABAP-код и/или представления CDS для ваших открытых служб. До сих пор это были внутренние системы SAP, в которых вы писали свой ABAP-код. Когда в следующем году RAP появится в локальных системах SAP S/4HANA, это останется неизменным. Но если у вас есть SAP S/4HANA Cloud, то ABAP-код находится в вашей учетной записи SAP Cloud Platform (в базе данных, отличной от базы данных основной ERP-системы), так что это и есть место назначения. Вполне возможно, что ваше приложение также использует некоторые функциональные возможности из реальной внутренней ERP-системы, но SAP web IDE не нужно об этом знать.

В любом случае (локальном или облачном) процедура создания приложения пользовательского интерфейса практически идентична той, которую вы видели в главе 12, когда создавали приложение SAPUI5 из шаблона на основе службы OData, созданной либо с помощью SEGW, либо непосредственно из представления CDS. Приложение, созданное в этой главе, также экспонирует сервис OData, даже если этот сервис был создан совершенно другим способом.

Однако SAP Web IDE совершенно безразлично, как была создана служба OData. Если вы передадите SAP Web IDE назначение и URL-адрес, и она сможет подключиться и вернуть метаданные службы, то у вас есть один счастливый кролик IDE.

Войдите в SAP Web IDE и на экране обзора (рис. 14.8) выберите значок *New Project from Template*.

Вам нужен список монстров, с помощью которого пользователь может создавать новых или выбирать существующих монстров, обновлять или удалять их и заставлять их выть на луну. Поэтому на следующем экране (Рисунок 14.9) выберите значок *List Report Application*, а затем нажмите Next в самом низу экрана (возможно, вам придется прокрутить страницу вниз, чтобы увидеть кнопку).

Следующие несколько экранов слишком скучны, чтобы показывать их. Вы получите экран *Основные сведения*, на котором вы введете имя и описание вашего приложения, например, ZWELL\_BEHAVEO\_MONSTERS. Следующим экраном является *Data Connection*, в котором вы указываете вашу ABAP в облачной системе в качестве места назначения и вводите URL, который был сгенерирован при создании при создании привязки службы. Нажмите кнопку Test, чтобы проверить, все ли в этом саду радужно.

844

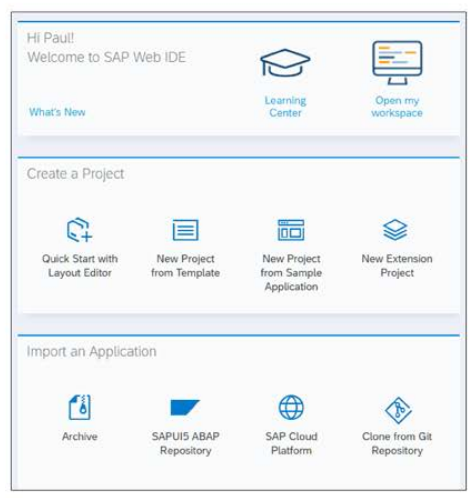


Figure 14.8 SAP Web IDE Welcome Screen

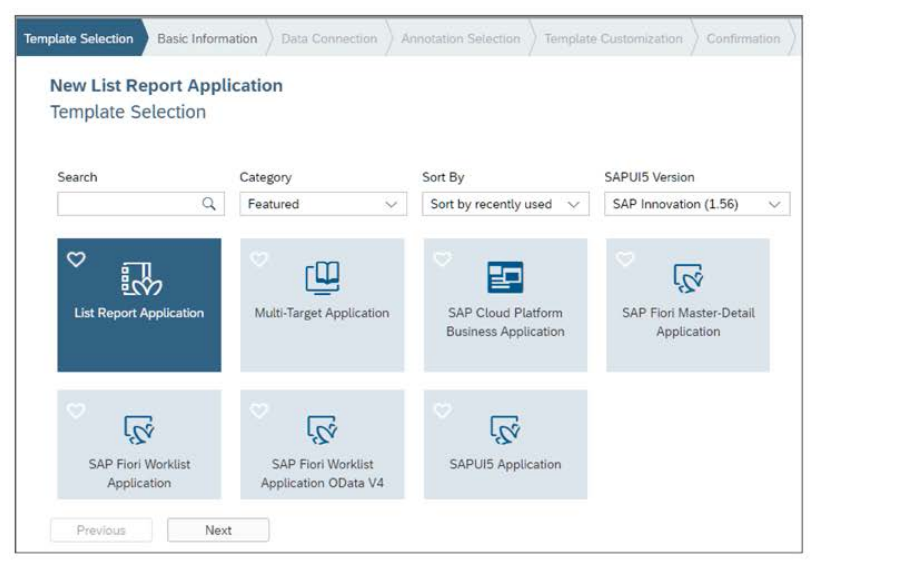


Figure 14.9 List of SAP Web IDE Templates

845

Если все в порядке, просто нажмите кнопку Next несколько раз, после чего на основе метаданных службы будет создано большое дерево файлов, из которых состоит приложение SAPUI5.

Теперь у вас есть настоящее транзакционное приложение SAPUI5, которое должно выглядеть и действовать точно так же, как тестовое приложение, которое вы видели при предварительном просмотре. Разница в том, что тестовое приложение было создано по волшебству; теперь у вас есть код, на который вы можете посмотреть (и возиться с ним), так что это больше не черный ящик. Что еще более важно, теперь у вас есть реальное приложение, которое вы можете использовать SAP Web IDE для развертывания где-нибудь, например, на телефоне барона, чтобы он мог выбрать несколько монстров и заставить их завыть, занимаясь своим хобби - дельтапланеризмом над действующими вулканами (не пытайтесь сделать это дома).

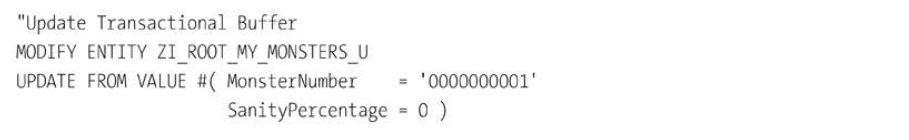
## 14.6 Вызов операций CRUD из ABAP

Вы уже заметили, что все методы, закодированные до сих пор, были абстрактными и частными - поэтому вы не можете инстанцировать класс и напрямую вызвать метод MODIFY, например.

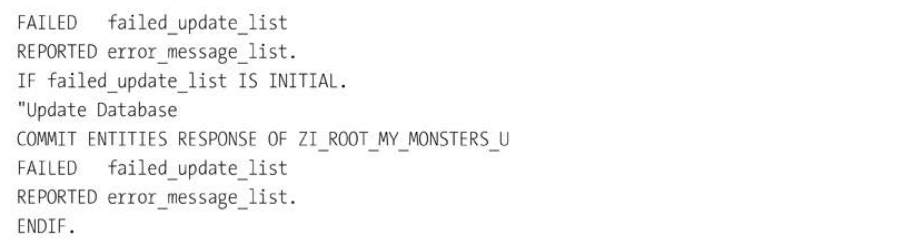
Но что делать, если вы действительно хотите вызвать метод CREATE прямо из ABAP - скажем, для модульного тестирования, потому что вы хотите провести модульное тестирование кода в RAP так же, как и традиционного кода?

Именно здесь в игру вступает язык управления сущностями (Entity Manipulation Language, EML). В прошлом вы привыкли использовать отдельные BAPIs для создания, обновления и чтения экземпляров бизнес-объекта, например, заказа на продажу или монстра. Даже в мире ОО вы вручную создавали методы для каждой операции (и мы все еще кодируем их вручную, как вы видели в этой главе), хотя методы CRUD будут автоматизированы в будущем).

Теперь эта концепция заложена в язык ABAP. Структура полей сущности определяется определением представления CDS, и вы манипулируете ею, используя код, подобный приведенному в листинге 14.29, в котором вы изменяете некоторые поля существующего монстра, используя новое ключевое слово MODIFY ENTITY. Как и в предыдущих примерах кода, результаты возвращаются в таблицы FAILED/REPORTED. При этом вызывается соответствующий метод, закодированный в фазе взаимодействия; то есть данные обновляются в транзакционном буфере. Чтобы сохранить изменений в базе данных, используется очень многословная новая конструкция *COMMIT ENTITIES RESPONSE OF* которая является аналогом *COMMIT WORK*.



846



Listing 14.29 Entity Manipulation Language

Ключевое отличие заключается в том, что в отличие от BOPF, который принимал общие параметры, здесь входные параметры являются типизированными. Если вы неправильно напишете MonsterNumber, вы получите синтаксическую ошибку, так же, как и в случае с BAPI или методом класса Z.

## 14.7 Резюме

В этой главе вы познакомились с идеей ABAP в облаке в целом и RESTful ABAP Programming Model (RAP) в частности. На момент написания этой главы они выглядят как одно и то же; RAP доступна только в облачной версии ABAP, и в этой версии вы должны использовать RAP.

Однако примерно в сентябре 2019 года RAP появится и в локальной версии SAP S/4HANA, хотя его использование будет необязательным.

Вы рассмотрели, как RAP относится к различным уровням - базе данных, бизнес-логике и пользовательскому интерфейсу - иначе, чем вы привыкли, и очень по-разному в случае с уровнем бизнес-логики.

Затем, чтобы продемонстрировать реальность, стоящую за теорией, вы прошли через шаги по созданию транзакционного монструозного приложения с помощью RAP, а затем представили его в качестве сервера. Этот сервис, в свою очередь, был использован SAP Web IDE для создания конечного приложения, готового к развертыванию.

Вся идея ABAP в облаке и RAP является революционной и всколыхнула мир ABAP подобно мегаземлетрясению. Однако это лишь самый первый шаг того, что может оказаться путешествием длиной в тысячу миль.

Цель SAP - не что иное, как полностью отделить пользовательский код от основного кода SAP - как для локальной, так и для облачной среды - и распределенная разработка ABAP, а не все в одном центральном хранилище, как это было всегда

847

до сих пор. Идея заключается в том, что это позволит повысить эффективность разработки и в то же время сделает обновление SAP не требующим особых усилий.

Эти цели, мягко говоря, несколько амбициозны, но, как говорил Кеннеди, "Мы делаем это не потому что это легко, а потому что это трудно".

ABAP в облаке и RAP являются важной частью этого. В настоящее время никто за пределами SAP не использует RAP, но так будет продолжаться недолго. Чем больше людей, которые его используют, тем больше будет ошибок и запросов на новые функции, а с трехмесячным циклом выпуска - а это трехмесячный цикл выпуска - и вы получаете новую версию независимо от того, нравится она вам или нет - количество улучшений каждый год будет драматическим, скорее похожим на то, что вы получаете в проекте с открытым исходным кодом.

Масштабы и скорость изменений в языке ABAP, несомненно, пугают, и, возможно на данный момент кажется довольно академичным, но это затронет вас раньше, чем вы успеете это понять. Поэтому держитесь за шляпу и отправляйтесь вместе с ABAP в будущее!

Recommended Reading

• SAP Cloud Platform, ABAP Environment

<https://blogs.sap.com/2078/09/_04/sap-cloud-platform-abap-environment/> (Harald Kuck)

• ABAP Evolution

<https://blogs.sap.com/2018/10/11/abap-is-evolving/> (Graham Robinson)

848