Система НКРЯ как многоуровневое программное обеспечение

Гладилин Сергей Александрович, с.н.с. лаборатории зрительных систем ИППИ РАН
gladilin@iitp.ru

работа выполнена совместно с *Морозовым Дмитрием Алексеевичем,* м.н.с. лаборатории компьютерной лингвистики ИППИ РАН

Пример: задача построения условий поиска словосочетания

- декодирование HTTP-запроса
- обработка повторяющихся и отсутствующих GET-переменных
- преобразование нумерованных переменных в массив
- обработка расстояний, заданных пустым полем
- разбор сложных (с И, ИЛИ, НЕ, скобками) выражений в запросах
- преобразование запросов к поиску в “комбинаторном взрыве”
- учет разметки неизменяемых слов с помощью “0”
- преобразование поиска lex:*лог в поиск rlex:гол*
- правильный учет е/ё и старой орфографии
- особый синтаксис запроса SAAS для только негативных термов
- вычисление hit-коэффициента

Корпус в целом

- поисковый запрос
  - построение ограничений на подкорпус
  - **построение условий поиска словосочетания**
  - построение поисковых параметров
- обработка ответа на поисковый запрос
  - стандартный поиск
  - графики
  - биграммы
  - отбор подкорпуса
- индексация
  - ...

Цели

1. Определять, где именно в системе требуется вносить изменения

2. Контролировать, на что влияют внесенные изменения

Идея: водонепроницаемые переборки

		Уровень веб-интерфейса	Пользовательский уровень	Уровень хранилища
поисковый запрос	подкорпус			
	словосочетание			
	параметры			
обработка ответа на поисковый запрос	поиск			
	графики			
	биграммы			
	подкорпус			
индексация	...			

Достоинства и недостатки

- Ограничение влияния изменений
- Понятные ограниченные экосистемы на каждом уровне

- Поддержание собственной инфраструктуры на каждом уровне
- Усложнение межуровневой архитектуры

- При эволюционном внедрении “переборок” на первом этапе требуются многочисленные “подпорки и растяжки”

Когда нужны переборки?

1. Переносимость
2. Существенное различие логики на разных уровнях
3. Распределенные системы
4. Разделение зон ответственности

Переносимость НКРЯ

В настоящее время используется SAAS со схемой данных “мешок” и интегрированным прямым индексом.

Возможные изменения (кратко и среднесрочно):

- замены схемы данных
- вынос прямого индекса из SAAS в SQL
- размещение таблиц (1-грамм, например) в SQL

Разная логика на разных уровнях

- “мешок” плохо аппроксимирует желаемое поведение
- технические ограничения SAAS приводят к фрагментации текстов
- фрагментация текстов приводит к фрагментации запросов
- SQL - мощный и эффективный инструмент, но с кардинально иной моделью запросов

Распределенность системы

В настоящее время НКРЯ “не очень распределенная” ;-) система:

- веб-интерфейс
- SAAS

Современные подходы к созданию веб-интерфейсов предполагают переход к Single Page Application (SPA). Если произойдет внедрение SPA, то НКРЯ станет полноценной распределенной системой.

Если НКРЯ используется для автоматических запросов какой-нибудь другой системой, то он (кроме веб-интерфейса) становится частью еще и этой распределенной системы.

Разделение зон ответственности

В настоящее время зоны ответственности разделены между:

- лингвистами-разметчиками
- разработчиками программного обеспечения
- командой сервиса SAAS

Разработка SPA требует особой квалификации frontend-програмистов, существенно отличной от backend. Вся нынешняя команда разработчиков ПО обладает квалификацией только в части backend. Возможным решением является передача frontend и сайта НКРЯ (новости и т.д.) на аутсорсинг.

Где поставить переборки?

1. Отгородить веб-интерфейс

2. Отгородить хранилище данных (сейчас - SAAS)

3. Между ними останется уровень, содержащий универсальные операции, не зависящие ни от выбранной базы данных, ни от веб-интерфейса (в т.ч. от того, Single Page Application или нет). Это и есть “суть” НКРЯ ;-)

   Традиционно я называю этот уровень “пользовательским” или “содержательным”, но может объявим конкурс на лучшее название? ;-)

Принцип “идеальной компоненты”

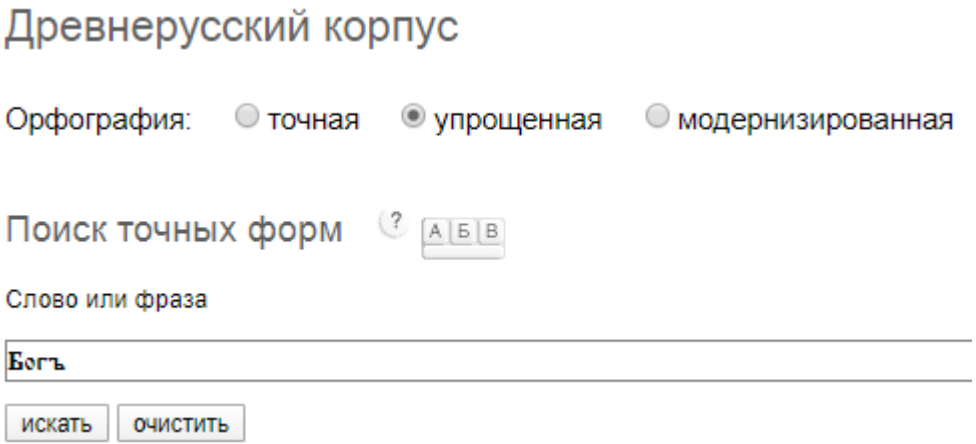
“Переборка” должна содержать “дверь” - программный интерфейс (API). Как он должен быть устроен?

Представим, что сейчас ничего нет и некоторые великие программисты (которые могут всё, что попросят) согласны разработать специально для нас самую лучшую в мире базу данных вместо SAAS. Какой у нее будет интерфейс? Какие запросы мы хотели бы ей задавать? В какой форме?

Ответив на этот вопрос, мы получим описание самого лучшего API для нужд корпуса.

Веб-интерфейс

Считаю, что тут важна максимальная близость API к тому, что видит пользователь глазами:




```
query = UserQueryEF(
    corpora       = 'old_rus',
    orthography   = 'simple',
    query         = 'Богъ'
)
```

Выдачу планируется оформлять в виде класса `UserQueryResult`, структура которого максимально близка к видимой пользователю выдаче, но (1) машиночитаема и (2) не описывает чисто внешнего оформления - цветов, отступов и т.д.

Пример: задача построения условий поиска словосочетания

- декодирование HTTP-запроса
- обработка повторяющихся и отсутствующих GET-переменных
- преобразование нумерованных переменных в массив
- обработка расстояний, заданных пустым полем
- разбор сложных (с И, ИЛИ, НЕ, скобками) выражений в запросах
- преобразование запросов к поиску в “комбинаторном взрыве”
- учет разметки неизменяемых слов с помощью “0”
- преобразование поиска lex:*лог в поиск rlex:гол*
- правильный учет е/ё и старой орфографии
- особый синтаксис запроса SAAS для только негативных термов
- вычисление hit-коэффициента

Хранилище данных - интерфейс

Планируется, что интерфейсом будут выступать классы `RepoQuery` и `RepoQueryResult`. Идеологически их структура будет похожа на структуру запроса и выдачи SAAS, но с рядом модификаций (по принцип идеального компонента):

- запрос будет задаваться не текстово, а в виде дерева объектов
- запрос формируется в виде ограничений на разбор, а не на слово
- операции с целыми текстами, а не фрагментами
- поддержка расстояний “с начала предложения” и “до конца предложения”
- полноценные AND, OR, NOT
- расширенная поддержка типов данных (как минимум - тип “множество”)
- сортировка не только по числовым полям
- выдача большого количества примеров (для экспорта в Excel)
- и т.д. и т.п.

Пример: задача построения условий поиска словосочетания

- декодирование HTTP-запроса
- обработка повторяющихся и отсутствующих GET-переменных
- преобразование нумерованных переменных в массив
- обработка расстояний, заданных пустым полем
- разбор сложных (с И, ИЛИ, НЕ, скобками) выражений в запросах
- преобразование запросов к поиску в “комбинаторном взрыве”
- учет разметки неизменяемых слов с помощью “0”
- преобразование поиска lex:*лог в поиск rlex:гол*
- правильный учет е/ё и старой орфографии
- особый синтаксис запроса SAAS для только негативных термов
- вычисление hit-коэффициента

Хранилище данных - реализация

Для реализации хранилища планируется создавать классы вида (названия условные, требуют проработки)

- `SAAS_Backend` - для текущей реализации хранилища через SAAS
- `SAAS2_Backend` - для реализации через SAAS, но с атрибутной схемой вместо “мешка”
- `SAASI_SQLF_Backend` - для реализации с обратным индексом в SAAS и прямым в SQL и т.д.

Хранилище данных - реализация

Общие принципы:

- все бэкенды должны разными способами реализовывать один и тот же программный интерфейс: вести себя как можно ближе к запланированному поведению “идеальной компоненты”
- при замене бэкенда не должно требоваться изменять другие части системы
- в системе есть диспетчер, который после того, как UserQuery сгенерил RepoQuery, определяет по нему, какой бэкенд нужен (потому что разные корпуса или виды запросов могут обрабатываться разными бэкендами), вызывает этот бэкенд и передает ему RepoQuery на вход
- ни UserQuery не знает списка бэкендов, ни бэкеды не знают списка возможных UserQuery, а они общаются только через RepoQuery

Национальный словарный фонд

1. Сайт
2. Поиск по словарям
3. Единое хранилище словарных статей

Заметим:

1. Сайт будут показывать премьеру, а может и президенту => требуется высокое качество дизайна, удобства пользования и т.д. Возможно, потребуется аутсорсинг.
2. На какой платформе строить хранилище словарных статей - не ясно. В процессе реализации проекта требования будут проясняться, но прототип должен быть построен уже на раннем этапе. Возможно, придется менять технологии хранилища в процессе проекта

Национальный словарный фонд

Выводы:

1. Прототип программного обеспечения Национального словарного фонда имеет смысл строить в виде многоуровневого ПО.

2. Уровни ПО НСФ имеет делать аналогичными уровням ПО НКРЯ.

Спасибо за внимание!