Глава 2

Закладываем фундамент

Итак, вы убедили клиента в том, что можете создать хороший веб-сайт, который существенно повысит эффективность его компании. Придя в себя после вечеринки по поводу заключения контракта, вам придется подумать, как воплотить в жизнь свои обещания. Как всегда, при попытке составить детальный план действий все начинает выглядеть сложнее, чем казалось на первый взгляд.

Чтобы обеспечить успех проекта, необходимо четко определить, как вы будете реализовывать то, что от вас требуется по контракту. Вам нужно, чтобы проект продвигался быстро и гладко, но самое главное — чтобы заказчик был доволен вашей работой. Соответственно, ваша задача — обеспечить сайту постоянно растущий поток посетителей. Для этого нужно сделать сайт удобным в использовании, функциональным и привлекательным.

Требования весьма жесткие, но для современных сайтов электронных магазинов это нормально. Чтобы довести до максимума вероятность успеха, мы постараемся учесть как можно больше технических требований и реализовать механизмы и элементы, которые позволят вносить дальнейшие усовершенствования с минимумом усилий. В целом наши задачи в этой главе таковы.

- Проанализировать проект с технической точки зрения
- Выбрать архитектуру для создаваемого приложения
- Решить, какие технологии, языки программирования и инструменты использовать
- Составить правила именования и оформления кода

ЗАМЕЧАНИЕ

Учтите, что в этой и нескольких последующих главах мы будем рассматривать материал весьма сжато, и они могут оказаться сложными для вас, если у вас нет опыта работы с PHP или MySQL. В таком случае мы можем только посоветовать предварительно почитать другие книги, например 3-е издание книги Мэтта Зандстры *Освой самостоятельно PHP за 24 часа* (ИД "Вильямс", 2008 г.).

Кроме того, мы очень рекомендуем вам использовать определенную методику управления проектом, чтобы максимизировать шансы на успешное завершение проекта в установленные сроки и без перерасхода денег. Большинство теорий управления проектами подразумевают, что вы как разработчик и ваш клиент подписали документ, содержащий начальные требования или спецификацию необходимого товара. Этот документ вы можете использовать в качестве ориентира в ходе разработки; если клиент позже выдвинет дополнительные требования, вы будете вправе потребовать дополнительную оплату и время на их реализацию.

Дизайн для дальнейшего роста

Термин "дизайн" применительно к веб-приложению может означать множество разных понятий. В самом распространенном случае он означает процесс создания пользовательского интерфейса и визуального оформления веб-страниц.

Этот аспект критически важен для проекта в целом. Давайте посмотрим правде в глаза: пользователю вряд ли понравится бесцветный, неряшливо оформленный или неустойчиво работающий сайт независимо от того, насколько хороша использованная в нем технологическая основа. Даже случайно зайдя на такой сайт, посетитель в дальнейшем вряд ли на него вернется.

К сожалению, этот момент заставляет многих неопытных разработчиков уделять все внимание только внешнему оформлению сайта, оставляя минимум времени на проработку самого приложения — управляющего кода, базы данных и всего остального. Да, красивое оформление может привлечь покупателя, но именно функциональность может убедить его вернуться на сайт. Часто веб-сайт удается создать очень быстро, если есть список требований к нему, но если при создании он был плохо структурирован, в дальнейшем его очень неудобно поддерживать в рабочем состоянии.

В проекте любого размера написанию кода должны предшествовать кое-какие действия. Но независимо от того, сколько подготовительной работы вы проведете, при написании кода всегда возникают неожиданные сложности, новые требования и изменяющиеся правила. Может показаться, что проект просто невозможно завершить в поставленные сроки. Даже если неприятных неожиданностей не случается, разработчиков часто просят добавить в проект не предусмотренную изначально функциональность — обычно уже после того, как проект завершен и товар успешно используется. Это случится и с нашим примером — проектом TShirtShop, который мы будем реализовывать в три этапа, как описано в главе 1.

Мы расскажем, как создать веб-сайт, который не развалится при попытках добавить в него новые элементы или функции. Поскольку это книга по программированию, мы сосредоточимся не на оформлении сайта или вопросах маркетинга, а на создании кода, который заставляет сайт работать.

Само по себе словосочетание "дизайн кода" может иметь разные значения, например в это понятие входит составление правил именования для переменных, функций и прочих элементов кода. Но самое важное для нашего проекта — это выбор архитектуры. К архитектуре относится и разбиение кода на функциональные элементы (например, механизм поиска в каталоге). Разумеется, можно создать работающий проект из нескольких крупных компонентов, но проект из множества мелких, простых компонентов модернизировать и поддерживать гораздо проще.

Прежде чем говорить о самой архитектуре, нужно решить, чего мы от нее хотим.

Выполнение долгосрочных требований с минимумом усилий

Помимо того, что нам нужен быстро работающий веб-сайт, на каждом этапе проектирования мы будем сталкиваться с новыми требованиями.

Каждый раз при переходе к новому этапу желательно, чтобы мы могли использовать большую часть ранее написанного кода. Будет очень плохо, если для реализации новой функции придется переделывать весь проект целиком — и визуальную часть, и код. Повысить эффективность повторного использования кода можно, хорошо спланировав и структурировав приложение, чтобы новая функция стала всего лишь еще одним модулем, а не потребовала переделки всего кода.

При создании веб-сайта реализация *гибкой архитектуры*, состоящей из подключаемых компонентов, позволяет легко добавлять в проект новые возможности, например корзину покупателя, список отделов или возможность поиска товаров по каталогу. Такие возможности пишутся в виде отдельных модулей, а затем подключаются к проекту. Достижение хорошей гибкости проекта — одна из основных задач при выборе архитектуры, и в этой главе мы постараемся показать вам, как достичь такой гибкости. Вы увидите, что проект тем гибче, чем больше времени потрачено на его дизайн и написание кода, так что мы попытаемся выдержать баланс между приемлемыми затратами времени и получением удобоваримого кода.

Еще одно важное требование, общее для всех онлайн-приложений — *масштабируемость*. Она определяется как способность приложений линейно наращивать производительность при линейном росте выделенных для них ресурсов. Другими словами, в идеально масштабирующейся системе соотношение между количеством обрабатываемых запросов в единицу времени и количеством выделенных ресурсов будет постоянным независимо от количества клиентов и других факторов. Плохо масштабирующаяся система не может справиться с ростом количества клиентов независимо от того, сколько аппаратных ресурсов мы для нее выделим. А поскольку мы надеемся, что покупателей у нас с течением времени будет все больше, мы должны сделать систему масштабируемой, чтобы сайт всегда работал быстро и плавно.

Надежность — еще одно критически важное требование к приложениям в области электронной торговли. Применив эффективные механизмы обработки ошибок и хорошую реляционную СУБД, вы сможете обеспечить целостность данных и гарантировать, что большинство ошибок не скажутся на стабильности работы сайта.

Трехуровневая архитектура

В целом под архитектурой понимают разбиение на части кода приложения. Мы постараемся выполнить это разбиение так, чтобы код, относящийся к одной части приложения, был объединен в один логический уровень.

В частности, трехуровневая архитектура подразумевает разделение кода на три уровня.

- Уровень представления (presentation layer)
- Уровень логики приложения (бизнес-уровень, или business layer)
- Уровень данных (data layer)

Уровень представления содержит код, отвечающий за интерфейс веб-сайта, и логику, обеспечивающую взаимодействие сайта с пользователями. Этот уровень "оживляет" сайт, и его качество критически важно для успеха сайта. Поскольку наше приложение — веб-сайт, его интерфейсом будет набор динамических веб-страниц.

Уровень логики приложения (иногда называемый просто *средним уровнем*, или бизнес-уровнем) получает запросы от уровня представления и возвращает уровню представления ответы на эти запросы, руководствуясь управляющей логикой, размещенной на бизнес-уровне. Почти каждое событие, происходящее на уровне представления, приводит к обращению к уровню логики приложения, за исключением событий, с которыми уровень представления может разобраться сам (например, провести простую проверку введенных пользователем данных). Если, к примеру, пользователь обращается к механизму поиска товара в каталоге, уровень представления обращается к уровню логики приложения и требует сообщить, какие товары соответствуют критериям поиска, которые задал пользователь. В большинстве слу-

чаев уровень логики приложения должен обращаться к уровню данных за информацией, необходимой для ответов на запросы с уровня представления.

Уровень данных (иногда называемый уровнем базы данных) отвечает за управление данными приложения и передачу этих данных уровню логики приложения при получении запросов. В проекте приложения TShirtShop нам понадобится хранить данные о товарах (включая сведения о категориях и отделах), о пользователях, о корзинах покупателей и т.д. Почти любой запрос пользователя в конечном итоге приводит к обращению приложения к уровню данных (если только данные ранее не были кэшированы на уровне логики приложения или уровне представления), поэтому уровень данных должен использовать быструю и эффективную базу данных. В главах 4 и 5 мы рассмотрим создание базы данных, оптимизированной для достижения максимальной производительности.

Сами по себе уровни являются сугубо логическими единицами — код, относящийся к разным уровням, не обязательно должен физически размещаться в разных файлах. Вы можете размещать все приложение, все его уровни, на одном сервере, или размещать разные уровни на разных машинах, если приложение это позволяет. Из главы 22 вы узнаете, как интегрировать в приложение функциональность с других веб-сайтов с помощью веб-служб XML. Эти веб-службы позволяют выполнять интеграцию с использованием минимума специального кода.

Рис. 2.1. Простое представление трехуровневой архитектуры

Важное ограничение трехуровневой архитектуры — то, что информация должна поочередно передаваться между соседними уровнями. Уровень представления может обращаться к уровню логики приложения, но не к уровню данных. Уровень логики приложения — это мозг приложения, который может общаться и с уровнем данных, и с уровнем представления, координируя потоки данных. Если уровень представления будет напрямую общаться с уровнем данных, правила трехуровневой архитектуры будут нарушены.

Возможно, эти правила покажутся вам искусственными ограничениями, но, чтобы получить выигрыш от использования архитектуры, вы должны следовать ее правилам. Правильное применение трехуровневой архитектуры обеспечит простоту модернизации сайта и добавления в него новых возможностей, а кроме того, позволит контролировать доступ к данным. Сейчас это может показаться ненужным и излишним, но в будущем такой контроль очень пригодится, когда придется менять функциональность сайта.

На рис. 2.1 изображена диаграмма потоков данных в приложении с трехуровневой архитектурой.

Простой пример использования трехуровневой архитектуры

Чтобы разобраться, как данные передаются между уровнями и преобразуются, давайте рассмотрим простой пример. А чтобы этот пример был полезен для нашего проекта TShirtShop, он будет основан на реальной ситуации, которая может возникнуть при общении пользователя с приложением.

Как и в большинстве приложений электронных магазинов, в TShirtShop будет корзина покупателя, которую мы подробно рассмотрим позже. А пока что вам достаточно знать, что покупатель сможет добавлять товары в корзину, щелкая на

кнопке Add to Cart на веб-странице. На рис. 2.2 показан обмен информацией, происходящий в приложении после нажатия этой кнопки.

На шаге 1 пользователь щелкает на кнопке Add to Cart для определенного товара. На шаге 2 уровень представления (реагирующий на щелчки на этой кнопке) передает уровню логики приложения запрос на добавление данного товара в корзину покупателя. На шаге 3 уровень логики приложения получает этот запрос и обрабатывает его, приказав уровню данных обновить содержимое корзины покупателя, добавив в нее выбранный товар. К уровню данных необходимо обращаться, потому что он отвечает за хранение данных веб-сайта (включая корзину покупателя) и доступ к ним.

На шаге 4 уровень данных обновляет содержимое базы данных и сообщает уровеню логики приложения, что обновление прошло успешно. На шаге 5 уровень логики приложения обрабатывает сообщение от уровня данных и/или коды ошибок, полученные от него при обновлении базы данных. Затем уровень логики приложения сообщает результаты обработки уровню представления.

На шаге 6 уровень представления генерирует обновленное отображение корзины покупателя. На шаге 7 генерируется соответствующий результатам работы приложения HTML-код, который передается клиенту. Этот код отображает в окне браузера обновленное содержимое корзины покупателя.

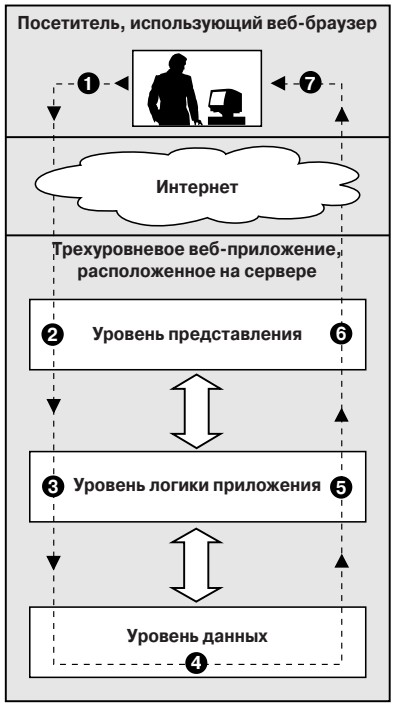

Рис. 2.2. Взаимодействие уровней в приложении с трехуровневой архитектурой

Заметьте, что в этом примере уровень логики приложения не выполняет какой-то сложной обработки данных, и сама управляющая логика на уровне приложения довольно проста. Однако если вам понадобится изменить эту логику, достаточно будет модифицировать только уровень логики приложения. Например, если вам нужно, чтобы управляющая логика позволяла добавлять товары в корзину покупателя, только если они есть в запасе, уровень логики приложения делал бы еще один запрос к уровню данных, проверяя, есть ли в запасе выбранный товар. Второй запрос, с приказом обновить содержимое корзины покупателя, делался бы только при наличии товара. Но в любом случае уровень представления отвечает за уведомление пользователя о результатах щелчка на кнопке Add to Cart.

Смысл данных

Интересно посмотреть, как разные уровни приложения воспринимают одни и те же данные. Для уровня данных сами данные ничего не значат — это просто числа и строки, которые он сохраняет в базе данных или считывает из нее по запросам от уровня логики приложения. Он понятия не имеет о том, что такое склад, товары и корзина покупателя. Например, отсутствие товара на складе для него есть просто число 0 в одной из ячеек базы данных — 32-битовое целое число 0.

Данные обретают смысл только на уровене логики приложения. Когда этот уровень запрашивает у уровня данных количество единиц товара на складе и получает

ответ "0", он интерпертирует этот ответ как "товара на складе нет". Этот ответ преобразуется в форму, понятную уровню представления, и уровень представления докладывает о результатах запроса пользователю, например, сообщением "Извините, в данный момент заказать этот товар нельзя".

Возможно, вы и не захотите запрещать пользователю добавлять в корзину товары, которых нет в запасе, но этот пример (показанный на рис. 2.3) достаточно хорошо иллюстрирует смысл данных на разных уровнях приложения с трехуровневой архитектурой.

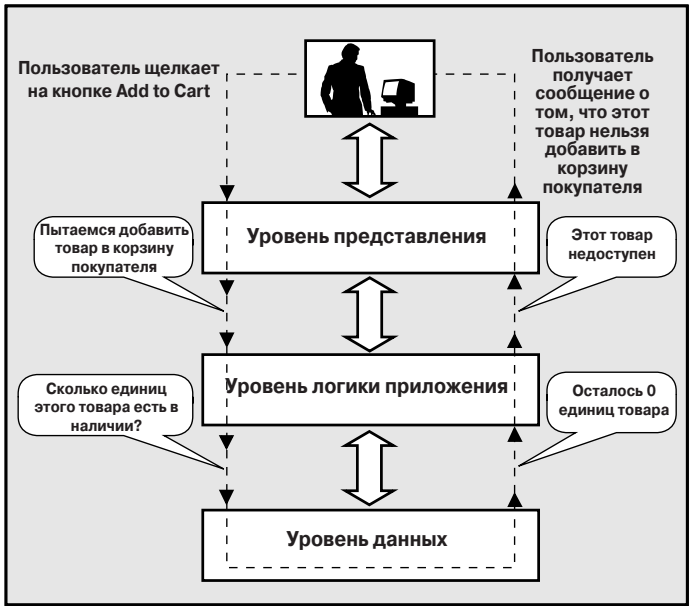

Рис. 2.3. Смысл данных на разных уровнях приложения

Распределение функций между слоями

Поскольку в каждом уровне приложения содержится своя управляющая логика, иногда трудно четко разделить приложение на уровни. Например, в приведенном выше сценарии вместо организации взаимодействия между слоями можно было бы написать одну процедуру `add_product_to_cart`, которая добавила бы товар в корзину, только если он есть в запасе.

В этом сценарии часть управляющей логики перемещается из уровня логики приложения на уровень данных. В других случаях управляющая логика может размещаться и на уровене логики приложения, и на уровне данных. Для большинства реальных приложений не существует единственно верного способа разделения на уровни, и разделение приходится выполнять, исходя из личных предпочтений разработчиков или каких-то внешних ограничений.

Кроме того, бывают случаи, когда вы знаете *правильный* (с точки зрения архитектуры) способ реализации какой-то функции, но этот способ не является оптимальным с точки зрения производительности приложения. В целом, если за счет нарушения разбиения на уровни можно повысить производительность приложения, это следует сделать (например, можно добавить какие-то механизмы из уровня логики приложения на уровень данных), если такое нарушение не сделает при-

ложение плохо модернизируемым. В противном случае лучше держать всю управляющую логику на уровне приложения — это позволит сделать приложение более простым в обслуживании.

Еще один совет: никогда не пытайтесь напрямую обращаться к уровню данных из уровня представления. Эта ошибка приводит к получению запутанных, трудно модернизируемых и плохо отлаживаемых приложений. Во многих учебниках показывается, как простые программы напрямую обращаются к базам данных. В таких программах весь код обычно представляет собой один небольшой исходный файл, и этот код не делится на уровни. Но это всего лишь маленькие учебные программы, а не серьезные коммерческие приложения, которые должны быть гибкими и масштабируемыми.

Трехуровневая архитектура для TShirtShop

Использование трехуровневой архитектуры в проекте TShirtShop упростит выполнение задач, перечисленных в начале главы. Правила оформления кода, которые могут показаться излишне жесткими, в длительной перспективе обеспечат приложению гибкость и расширяемость.

Разделение больших частей приложения на маленькие отдельные компоненты сделает код более удобным для многократного использования. В дальнейшем, реализуя в приложении новые функции, вы будете обнаруживать, что эти функции можно собирать из уже существующих фрагментов кода.

Еще одно преимущество трехуровневой архитектуры — при правильной реализации она делает систему в целом устойчивой к изменениям. Когда изменяется часть одного из уровней, остальные уровни обычно остаются неизменными, даже если изменения весьма заметны. Например, если вы решите вместо MySQL реализовать уровень данных с помощью PostgreSQL, вам придется поменять только уровень данных (и, возможно, немного модифицировать уровень логики приложения).

А почему бы не использовать больше уровней?

Трехуровневая архитектура, которую мы рассматриваем, — это частный (хотя и наиболее популярный) вариант *n*-уровневой архитектуры, в которой приложение делится на *n* логических слоев. В больших и сложных проектах иногда есть смысл разделить уровень логики приложения на несколько более мелких уровней. Однако для нашего веб-сайта вполне достаточно трехуровневой архитектуры, которая позволит пользоваться преимуществами разбиения кода на уровни без лишней возни с написанием поддерживающего архитектуру кода.

Возможно, более сложная архитектура позволит сделать приложение еще более гибким и масштабируемым, но вам придется потратить гораздо больше времени на проектирование, прежде чем вы сможете приступить к написанию кода. Трехуровневая архитектура, пожалуй, оптимальна для проектов средней сложности наподобие веб-сайта TShirtShop.

Возможно, вы зададитесь противоположным вопросом “А почему бы не использовать меньше уровней?” Двухуровневая архитектура, более известная как *архитектура “клиент/сервер”*, вполне подходит для несложных проектов. В целом она требует меньше времени на планирование приложений и позволяет быстрее приступить к написанию кода; однако в дальнейшем такие приложения труднее обслуживать и модернизировать. А поскольку мы планируем в дальнейшем расширять возможности приложения, мы не будем использовать эту архитектуру.

Итак, выбрав базовую архитектуру приложения, давайте посмотрим, какие технологии и инструменты можно применить, чтобы воплотить ее в жизнь. Сейчас

мы кратко обсудим эти технологии и инструменты, а в главе 3 заложим основы уровня данных и уровня представления, создав первую страницу веб-сайта и базу данных нашего приложения. Реальную функциональность для каждого из трех уровней мы начнем создавать в главе 4, когда займемся каталогом товаров для электронного магазина.

Выбор технологий и инструментов

Независимо от выбранной архитектуры в каждом проекте по разработке приложений нужно выбирать какие-то технологии, языки программирования и инструменты, с помощью которых будут создаваться эти приложения. При этом нужно учитывать, что выбор может ограничиваться требованиями заказчиков.

В этой книге мы будем создавать веб-приложение с использованием PHP 5, MySQL 5 и связанных с ними технологий. Это распространенные технологии, но это не обязательно значит, что они будут наилучшим выбором для любого проекта в любых обстоятельствах. Кроме того, часто клиент требует, чтобы при разработке приложения использовались определенные технологии и инструменты. Этап анализа требований, который присутствует в большинстве проектов по разработке программ, как раз и предназначен для выбора технологий, которые будут использоваться.

Хотя в этой книге и подразумевается, что у вас есть опыт использования PHP и MySQL, мы кратко рассмотрим их, чтобы разобраться, как их можно применить в нашем проекте и в трехуровневой архитектуре в целом.

Использование PHP для генерации динамического веб-контента

PHP — это технология с открытым исходным кодом, предназначенная для генерации динамического, интерактивного веб-контента. Как говорится в ее официальном описании (на сайте `http://www.php.net`), "PHP — широко используемый интерпретируемый язык общего назначения, особенно удобный для веб-приложений, который может встраиваться в HTML".

"PHP" расшифровывается как "PHP": "Hypertext Preprocessor" (да, это рекурсивная аббревиатура), и его можно бесплатно загрузить с официального сайта. История PHP началась примерно с 1994 года и представляет собой пример впечатляющего успеха. Среди факторов, обусловивших этот успех, можно упомянуть следующие.

- PHP бесплатен. Если его использовать в сочетании с серверными программами под управлением Linux, он представляет собой очень экономически эффективную платформу для генерации динамического веб-контента.
- PHP легче изучать, чем большинство других интерпретируемых языков.
- Существует большое сообщество разработчиков, использующих PHP. Есть множество разнообразных библиотек для PHP (например, их можно найти в репозитории PEAR или на сайте `http://www.phpclasses.org`), и в них регулярно добавляются новые возможности.
- PHP хорошо работает с разными веб-серверами и операционными системами (семейств Unix, Windows или MacOS).

Однако PHP — отнюдь не единственный серверный интерпретатор, позволяющий генерировать динамические веб-страницы. Среди его конкурентов можно

упомянуть JavaServer Pages (JSP), Perl, ColdFusion и ASP.NET. У этих технологий есть свои особенности, но в целом они предназначены для решения одних и тех же задач. Например, страницы, сгенерированные с помощью любой из этих технологий, состоят из простого HTML, отвечающего за статичную часть страниц (шаблон) и кода, генерирующего динамическую часть.

ЗАМЕЧАНИЕ

Если вам интересно, прочтите 3-е издание книги Мэтью Мак-Дональда и др. *Microsoft ASP.NET 3.5 с примерами на C# 2008 и Silverlight 2 для профессионалов* ("Диалектика", 2009 г.). В ней рассматривается создание сайтов с помощью ASP.NET, C# и SQL Server.

Разделение кода и оформления с помощью Smarty

Из-за простоты PHP многие использующие его разработчики начинают писать код приложения, не продумав его архитектуру и не сформировав базовую платформу.

Еще хуже то, что PHP можно легко смешивать с HTML-кодом веб-страниц, поскольку PHP по умолчанию не содержит механизмов для четкого разделения кода на PHP и кода на HTML.

Смешивание кода на PHP и HTML приводит к двум довольно неприятным проблемам.

- В результате такого смешивания вы получаете объемный, запутанный и плохо поддающийся модернизации код. Возможно, вы видели длинные непонятные листинги с PHP и HTML, в которых сами их авторы не могут разобраться через неделю после написания.
- В таких файлах вынуждены копаться и программисты, и дизайнеры вебсайта, что существенно усложняет взаимодействие разработчиков. Кроме того, дизайнеры часто вносят в код ошибки, занимаясь оформлением вебстраниц.

Эти проблемы привели к появлению систем шаблонов, позволяющих разделять логику уровня представления и HTML-код. Smarty (`http://smarty.php.net`) — это одна из самых популярных и эффективных систем шаблонов для PHP. Ее основное назначение — именно разделение логики приложения (кода на PHP) и кода представления (HTML).

Такое разделение позволяет программистам и дизайнерам шаблонов независимо работать над одним и тем же приложением. Программист может менять логику в PHP-коде, не трогая файлы шаблонов, а дизайнер шаблонов может менять эти шаблоны, не беспокоясь о том, как работает код, в котором они используются.

На рис. 2.4 показаны взаимоотношения между шаблонами Smarty и соответствующими им подключаемыми файлами.

Дизайн-шаблон Smarty (файл `.tpl` с тегами HTML и тегами, специфичными для Smarty) и соответствующий ему подключаемый файл Smarty (файл `.php` с кодом, взаимодействующим с шаблоном) образуют *компонентный шаблон Smarty*.

На практике мы не будем создавать подключаемый файл Smarty для каждого дизайн-шаблона, как показано на рис. 2.4. Вместо этого мы создадим обобщенный подключаемый файл, который будет интегрироваться со всеми дизайн-шаблонами, загружая необходимые объекты представления. Объекты представления — это классы, снабжающие файлы шаблонов необходимыми данными.

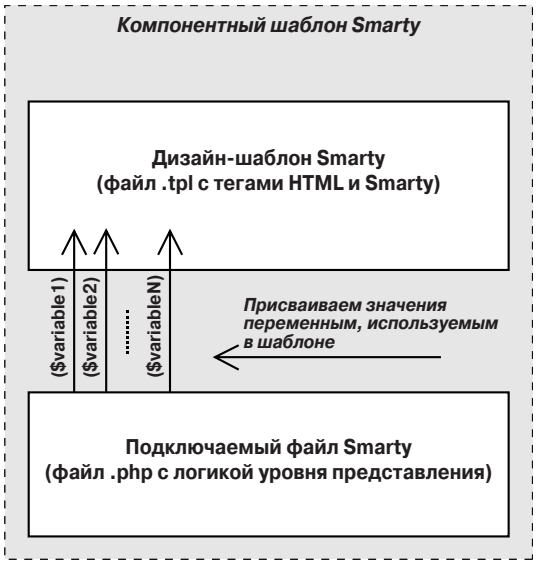

Рис. 2.4. Шаблоны Smarty и подключаемые файлы

Вы узнаете больше о Smarty, создавая наш сайт электронного магазина. Если вам нужен краткий вводный курс, прочтите Smarty Crash Course по адресу `http://smarty.php.net/crashcourse.php`. Если же вам требуется нечто более подробное, могу порекомендовать книгу *Smarty PHP Template Programming and Applications* (Hasin Hayder, J. P. Maia и Lucian Gheorghe; Packt Publishing, 2006).

ЗАМЕЧАНИЕ

Применение в проекте Smarty или другой системы шаблонов потребует написания кое-какого дополнительного кода и, разумеется, времени на обучение использованию этой системы. Но все же рекомендую вам попробовать это — в конечном счете вы немало выиграете от использования системы шаблонов.

Как насчет альтернатив?

Smarty — отнюдь не единственная существующая система шаблонов для PHP. Если вас интересуют альтернативы, попробуйте задать в Google поиск на тему "PHP template engines" или найти статью *Top 25 PHP Template Engines* (автор — Justin Silverton) с помощью того же Google.

В целом все системы шаблонов используют один и тот же базовый принцип, и мы выбрали Smarty для нашего проекта потому, что он обеспечивает очень хорошую производительность, развитую функциональность (например, компиляцию и кэширование шаблонов) и широко распространен.

Использование MySQL на уровне данных

Большую часть данных, которые будут видеть посетители нашего сайта, наше приложение должно будет извлекать из реляционной базы данных. Система управления реляционными базами данных (РСУБД) — это сложная программа, обеспечивающая максимально надежное хранение информации и быстрый доступ

к ней. На сайте TShirtShop в реляционной базе данных будут храниться сведения о товарах, отделах, пользователях, корзинах покупателей и т.д.

Для совместного использования с PHP подходит множество РСУБД, включая MySQL, PostgreSQL, Oracle и др. Однако, как показывает практика, чаще всего совместно с PHP используют именно MySQL.

MySQL — самая популярная в мире РСУБД с открытым исходным кодом, она бесплатна (для некоммерческого применения), быстро работает и весьма надежна. Еще одно ее преимущество — то, что многие провайдеры веб-хостинга предлагают доступ к базам данных MySQL, заметно упрощая жизнь разработчикам веб-приложений. Поэтому мы будем использовать именно MySQL при разработке электронного магазина TShirtShop.

Для обращения к реляционной базе данных мы будем использовать язык SQL (Structured Query Language — язык структурированных запросов). Каждая РСУБД использует свой диалект SQL, и, если вы решите использовать вместо MySQL другую РСУБД, вам, вероятно, придется модифицировать запросы, приведенные в листингах в этой книге.

Установка связи с MySQL

Чтобы обратиться к серверу РСУБД, нужно составить SQL-запрос, передать его серверу и получить от сервера ответ. SQL-запрос может содержать любое требование, касающееся базы данных, например "создать таблицу", "вывести список отделов", "удалить из базы данных товар 223" или "выдать всю информацию о желтых футболках в каталоге".

Независимо от того, что требует SQL-запрос от базы данных, нам нужно как-то передать его MySQL. MySQL поставляется с простым интерфейсом, использующим командную строку, который позволяет выполнять запросы и получать результаты их выполнения в текстовой форме. Если этот интерфейс покажется вам неудобным, не волнуйтесь: для него есть множество альтернатив. В частности, есть несколько товаров от сторонних разработчиков, позволяющих работать с базами данных MySQL с помощью удобного графического интерфейса. Многие провайдеры веб-хостинга предлагают работать с базами данных с помощью phpMyAdmin (это самый распространенный интерфейс для веб-клиентов MySQL). Очень советую вам познакомиться с этим инструментом. Вы можете также использовать какой-нибудь из графических клиентов, например Toad for MySQL (`http://www.quest.com/toad-for-mysql/`).

Интеграция с базой данных с помощью PDO

PDO (PHP Data Objects — объекты данных PHP) — это библиотека для доступа к данным, поставляемая вместе с PHP начиная с версии 5.1. Она также доступна в виде расширения PECL для PHP 5.0 (PECL — это репозиторий расширений для PHP, доступный по адресу `http://pecl.php.net/`). Официальное руководство по PDO, включая инструкции по установке, доступно по адресу `http://www.php.net/pdo`.

PDO предлагает единый способ доступа к различным источникам данных. Эта библиотека упрощает портирование приложений между разными платформами и повышает гибкость приложений, поскольку при изменениях уровня данных изменения в коде остаются минимальными (во многих случаях достаточно только изменить строку, создающую соединение с базой данных).

Познакомившись с уровнем абстрагирования данных в PDO, вы сможете использовать одни и те же приемы в разных проектах, предъявляющих разные требования к базам данных.

Чтобы продемонстрировать разницу между доступом к базе данных из обычного PHP-приложения и приложения, использующего PDO, давайте рассмотрим два фрагмента исходного кода.

ЗАМЕЧАНИЕ

Если вы не понимаете, что делает этот код, ничего страшного. В последующих главах мы подробно проанализируем его работу.

Вот код, обращающийся к базе данных MySQL с помощью встроенных функций PHP.

```
// Соединяемся с MySQL
$link = mysql_connect('localhost', $username, $password);

if (!$link)
{
   die ('Could not connect: ' . mysql_error());
}

$db_selected = mysql_select_db('tshirtshop', $link);

if (!$db_selected)
{
   die ('Could not select database : ' . mysql_error());
}

// Выполняем SQL-запрос
$queryString = 'SELECT * FROM product';

$result = mysql_query($queryString);
if (!$result)
{
  die ('Query failed : ' . mysql_error());
}

// Закрываем соединение
mysql_close($link);
```

ЗАМЕЧАНИЕ

Если вы все же хотите вместо PDO применять встроенные функции расширения PHP MySQL, подумайте о том, чтобы использовать улучшенную версию этого расширения (mysqli). Более подробную информацию об этом расширении можно найти по адресу `http://www.php.net/manual/en/ref.mysqli.php`.

А вот код, выполняющий те же действия, но использующий PDO.

```
try
{
   // Создаем новый экземпляр PDO
   $database_handler =
      new PDO('mysql:host=localhost;dbname=tshirtshop',
      $username, $password);

   // Составляем SQL-запрос
   $sqlQuery = 'SELECT * FROM product';

   // Выполняем SQL-запрос
```

```
    $statement_handler = $database_handler->query($sqlQuery);

    // Получаем данные
    $result = $statement_handler->fetchAll(PDO::FETCH_ASSOC);

    // Очищаем экземпляр объекта PDO
    $database_handler = null;
}
catch (PDOException $e)
{
    /* Если что-то пойдет не так, мы можем перехватить исключение
       из объекта, вывести сообщение и прекратить выполнение
       сценария */
    print 'Error! <br />' . $e->getMessage() . '<br />';
    exit;
}
```

Версия кода, использующая PDO, длиннее, но в ней используется эффективный механизм обработки ошибок — это очень полезно при отладке приложения. Если вам непонятно, о чем идет речь, подождите, пока мы не начнем реально использовать PDO в последующих главах и подробно анализировать работу кода.

Кроме того, если вы используете PDO, вам не придется менять код, отвечающий за доступ к данным, если вы, например, решите вместо MySQL использовать PostgreSQL. В то же время первый фрагмент кода пришлось бы переделывать целиком (в частности, использовать функции `pg_connect` и `pg_query` вместо `mysql_connect` и `mysql_query`). Кроме того, некоторые функции для работы с PostgreSQL используют параметры, которых нет в функциях для MySQL.

Применяя уровень абстрагирования данных (например, PDO), вам, вероятно, придется менять строку соединения при смене базы данных. Учтите, что пока что мы говорим только о PHP-коде, взаимодействующем с базой данных. Если новая база данных будет использовать другой диалект SQL, вам, вероятно, придется изменять и строки запросов.

ЗАМЕЧАНИЕ

Чтобы свести к минимуму необходимость изменения запросов, старайтесь делать их как можно более соответствующими стандарту SQL. Более подробно мы поговорим об SQL в главе 4.

MySQL и трехуровневая архитектура

Думаю, вам уже ясно, что MySQL относится к уровню данных. Но если до сих пор вам не приходилось работать с базами данных, вы могли не понять, что MySQL — это не просто хранилище для данных. Кроме самих данных MySQL может хранить и управляющую логику в виде хранимых процедур, а также контролировать целостность данных, проверять их корректность и многое другое.

Для общения с MySQL вам не придется применять SQL — язык, используемый в реляционных базах данных. SQL позволяет отдавать РСУБД приказы наподобие "выдай мне детали последних десяти заказов" или "удали товар номер 228".

Хотя и можно составлять SQL-запросы непосредственно в коде на PHP и сразу же отправлять их на выполнение, это не лучшее решение, поскольку оно делает программу уязвимой к ошибкам, снижает производительность и усложняет последующие модернизации. В нашем проекте мы будем хранить управляющую логику уровня данных в *функциях базы данных* (database functions).

Код, приведенный в этой книге, проверялся на MySQL 5.0 и MySQL 5.1. Место сервера MySQL в трехуровневой архитектуре показано на рис. 2.5.

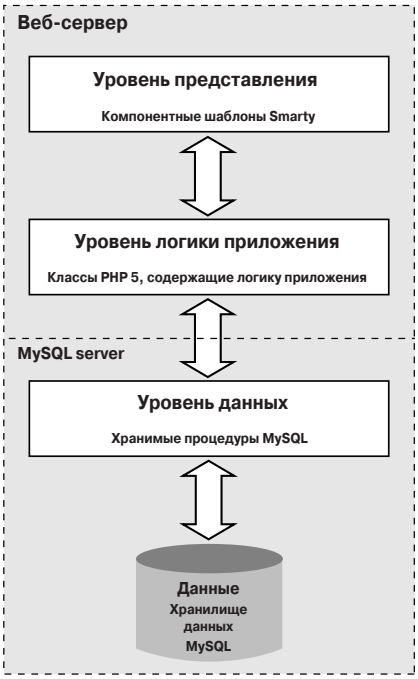

Рис. 2.5. Место MySQL в архитектуре приложения

Выбор стандартов именования и оформления кода

Возможно, на первый взгляд, стандарты именования и оформления кода покажутся вам не слишком важными, но их не стоит игнорировать. Если код писать, не следуя правилам оформления, в результате получится приложение, в котором будет очень сложно разобраться, не говоря уже о том, чтобы отлаживать его или модифицировать. Но если вы будете четко следовать правилам оформления, вы сможете с самого начала считать свое приложение наполовину задокументированным. Это важный вклад в будущее проекта, особенно если над ним одновременно работает много людей.

ПОДСКАЗКА

В одних компаниях есть официальные стандарты именования и оформления кода, а в других вы можете использовать такие стандарты, какие сочтете нужным. В любом случае самое главное — *единообразие в оформлении всего написанного кода*. Еще один важный совет: обязательно пишите комментарии к своему коду (чем больше, тем лучше).

Правила именования относятся ко многим элементам кода в проекте просто потому, что почти у всех элементов кода есть имена: сам проект, файлы, классы, переменные, методы, параметры методов, таблицы базы данных, столбцы этих

таблиц и т.д. Если вы не будете следовать каким-то правилам в задании имен, через неделю вы не поймете ни строчки из кода, который сами написали.

При разработке TShirtShop мы использовали набор правил именования, популярный у разработчиков на PHP. Некоторые правила из этого набора приведены ниже и проиллюстрированы фрагментом кода.

- Имена классов и методов записываются слитно, первые буквы слов в именах — заглавные (например, `WarZone`).
- Имена открытых атрибутов классов записываются по тем же правилам, но в начале этих имен ставится буква `m`, например `$mSomeSoldier`.
- Имена закрытых атрибутов классов записываются так же, как имена открытых, но в начале этих имен ставится знак подчеркивания, например `$_mSomeOtherSoldier`.
- Имена методов записываются слитно, слова в этих именах, кроме первого слова, записываются с прописной буквы, например `$someEnemy`, `$someOtherEnemy`.
- Имена переменных записываются строчными буквами, слова в них разделяются знаками подчеркивания, например `$master_of_war`.
- Элементы баз данных именуются по тем же правилам, что и переменные, например столбец `department_id`.
- Старайтесь использовать в коде отступы фиксированной величины (скажем, четыре символа на уровень, хотя в этой книге используются три символа из-за ограниченной ширины страниц).
- Вот пример кода, оформленного по этим правилам.

```
class WarZone
{
   public $mSomeSoldier;
   private $_mSomeOtherSoldier;

   function SearchAndDestroy($someEnemy, $someOtherEnemy)
   {
      $master_of_war = 'Soldier';
      $this->mSomeSoldier = $someEnemy;
      $this->_mSomeOtherSoldier = $someOtherEnemy;
   }
}
```

Кроме того, вам стоит сразу решить, какие кавычки вы будете использовать для ограничения строк. PHP, HTML и JavaScript позволяют использовать и одинарные кавычки (апострофы), и двойные. В этой книге одинарные кавычки используются в коде HTML и JavaScript, а двойные — в PHP. Хотя для JavaScript выбор кавычек — это дело вкуса (можете использовать одинарные, но только делайте это постоянно), в PHP одинарные кавычки обрабатываются быстрее, более безопасны и вызывают меньше ошибок при написании кода. Более подробно строки в PHP описаны в документации по адресу `http://php.net/types.string`. Кроме того, две полезные статьи по строкам в PHP можно найти по адресам `http://www.sitepoint.com/print/quick-php-tips` и `http://www.jeroenmulder.com/weblog/2005/04/php_single_and_double_quotes.php`.

Резюме

В этой главе мы коснулись многих тем. Мы обсудили трехуровневую архитектуру и разобрались, каким образом она помогает создавать гибкие и масштабируемые приложения. Кроме того, мы выяснили, какие места в этой архитектуре будут занимать используемые технологии.

Если вы чувствуете себя перегруженным информацией, не волнуйтесь. В следующей главе мы приступим к реальной работе по созданию первой части нашего сайта. Каждый шаг этой работы мы будем подробно описывать, и вы поймете, как работает каждый элемент приложения.