# Содержание

**1. Задание**

Реализовать на Java1.8+soap web-service, содержащий в себе метод **Result findNumber(Integer number)**, ищущий в 20 разных по составу больших текстовых файлах (каждый должен быть порядка Гб.) полученное на вход число n. Файлы состоят только из чисел, которые разделены между собой запятой. Результат работы необходимо записать в таблицу БД и вернуть объект Result в вызывающую систему.

Объект Result должен иметь следующую структуру:


где:
code—код выполнения программы (коды см. ниже).
fileNames—имена файлов, в которых удалось найти число.
error—описание ошибки, в случае её возникновения.

Структура таблицы должна быть следующей:

| | Name | DataType | Description |
|---|---|---|---|
| 1 | ID | NUMBER (*,0) | PK, auto-increment |
| 2 | CODE | VARCHAR2 (50 BYTE) | код выполнения программы (коды см. ниже) |
| 3 | NUMBER | NUMBER (*,0) | Число, переданное на вход. |
| 4 | FILENAMES | VARCHAR2 (100 BYTE) | имена файлов, в которых удалось найти число |
| 5 | ERROR | VARCHAR2 (100 BYTE) | Описание ошибки, в случае её возникновения. |

Коды выполнения программы, которые необходимо записать в колонку CODE и в поле code объекта Result:

| Code | В каком случае используется |
|---|---|
| 00.Result.OK | В случае, если число найдено. |
| 01.Result.NotFound | В случае, если число не найдено. |
| 02.Result.Error | В случае, если при выполнении программы возникла ошибка. |

Ограничений на использование фреймворков, технологий и приёмов программирования нет. Скрипт для создания таблицы приложить к проекту.
Будет плюсом если:
1.Будет использован Spring.
2.Будет использована одна из библиотек для логгирования.
3.Код будет задокументирован.
4.Будет приложен код для генерации тестовых файлов.

# 2. Алгоритм

Клиент, например, через SoapUI обращается к моему сервису по url:
**http://localhost:8080/finderfiles/FindNumberInFiles.wsdl**

После этого сервер читает файл **web.xml**, в котором указано, что все запросы на *http://localhost:8080/* будут читаться servlet-классом:
org.springframework.ws.transport.http.**MessageDispatcherServlet**

Spring-конфигурация сервиса: **service-ws-servlet.xml**
В ней прописано 2 класса для приема и преобразовании запроса из xml в Java-объект и дальнейшего обратного преобразования:
1)org.springframework.ws.server.endpoint.adapter.GenericMarshallingMethodEndpointAdapter
2) org.springframework.oxm.xmlbeans.XmlBeansMarshaller

*sws:dynamic-wsdl* отвечает за автоматическую генерацию WSDL-документа на основе созданной Xml-схемы.
*location* указывает на путь к схеме.
*locationUri* – адрес (относительно контейнера), по которому будет доступна WSDL-схема.
**<sws:xsd location="/WEB-INF/FindNumberInFiles.xsd" />**
За счёт этого когда клиент обращается по этому url (http://localhost:8080/finderfiles/FindNumberInFiles.wsdl ) он получает схему, описанную в /WEB-INF/FindNumberInFiles.xsd

В этой схеме прописано, что пользователь передаёт на сервер int параметр number, а в ответ получает String параметр reply

Атрибут *targetNamespace* – используемое пространство имен. Т.е. все созданные объекты будут располагаться в пакете org.example.FindNumberInFiles

Файл **build.xml** (используемый сборщиком **ant**) нужен для того, чтобы на основании созданной схемы ( FindNumberInFiles.xsd ) создать Java классы. Также для этого необходим **XMLBeans**. После сборки ant-ом в директории WEB-INF/lib появится **findnumberinfiles.jar**

Когда пользователь, подключившись к сервису по url http://localhost:8080/finderfiles/FindNumberInFiles.wsdl посылает запрос на сервер с параметром number запрос приходит классу **FindNumberInFilesEndpoint.java** (точку входа запроса клиента определяет аннотация **@Endpoint**)
Этой строкой кода:
**@PayloadRoot(localPart = "ServiceRequest", namespace = namespaceUri)**
определено, что при получении запроса от клиента будет вызван метод обозначенный этой аннотацией (**getService**)
Этот метод для поиска числа в файлах, формирования объекта Result, формирования xml и отправки значений объекта Result в вызывающую систему и в БД вызывает метод **findNumber класса FindNumberInFilesImpl.java**, передавая ему число (полученный от пользователя в запросе параметр) и возвращает пользователю ответ этого метода.
**String findMessage = findNumberInFiles.findNumber(userNumber).toString();**

## Алгоритм работы класса FindNumberInFilesImpl.java

Метод **findNumber** этого класса получает от класса FindNumberInFilesEndpoint.java число для поиска и возвращает классу FindNumberInFilesEndpoint.java объект Result. В этом методе прописываются файлы, в которых будет произведён поиск числа (добавляются в ArrayList filesForSearch)
Для поиска числа для каждого файла из этого ArrayList запускается метод **findNumberInOneFile(path)**.
После завершения поиска формируется объект Result, который и возвращается в вызывающую систему и записывается при помощи Hibernate в БД Oracle.Настройки для подключения к БД указаны в hibernate.cfg.xml. При этом работа этого класса логируется с помощью библиотеки log4j (настройки указаны в log4j.properties)
static Logger log = Logger.getLogger("example/FindNumberInFilesImpl");

**Алгоритм findNumberInOneFile**
Поиск числа в каждом фале осуществляется в методе:
boolean findNumberInOneFile(String path)
Этот метод принимает путь к файлу для поиска.
Т.к. каждый файл размером 1 Гб и не может быть сразу целиком прочитан в одну строку (не хватит памяти и будет вызвано исключение Java heap), поэтому этот метод читает файл по частям в символьный массив char[10000]

При таком чтении файла есть 2 особенности:
1) может получиться так что одно число будет разрезано на 2 потока и из-за этого мы можем посчитать его за 2 разных числа.
Решение:
В каждом потоке отрезать число после последней запятой, запомнить его и потом конкатенировать его с началом следующего потока.
2) Мы можем отрезать число (как описано в п.1), а этот поток последний и следующего за ним потока не будет. В таком случае после чтения всех потоков мы это число отдельно сравним со значением, которое передано для поиска.

**Порядок действий этого метода:**

1) Чтение файла в несколько потоков в ограниченный массив символов char[10000].

2) Формирование строки, прибавляя в начало потока"остаток" предыдущего потока.

3) Получение из этой строки массива строк, выполняя разделение по элементам относительно запятых.

4) Вычисление "остатка", чтобы не учитывать его при поиске числа в этом потоке, а присоединить этот "остаток" в начало следующего потока.

5) Ищем в полученном массиве строк заданное число

6) Сравниваем последнее число файла с искомым числом

На основании поиска будут получены значения статических переменных code, fileNames и error.

### 3. Как запускать?

1) В клиенте (например, SoapUI) обратиться к соответствующему url:

```
http://localhost:8080/finderfiles/FindNumberInFiles.wsdl
```

2) передать в поле number число для поиска

### 4. Требования для запуска

4.1) клиент (я использовал SoapUI)

4.2) БД Oracle (я использовал 11 версию)

Для того чтобы моя программа работала с БД требуется:

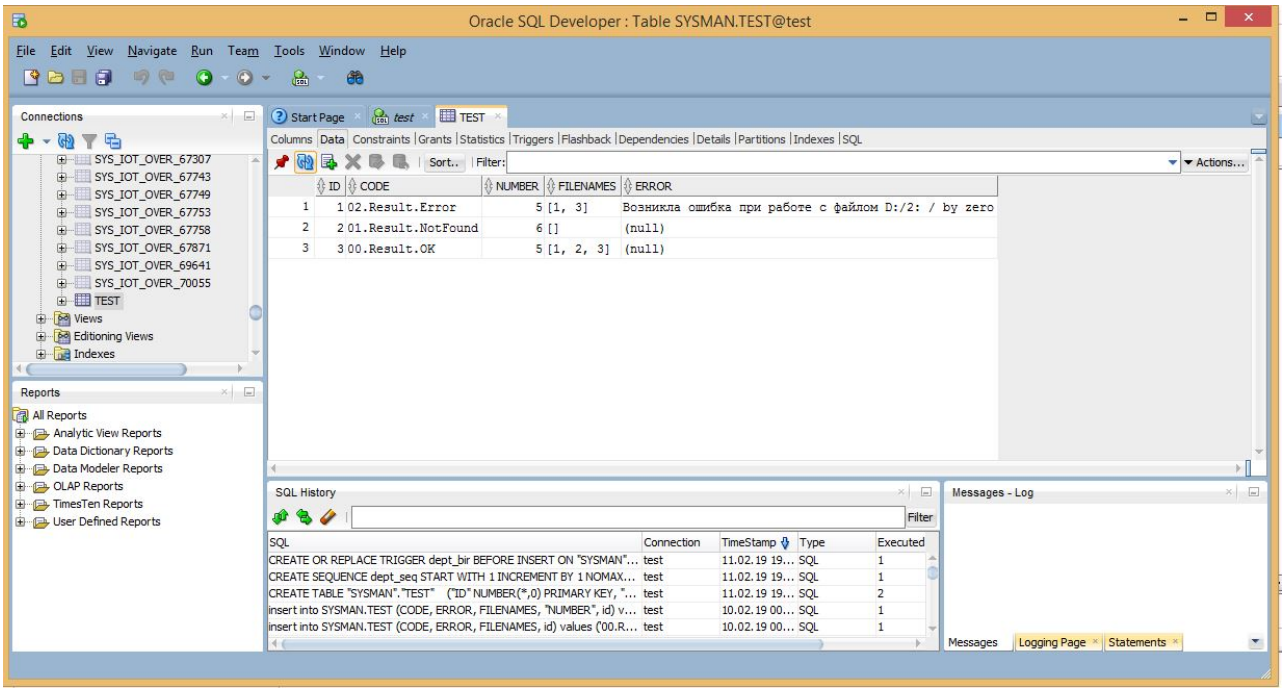
4.2.1) настроить доступ из java web service к БД (прописать параметры доступа к БД в файле /FindNumberInFiles/src/hibernate.cfg.xml )

4.2.2) накатить sql скрипт (findernumber.sql) передаю вместе с кодом

4.3) сгенерированные большие тестовые файлы (для генерации передаю проект FormFiles)

### 5. Пример результатов моего тестирования

Ошибку деления на ноль специально сам создал (потом убрал из кода)




Также потом протестировал свою программу на 20 больших 1 Гб разных файлах (как по условию задачи). На моём железе (процессор: AMD A8 2Ггц, ОЗУ: 8 Гб) программа успешно выполнилась за 22 минуты.

**6. Состав проекта**

1) java web-service (FindNumberInFiles)

2) проект для формирования больших (1 Гб) CSV-файлов, состоящих из Integer-значений (как по условию задачи) ( FormFiles )

3) sql-скрипт (findernumber.sql)

4) эта инструкция