Использование собственных событий в OpenLayers

Обсудить в форуме Комментариев — 0

В статье на примере реальной задачи рассмотрен процесс регистрации и обработки собственных событий.

Предположим, перед нами стоит следующая задача: имеется векторный слой (vectors) и некоторое множество объектов (feature) класса OpenLayers.Feature.Vector на нём. Необходимо при изменении расположения (feature.geometry), стиля (feature.style) или названия (feature.attributes.title) сериализовать соответствующие свойства в JSON-объект и передать на сервер. Другими словами, мы хотим сохранять данные непосредственно в процессе их изменения, не создавая дополнительных инструментов типа "Сохранить данные".

В принципе нас не интересуют детали того, как именно эти свойства будут изменяться (с помощью каких-либо контролов или в ручную или еще как-нибудь), для нас важно научиться отслеживать и должным образом обрабатывать эти события.

По умолчанию при изменении геометрии в OpenLayers происходит срабатывание стандартного события "featuremodified", нативных же событий, срабатывающих при изменении стиля или какого-нибудь из свойств объекта feature.attributes не предусмотрено.

Придумаем названия для события изменения стиля - "stylemodified", изменения аттрибута title - "titlemodified". Переходим к регистрации событий (небольшая справка по использованию событий в OpenLayers доступна здесь).

```
vectors.events.register("featuremodified", null, update)
vectors.events.register("stylemodified", null, update)
vectors.events.register("titlemodified", null, update)
```

В данном примере update - это callback-функция, вызываемая при срабатывании события. Обратите внимание на то, что во всех трёх случаях используется одна и та же callback-функция. Конечно, можно было для каждого события написать свой обработчик, но так как фактически всё время будет выполнятся одна и та же задача - передача данных на сервер с разницей лишь в самих данных, то в таком случае код, отвечающий за передачу информации на сервер дублировался бы трижды.

Так как "featuremodified" - стандартное событие, то вызывать нам вручную его не требуется, соответствующие вызовы уже расставлены внутри кода OpenLayers. Вызывать же события "stylemodified" и "titlemodified" придётся вручную.

Предположим, что за изменение стиля отвечает некоторая функция changeStyle, а за изменение названия changeTitle. Тогда вызовы соответствующих событий должны быть добавлены в конец этих функций:

```
function changeStyle(feature) {
    ...
    feature.layer.events.triggerEvent("stylemodified", {feature: feature});
}

function changeTitle(feature) {
    ...
    feature.layer.events.triggerEvent("titlemodified", {feature: feature});
}
```

Второй аргумент функции triggerEvent - объект, который будет передаваться в callback-функцию. Аналогичным образом стандартное событие "featuremodified" передаёт в callback объект, свойство feature которого - есть ссылка на модифицированный объект, подробнее здесь.

Как же теперь внутри callback-а update определить от какого именно из событий пришли данные, чтобы соответствующим образом сформировать строку, которую нужно передать на сервер, ведь мы вроде-бы подаём на вход только объект {feature: feature} и он не содержит в себе имени типа события. Однако, если залезть в исходники OpenLayers и найти определение функции triggerEvent, то можно увидеть следующее:

```
triggerEvent: function (type, evt) {
    ...
    if(!evt.type) {
        evt.type = type;
    }
    ...
}
```

То есть в объект (evt), подающийся на вход callback-у добавляется свойство type, совпадающее с именем события.

Используя полученные знания, напишем callback-функцию update:

```
json = new OpenLayers.Format.JSON();
geojson = new OpenLayers.Format.GeoJSON();

function update(evt) {
    switch (evt.type) {
        case "featuremodified":
            data = '{"geom":'+geojson.write(evt.feature.geometry)+'}';
            break;
        case "stylemodified":
            data = '{"style":'+json.write(evt.feature.style)+'}';
            break;
        case "titlemodified":
            data = '{"title":'+json.write(evt.feature.attributes.title)+'}';
            break;
    }
    OpenLayers.Request.PUT({
        url: "your_url/" + evt.feature.attributes.id,
        data: data
    });
}
```

Вот и всё, ничего сложного.

Дополнение

Не возникло ли у вас вопроса, почему у функции register второй аргумент равен null и для чего он в принципе может использоваться? Попробуем разобраться. Второй аргумент в терминах JavaScript - это scope, в контексте которого будет вызываться callback-функция обработчик.

Поясним на примере:

```
vectors.events.register("featuremodified", {"type": "geometry"}, update)
```

То есть при вызове функции update её this будет ссылаться на объект:

```
{"type": "geometry"}
```

В этом случае update будет выглядеть следующим образом:

```
function update(evt) {
```

```
    switch (this.type) {
        case "geometry":
            data = '{"geom":'+geojson.write(evt.feature.geometry)+'}';
            break;
      case "style":
          data = '{"style":'+json.write(evt.feature.style)+'}';
          break;
      case "title":
          data = '{"title":'+json.write(evt.feature.attributes.title)+'}';
          break;
    }
    OpenLayers.Request.PUT({
        url: "your_url/" + evt.feature.attributes.id,
        data: data
    });
}
```

А функции изменения стиля и названия:

```
function changeStyle(feature) {
    ...
    update.call({"type": "style"}, {feature: feature})
}

function changeTitle(feature) {
    ...
    update.call({"type": "title"}, {feature: feature})
}
```

Метод call - стандартная возможность JavaScript, позволяющая вызывать функции в контексте нужного объекта.

Обсудить в форуме Комментариев — 0

Последнее обновление: 2014-05-15 00:11

Дата создания: 27.06.2012
Автор(ы): Денис Рыков