Ember Model Aufbau

```
App.Person = DS.Model.extend({
  firstName: DS.attr('string'),
  lastName: DS.attr('string'),
  isPersonOfTheYear:
DS.attr('boolean')
});
```

Aufbau eines Models

DS.attr defaultValue

String	DS.attr('boolean', {defaultValue: false})
Funktion	DS.attr('string', { defaultValue: function() { return new Date(); }})

Der zweite optionale Parameter steht für den "defaultValue". Kann ein String oder Funktion sein.

Custom Transformations

Transformations bieten die Möglichkeit neue Datentypen anzulegen die sich auf ein Model Attribute beziehen.

http://emberjs.com/guides/models/the-rest-adapter/#toc_creating-custom-transformations

Rekursion gutes Beispiel

http://emberjs.com/guides/models/persisting-records/

Am Ende der Seite letzte Funktion.

test

isPersonOfTheYear	Mehrer Wörter camilezied

Ist der Attribut Name im Model camelized, muss dieser auch von der API camelized

Store find

this.store.find('post', 1)	GET /posts/1
this.store.find('person', { name: "Peter" });	GET /persons?name='Peter'

Liefert ein DS.PromiseArray zurück was zu einem DS.RecordArray wird, wenn alle Daten im Store sind.

Record Methods

person.get('isDirty')	true/false Ist ein Wert geändert, aber noch nicht gespeichert
person.changedAttributes()-	welche werte wurden geändert
person.rollback()	setzt geändert (aber noch nicht gespeicherte??) werte zurück;

Alle Methoden beziehen sich auf einen Record

EXPLICIT INVERSES

Bei mehreren HasMany Beziehungen zum gleichen Model kann mit expliziten Inverses Ember gesagt werden welches Attribut aktualisiert werden muss.

http://emberjs.com/guides/models/defining-models/#toc_explicit-inverses

Etwas anlegen

store.createRecord()

> Eintrag erstellen

Wichtig. Alle übergebenen Daten müssen zum Zeitpunkt der Erstellung
existieren. Siehe auch:
http://emberjs.com/guides/models/creating-and-deleting-records/

Etwas loeschen

```
var post = store.find('post', 1);
post.deleteRecord();
post.get('isDeleted'); // => true
post.save(); // => DELETE to
/posts/1
// ODER
var post = store.find('post', 1);
post.destroyRecord(); // => DELETE
to /posts/1
```

Der Unterschied der Funktion ist: 1. Solange bei der ersten Möglichkeit nicht das save() ausgeführt wurde ist die Löschung nur auf der Client Site, und wird nicht an den Server geschickt. Funktion zwei ist die Kurzform von 1