Opérations CRUD

Store

Récupération du store, dans un contrôleur ou un route handler :

```
let store=this.get('store');
```

Read : chargement d'enregistrements

1 enregistrement

Chargement d'une instance de **person**

```
let person = this.get('store').findRecord('person', 1);
```

Recherche d'une instance de **person** déjà chargée (ne retourne que si elle est déjà présente dans le store, sans effectuer de requête vers le serveur) :

```
let person = this.get('store').peekRecord('person', 1);
```

Plusieurs enregistrements

Chargement de toutes les instances de **person** :

```
let persons = this.get('store').findAll('person');
```

Retour des instances de **person** déjà chargée dans le store :

```
let persons = this.get('store').peekAll('person');
```

Recherche par critères

De plusieurs enregistrements

```
this.get('store').query('person', {
  filter: {
    name: 'Peter'
  }
}).then(function(peters) {
  // Do something with `peters`
```

```
});
```

D'un seul enregistrement

```
this.get('store').queryRecord('person', {
  filter: {
    id: 1234
  }
}).then(function(user) {
  // Do something with `user`
});
```

Create

Toutes les opérations de modification (ajout, modification, suppression) fonctionnent en 2 étapes :

- Réalisation de la modification (appel de createRecord, deleteRecord, updateRecord...)
- Mise à jour sur le serveur (save)

Création d'une instance

```
let post = store.createRecord('post', {
  title: 'Rails is Omakase',
  body: 'Lorem ipsum'
});
```

Mise à jour de la création (persistance) :

```
post.save();
```

Update

Mise à jour de l'instance avec setter et persistance :

```
this.get('store').findRecord('person', 1).then(function(tyrion) {
  // ...after the record has loaded
  tyrion.set('firstName', 'Yollo');
  tyrion.save();
});
```

Delete

Suppression d'une instance et persistance :

```
  post.deleteRecord();
  post.get('isDeleted'); // => true
  post.save(); // => DELETE to /posts/1
```

Il est possible d'utiliser destroyRecord pour effectuer directement la suppression :

```
  post.destroyRecord();
```

Flags, dirty attributes et annulation

Tant que la méthode **save** n'est pas appelée, la persistance n'est pas réalisée, et les instances modifiées portent les flags mentionnant leurs modifications :

- isDirty
- isDeleted
- isNew

```
person.get('isAdmin');            // => false
person.get('hasDirtyAttributes'); // => false
person.set('isAdmin', true);
person.get('hasDirtyAttributes'); // => true
person.changedAttributes();       // => { isAdmin: [false, true] }
```

Il est possible d'annuler les modifications avec **rollbackAttributes** tant que l'appel de la méthode **save** n'a pas été effectué.

```
person.get('hasDirtyAttributes'); // => true
person.changedAttributes();       // => { isAdmin: [false, true] }

person.rollbackAttributes();

person.get('hasDirtyAttributes'); // => false
person.get('isAdmin');            // => false
person.changedAttributes();       // => {}
```

Erreurs de validation

Côté router, il est possible d'utiliser le **error hook** pour intercepter les erreurs de chargement/mise à jour,

```
import Route from '@ember/routing/route';

export default Route.extend({
  model(params) {
    return this.get('store').findAll('privileged-model');
  },
```

```
  actions: {
    error(error, transition) {
      if (error.status === '403') {
        this.replaceWith('login');
      } else {
        // Let the route above this handle the error.
        return true;
      }
    }
  }
});
```

Si le serveur retourne des erreurs de validation au moment du save, la propriété **errors** du model permet de les afficher :

```
{{#each post.errors.title as |error|}}
  <div class="error">{{error.message}}</div>
{{/each}}
{{#each post.errors.body as |error|}}
  <div class="error">{{error.message}}</div>
{{/each}}
```

Il existe également un **loading event** permettant d'intervenir sur le chargement :

```
import Route from '@ember/routing/route';

export default Route.extend({
  ...
  actions: {
    loading(transition) {
      let start = new Date();
      transition.promise.finally(() => {
        this.get('notifier').notify(`Took ${new Date() - start}ms to load`);
      });

      return true;
    }
  }
});
```

Promises

Toutes les méthodes du store interrogeant le serveur (find, findAll, query, save...) retournent une “promise” permettant d'intercepter la fin du chargement, ou les erreurs éventuelles.

```
let persons=this.get('store').findAll('person').then(function(datas){
    //les données datas sont chargées
    }).catch(function(reason){
```

```
    //Erreurs de chargement
    });
```

```
let post = store.createRecord('post', {
  title: 'Rails is Omakase',
  body: 'Lorem ipsum'
});

let self = this;

function transitionToPost(post) {
  self.transitionToRoute('posts.show', post);
}

function failure(reason) {
  // handle the error
}

post.save().then(transitionToPost).catch(failure);
```