Веб-сервис на основе SDK для распознавания документов*

Д. П. Маталов[I,II], Е. Л. Плискин[III]

[I] Московский физико-технический институт (ГУ), г.Долгопрудный, Россия
[II] ООО «Смарт Энджинс Сервис», Москва, Россия
[III] Федеральное государственное учреждение «Федеральный исследовательский центр "Информатика и управление" Российской академии наук», г. Москва, Россия

Аннотация. Статья обобщает опыт разработки веб-службы на основе SDK для распознавания документов (OCR). Рассматриваются вопросы устойчивости и производительности веб-службы: способность не терять данные при высокой нагрузке и после перезапуска; способность своевременно обнаруживать ошибки, ограничивать их распространение и длительность; а также свойство детерминированности веб-службы в условиях параллельной обработки множества запросов. Высокая производительность подразумевает умеренные накладные расходы, связанные с приёмом запросов и отправкой ответов клиентам, помимо собственно распознавания. Описанное решение может применяться для создания веб-службы из любого SDK, который позволяет обрабатывать входные документы и получать из них выходные файлы, в том числе не обязательно в связи с технологиями оптического распознавания.

Ключевые слова: SDK, SOAP, REST, Java, Web service, оптическое распознавание, OCR, многопоточность.

DOI 10.14357/20718632190204

Введение

Эта статья обобщает опыт разработки веб-службы на основе SDK (комплекта для разработки программного обеспечения, software development kit) для оптического распознавания документов (OCR). Отправной точкой нашего проекта был продукт «Smart IDReader SDK» [1-3] в виде набора библиотек (DLL, LIB), заголовочных файлов (.H), файлов данных различного формата и документации. С помощью этого комплекта любой программист, не знакомый с алгоритмами оптического распознавания, может написать программу на языке C++ с использованием готового прикладного интерфейса (API), включающего следующие типы операций: инициализировать SDK, инициализировать новый документ, задать тип документа и другие параметры распознавания, загрузить изображение из файла, выполнить распознавание, получить тексты полей документа и оценки вероятности в формате JSON, получить исправленное изображение документа и изображения полей. Здесь мы перечислили эти типы операций только для того, чтобы обозначить контекст предметной области. Однако, для нашего проекта веб-службы специфика именно *оптического распознавания* не существенна и в остальной части статьи мы её не

* Работа выполнена при финансовой поддержке РФФИ, гранты № 17-29-03236 и 17-29-03263.

касаемся. Для нас важно то, что SDK позволяет обрабатывать входные документы (JSON с параметрами задания + изображение документа) и получать из них выходные файлы (JSON с распознанными текстами + возможно, дополнительные файлы изображений).

Целью проекта была разработка веб-службы, которая сможет быть развёрнута либо на публично доступном Интернет-сервере, либо внутри корпоративной сети. В 2019 году не нужно долго объяснять, зачем создаются веб-службы, но всё же скажем несколько слов. Во-первых, веб-технологии модны и повсеместны. Во-вторых, в отличие от SDK, веб-службой легко пользоваться правильно, но трудно воспользоваться неправильно, иначе, чем было задумано авторами. В-третьих, даже и в 2019 году устойчивую и высокопроизводительную веб-службу разработать всё ещё не так-то просто, и мы надеемся, что наша разработка и эта статья могут способствовать широкому внедрению технологий оптического распознавания. Говоря об устойчивости, мы подразумеваем способность веб-службы не терять данные, в том числе при высокой нагрузке и после перезапуска; способность своевременно обнаруживать ошибки, ограничивать их распространение и длительность; а также свойство детерминированности: одинаковые запросы должны давать одинаковые результаты, в том числе, независимо от параллельного обслуживания других запросов. Высокая производительность подразумевает умеренные накладные расходы, связанные с приёмом запросов и отправкой ответов клиентам, помимо собственно распознавания.

Двухуровневое решение

Под двухуровневым решением мы понимаем работу веб-службы в отдельном процессе П1 от процесса П2, в котором выполняется обработка документов при помощи заданного SDK. Изоляция процессов в операционной системе означает, что возможные ошибки внутри процесса П2 не сломают процесс П1, и что мы можем встроить в веб-службу логику наблюдения и управления процессами П2.

Запускать ли новый процесс П2 для обработки *каждого* документа? Это не лишено смысла: в таком случае ошибки SDK при обработке «трудного» документа не смогут повлиять на другие документы. Однако отдельные процессы для каждого документа могут быть не всегда практичны по следующим причинам.

- Нас интересует много-платформенное решение. Хотя запуск процессов в Unix-подобных ОС ненамного дороже запуска рабочих потоков (threads), но в ОС Windows такое решение обходится гораздо дороже. При переносе программ с Unix на Windows множественные запуски короткоживущих программ рекомендуют заменять множественными рабочими потоками внутри одного процесса.
- Процедура инициализации SDK может быть затратной по времени и по памяти, если авторы SDK не рассчитывали на запуск нового процесса для каждого документа.

В силу этих соображений в нашем решении процесс П2 запускается один раз в начале работы веб-службы и перезапускается только в случае ошибок. Далее мы называем процесс П2 «демоном».

Следующий вопрос: какой способ межпроцессного взаимодействия выбрать для передачи информации между процессом веб-службы (ВС) и демоном? На этот архитектурный выбор влияет желание хранить очереди заданий в файловой системе. Если HTTP-запросы поступают с большей интенсивностью, чем способен обработать демон, то между ВС и демоном возникает *входная* очередь заданий. А *выходная* очередь заданий возникает в обратном случае, если демон выдаёт результаты распознавания быстрее, чем веб-служба успевает отправлять ответы клиентам. Очереди заданий хотелось бы сохранять в случае штатного и нештатного перезапуска решения. Поэтому мы не используем иные способы обмена сообщениями между процессами ВС и демона, кроме обычных файлов и папок. При этом придерживаемся консервативного подхода, в соответствии с которым файл должен быть закрыт одним процессом, прежде чем станет доступным другому процессу.

Сочетание технологий REST и SOAP

Архитектурный стиль REST и стандартизованный протокол SOAP часто обсуждаются в литературе как альтернативные подходы к ди-

зайну веб-сервисов [4]. И тот и другой подходы имеют свою преданную клиентскую базу, но на деле эти технологии не исключают одна другую. В нашем проекте мы убедились, что в рамках веб-приложения Java нетрудно реализовать и REST, и SOAP «в одном флаконе». Для реализации REST мы использовали пакеты `javax.ws.rs.*` из библиотеки JAX-RS [18], а для реализации SOAP мы использовали пакеты `javax.jws.*` из библиотеки JAX-WS [19]. Указанные инфраструктурные пакеты избавляют разработчика веб-службы от деталей передачи информации по сети. Хотя для REST может использоваться формат данных JSON, а для SOAP формат данных XML, но и в том и в другом случаях инфраструктура способна доводить запросы до веб-службы и принимать от веб-службы ответы для отправки клиентам в виде однотипных, спроектированных разработчиком, объектов Java. Мы предлагаем снабжать веб-службу распознавания точками входа REST и SOAP одновременно, чтобы лучше отвечать потребностям клиентов с различными вкусами и ограничениями.

Функциональная схема решения

Функциональная схема решения изображена на Рис. 1. В верхней половине рисунка показаны две компоненты решения: слева веб-служба (ВС) на языке Java, справа программа распознавания (демон) на языке C++. В нижней половине рисунка показаны объекты файловой системы, которые передаются между веб-службой и демоном. Модуль веб-службы, условно обозначенный как «менеджер демона» (МД), отвечает за запуск программы распознавания и затем следит за её работой двумя способами: периодически проверяя дату изменения файла «пульса» демона, получая от операционной системы (ОС) сигнал о завершении процесса демона. За исключением такого сигнала от ОС, всё остальное взаимодействие между ВС и демоном осуществляется через обычные файлы и папки файловой системы. Начиная от приёма задания и изображения документа для распознавания от веб-клиента, и до отправки результата распознавания клиенту, на каждом шаге обработки задания информация сохраняется в файловой системе.

Этим обеспечивается наблюдаемость процесса и устойчивость в случае перезапуска ВС и/или демона. В файловой системе хранится и статистика обработанных заданий по часам, дням и месяцам. Веб-служба включает следующие модули.

- Модуль REST и модуль SOAP веб-службы принимают внешние HTTP-запросы. Каждый из этих модулей является не более чем тонкой обёрткой веб-службы и обеспечивает альтернативную «точку входа». Клиенты могут выбирать точку входа REST или SOAP по своему вкусу. Новые задания передаются Приёмщику заданий, а запросы на выдачу результатов передаются Кладовщику. Рабочими потоками для обслуживания запросов управляет инфраструктура веб-сервера, работающего по технологии J2EE, такого как Apache Tomcat [17]. Параллельное обслуживание запросов осуществляется в различных рабочих потоках.
- Модуль «приёмщик заданий» при получении нового задания на распознавание проверяет его, создаёт новую папку задания в рабочей папке «work» и сохраняет в ней изображение документа и файл `input.json`.
- Модуль «синхронизатор» используется для «синхронных» заданий, т.е. таких, в которых указано, что клиент в ответ на свой запрос желает получить результаты распознавания. В таком случае рабочий поток запроса направляется в Синхронизатор, где «затормаживается» и ожидает в продолжение времени, пока выполняется обработка документа. Альтернативно клиент может либо задать адрес для уведомления (обратного вызова), либо опрашивать сервер для получения результатов распознавания.
- Модуль «менеджер демона» (МД) обеспечивает запуск демона и следит за его активностью. При необходимости перезапуска демона МД создаёт файл «`daemon.die`». Увидев такой файл, демон пытается корректно завершить свою работу. Если этого не происходит в течение установленного количества секунд, то МД убивает процесс демона. Затем МД заново запускает новый экземпляр демона.
- Модуль архивации и статистики отвечает за перемещение заданий из локального архива во внешний архив. Обработанные задания могут быстро перемещаться из рабочей папки в локаль-

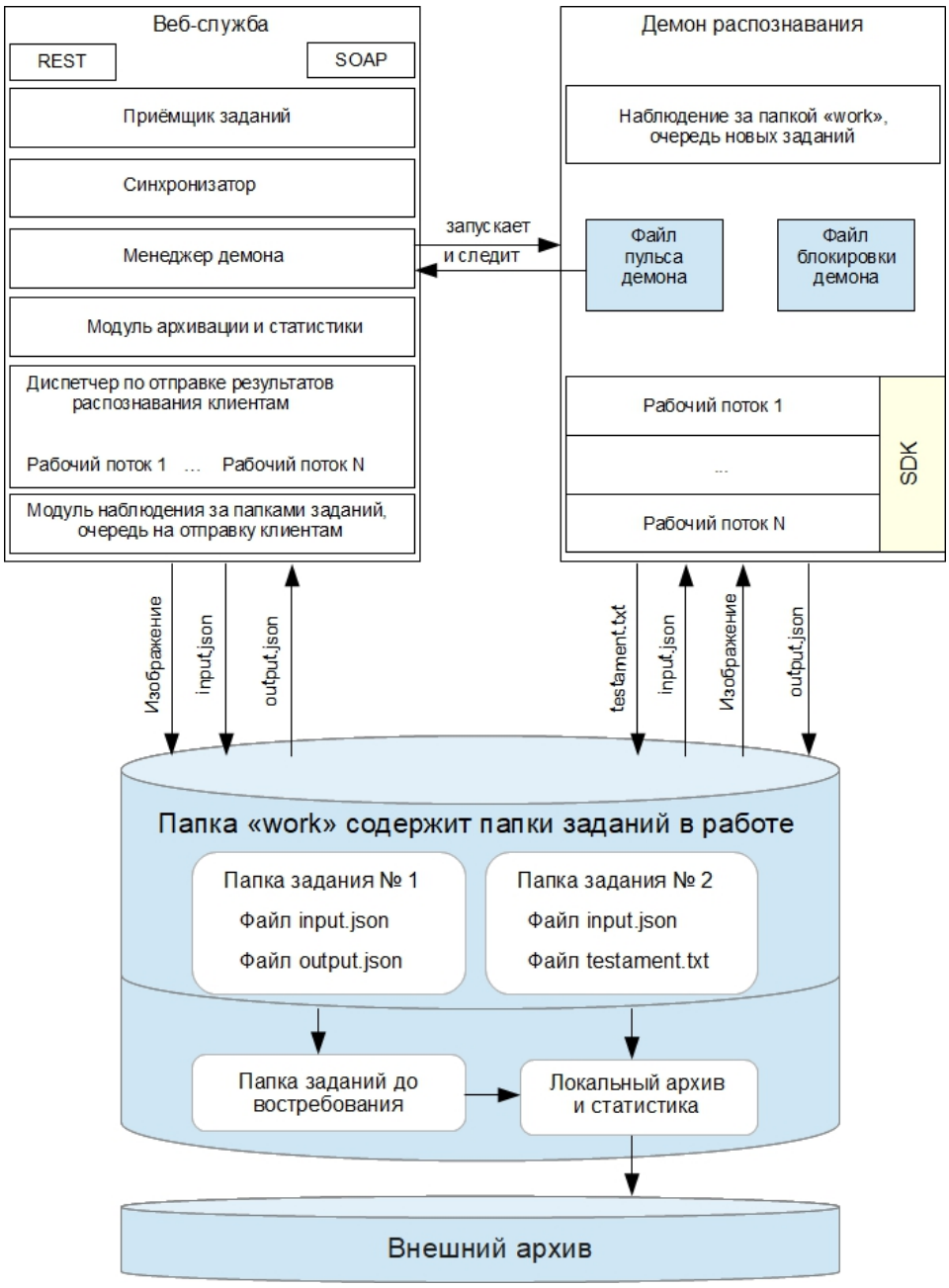

Рис. 1. Функциональная схема решения

ный архив, а затем после статистического учёта перемещаются в (возможно, расположенный на другом компьютере) внешний архив. Статистика текущего месяца, дня и часа загружается из соответствующих JSON-файлов при запуске веб-службы, непрерывно обновляется в памяти и периодически сохраняется на диске.

- Модуль «диспетчер» работает совместно с модулем наблюдения за папками заданий. Эти два модуля отвечают за доставку результатов распознавания клиентам. Модуль наблюдения отслеживает появление в папках заданий выходного файла «`output.json`». Появление такого файла в папке задания означает, что демон полностью закончил обработку задания. Такое задание ставится в очередь на отправку, откуда его извлекает диспетчер. В распоряжении диспетчера имеется несколько рабочих потоков, каждый из

которых последовательно обрабатывает полученные от диспетчера готовые задания.

- Рабочие потоки диспетчера (РПД). В зависимости от заданных клиентом параметров запроса, РПД может выполнить одно из следующих действий: (1) разбудить ожидающий в синхронизаторе поток для возврата результатов клиенту; (2) переместить задание в папку «до востребования», откуда клиент сможет получить результаты при помощи дополнительного запроса; или (3) отправить клиенту уведомление (обратный вызов) с результатами распознавания, при этом может быть сделано несколько попыток отправки уведомления, с возрастающими паузами. В случаях (1) и (3) задание перемещается из папки «work» в папку локального архива.

- Модуль «кладовщик» (на рисунке не показан) выдаёт клиентам задания из папки «до востребования», включая файл `output.json` и возможно, дополнительные файлы изображений. После того как все артефакты будут выданы клиенту, задание перемещается из папки «до востребования» в локальный архив.

Решение включает следующие объекты файловой системы.

- Рабочая папка «work» содержит новые задания и задания, находящиеся в процессе обработки демоном. Для каждого задания в папке «work» создаётся папка, имя которой соответствует уникальному, заданному клиентом, коду задания. Рядом с папкой «work» находятся папка «до востребования» и папка «локального архива».

- В каждой папке задания содержатся следующие файлы: (1) файл изображения, подлежащий распознаванию; (2) входной файл задания «`input.json`»; (2) временный «файл завещания демона» «`testament.txt`» создаётся демоном в начале обработки задания, чтобы не начинать обрабатывать аварийное задание повторно после перезапуска демона; (3) выходной файл с результатами распознавания «`output.json`»; (4) по желанию клиента, дополнительные выходные файлы с изображениями полей документа.

- Папка «до востребования» содержит папки таких заданий, в параметрах которых указано, что клиент планирует опрашивать сервер для получения результатов распознавания. После успешной выдачи результатов клиенту по требованию, папка задания перемещается в локальный архив.

- Папка локального архива содержит задания, обработка которых полностью закончена, включая отправку результатов клиентам. Здесь задания учитываются для статистики и отсюда периодически перемещаются во внешний архив.

Помимо папок заданий, следующие три файла используются для управления работой демона.

- Файл пульса демона «`daemon.pulse`» создаётся демоном в начале работы, а затем через каждые несколько секунд открывается и закрывается, для обновления даты изменения. Со своей стороны, модуль МД веб-службы периодически проверяет дату изменения этого файла. Если дата файла не меняется в течение установленного времени, то МД предполагает, что демон «завис» и пытается его остановить и перезапустить.

- Файл остановки демона «`daemon.die`» создаётся модулем МД веб-службы при необходимости остановки демона. Со своей стороны, демон периодически проверяет, не появился ли такой файл. Увидев данный файл, демон прекращает слежение за новыми заданиями, очищает очередь заданий, дожидается завершения обработки текущих заданий рабочими потоками и (по возможности) корректно завершает свою работу.

- Файл блокировки демона «`daemon.lock`» создаётся (если не существует в установленном месте) и эксклюзивно открывается на запись демоном в начале работы. Если открыть файл не удалось, это означает, что работает другой экземпляр демона. В этом случае демон не начинает обработку заданий, а немедленно завершается. Файл блокировки при нормальной работе никогда не удаляется.

Программа распознавания (демон) включает главный поток и несколько рабочих потоков.

- Главный поток демона непрерывно просматривает оглавление папки «work», находит в ней новые папки заданий и ставит их в очередь на выполнение. При каждом просмотре оглавления папки «work» в очередь добавляются такие папки заданий, в которых (1) имеется входной файл `input.json`, и при этом (2) нет ни

выходного файла `output.json`, и при этом (3) задание не находится на выполнении ни в одном из рабочих потоков демона. Если ни одного нового задания не нашлось, то перед следующим просмотром оглавления папки «work» главный поток делает небольшую паузу 100 миллисекунд, чтобы не перегружать процессор на холостом ходу демона. Кроме того, главный поток периодически открывает и закрывает файл пульса `daemon.pulse`. Кроме того, главный поток периодически проверяет, не появился ли файл остановки демона `daemon.die`.

- Каждый рабочий поток демона (РП) последовательно обрабатывает одно задание за другим, извлекая их из очереди. Обработка задания начинается с того, что РП проверяет, нет ли в папке задания файла завещания `testament.txt`. Если такой файл имеется, то распознавание повторно не выполняется, а вместо этого в папке задания формируется выходной файл «`output.json`» с сообщением об ошибке. Иначе, РП создаёт файл завещания, загружает входной файл задания `input.json` и приступает к распознаванию документа. По завершении распознавания РП формирует необходимые клиенту выходные файлы изображений, удаляет файл завещания и наконец, в последнюю очередь, создаёт выходной файл `output.json`. Появление этого файла в папке задания служит сигналом веб-службе о готовности результатов задания к отправке клиенту.

Взаимодействие веб-службы с демоном

Для целей настоящей работы предположим, не в обиду разработчикам SDK, что программа распознавания может ломаться двумя способами: 1) аварийно завершаться и 2) «зависать». Зависание можно диагностировать двояко: 2.1) как превышение априорного лимита времени распознавания документа без получения ответа и 2.2) как отсутствие «пульса» – ожидаемых периодических действий программы распознавания, таких как обновление определённого файла. Второй способ 2.2) позволяет быстрее диагностировать «жёсткое» зависание программы распознавания, когда останавливаются все её рабочие потоки и производительность веб-службы распознавания фактически падает до нуля. Такие критические ситуации, сколь бы они ни были редки, следует диагностировать как можно быстрее. И хотя способ 2.1) позволяет диагностировать зависание даже одного рабочего потока внутри программы распознавания, но одного его недостаточно. Если большинство (простых) документов распознаются намного быстрее, чем максимально допустимое время распознавания для сложных документов, то способ 2.1) может допускать неприемлемо длительные (десятки секунд) задержки диагностики жёстких зависаний. Поэтому целесообразно сочетать два указанных способа диагностики зависаний. Заметим, что способ 2.1) можно усовершенствовать, если ввести дифференцированные априорные оценки максимального времени распознавания для различных типов документов.

Мы хотим, чтобы в случае аварийного завершения программа распознавания была бы автоматически (без участия человека) и оперативно (с минимальной паузой) перезапущена, а в случае диагностики зависания программу распознавания следует убить и перезапустить.

Если программа распознавания сломалась при распознавании некоторого документа, то мы не хотим давать её «второй шанс» снова сломаться на том же документе, что могло бы привести к бесконечному циклу уже на уровне управляющей программы. Эта проблема решается с помощью файла «завещания». Прежде чем начать распознавание документа, программа распознавания создаёт в папке задания пустой файл завещания «`testament.txt`». Если такой файл уже существует, то документ повторно не обрабатывается, а в папке задания формируется выходной файл «`output.json`» с сообщением об ошибке. Иначе, после успешного распознавания документа файл завещания удаляется.

Взаимодействие с клиентами

Клиенты могут взаимодействовать с веб-сервисом распознавания либо синхронно, либо асинхронно. Под синхронным взаимодействием мы понимаем ситуацию, когда клиент отправляет в веб-службу запрос на распознавание,

включая изображение и сопутствующие параметры, и ожидает ответа на свой запрос в продолжение времени, пока на сервере выполняется обработка документа. Результаты распознавания возвращаются клиенту в виде ответа на его запрос. Напротив, при асинхронном способе взаимодействия клиент отправляет HTTP-запрос, который только ставит задание в очередь на сервере и сразу завершается. После обработки документа сервером ответ может быть либо отправлен клиенту обратным вызовом (callback), либо клиент может опрашивать сервер до тех пор, пока не получит результаты распознавания.

В общем случае программа распознавания может производить не один, а несколько выходных файлов. Файл с текстовыми результатами распознавания может иметь формат JSON или XML. Помимо этого, программа распознавания по требованию клиента может генерировать файлы изображений, включая исправленное изображение документа и изображения отдельных полей, для удобства ручной проверки и корректировки результатов распознавания. Если клиент прислал снятое под углом, далёкое от прямоугольной формы фото документа, то исправленное изображение может быть ближе к фронтальному.

Достоинство «наивной» синхронной схемы с единственным HTTP-запросом состоит в её простоте для клиента, но возможность нескольких выходных файлов усложняет картину. Для получения дополнительных файлов клиенту могут понадобиться дополнительные запросы. При этом эффективная и устойчивая реализация *синхронной* схемы на сервере может быть сложнее и потребовать бóльших вычислительных ресурсов, чем реализация *асинхронной* обработки запросов. Это связано, с одной стороны, с тем, что в синхронной схеме во время обработки документа на сервере необходимо сохранять канал связи с клиентом. С другой стороны, жёсткая синхронная связь (tight coupling) между веб-сервисом и программой распознавания может быть менее устойчивой, чем слабо связанное (loose coupling) асинхронное взаимодействие через очередь заданий. Хотя для клиентов синхронная схема вызова службы может выглядеть простой и естественной, но с точки зрения серверной реализации, напротив, естественной является асинхронная, конвейерная схема обслуживания запросов.

Мы предлагаем реализовать синхронный способ обслуживания клиентов как надстройку над базовым асинхронным серверным механизмом. Клиент указывает в запросе признак синхронного режима, если желает дожидаться результатов распознавания в ответ на свой запрос. Серверный рабочий поток T, принявший такой синхронный запрос, ставит задание в очередь, но не завершается сразу, а погружается в состояние ожидания. После того, как общим (асинхронным по сути) серверным механизмом будут получены результаты распознавания, поток T пробуждается и возвращает ответ клиенту. Синхронный режим можно рекомендовать либо для простых документов, распознавание которых, как правило, не занимает более нескольких секунд, либо для начальных экспериментов на этапе тестирования веб-службы пользователями.

Вопросы синхронизации и детерминированности

В этом разделе рассматриваются «под лупой» (1) некоторые вопросы двусторонней передачи информации между процессом ВС и процессом демона через файловую систему; а также (2) некоторые детали синхронизации рабочих потоков отдельно внутри каждой из двух компонент решения: внутри ВС и внутри демона. Этими деталями определяется детерминированность алгоритма работы веб-службы: одинаковые входные данные заданий на распознавание должны приводить к одинаковым результатам, независимо от параллельного обслуживания других запросов. Мы также коснёмся вопроса о корректной остановке программы.

В предлагаемом решении взаимодействие между процессом ВС и процессом демона опирается на объекты (файлы и папки) файловой системы. Этим определяется сохранение информации при перезапуске процессов. В современных файловых системах (ФС) для таких популярных серверных операционных систем (ОС), как Windows, Linux и macOS, имеются инструменты для координации межпрограмм-

ного взаимодействия, включая различные виды «замков» или блокировок файлов, областей внутри файлов и метаданных. Для целей настоящего решения достаточно использовать следующие базовые механизмы синхронизации процессов средствами файловой системы:

- Эксклюзивное открытие файла. В языке C++ для эксклюзивного открытия файла указываются флаги (`O_CREAT|O_EXCL`). В языке Java для эксклюзивной блокировки файла используется класс `FileLock`. Если процессу удалось заблокировать файл, то другой процесс не сможет сделать то же самое, пока файл не будет разблокирован первым процессом. Атомарность данной операции означает, что если два процесса примерно в одно и то же время пытаются заблокировать файл, то одному из них это обязательно удастся сделать. В данном решении эксклюзивное открытие применяется для файла блокировки демона `daemon.lock`, чтобы проверить, что работает только один экземпляр демона.
- Атомарность операции переименования файла. Для того чтобы передать данные из процесса П1 процессу П2 через файл Ф1 в папке А1, процесс П1 сначала создаёт в папке А1 временный файл Ф2, записывает в него данные, закрывает файл Ф2, а затем выполняет операцию переименования (Ф2 -> Ф1). Операция переименования файла в современных файловых системах выполняется *атомарно*. Это означает, что если процесс П2 наблюдает за оглавлением папки А2, то он увидит (и сможет открыть) файл Ф1 во всей его полноте, включая такие атрибуты файла, как дата создания и размер, и все записанные процессом П1 байты данных. В данном решении атомарное переименование файлов применяется при передаче задания веб-службой демону в файле `input.json` и при обратной передаче результатов распознавания демоном веб-службе в файле `output.json`.

Вопросы меж-поточной синхронизации возникают при работе модуля МН веб-службы, выполняющего наблюдение за папками заданий и поддерживающего очередь на отправку клиентам результатов распознавания. Для наблюдения за множественными папками заданий в модуле МН на языке Java целесообразно сочетать две техники: (1) перед записью входного файла `input.json` в папке задания создавать подписку на уведомления о событиях в папке задания при помощи службы `WatchService` [14]; (2) на старте веб-службы и затем периодически, через каждые несколько десятков минут, проводить «инвентаризацию» заданий в папке «work». Такая тактика позволяет сочетать оперативную (в масштабе миллисекунд) отправку результатов заданий клиентам со «сборкой мусора». Мусор здесь – это старые не до конца обработанные задания, которые могли образоваться вследствие различных ошибок, включая зависание или нештатное завершение веб-сервера, веб-службы или демона. Помимо очистки рабочей папки «work» от мусора, процедура инвентаризации в модуле МН веб-службы может инициировать перезапуск демона через модуль МД веб-службы, в случае если старый файл завещания `testament.txt` в папке задания позволяет предположить зависание одного из рабочих потоков демона.

Возвращаясь к между-поточной синхронизации в модуле МН, заметим, что здесь сходятся и обмениваются информацией, с одной стороны, рабочие потоки службы наблюдения `WatchService`, от которых поступают уведомления о событиях в папках заданий; а с другой стороны, рабочие потоки диспетчера (РПД), выполняющие отправку результатов заданий клиентам. Мы упоминали выше «очередь на отправку клиентам» в модуле МН. Теперь уточним, что в модуле МН не одна, а три списочные структуры данных. Во-первых, словарь, каждый элемент которого E включает (1) код задания `E.task_id` и (2) ключ подписки `E.key` от службы наблюдения. Ключом словаря служит `E.key`. Словарь используется для обработки уведомлений от службы `WatchService` об изменениях в папках заданий. Во-вторых, собственно очередь (first in – first out, FIFO) готовых к отправке клиентам заданий. Наконец, в-третьих, в модуле МН имеется словарь (СКЗ) кодов заданий, находящихся либо в очереди, либо в обработке диспетчером. Порядок работы модуля МН следующий. При обнаружении файла `output.json` в некоторой папке задания проверяется СКЗ, и если такого задания ещё нет в СКЗ, то оно одновременно включается в оче-

редь и в СКЗ. Из очереди задания извлекаются диспетчером. Отправку заданий клиентам выполняют потоки РПД, и лишь по окончании обработки задания поток РПД окончательно удаляет код задания из СКЗ в модуле МН. Такая тактика гарантирует, что ни одно задание не может быть повторно поставлено в очередь на отправку в модуле МН.

На стороне демона наблюдение за папками заданий несколько проще: демону достаточно просматривать оглавление единственной папки «work». Для предотвращения повторной обработки задания в демоне можно применить тактику, аналогичную вышеописанной. А именно, обнаруженные главным потоком демона новые задания включаются одновременно в очередь FIFO и в СКЗ. Из очереди задания извлекаются рабочими потоками демона. По окончании обработки задания оно удаляется также и из СКЗ.

Важное значение для предсказуемой работы решения имеет порядок остановки процессов, либо по требованию администратора, либо в случае автоматического обнаружения серьёзных ошибок. При остановке работы следует завершить то, что можно завершить, и сохранить то, что можно сохранить. При этом к началу процедуры завершения программы может оказаться, что некоторые рабочие потоки находятся в состоянии паузы. Кратковременные паузы (до секунды) не представляют проблемы, их можно переждать и продолжить процедуру завершения программы. Однако некоторые технологические процессы, такие как архивация и сборка мусора, могут штатно выполняться с большими, в десятки и сотни секунд, паузами. Другой пример – значительные паузы между несколькими попытками отправки результатов распознавания клиенту. В алгоритме работы соответствующих модулей желательно предусмотреть возможность корректного завершения работы по требованию, без затрудняющих анализ детерминированности программных прерываний. Так, вместо метода `Thread.sleep()` в Java для длительных (более секунды) пауз лучше использовать метод `Object.wait()`. Такое ожидание легко можно прервать методом `Object.notifyAll()`, при этом метод `wait()` штатно завершится без создания исключения. Альтернативно, вместо длительных пауз `Thread.sleep()` иногда уместно запускать порции работы по таймеру, который легко отключить в процедуре завершения программы.

Другие исследования

Веб-сервисы для обработки изображений и оптического распознавания не новость [20-24], но внутреннее устройство подобных решений почти не описано в литературе. В открытом исследовательском проекте DIVAServices [12, 13] веб-служба предоставляет клиентам набор методов для анализа изображений документов. Для запуска методов на сервере в DIVAServices используются контейнеры Docker [26], представляющие собой облегчённые виртуальные машины. Запуск контейнера Docker по требованию может занимать менее секунды [25]. Такой подход позволяет изолировать веб-службу от непредвиденных ошибок чужого разнородного программного обеспечения для обработки документов. Ключевые слова здесь «чужое» и «разнородное» и это не совсем наш случай, если мы говорим об использовании определённого SDK.

В современной литературе веб-технологии часто ассоциируются с облачными платформами, в том числе для задач индексации и поиска изображений [29]. Многих исследователей привлекают вопросы организации устойчивых параллельных вычислений на многопроцессорных кластерах [5, 6] или вопросы приватности и безопасности, связанные со взаимным недоверием между сервером и клиентами [27, 28]. По контрасту, наша задача разработки веб-службы на основе SDK для оптического распознавания лежит в стороне от указанных направлений. Мы предполагаем, что (1) веб-служба работает на «обычном» сервере, а не на кластере, а для масштабирования может применяться архитектура веб-фермы [30]; и (2) хотя передача информации по сети может происходить с применением шифрования на уровне транспорта (HTTPS), но во всяком случае, в условиях полного доверия между клиентами и сервером.

Отметим кратко ещё несколько направлений исследований, примыкающих по контексту к нашей работе. Немало работ, таких как [9], посвящены вопросам «потребления» веб-служб клиентами. Работа [15] может служить примером создания веб-службы для доступа к широ-

кому набору вычислительных ресурсов. Новый актуальный аспект веб-технологий связан с созданием среды для *воспроизводимых* исследований в области обработки изображений [7, 16]. К интересующим нас вопросам надёжности примыкает исследование [8] способов измерения и прогнозирования времени отклика веб-сервиса. Недавно некоторые исследователи провозгласили переход от сервисно-ориентированной архитектуры к новой *микро*-сервисной парадигме [10, 11]. Микро-сервисы отличаются узкой функциональностью и слабыми взаимосвязями. Наша веб-служба распознавания может стать элементом такой микро-сервисной архитектуры.

Литература

1. Арлазаров В. В., Булатов К. Б., Усков А. В. Модель системы распознавания объектов в видеопотоке мобильного устройства // Труды ИСА РАН. — 2018. — Спецвыпуск, 2018. — С. 73-82. — DOI: 10.14357/20790279180508.
2. V. V. Arlazarov, O. A. Slavin, A. V. Uskov and I. M. Yanishevskiy, “Modelling the flow of character recognition results in video stream,” Bulletin of the South Ural State University. Ser. Mathematical Modelling, Programming & Computer Software, vol. 11, no 2, pp. 14-28, 2018, DOI: 10.14529/mmp180202.
3. K. B. Bulatov, V. V. Arlazarov, T. S. Chernov, O. A. Slavin and D. P. Nikolaev, “Smart IDReader: Document Recognition in Video Stream,” ICDAR2017, IEEE Computer Society, ISSN 2379-2140, ISBN 978-15-38635-86-5, pp. 39-44, 2017, DOI: 10.1109/ICDAR.2017.347.
4. Zur Muehlen M., Nickerson J. V., Swenson K. D. Developing web services choreography standards—the case of REST vs. SOAP //Decision Support Systems. – 2005. – Т. 40. – №. 1. – С. 9-29.
5. Amin Z., Singh H., Sethi N. Review on fault tolerance techniques in cloud computing //International Journal of Computer Applications. – 2015. – Т. 116. – №. 18.
6. Jhawar R., Piuri V. Fault tolerance and resilience in cloud computing environments //Computer and information security handbook. – Morgan Kaufmann, 2017. – С. 165-18.
7. Lamiroy B., Lopresti D. P. The DAE platform: a framework for reproducible research in document image analysis //International Workshop on Reproducible Research in Pattern Recognition. – Springer, Cham, 2016. – С. 17-29.
8. Jayathilaka H., Krintz C., Wolski R. Service-level agreement durability for web service response time //2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). – IEEE, 2015. – С. 331-338.
9. Wittern E. et al. Opportunities in software engineering research for web API consumption //Proceedings of the 1st International Workshop on API Usage and Evolution. – IEEE Press, 2017. – С. 7-10.
10. Dragoni N. et al. Microservices: yesterday, today, and tomorrow //Present and Ulterior Software Engineering. – Springer, Cham, 2017. – С. 195-216.
11. Karlsson E. The evolution and erosion of a service-oriented architecture in enterprise software: A study of a service-oriented architecture and its transition to a microservice architecture. – 2018.
12. Würsch M., Ingold R., Liwicki M. Sdk reinvented: Document image analysis methods as restful web services //2016 12th IAPR Workshop on Document Analysis Systems (DAS). – IEEE, 2016. – С. 90-95.
13. Würsch M. et al. Turning Document Image Analysis Methods into Web Services-An Example Using OCRopus //2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). – IEEE, 2017. – Т. 4. – С. 48-52.
14. Watching a Directory for Changes. https://docs.oracle.com/javase/tutorial/essential/io/notification.html
15. Cholia S., Skinner D., Boverhof J. NEWT: A RESTful service for building High Performance Computing web applications //2010 Gateway Computing Environments Workshop (GCE). – IEEE, 2010. – С. 1-11.
16. Boettiger C. An introduction to Docker for reproducible research //ACM SIGOPS Operating Systems Review. – 2015. – Т. 49. – №. 1. – С. 71-79.
17. Apache Tomcat. http://tomcat.apache.org/
18. Building RESTful Web Services with JAX-RS. https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cp1v/index.html
19. Building Web Services with JAX-WS. https://docs.oracle.com/cd/E19798-01/821-1841/bnayl/index.html
20. Google Cloud Vision API. https://cloud.google.com/vision/
21. Microsoft’s Computer Vision API. https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/
22. FreeOCR API. https://ocr.space/ocrapi
23. OCR Cloud 2.0 API. http://www.ocr-it.com/ocr-cloud-2-0-api/
24. Tabex-OCR-REST-API-Precise-Developers-OCR. http://pdfextractoronline.com/tabex-ocr-rest-api/
25. https://blog.iron.io/the-overhead-of-docker-run/
26. https://www.docker.com/
27. Xu Y. et al. A privacy-preserving content-based image retrieval method in cloud environment //Journal of Visual Communication and Image Representation. – 2017. – Т. 43. – С. 164-172.
28. Zhang L. et al. Pic: Enable large-scale privacy preserving content-based image search on cloud //IEEE Transactions on Parallel and Distributed Systems. – 2017. – Т. 28. – №. 11. – С. 3258-3271.
29. Hu H. et al. Web-scale responsive visual search at bing //Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. – ACM, 2018. – С. 359-367.
30. Aweya J. et al. An adaptive load balancing scheme for web servers //International Journal of Network Management. – 2002. – Т. 12. – №. 1. – С. 3-39.

Маталов Даниил Павлович. Студент магистратуры МФТИ (ГУ), г. Москва. Научный сотрудник – программист ООО «Смарт Энджинс Сервис», г. Москва. Количество печатных работ: 4. Область научных интересов: обработка изображений, распознавание образов, машинное обучение. E-mail: mataloff@gmail.com

Плискин Евгений Львович. Федеральное государственное учреждение «Федеральный исследовательский центр "Информатика и управление" Российской академии наук», г. Москва, Россия. Ведущий научный сотрудник, кандидат технических наук. Количество печатных работ: 24. Область научных интересов: автоматизированные информационные системы. E-mail: pliskin@isa.ru

Making a Web service from OCR SDK

D. P. Matalov[I, II], E. L. Pliskin[III]

[I] National Research University MIPT (Moscow Institute of Physics and Technology), 141700, Dolgorpudny, Russia
[II] LLC "Smart Engines Service", Moscow, Russia
[III] Federal Research Center "Computer Science and Control" RAS, 119333, Moscow, Russia

Abstract. This article summarizes authors' experience of developing a web service (WS) based on a document optical character recognition (OCR) software development kit (SDK). We consider issues of WS stability and performance, including: ability not to lose data under high load and after restart; ability to timely detect errors and limit their spread and duration; as well as deterministic WS behavior under conditions of parallel processing of multiple requests. High WS performance implies moderate overhead costs associated with receiving web requests and sending web responses to clients, besides of OCR engine costs itself. The described solution can be used to create a web service from any SDK which enables developer to process input documents and obtain output files from them, not necessarily in connection with optical recognition technologies.

Keywords: SDK, SOAP, REST, Java, Web service, optical character recognition, OCR, multithreading.

DOI 10.14357/20718632190204

Reference

1. Arlazarov V. V., Bulatov K. B., Uskov A. V. Model of object recognition system in a mobile device video stream // Proceedings of the ISA RAS. Special issue, 2018, pp. 73-82, DOI: 10.14357/20790279180508.
2. V. V. Arlazarov, O. A. Slavin, A. V. Uskov and I. M. Yanishevskiy, "Modelling the flow of character recognition results in video stream," Bulletin of the South Ural State University. Ser. Mathematical Modelling, Programming & Computer Software, vol. 11, no 2, pp. 14-28, 2018, DOI: 10.14529/mmp180202.
3. K. B. Bulatov, V. V. Arlazarov, T. S. Chernov, O. A. Slavin and D. P. Nikolaev, "Smart IDReader: Document Recognition in Video Stream," ICDAR2017, IEEE Computer Society, ISSN 2379-2140, ISBN 978-15-38635-86-5, pp. 39-44, 2017, DOI: 10.1109/ICDAR.2017.347.
4. Zur Muehlen M., Nickerson J. V., Swenson K. D. Developing web services choreography standards—the case of REST vs. SOAP //Decision Support Systems. – 2005. – Т. 40. – №. 1. – С. 9-29.
5. Amin Z., Singh H., Sethi N. Review on fault tolerance techniques in cloud computing //International Journal of Computer Applications. – 2015. – Т. 116. – №. 18.
6. Jhawar R., Piuri V. Fault tolerance and resilience in cloud computing environments //Computer and information security handbook. – Morgan Kaufmann, 2017. – С. 165-18
7. Lamiroy B., Lopresti D. P. The DAE platform: a framework for reproducible research in document image analysis //International Workshop on Reproducible Research in Pattern Recognition. – Springer, Cham, 2016. – С. 17-29.
8. Jayathilaka H., Krintz C., Wolski R. Service-level agreement durability for web service response time //2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). – IEEE, 2015. – С. 331-338.
9. Wittern E. et al. Opportunities in software engineering research for web API consumption //Proceedings of the 1st International Workshop on API Usage and Evolution. – IEEE Press, 2017. – С. 7-10.
10. Dragoni N. et al. Microservices: yesterday, today, and tomorrow //Present and Ulterior Software Engineering. – Springer, Cham, 2017. – С. 195-216.
11. Karlsson E. The evolution and erosion of a service-oriented architecture in enterprise software: A study of a service-oriented architecture and its transition to a microservice architecture. – 2018.
12. Würsch M., Ingold R., Liwicki M. Sdk reinvented: Document image analysis methods as restful web services //2016 12th IAPR Workshop on Document Analysis Systems (DAS). – IEEE, 2016. – С. 90-95.

13. Würsch M. et al. Turning Document Image Analysis Methods into Web Services-An Example Using OCRopus //2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). – IEEE, 2017. – Т. 4. – С. 48-52.
14. Watching a Directory for Changes. https://docs.oracle.com/javase/tutorial/essential/io/notification.html
15. Cholia S., Skinner D., Boverhof J. NEWT: A RESTful service for building High Performance Computing web applications //2010 Gateway Computing Environments Workshop (GCE). – IEEE, 2010. – С. 1-11.
16. Boettiger C. An introduction to Docker for reproducible research //ACM SIGOPS Operating Systems Review. – 2015. – Т. 49. – №. 1. – С. 71-79.
17. Apache Tomcat. http://tomcat.apache.org/
18. Building RESTful Web Services with JAX-RS. https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cp1v/index.html
19. Building Web Services with JAX-WS. https://docs.oracle.com/cd/E19798-01/821-1841/bnayl/index.html
20. Google Cloud Vision API. https://cloud.google.com/vision/
21. Microsoft's Computer Vision API. https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/
22. FreeOCR API. https://ocr.space/ocrapi
23. OCR Cloud 2.0 API. http://www.ocr-it.com/ocr-cloud-2-0-api/
24. Tabex-OCR-REST-API-Precise-Developers-OCR. http://pdfextractoronline.com/tabex-ocr-rest-api/
25. https://blog.iron.io/the-overhead-of-docker-run/
26. https://www.docker.com/
27. Xu Y. et al. A privacy-preserving content-based image retrieval method in cloud environment //Journal of Visual Communication and Image Representation. – 2017. – Т. 43. – С. 164-172.
28. Zhang L. et al. Pic: Enable large-scale privacy preserving content-based image search on cloud //IEEE Transactions on Parallel and Distributed Systems. – 2017. – Т. 28. – № 11. – С. 3258-3271.
29. Hu H. et al. Web-scale responsive visual search at bing //Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. – ACM, 2018. – С. 359-367.
30. Aweya J. et al. An adaptive load balancing scheme for web servers //International Journal of Network Management. – 2002. – Т. 12. – №. 1. – С. 3-39.

Matalov D. P. MIPT master's student, Moscow, Russia. Researcher-programmer at LLC "Smart Engines Service", pr. 60-letiya Oktyabrya, 9, Moscow, Russia. Number of publications: 4. Research interests: image processing, pattern recognition, machine learning. E-mail: mataloff@gmail.com

Pliskin E. L. PhD. Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russia. Leading Researcher. Number of publications: 24. Research interests: automated information systems. E-mail: pliskin@isa.ru